Enterprise COBOL for z/OS and OS/390

プログラミング・ガイド

バージョン 3.1

SC88-9121-00
(英文原典：SC27-1412-00)

Enterprise COBOL for z/OS and OS/390

プログラミング・ガイド

バージョン 3.1

SC88-9121-00
(英文原典：SC27-1412-00)

お願い!

本書、および本書がサポートする製品をご使用になる前に、 755 ページの『特記事項』にある一般的な情報を必ずお読みください。

本書は、IBM Enterprise COBOL for z/OS and OS/390 のバージョン 3 リリース 1 (プログラム番号 5655-G53) に適用されます。また、新版において特に断りのない限り、それ以降のすべてのリリースおよび修正レベルにも適用されます。製品のレベルに応じた正しい版を使用していることを確認してください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原　典： SC27–1412–00
Enterprise COBOL for z/OS and OS/390
Programming Guide
Version 3 Release 1

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2002.3

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1991, 2001. All rights reserved.

© Copyright IBM Japan 2002

目次

本書について xiii
本書の使い方 xiii
略語 xiv
一般に使用される用語の比較 xiv
構文図の読み方 xv
例の示し方 xvi
改訂の要約 xvi

第 1 部 プログラムのコーディング . . 1

第 1 章 プログラムの構造 7
プログラムの識別. 7
プログラムを再帰的として識別する. 8
収容プログラムによってプログラムに呼び出し可能のマークを付ける. 8
プログラムを初期状態に設定する 9
ソース・リストのヘッダーの変更 9
コンピューター環境の記述. 9
例: FILE-CONTROL の記入項目 10
照合シーケンスの指定. 10
シンボリック文字を定義する 12
ユーザー定義のクラスを定義する 12
オペレーティング・システムに対してファイルを定義する 12
データの記述. 15
入出力操作でのデータの使用 15
WORKING-STORAGE と LOCAL-STORAGE の比較. 18
別のプログラムからのデータの使用 19
データの処理. 21
PROCEDURE DIVISION 内でロジックが分割される方法 21
宣言. 25

第 2 章 データの使用 27
変数、構造、リテラル、および定数の使用 27
変数. 27
データ構造: データ項目とグループ項目 27
リテラル 28
定数. 28
形象定数 29
データ項目への値の割り当て 29
例: 変数の初期化 30
構造の初期化 (INITIALIZE) 31
変数または構造への値の割り当て (MOVE) . . . 32
算術結果の割り当て (MOVE または COMPUTE) 32
画面またはファイルからの入力の割り当て (ACCEPT) 33
画面上またはファイル内での値の表示 (DISPLAY) 34
システム論理出力装置上でのデータの表示 . . . 35
WITH NO ADVANCING の使用 36
組み込み関数の使用 (組み込み関数) 37
組み込み関数の型 37
関数のネスト. 38
テーブル (配列) とポインターの使用. 38
ストレージとそのアドレス可能度 38
RMODE の設定. 40
データの受け渡しに関するストレージ制限 . . . 40
データ域のロケーション. 40
外部データ用のストレージ 41
QSAM 入出力バッファー用のストレージ . . . 41

第 3 章 数値および算術演算 43
数値データの定義 43
数値データの表示 44
数値データの保管方法の制御 45
数値データの形式 46
外部 10 進数 (DISPLAY) 項目 46
外部浮動小数点 (DISPLAY) 項目 47
2 進数 (COMP) 項目 47
固有 2 進数 (COMP-5) 項目. 48
パック 10 進数 (COMP-3) 項目 48
浮動小数点 (COMP-1 および COMP-2) 項目 . . 49
例: 数値データおよび内部表現 49
データ形式の変換 50
変換および精度 51
符号表現および処理 52
NUMPROC(PFD). 52
NUMPROC(NOPFD). 52
NUMPROC(MIG). 52
非互換データの検査 (数値のクラス・テスト) . . . 53
算術の実行 54
COMPUTE およびその他の算術ステートメント 54
算術式 55
数字組み込み関数 55
関数および算術式のネスト 56
ALL 添え字付けおよび特殊レジスター 56
数学および日付の言語環境プログラム・サービス 56
例: 数字組み込み関数 58
一般数値処理. 59
日付および時刻 59
金融. 59
数学. 60
統計. 60
固定小数点演算と浮動小数点演算の対比 60
浮動小数点計算 61
固定小数点計算 61
算術比較 (比較条件) 61
例: 固定小数点計算および浮動小数点計算 . . . 62
通貨記号の使用 63
例: 複数の通貨記号. 64

© Copyright IBM Corp. 1991, 2001

第 4 章 テーブルの処理 65
テーブルの定義 (OCCURS) 65
テーブルのネスト 66
添え字付け 67
指標付け 67
テーブルでの項目の参照 68
添え字付け 68
指標付け 69
テーブルに値を入れる方法 70
テーブルの動的なロード 70
テーブルの初期化 (INITIALIZE) 71
テーブルの定義時の値の割り当て (VALUE). . . 71
例: PERFORM と添え字付け 72
例: PERFORM と指標付け 73
可変長テーブルの作成 (DEPENDING ON) 74
可変長テーブルのロード 76
可変長テーブルへの値の割り当て 77
テーブルの探索 77
シリアル検索 (SEARCH) 78
二分探索 (SEARCH ALL). 79
組み込み関数を使用したテーブル項目の処理 . . . 80
例: 組み込み関数 81

第 5 章 プログラム・アクションの選択と反復 83
プログラム・アクションの選択 83
アクションの選択項目のコーディング 83
条件式のコーディング 88
プログラム・アクションの反復 91
インラインまたはライン外 PERFORM の選択 . . 91
ループのコーディング 92
テーブルを操作するループのコーディング . . . 93
複数の段落またはセクションの実行 94

第 6 章 ストリングの処理 95
データ項目の結合 (STRING). 95
例: STRING ステートメント. 96
データ項目の分割 (UNSTRING). 97
例: UNSTRING ステートメント 98
ヌル終了ストリングの処理 100
例: ヌル終了ストリング 100
データ項目のサブストリングの参照 101
参照修飾子 102
例: 参照修飾子としての演算式 103
例: 参照修飾子としての組み込み関数 103
データ項目の計算および置換 (INSPECT) 104
例: INSPECT ステートメント 104
データ項目の変換 (組み込み関数) 105
大文字または小文字への変換 (UPPER-CASE、LOWER-CASE) 105
逆順への変換 (REVERSE) 106
数値への変換 (NUMVAL、NUMVAL-C) . . . 106
データ項目の評価 (組み込み関数) 107
照合シーケンスに関する単一文字の評価 . . . 107
最大または最小データ項目の検出. 108
データ項目の長さの検出. 110
コンパイルの日付の検出. 110

第 7 章 実行時に各国語を使用するためのコーディング 113
Unicode および言語文字のエンコード 115
COBOL での国別データ (Unicode) の使用. . . . 116
国別データ項目. 116
国別リテラル 116
形象定数としての国別文字 117
国別データのストレージ. 117
国別データの変換 118
英数字および整数の国別データへの変換 (MOVE) 118
英数字データの国別データへの変換 (NATIONAL-OF) 118
国別データの英数字データへの変換 (DISPLAY-OF) 118
変換例外 119
例: 国別データの変換 119
UTF-8 データの処理 120
中国語 GB 18030 データの処理 121
国別データ項目の比較 121
国別オペランドの比較 122
国別オペランドと数値オペランドの比較 . . . 122
国別オペランドと英字または英数字オペランドの比較 122
国別オペランドとグループ・オペランドの比較 122
DBCS データを含む英数字データ項目の処理 . . . 123

第 8 章 ファイルの処理 125
ファイル編成および入出力装置 125
ファイル編成およびアクセス・モードの選択 . . . 127
入出力のコーディング形式 128
ファイルの割り振り 129
入出力エラーの検査 131

第 9 章 QSAM ファイルの処理 133
COBOL での QSAM ファイルおよびレコードの定義 133
レコード形式の指定 134
ブロック・サイズの設定. 142
QSAM ファイルの入出力ステートメントのコーディング. 145
QSAM ファイルのオープン. 145
CBLQDA による QSAM ファイルの動的作成 146
QSAM ファイルへのレコードの追加. 147
QSAM ファイルの更新 147
QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み 147
QSAM ファイルのクローズ. 148
QSAM ファイルのエラーの処理 149
QSAM ファイルに対する作業 150
QSAM ファイルの定義および割り振り 150
QSAM ファイルの検索 152
ファイル属性をプログラムと一致させる . . . 154

ストライプ拡張形式 QSAM データ・セットの使用 156
QSAM を使用する HFS ファイルへのアクセス . . 157
QSAM ファイルのラベル 158
トレーラー・ラベルおよびヘッダー・ラベルの使用 159
標準ラベルの形式 160
テープ上の QSAM ASCII ファイルの処理. . . . 161
ASCII アルファベットの要求 162
レコード形式の定義 162
DD 名の定義 162
ASCII ファイルのラベルの処理 163

第 10 章 VSAM ファイルの処理 . . . 165

VSAM ファイル 166
VSAM ファイル編成およびレコードの定義 . . . 168
VSAM ファイルの順次編成の指定方法 168
VSAM ファイルの索引編成の指定方法 169
VSAM ファイルの相対編成の指定方法 170
VSAM ファイルのアクセス・モードの指定 . . 172
VSAM ファイルのレコード長の定義. 173
VSAM ファイルの入出力ステートメントのコーディング. 174
ファイル位置標識 176
ファイルのオープン (ESDS、KSDS、または RRDS). 176
VSAM ファイルからのレコードの読み取り . . 179
VSAM ファイル内のレコードの更新. 180
VSAM ファイルへのレコードの追加. 181
VSAM ファイル内のレコードの置換. 182
VSAM ファイルからのレコードの削除 182
VSAM ファイルのクローズ. 183
VSAM ファイルのエラーの処理 183
パスワードによる VSAM ファイルの保護. . . . 184
例: VSAM 索引付きファイルのパスワード保護 185
z/OS および UNIX のもとでの VSAM データ・セットの操作 185
VSAM ファイルの定義 186
代替索引の作成. 187
VSAM ファイルの割り振り. 189
RLS による VSAM ファイルの共用. 191
VSAM パフォーマンスの向上 193

第 11 章 行順次ファイルの処理 195

COBOL での行順次ファイルおよびレコードの定義 195
許可される制御文字 196
行順次ファイルの構造の記述 196
行順次ファイルの定義および割り振り 197
行順次ファイル用の入出力ステートメントのコーディング. 198
行順次ファイルのオープン 198
行順次ファイルからのレコードの読み取り. . . 199
行順次ファイルへのレコードの追加 199
行順次ファイルのクローズ 200
行順次ファイルのエラーの処理 201

第 12 章 ファイルのソートおよびマージ 203

ソートおよびマージ・プロセス 204
ソートまたはマージ・ファイルの記述 204
ソートまたはマージへの入力の記述 205
例: SORT 用のソート・ファイルおよびの入力ファイルの記述 206
入力プロシージャーのコーディング 207
ソートまたはマージからの出力の記述 208
出力プロシージャーのコーディング 208
DFSORT を使用する際のコーディングの考慮事項 209
例: DFSORT を使用する際の出力プロシージャーのコーディング 209
入出力プロシージャーに関する制約事項 210
ソートおよびマージ・データ・セットの定義 . . . 210
可変長レコードのソート. 211
ソートまたはマージの要求 211
ソートまたはマージ基準の設定 212
例: 入出力プロシージャーを使用したソート . . 213
代替照合シーケンスの選択 214
ウィンドウ化日付フィールドに基づいたソート 214
同じキーを持つレコードのオリジナル・シーケンスの保持 215
ソートまたはマージの成否の判断. 215
ソートまたはマージ操作の途中停止 216
FASTSRT を使ってのソートのパフォーマンスの向上 217
JCL に関する FASTSRT の要件 217
ソート入出力ファイルに関する FASTSRT の要件 217
NOFASTSRT によるソート・エラーの検査 . . . 219
ソート動作の制御 220
ソート特殊レジスター 221
制御ステートメントによる DFSORT デフォルトの変更. 221
ソートまたはマージ操作のためのストレージの割り振り. 222
ソート・ファイル用のスペースの割り振り. . . 223
DFSORT によるチェックポイント・リスタートの使用 223
CICS のもとでのソート 224
CICS SORT アプリケーションの制約事項 . . . 224

第 13 章 XML 文書の処理 225

COBOL でのXML パーサー 225
XML 文書へのアクセス 227
XML 文書の構文解析. 227
XML イベントの処理. 228
XML を処理するためのプロシージャーの作成 234
XML 文書のエンコードについての理解. 240
コード・ページの指定 240
その他のコード・ページを使用した文書の構文解析 241
XML 文書のエラー処理 241
未処理の例外 243

例外の処理 243
構文解析の終了. 244
CCSID 矛盾例外 244

第 14 章 エラーの処理 247
ダンプの要求 247
定様式ダンプの作成 247
システム・ダンプの作成. 248
ストリングの結合および分割におけるエラーの処理 248
算術演算でのエラーの処理 249
例: 0 による除算の検査 249
入出力操作でのエラーの処理 250
ファイル終わり条件 (AT END) の使用. . . . 252
ERROR 宣言のコーディング 253
ファイル状況キーの使用. 254
例: ファイル状況キー 255
VSAM 戻りコードの使用 (VSAM ファイルのみ) 256
例: VSAM 状況コードの検査 256
INVALID KEY 句のコーディング 258
例: FILE STATUS および INVALID KEY. . . 258
プログラム呼び出し時のエラーの処理 259
エラー処理用のルーチンの作成 259

第 2 部 プログラムのコンパイルおよびデバッグ 261

第 15 章 z/OS のもとでのコンパイル 263
JCL を使用したコンパイル. 264
カタログ式プロシージャーの使用. 264
プログラムをコンパイルするための JCL の作成 274
TSO のもとでのコンパイル. 276
例: TSO のもとでコンパイルするための ALLOCATE および CALL 277
例: TSO のもとでコンパイルするための CLIST 277
アセンブラー・プログラムからコンパイラーを開始する 278
コンパイラー入出力の定義 279
z/OS のもとでコンパイラーによって使用されるデータ・セット. 280
ソース・コード・データ・セットの定義 (SYSIN) 282
ソース・ライブラリーの指定 (SYSLIB). . . . 283
出力データ・セットの定義 (SYSPRINT) . . . 283
コンパイラー・メッセージの端末への送信 (SYSTERM) 283
オブジェクト・コードの作成 (SYSLIN または SYSPUNCH). 284
関連データ・ファイルの作成 (SYSADATA) . . 284
出力 Java データ・セット (SYSJAVA) の定義 285
デバッグ・データ・セットの定義 (SYSDEBUG) 285
z/OS のもとでのコンパイラー・オプションの指定 285
PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定. 286
例: JCL を使用したコンパイラー・オプションの指定 287
例: TSO のもとでのコンパイラー・オプションの指定. 287
z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力 287
複数プログラムのコンパイル (バッチ・コンパイル) 289
例: バッチ・コンパイル 290
バッチ・コンパイルでのコンパイラー・オプションの指定 291
例: バッチ・コンパイルでのオプションの優先順位 292
例: バッチ・コンパイルでの LANGUAGE オプション. 292
ソース・プログラムのエラーの訂正 293
コンパイル・エラー・メッセージのリストの生成 294
コンパイラー検出エラーに関するメッセージおよびリスト 294
コンパイル・エラー・メッセージの形式 . . . 295
コンパイル・エラー・メッセージの重大度コード 296

第 16 章 UNIX のもとでのコンパイル 299
UNIX のもとでの環境変数の設定. 299
UNIX のもとでのコンパイラー・オプションの指定 300
cob2 コマンドを使用したコンパイルおよびリンク 301
入出力の定義 302
DLL の作成. 302
例: UNIX のもとでコンパイルするための cob2 の使い方 303
cob2 303
cob2 入出力ファイル. 305
スクリプトを使用したコンパイル. 306

第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行. . . 307
UNIX のもとでのオブジェクト指向アプリケーションの準備 308
UNIX 環境外でのオブジェクト指向プログラムのコンパイル 309
オブジェクト指向プログラムの実行 309

第 18 章 コンパイラー・オプション 311
COBOL 85 標準に準拠するオプション設定値. . . 313
矛盾するコンパイラー・オプション 313
ADATA 315
ADV 315
ARITH. 316
AWO 316
BUFSIZE 317
CICS 317
CODEPAGE 318
COMPILE. 319
CURRENCY 320
DATA 321
DATEPROC 322
DBCS 323
DECK 323
DIAGTRUNC 324

DLL 324
DUMP 325
DYNAM 326
EXIT 327
EXPORTALL 327
FASTSRT 327
FLAG 328
FLAGSTD 329
INTDATE 330
LANGUAGE 331
LIB 332
LINECOUNT 332
LIST 333
MAP 333
NAME 334
NSYMBOL 335
NUMBER 336
NUMPROC 336
OBJECT 338
OFFSET 338
OPTIMIZE 339
未使用データ項目 339
OUTDD 340
PGMNAME 340
PGMNAME(COMPAT) 341
PGMNAME(LONGUPPER) 341
PGMNAME(LONGMIXED) 342
QUOTE/APOST 343
RENT 343
RMODE 344
SEQUENCE 345
SIZE 346
SOURCE 346
SPACE 347
SQL 347
SSRANGE 348
TERMINAL 349
TEST 349
THREAD 352
TRUNC 353
TRUNC の例 1 355
TRUNC の例 2 355
VBREF 356
WORD 357
XREF 357
YEARWINDOW 358
ZWB 359
コンパイラー指示ステートメント 360

第 19 章 デバッグ 365
ソース言語によるデバッグ 366
プログラム・ロジックのトレース 366
入出力エラーの検出および処理 367
データの妥当性検査 367
初期化されていないデータの検出 368
プロシージャーに関する情報の生成 368
コンパイラー・オプションを使用したデバッグ . . 370
コーディング・エラーの検出 371
行シーケンス問題の検出 372
有効範囲の検査 372
診断するエラーのレベルの選択 373
プログラム・エンティティー定義および参照の検出 375
データ項目のリスト 376
リストの入手 377
例: 短縮リスト 379
例: SOURCE および NUMBER 出力 381
例: 組み込みマップ要約 383
MAP 出力で使用される用語 384
LIST および MAP 出力で使用される記号 . . . 385
例: ネストされたプログラム・マップ 386
LIST 出力の読み取り 386
例: XREF 出力 - データ名相互参照 399
例: XREF 出力 - プログラム名相互参照 . . . 400
例: 組み込み相互参照 400
例: OFFSET コンパイラー出力 401
例: VBREF コンパイラー出力 402
デバッガーの使用準備 402

第 3 部 特定の環境に合わせた COBOL プログラムの目標 405

第 20 章 COBOL プログラムの開発 (CICS の場合) 407
CICS のもとで実行する COBOL プログラムのコーディング 408
ファイル入出力のコーディング 408
システム日付および時刻の検索 408
データ項目の内容の表示 409
COBOL プログラムとの間の呼び出し 409
ネストされたプログラムのコーディング . . . 410
16MB 境界より上で実行する COBOL プログラムのコーディング 410
ECI 呼び出しの成否の判断 411
CICS オプションを使用したコンパイル 411
一連のプログラムのコンパイル 412
CICS サブオプションの分離 412
組み込みの CICS トランスレーター 413
分離型の CICS トランスレーターの使用 414
CICS 予約語テーブル 416
CICS HANDLE を使用したエラー処理 416
例: CICS HANDLE を使用したエラー処理 . . 417

第 21 章 DB2 環境用のプログラミング 419
SQL ステートメントのコーディング 419
DB2 coprocessor を用いた SQL INCLUDE の使用 420
2 進数項目の使用 420
SQL ステートメントの成否の判断 420
SQL オプションを使用したコンパイル 421
バッチでのコンパイル 421

DB2 サブオプションの分離. 422
DB2 coprocessor 422

第 22 章 IMS のもとでの COBOL プログラムの実行 425
IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク 425

第 23 章 UNIX のもとでの COBOL プログラムの実行 427
UNIX 環境での実行 427
環境変数の設定およびアクセス 429
実行に影響を与える環境変数の設定 429
環境変数のリセット 429
環境変数へのアクセス 429
例: 環境変数へのアクセス 430
UNIX/POSIX API の呼び出し 430
fork、exec および spawn. 431
サンプル 431
メインプログラム・パラメーターへのアクセス . . 432
例: メインプログラム・パラメーターへのアクセス 433

第 4 部 複雑なアプリケーションの構造 435

第 24 章 サブプログラムの使用 437
メインプログラム、サブプログラム、および呼び出し 438
メインプログラムまたはサブプログラムの終了と再入 439
別のプログラムへの制御権の移動. 440
静的呼び出しの作成 441
動的呼び出しの作成 442
静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項 444
静的呼び出しと動的呼び出しの両方の作成. . . 445
例: 静的および動的 CALL ステートメント . . 445
ネストされた COBOL プログラムの呼び出し 447
再帰呼び出しの作成 451
オブジェクト指向プログラムとの間での呼び出し 451
プロシージャー・ポインターと関数ポインターの使用 452
どのタイプのポインターを使用するかの決定 . . 453
C 関数ポインターの呼び出し 453
代替入り口点への呼び出し 453
プログラムを再入可能にする 454

第 25 章 データの共用 455
データの受け渡し 455
呼び出しプログラムの中での引き数の記述. . . 456
呼び出されるプログラムの中でのパラメーターの記述 457
OMITTED 引き数に関するテスト. 457
LINKAGE SECTION のコーディング 458
引き数を受け渡すための PROCEDURE DIVISION のコーディング. 458
受け渡されるデータのグループ化. 459
ヌル終了ストリングの処理 459
チェーン・リストを使用するためのポインターの使用 460
戻りコード情報の受け渡し 463
RETURN-CODE 特殊レジスターの理解. . . . 463
PROCEDURE DIVISION RETURNING . . . の使用 463
CALL . . . RETURNING の指定 464
EXTERNAL 文節によるデータの共用 464
プログラム間でのファイルの共用 (外部ファイル) 464
例: 外部ファイルの使用 465

第 26 章 DLL または DLL アプリケーションの作成 469
ダイナミック・リンク・ライブラリー (DLL) . . . 469
DLL を作成するためのプログラムのコンパイル 470
DLL のリンク 471
特定の DLL のプリリンク 472
例: プロシージャー型 DLL アプリケーションのサンプル JCL 473
DLL での CALL ID の使用 474
HFS での DLL の探索順序. 475
DLL リンケージと動的呼び出しの併用 475
DLL でのプロシージャー・ポインターまたは関数ポインターの使用 476
非 DLL からの DLL の呼び出し. 477
例: 非 DLL からの DLL の呼び出し 478
C/C++ プログラムでの COBOL DLL の使用 . . . 479
OO COBOL アプリケーションでの DLL の使用 480

第 27 章 マルチスレッド化のための COBOL プログラムの準備 481
マルチスレッド化 482
マルチスレッド化サポートのための THREAD の選択 483
マルチスレッド化による制御権移動 484
スレッド化プログラムでの取り消しの使用. . . 484
プログラムの終了 484
COBOL 環境の事前初期設定 484
マルチスレッド化によるファイルの処理 485
ファイル定義ストレージ. 485
ファイル・アクセスの場合の推奨使用法 . . . 485
例: マルチスレッド化によるファイル入出力の使用パターン 486
マルチスレッド化による COBOL 制限の処理. . . 487

第 5 部 オブジェクト指向プログラムの開発 489

第 28 章 オブジェクト指向プログラムの作成 491
例: 口座 492

サブクラス 493
クラスの定義 495
クラスを定義する CLASS-ID 段落 496
クラスを定義する REPOSITORY 段落 497
クラス・インスタンス・データを定義する WORKING-STORAGE SECTION 498
例: クラスの定義 499
クラス・インスタンス・メソッドの定義 500
クラス・インスタンス・メソッドを定義する METHOD-ID 段落 501
クラス・インスタンス・メソッドを定義する INPUT-OUTPUT SECTION 501
クラス・インスタンス・メソッドを定義する DATA DIVISION 502
クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION 503
インスタンス・メソッドのオーバーライド. . . 504
インスタンス・メソッドの多重定義 505
属性 (get および set) メソッドのコーディング 506
例: メソッドの定義 507
クライアントの定義 509
クライアントを定義する REPOSITORY 段落 510
クライアントを定義する DATA DIVISION . . 511
オブジェクト・リファレンスの比較と設定. . . 513
メソッドの呼び出し (INVOKE) 514
クラスのインスタンスの生成と初期化 516
クラスのインスタンスの解放 518
例: クライアントの定義 519
サブクラスの定義 520
サブクラスを定義する CLASS-ID 段落 521
サブクラスを定義する REPOSITORY 段落 . . 521
サブクラス・インスタンス・データを定義する WORKING-STORAGE SECTION 522
サブクラス・インスタンス・メソッドの定義 . . 522
例: サブクラスの定義 (メソッドを含む) . . . 523
ファクトリー・セクションの定義. 524
ファクトリー・データを定義する WORKING-STORAGE SECTION 525
ファクトリー・メソッドの定義 525
プロシージャー指向 COBOL プログラムのラッピング. 528
例: ファクトリーの定義 (メソッドを含む). . . 528

第 29 章 Java メソッドとの通信 . . . 535
JNI サービスへのアクセス 535
Java 例外の取り扱い 537
ローカルおよびグローバル参照の管理 538
Java アクセス制御. 539
Java とのデータ共用 540
COBOL および Java での相互運用可能データ型のコーディング. 540
Java の配列とストリングの宣言 541
Java 配列の取り扱い 542
Java ストリングの取り扱い. 543
例: COBOL で書き込む J2EE クライアント . . . 546

第 6 部 特殊処理 551

第 30 章 割り込みとチェックポイント・リスタート 553
チェックポイントの設定. 553
チェックポイントの設計. 554
チェックポイントの正常実行に関するテスト . . 555
チェックポイント・データ・セットを定義する DD ステートメント 555
チェックポイント時に生成されるメッセージ . . 556
プログラムの再始動 557
自動再始動の要求 557
据え置き再始動の要求 558
据え置き再始動を要求する形式 559
再始動のためのジョブの再実行依頼 560
例: 特定のチェックポイント・ステップでジョブを再始動する 560
例: ステップ再始動を要求する 560
例: ステップ再始動のためにジョブを再実行依頼する 561
例: チェックポイント・リスタートのためにジョブを再実行依頼する 561

第 31 章 2 桁年号の日付の処理 563
2000 年言語拡張 (MLE) 564
言語拡張の原則および目標 564
日付関連ロジック問題の解決 566
世紀ウィンドウの使用 567
内部ブリッジングの使用. 568
全フィールド拡張への移行 569
年先頭型、年単独型、および年末尾型日付フィールドの使用 571
互換性のある日付 572
例: 年先頭型日付フィールドの比較 573
他の日付形式の使用 573
例: 年部分の分離 574
日付としてのリテラルの操作 574
仮定による世紀ウィンドウ 575
非日付の取り扱い 576
トリガーおよび限界の設定 577
例: 限界の使用 578
符号条件の使用. 579
日付によるソートおよびマージ 579
例: 日付および時刻によるソート 580
日付フィールドへの算術の実行 581
ウィンドウ化日付フィールドからのオーバーフローを考慮する 581
評価の順序の指定 582
日付処理の明示的制御 583
DATEVAL の使用. 583
UNDATE の使用 584
例: DATEVAL 584
例: UNDATE 584
日付関連診断メッセージの分析および回避. . . . 585
日付の処理における問題の回避 587

パック 10 進数フィールドでの問題の回避 . . . 587
拡張日付フィールドからウィンドウ化日付フィールドへの移動 587

第 7 部 パフォーマンスおよび生産性の向上 589

第 32 章 プログラムの調整 591
最適プログラミング・スタイルの使用 592
構造化プログラミングの使用 592
一括表示表現 592
シンボリック定数の使用 593
定数計算のグループ化 593
重複計算のグループ化 593
効率的なデータ型の選択 594
計算データ項目 594
一貫性のあるデータ型 595
算術式 595
べき乗計算 595
テーブルの効率的な処理 595
テーブル参照の最適化 597
コードの最適化 599
最適化 599
例: PERFORM プロシージャー統合 601
パフォーマンスを向上させるためのコンパイラー機能の選択 602
パフォーマンスに関連するコンパイラー・オプション 603
パフォーマンスの評価 606
CICS、IMS、または VSAM での効率的な実行 . . 607
CICS 607
IMS 607
VSAM 607

第 33 章 コーディングの簡略化 609
反復コーディングの除去 609
例: COPY ステートメントの使用 610
言語環境プログラム呼び出し可能サービスの使用 611
言語環境プログラム呼び出し可能サービスのサンプル・リスト 612
言語環境プログラム・サービスの呼び出し . . . 613
例: 言語環境プログラム呼び出し可能サービス 614

第 8 部 付録 617

付録 A. 中間結果と算術精度 619
中間結果として使用される用語 620
例: 中間結果の計算 621
固定小数点データと中間結果 621
加算、減算、乗算、および除算 621
指数 622
例: 固定小数点算術演算でのべき乗計算 . . . 624
中間結果の切り捨て 624
2 進数データと中間結果 624
固定小数点算術演算で評価される組み込み関数 . . 625
整数関数 625
混合関数 625
浮動小数点データと中間結果 627
浮動小数点算術演算で評価されるべき乗計算 . . 627
浮動小数点算術演算で評価される組み込み関数 628
非算術ステートメントでの算術式 628

付録 B. 複合 OCCURS DEPENDING ON 631
例: 複合 ODO 631
長さの計算方法 632
ODO オブジェクトの値の設定 632
ODO オブジェクトの値における変更の影響 . . . 632
ODO オブジェクトの値を変更するときの指標エラーの回避 633
可変テーブルにエレメントを追加するときのオーバーレイ回避 634

付録 C. 2 バイト文字セット (DBCS) データの変換 637
DBCS 表記 637
英数字から DBCS データへの変換 (IGZCA2D) . . 637
IGZCA2D の構文 638
IGZCA2D 戻りコード 639
例: IGZCA2D 639
DBCS データから英数字データへの変換 (IGZCD2A) 640
IGZCD2A の構文 640
IGZCD2A 戻りコード 641
例: IGZCD2A 641

付録 D. XML 参照資料 643
続行を許可する XML 例外 643
続行を許可しない XML 例外 648
XML 適合性 652

付録 E. EXIT コンパイラー・オプション 657
ユーザー出口作業域の使用 658
出口モジュールからの呼び出し 659
INEXIT の処理 659
INEXIT のパラメーター・リスト 660
LIBEXIT の処理 661
ネストされた COPY ステートメントによる LIBEXIT の処理 661
LIBEXIT のパラメーター・リスト 663
PRTEXIT の処理 664
PRTEXIT のパラメーター・リスト 664
ADEXIT の処理 665
ADEXIT のパラメーター・リスト 666
出口モジュールのエラー処理 667
EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用 667
INEXIT 667
LIBEXIT 668

PRTEXIT 668
ADEXIT 668
例: INEXIT ユーザー出口 669

付録 F. JNI.cpy 673

付録 G. COBOL SYSADATA ファイル内容 679
SYSADATA ファイルに影響する既存のコンパイラー・オプション. 679
レコード・タイプ 680
例: SYSADATA 682
SYSADATA レコード記述 683
共通ヘッダー・セクション 683
ジョブ識別レコード - X'0000'. 685
ADATA 識別レコード - X'0001' 686
コンパイル単位の開始 / 終了レコード - X'0002' 687
オプション・レコード - X'0010' 687
外部シンボル・レコード - X'0020' 698
構文解析ツリー・レコード - X'0024' 699
トークン・レコード - X'0030'. 713
ソース・エラー・レコード - X'0032' 714
ソース・レコード - X'0038' 714
COPY REPLACING レコード - X'0039' 715
記号レコード - X'0042' 715
記号相互参照レコード - X'0044' 727
ネストされたプログラム・レコード - X'0046' . . 728
ライブラリー・レコード - X'0060' 729
統計レコード - X'0090' 730
EVENTS レコード - X'0120' 731

付録 H. サンプル・プログラム 733
IGYTCARA: バッチ・アプリケーション 733
IGYTCARA の入力データ 734
IGYTCARA によって作成される報告書. . . . 735
IGYTCARA の実行準備 736
IGYTCARB: 対話式プログラム 738
IGYTCARB の実行準備 738
IGYTSALE: ネストされたプログラム・アプリケーション. 740
IGYTSALE の入力データ 742
IGYTSALE によって作成される報告書 744
IGYTSALE の実行準備 748
例示されている言語エレメントおよび概念. . . . 749

特記事項. 755
商標 755

用語集 757

リソース・リスト 785
Enterprise COBOL for z/OS and OS/390. 785
関連資料 785

索引 789

本書について

IBM Enterprise COBOL for z/OS and OS/390 へようこそ。IBM Enterprise COBOL for z/OS and OS/390 は、最新のホスト COBOL コンパイラーです。

IBM COBOL のこのバージョンには、COBOL ビジネス・プロセスと Web 主導型ビジネス・プロセスを以下の方法で統合するための新たな COBOL 機能が追加されています。

- COBOL プログラムのコンポーネント化を単純にし、Java コンポーネントとの相互運用を可能にする
- 標準化された形式 (XML や Unicode など) のデータの交換および使用を促進する

本書において、『Enterprise COBOL』は IBM Enterprise COBOL for z/OS and OS/390 を指しています。

本書において、『z/OS』は OS/390 バージョン 2 リリース 10 および z/OS バージョン 1 リリース 1 を指しています。

本書の使い方

本書は、Enterprise COBOL プログラムの作成およびコンパイルに役立ちます。さらに、オブジェクト指向クラスおよびメソッドの定義、メソッドの呼び出し、およびプログラムでのオブジェクトの参照を行うのに役立ちます。

本書は、アプリケーション・プログラムの開発の経験と、COBOL に関する多少の知識を前提としています。本書では、COBOL 言語の定義ではなく、プログラミングの目的に合った Enterprise COBOL の使用に重点を置いています。COBOL 構文に関する完全な情報については、*IBM Enterprise COBOL 言語解説書* を参照してください。

Enterprise COBOL へのプログラムの移行については、*IBM Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド* を参照してください。

IBM z/OS 言語環境プログラムが、Enterprise COBOL プログラムを実行するのに必要な実行時環境と実行時サービスを提供します。プログラムのリンク・エディットおよび実行については、*IBM z/OS 言語環境プログラム プログラミング・ガイド* および *IBM z/OS 言語環境プログラム プログラミング・リファレンス* を参照してください。

一般に使用される Enterprise COBOL 用語と IBM z/OS 言語環境プログラム用語の比較については、xiv ページの『一般に使用される用語の比較』を参照してください。

© Copyright IBM Corp. 1991, 2001

略語

本書では、短縮形で使用される用語があります。本書で最も頻繁に使用される製品名の省略形を以下にリストします。

使用される用語	長形式
CICS	CICS Transaction Server
Enterprise COBOL	IBM Enterprise COBOL for z/OS and OS/390
IMS	IMS/ESA
言語環境プログラム	IBM z/OS 言語環境プログラム
MVS	MVS/ESA

本書では、OS/390 バージョン 2 リリース 10 と z/OS バージョン 1 リリース 1 をまとめて『z/OS』と呼びます。

本書では、これらの略語のほかに、「COBOL 85 標準」という用語が使用されています。この用語は、以下の規格の組み合わせを示します。

- ISO 1989:1985、プログラム言語 - COBOL
- ISO/IEC 1989/AMD1:1992、プログラム言語 - COBOL - 改訂 1: 組み込み関数モジュール
- ISO/IEC 1989/AMD2:1994、プログラム言語 - COBOL - 改訂 2: COBOL 用修正および説明改訂
- X3.23-1985、情報システム用米国標準規格 - プログラミング言語 - COBOL
- X3.23a-1989、情報システム用米国標準規格 - プログラミング言語 - COBOL 用組み込み関数モジュール
- X3.23b-1989、情報システム用米国標準規格 - プログラミング言語 - COBOL 用修正改訂

2 つの ISO 規格は、米国標準規格と一致しています。

その他の用語 (一般に理解されていない場合) は、初出時に*イタリック体* で示され、巻末の用語集にリストされています。

一般に使用される用語の比較

IBM z/OS 言語環境プログラムおよび IBM Enterprise COBOL for z/OS and OS/390 の資料で使用される用語、およびどの用語が同等であるかを理解するためには、以下の表を参照してください。

言語環境プログラム用語	Enterprise COBOL での同等語
集合体	グループ項目
配列	OCCURS 文節を使用して作成されたテーブル
配列エレメント	テーブル・エレメント
エンクレーブ	実行単位
EXTERNAL データ	EXTERNAL 文節を使用して定義される WORKING-STORAGE データ

言語環境プログラム用語	Enterprise COBOL での同等語
ローカル・データ	非 EXTERNAL データ項目
値によるパラメーターの直接受け渡し	BY VALUE
参照によるパラメーターの間接受け渡し	BY REFERENCE
値によるパラメーターの間接受け渡し	BY CONTENT
ルーチン	プログラム
スカラー	基本項目

構文図の読み方

構文図を読み取るには、以下の規則に従ってください。

- 構文図は線の経路に沿って、左から右に、上から下に見るようになっています。

  ▶▶ 記号は、ステートメントの始まりを示します。

  → 記号は、ステートメントの構文が次の行に続くことを示します。

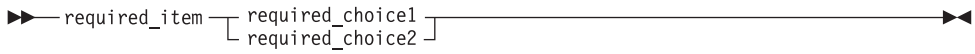
  ▶ 記号は、ステートメントが前の行から続いていることを示します。

  →◀ 記号は、ステートメントの終わりを示します。

  完全なステートメントではない構文単位の図は、▶ 記号で始まり、→ 記号で終わります。
- 必須項目は、水平線 (幹線) と同じ高さに示されます。

- オプション項目は、幹線の下側に示されます。

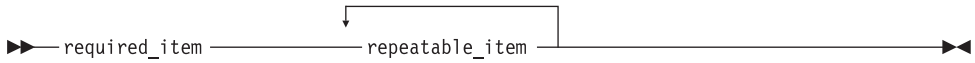
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。それらの項目のうち 1 つを選択しなければならない場合には、それらの項目のうちの 1 つが幹線上に示されます。



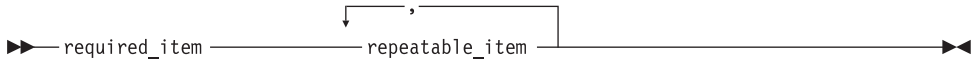
  項目の選択が任意である場合には、それらの項目全体が幹線より下に置かれます。

  項目のうちの 1 つがデフォルトであれば、それが幹線より上に示され、残りの選択項目は幹線より下に示されます。

- 幹線の上を左に戻る矢印は、その項目を繰り返して指定できることを示しています。



反復矢印にコンマが含まれている場合は、繰り返す項目をコンマで区切って指定する必要があります。



- キーワードは大文字で示されています (たとえば、FROM)。キーワードは、示されているとおりに入力しなければなりません。変数は小文字のイタリックで示されます (たとえば、*column-name*)。それらの変数は、ユーザーが指定する名前や値を表します。
- 句読記号、括弧、算術演算子などの記号が示されている場合には、これらも構文の一部として入力しなければなりません。

例の示し方

本書では、説明する概念を示すのに役立つように、サンプル COBOL ステートメント、プログラムの一部分、および小さなプログラムの例を多く示しています。プログラム・コードの例は、小文字、大文字、または大文字小文字混合で書かれており、これらのいずれでもプログラムを作成できることを示しています。

例を説明テキストと明確に区別する場合は、例は**モノスペース・フォント**で示されます。

テキストに現れる COBOL キーワードとコンパイラー・オプションは、通常は、SMALL UPPERCASE (小さい英大文字フォント) で示されます。プログラム変数名などのようなその他の用語は、明確にするために、*イタリック・フォント* 示される場合があります。

改訂の要約

ここには、Enterprise COBOL for z/OS and OS/390 に加えられた主要な変更を示します。本書記載する変更には、便宜上、参照ページが記載されています。最新の技術上の変更については、左側の余白にリビジョン・バーが付いています。

- オブジェクト指向型の構文による COBOL と Java の相互運用性 (インターオペラビリティー) により、COBOL プログラムで Java クラスのインスタンスを生成したり、Java オブジェクトのメソッドを呼び出すことができるようになりました。また、Java または COBOL でインスタンスを生成することができ、かつ、Java または COBOL で呼び出すことができるメソッドをもつクラスを COBOL プログラムで定義することができます。(491 ページの『第 28 章 オブジェクト指向プログラムの作成』)
- Java Native Interface (JNI) によって提供されるサービスを呼び出すことで、Java の追加機能を取得することができます。また、JNI へのアクセスを容易にするためのコピーブック JNI.cpy および特殊レジスター JNIEnvPtr が提供されています。(535 ページの『JNI サービスへのアクセス』)
- 高速 XML パーサーを含む XML サポートによってプログラムでインバウンド XML メッセージを利用でき、これらの XML メッセージが整形式 (well-formed) かどうかをチェックすることができます。また、メッセージの内容を COBOL デ

ータ構造へ変換できます。Unicode UTF-16 またはいくつかの単一バイト EBCDIC (拡張 2 進化 10 進コード) または ASCII コード・ページでエンコードされた XML 文書がサポートされます。(225 ページの『第 13 章 XML 文書の処理』)

- 個別の変換ステップなしで、CICS ステートメントを含むプログラムをコンパイルできるようになりました。(413 ページの『組み込みの CICS トランスレーター』)
  - コンパイラー・オプション CICS が追加され、組み込みの CICS トランスレーターの使用と CICS オプションの指定が可能になりました。(317 ページの『CICS』)
- Unicode がサポートされるようになりました。Unicode サポートは、NATIONAL データ型および国別 (N, NX) リテラル、文字変換の組み込み関数 DISPLAY-OF および NATIONAL-OF、およびコンパイラー・オプション NSYMBOL と CODEPAGE により提供されます。(113 ページの『第 7 章 実行時に各国語を使用するためのコーディング』)
  - コンパイラー・オプション CODEPAGE が追加され、国別リテラル、英数字、DBCS (2 バイト文字セット) データ項目、リテラルをエンコードするために使用されるコード・ページを指定できるようになりました。(318 ページの『CODEPAGE』)
  - コンパイラー・オプション NSYMBOL が追加され、N 記号を使用するリテラルおよびデータ項目に対して国別処理と DBCS 処理のどちらを有効にするかを制御できるようになりました。(335 ページの『NSYMBOL』)
- マルチスレッド化がサポートされます。POSIX スレッドおよび非同期 (ASYNC) シグナルのサポートにより、COBOL プログラムを含むアプリケーションを、1 つのプロセス内の複数のスレッドで実行できるようにします。(481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』)
  - コンパイラー・オプション THREAD が追加され、言語環境プログラムのエンクレーブで複数の POSIX スレッドまたは PL/I サブタスクと共にプログラムを実行できるようになりました。(352 ページの『THREAD』)
- BINARY データ項目の VALUE 文節により、ネイティブ・バイナリー表現の大きさまでの値を数字リテラルに指定できるようになり、PICTURE 文節の 9 の数によって暗黙に定義される値に制限されることがなくなりました。(46 ページの『数値データの形式』)
- 4 バイトのデータ項目 FUNCTION-POINTER に COBOL または COBOL 以外の入り口点のアドレスを指定することができ、これにより、C の関数ポインターとのインターオペラビリティーが向上しました。(452 ページの『プロシージャー・ポインターと関数ポインターの使用』)
- 以下のサポートはこのバージョンではもはや提供されません (*Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド* を参照)。
  - SOM ベースのオブジェクト指向型構文およびサービス
  - コンパイラー・オプション CMPR2、ANALYZE、FLAGMIG、TYPECHK、および IDLGEN

以前の COBOL コンパイラーに対する変更のヒストリーについては、*Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド* を参照してください。

第 1 部 プログラムのコーディング

第 1 章 プログラムの構造 7
プログラムの識別. 7
プログラムを再帰的として識別する. 8
収容プログラムによってプログラムに呼び出し可能のマークを付ける. 8
プログラムを初期状態に設定する 9
ソース・リストのヘッダーの変更 9
コンピューター環境の記述. 9
例: FILE-CONTROL の記入項目 10
照合シーケンスの指定. 10
例: 照合シーケンスの指定 11
シンボリック文字を定義する 12
ユーザー定義のクラスを定義する 12
オペレーティング・システムに対してファイルを定義する 12
実行時の入出力ファイルの変更. 13
バッファーおよび装置スペースの最適化. . . 14
データの記述. 15
入出力操作でのデータの使用 15
FILE SECTION の記入項目 16
WORKING-STORAGE と LOCAL-STORAGE の比較. 18
例: ストレージ・セクション. 18
別のプログラムからのデータの使用 19
別個にコンパイルされたプログラムでのデータの共用 19
ネストされたプログラムでのデータの共用 . . 20
再帰的またはマルチスレッド化されたプログラムでのデータの共用 20
データの処理. 21
PROCEDURE DIVISION 内でロジックが分割される方法 21
命令ステートメント 22
条件ステートメント 23
コンパイラー指示ステートメント 24
範囲終了符号. 24
宣言. 25

第 2 章 データの使用. 27
変数、構造、リテラル、および定数の使用 27
変数. 27
データ構造: データ項目とグループ項目 27
リテラル 28
定数. 28
形象定数 29
データ項目への値の割り当て 29
例: 変数の初期化 30
構造の初期化 (INITIALIZE) 31
変数または構造への値の割り当て (MOVE) . . . 32
算術結果の割り当て (MOVE または COMPUTE) 32
画面またはファイルからの入力の割り当て (ACCEPT) 33
画面上またはファイル内での値の表示 (DISPLAY) 34
システム論理出力装置上でのデータの表示 . . . 35
WITH NO ADVANCING の使用 36
組み込み関数の使用 (組み込み関数) 37
組み込み関数の型 37
関数のネスト. 38
テーブル (配列) とポインターの使用. 38
ストレージとそのアドレス可能度 38
RMODE の設定. 40
データの受け渡しに関するストレージ制限 . . . 40
データ域のロケーション. 40
外部データ用のストレージ 41
QSAM 入出力バッファー用のストレージ . . . 41

第 3 章 数値および算術演算 43
数値データの定義 43
数値データの表示 44
数値データの保管方法の制御 45
数値データの形式 46
外部 10 進数 (DISPLAY) 項目. 46
外部浮動小数点 (DISPLAY) 項目 47
2 進数 (COMP) 項目 47
固有 2 進数 (COMP-5) 項目. 48
パック 10 進数 (COMP-3) 項目 48
浮動小数点 (COMP-1 および COMP-2) 項目 . . 49
例: 数値データおよび内部表現 49
データ形式の変換 50
変換および精度 51
精度を保つ変換 51
丸めを生じさせる変換. 51
符号表現および処理 52
NUMPROC(PFD). 52
NUMPROC(NOPFD). 52
NUMPROC(MIG). 52
非互換データの検査 (数値のクラス・テスト) . . . 53
算術の実行 54
COMPUTE およびその他の算術ステートメント 54
算術式 55
数字組み込み関数 55
関数および算術式のネスト 56
ALL 添え字付けおよび特殊レジスター 56
数学および日付の言語環境プログラム・サービス 56
数学用の呼び出し可能サービス. 56
日付呼び出し可能サービス 57
例: 数字組み込み関数 58
一般数値処理. 59
日付および時刻 59
金融. 59
数学. 60

© Copyright IBM Corp. 1991, 2001

統計 60
固定小数点演算と浮動小数点演算の対比 60
浮動小数点計算 61
固定小数点計算 61
算術比較 (比較条件) 61
例: 固定小数点計算および浮動小数点計算 . . . 62
通貨記号の使用 63
例: 複数の通貨記号 64

第 4 章 テーブルの処理 65
テーブルの定義 (OCCURS) 65
テーブルのネスト 66
添え字付け 67
指標付け 67
テーブルでの項目の参照 68
添え字付け 68
指標付け 69
テーブルに値を入れる方法 70
テーブルの動的なロード 70
テーブルの初期化 (INITIALIZE) 71
テーブルの定義時の値の割り当て (VALUE). . . 71
それぞれのテーブル項目の個別の初期化 . . . 71
01 レベルでのテーブルの初期化 72
テーブル・エレメントのすべてのオカレンスの初期化 72
例: PERFORM と添え字付け 72
例: PERFORM と指標付け 73
可変長テーブルの作成 (DEPENDING ON) 74
可変長テーブルのロード 76
可変長テーブルへの値の割り当て 77
テーブルの探索 77
シリアル検索 (SEARCH) 78
例: シリアル検索 79
二分探索 (SEARCH ALL). 79
例: 二分探索 80
組み込み関数を使用したテーブル項目の処理 . . . 80
例: 組み込み関数 81

第 5 章 プログラム・アクションの選択と反復 . . 83
プログラム・アクションの選択 83
アクションの選択項目のコーディング 83
ネストされた IF ステートメントの使用 . . . 84
EVALUATE ステートメントの使用 85
条件式のコーディング 88
スイッチおよびフラグ 88
スイッチおよびフラグの定義 89
例: スイッチ 89
例: フラグ 89
スイッチおよびフラグのリセット 90
例: スイッチをオンに設定する 90
例: スイッチをオフに設定する 91
プログラム・アクションの反復 91
インラインまたはライン外 PERFORM の選択 . . 91
例: インライン PERFORM ステートメント . . 92
ループのコーディング 92
テーブルを操作するループのコーディング . . . 93
複数の段落またはセクションの実行 94

第 6 章 ストリングの処理 95
データ項目の結合 (STRING). 95
例: STRING ステートメント. 96
STRING プログラムの結果 96
データ項目の分割 (UNSTRING). 97
例: UNSTRING ステートメント 98
UNSTRING プログラムの結果 99
ヌル終了ストリングの処理 100
例: ヌル終了ストリング 100
データ項目のサブストリングの参照 101
参照修飾子 102
例: 参照修飾子としての演算式 103
例: 参照修飾子としての組み込み関数 103
データ項目の計算および置換 (INSPECT) 104
例: INSPECT ステートメント 104
データ項目の変換 (組み込み関数) 105
大文字または小文字への変換 (UPPER-CASE、LOWER-CASE) 105
逆順への変換 (REVERSE) 106
数値への変換 (NUMVAL、NUMVAL-C) . . . 106
データ項目の評価 (組み込み関数) 107
照合シーケンスに関する単一文字の評価 . . . 107
最大または最小データ項目の検出. 108
MAX および MIN. 108
ORD-MAX および ORD-MIN 108
英数字関数で可変長の結果を戻す場合 . . . 109
データ項目の長さの検出. 110
コンパイルの日付の検出. 110

第 7 章 実行時に各国語を使用するためのコーディング 113
Unicode および言語文字のエンコード 115
COBOL での国別データ (Unicode) の使用. . . . 116
国別データ項目. 116
国別リテラル 116
形象定数としての国別文字 117
国別データのストレージ. 117
国別データの変換 118
英数字および整数の国別データへの変換 (MOVE) 118
英数字データの国別データへの変換 (NATIONAL-OF) 118
国別データの英数字データへの変換 (DISPLAY-OF) 118
デフォルト・コード・ページのオーバーライド 119
変換例外 119
例: 国別データの変換 119
UTF-8 データの処理 120
中国語 GB 18030 データの処理 121
国別データ項目の比較 121
国別オペランドの比較 122
国別オペランドと数値オペランドの比較 . . . 122

国別オペランドと英字または英数字オペランドの比較 122
国別オペランドとグループ・オペランドの比較 122
DBCS データを含む英数字データ項目の処理 . . . 123

第 8 章 ファイルの処理 125
ファイル編成および入出力装置 125
ファイル編成およびアクセス・モードの選択 . . . 127
入出力のコーディング形式 128
ファイルの割り振り 129
入出力エラーの検査 131

第 9 章 QSAM ファイルの処理 133
COBOL での QSAM ファイルおよびレコードの定義 133
レコード形式の指定 134
論理レコード 135
固定長形式の要求 135
可変長形式の要求 136
スパン形式の要求 138
不定形式の要求. 141
ブロック・サイズの設定. 142
z/OS にブロック・サイズを決定させる . . . 142
ブロック・サイズの明示的設定 143
LBI の利用 143
ブロック・サイズおよび DCB RECFM サブパラメーター 144
QSAM ファイルの入出力ステートメントのコーディング. 145
QSAM ファイルのオープン. 145
CBLQDA による QSAM ファイルの動的作成 146
QSAM ファイルへのレコードの追加. 147
QSAM ファイルの更新 147
QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み 147
ページ・サイズの制御 147
レコードの垂直位置決めの制御 148
QSAM ファイルのクローズ. 148
QSAM ファイルのエラーの処理 149
QSAM ファイルに対する作業 150
QSAM ファイルの定義および割り振り 150
QSAM ファイルを作成するためのパラメーター 152
QSAM ファイルの検索 152
QSAM ファイルを検索するためのパラメーター 153
ファイル属性をプログラムと一致させる . . . 154
既存ファイルの処理 154
可変長 (形式 V) レコードの定義. 154
形式 U レコードの定義 154
固定長レコードの定義 155
新規ファイルの処理 155
COBOL によって動的に作成されるファイルの処理. 156
ストライプ拡張形式 QSAM データ・セットの使用 156
QSAM ファイル用のバッファーの割り振り 157
QSAM を使用する HFS ファイルへのアクセス . . 157
QSAM ファイルのラベル 158
トレーラー・ラベルおよびヘッダー・ラベルの使用 159
ユーザー・ラベル・トラックの入手 159
ユーザー・ラベルの処理. 160
標準ラベルの形式 160
標準ユーザー・ラベル 161
テープ上の QSAM ASCII ファイルの処理. . . . 161
ASCII アルファベットの要求 162
レコード形式の定義 162
DD 名の定義 162
ASCII ファイルのラベルの処理 163

第 10 章 VSAM ファイルの処理 165
VSAM ファイル 166
VSAM ファイル編成およびレコードの定義 . . . 168
VSAM ファイルの順次編成の指定方法 168
VSAM ファイルの索引編成の指定方法 169
代替キー 169
代替索引 169
VSAM ファイルの相対編成の指定方法 170
固定長および可変長 RRDS 170
可変長 RRDS のシミュレート. 170
VSAM ファイルのアクセス・モードの指定 . . 172
例: VSAM ファイルでの動的アクセスの使用 172
VSAM ファイルのレコード長の定義. 173
固定長レコードの定義 173
可変長レコードの定義 173
VSAM ファイルの入出力ステートメントのコーディング. 174
ファイル位置標識 176
ファイルのオープン (ESDS、KSDS、または RRDS). 176
空のファイルのオープン. 177
VSAM ファイルにレコードをロードするために使用されるステートメント 179
ロード済みファイル (レコードが入っているファイル) のオープン 179
VSAM ファイルからのレコードの読み取り . . 179
VSAM ファイル内のレコードの更新. 180
VSAM ファイルへのレコードの追加. 181
レコードを順次に追加する場合 181
レコードをランダムまたは動的に追加する場合 182
VSAM ファイル内のレコードの置換. 182
VSAM ファイルからのレコードの削除 182
VSAM ファイルのクローズ. 183
VSAM ファイルのエラーの処理 183
パスワードによる VSAM ファイルの保護. . . . 184
例: VSAM 索引付きファイルのパスワード保護 185
z/OS および UNIX のもとでの VSAM データ・セットの操作 185
VSAM ファイルの定義 186
代替索引の作成. 187

例: 代替索引の項目 189
VSAM ファイルの割り振り. 189
RLS による VSAM ファイルの共用. 191
RLS モードでの VSAM ファイルに関する更新問題の回避 191
RLS を使用する際の制約事項 192
RLS モードでの VSAM ファイルのエラーの処理 192
VSAM パフォーマンスの向上 193

第 11 章 行順次ファイルの処理 195
COBOL での行順次ファイルおよびレコードの定義 195
許可される制御文字 196
行順次ファイルの構造の記述 196
行順次ファイルの定義および割り振り 197
行順次ファイル用の入出力ステートメントのコーディング. 198
行順次ファイルのオープン 198
行順次ファイルからのレコードの読み取り. . . 199
行順次ファイルへのレコードの追加 199
行順次ファイルのクローズ 200
行順次ファイルのエラーの処理 201

第 12 章 ファイルのソートおよびマージ 203
ソートおよびマージ・プロセス 204
ソートまたはマージ・ファイルの記述 204
ソートまたはマージへの入力の記述 205
例: SORT 用のソート・ファイルおよびの入力ファイルの記述 206
入力プロシージャーのコーディング 207
ソートまたはマージからの出力の記述 208
出力プロシージャーのコーディング 208
DFSORT を使用する際のコーディングの考慮事項 209
例: DFSORT を使用する際の出力プロシージャーのコーディング 209
入出力プロシージャーに関する制約事項 210
ソートおよびマージ・データ・セットの定義 . . . 210
可変長レコードのソート. 211
ソートまたはマージの要求 211
ソートまたはマージ基準の設定 212
例: 入出力プロシージャーを使用したソート . . 213
代替照合シーケンスの選択 214
ウィンドウ化日付フィールドに基づいたソート 214
同じキーを持つレコードのオリジナル・シーケンスの保持 215
ソートまたはマージの成否の判断. 215
ソートまたはマージ操作の途中停止 216
FASTSRT を使ってのソートのパフォーマンスの向上 217
JCL に関する FASTSRT の要件 217
ソート入出力ファイルに関する FASTSRT の要件 217
QSAM の要件 218
VSAM の要件 219
NOFASTSRT によるソート・エラーの検査 . . . 219
ソート動作の制御 220
ソート特殊レジスター 221
制御ステートメントによる DFSORT デフォルトの変更. 221
IGZSRTCD データ・セットのデフォルトの特性 222
ソートまたはマージ操作のためのストレージの割り振り. 222
ソート・ファイル用のスペースの割り振り. . . 223
DFSORT によるチェックポイント・リスタートの使用 223
CICS のもとでのソート 224
CICS SORT アプリケーションの制約事項 . . . 224

第 13 章 XML 文書の処理 225
COBOL でのXML パーサー 225
XML 文書へのアクセス 227
XML 文書の構文解析. 227
XML イベントの処理. 228
XML を処理するためのプロシージャーの作成 234
XML-CODE の内容を理解する 234
XML-TEXT および XML-NTEXT を使用する 236
XML テキストを COBOL データ項目に変換する 236
処理プロシージャーに関する制約. 236
処理プロシージャーを終了する 237
例: XML の構文解析 237
XML 文書のエンコードについての理解. 240
コード・ページの指定 240
その他のコード・ページを使用した文書の構文解析 241
XML 文書のエラー処理 241
未処理の例外 243
例外の処理 243
構文解析の終了. 244
CCSID 矛盾例外 244

第 14 章 エラーの処理 247
ダンプの要求 247
定様式ダンプの作成 247
システム・ダンプの作成. 248
ストリングの結合および分割におけるエラーの処理 248
算術演算でのエラーの処理 249
例: 0 による除算の検査 249
入出力操作でのエラーの処理 250
ファイル終わり条件 (AT END) の使用. . . . 252
ERROR 宣言のコーディング 253
ファイル状況キーの使用. 254
例: ファイル状況キー 255
VSAM 戻りコードの使用 (VSAM ファイルのみ) 256
例: VSAM 状況コードの検査 256
INVALID KEY 句のコーディング 258
INVALID KEY および ERROR 宣言 . . . 258
NOT INVALID KEY 258
例: FILE STATUS および INVALID KEY. . . 258
プログラム呼び出し時のエラーの処理 259

エラー処理用のルーチンの作成 259

第 1 章 プログラムの構造

COBOL プログラムは、それぞれ特定の論理機能を持つ次の 4 つの部から構成されます。

- IDENTIFICATION DIVISION (見出し部)
- ENVIRONMENT DIVISION (環境部)
- DATA DIVISION (データ部)
- PROCEDURE DIVISION (手続き部)

IDENTIFICATION DIVISION だけが必須です。

COBOL クラスまたはメソッドを定義するには、プログラムの場合とは違った方法でいくつかの部を定義することが必要です。

関連タスク
『プログラムの識別』
9 ページの『コンピューター環境の記述』
15 ページの『データの記述』
21 ページの『データの処理』
495 ページの『クラスの定義』
500 ページの『クラス・インスタンス・メソッドの定義』

プログラムの識別

IDENTIFICATION DIVISION (見出し部) は、プログラムの名前を指定し、また必要があればその他の識別情報を与えるために使用されます。

任意指定の AUTHOR、INSTALLATION、DATE-WRITTEN、および DATE-COMPILED 段落を使用して、プログラムに関する記述情報を与えることができます。DATE-COMPILED 段落に入力した日付は、最新のコンパイル日付で置き換えられます。

```
IDENTIFICATION DIVISION.
Program-ID. Helloprog.
Author. A. Programmer.
Installation.  Computing Laboratories.
Date-Written.  08/18/2001.
Date-Compiled. 09/30/2001.
```

PROGRAM-ID 段落を使用して、プログラムの名前を指定します。割り当てるプログラム名は、以下のように使用されます。

- プログラムを呼び出すために他のプログラムがその名前を使用します。
- プログラムのコンパイル時に生成されるプログラム・リストの各ページ (最初のページを除く) のヘッダーにその名前が入れられます。
- NAME コンパイラー・オプションを使用した場合は、コンパイルによって作成されるオブジェクト・モジュールを識別するために、その名前がリンケージ・エディターまたはバインダーの NAME 制御ステートメントに入れられます。

© Copyright IBM Corp. 1991, 2001

ヒント: IBM 製品が使用している接頭部で始まるプログラム名は使用しないでください。名前が以下のいずれかで始まるプログラムの使用を試みると、CALL ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラー・ルーチンを呼び出してしまう可能性があります。

AFB	AFH	CBC	CEE	EDC
IBM	IFY	IGY	IGZ	ILB

ヒント: プログラム名に大文字小文字の区別がある場合は、コンパイラーの探索対象である名前とのミスマッチが起こらないようにしてください。適切な PGMNAME コンパイラー・オプションが設定されているかどうか確かめてください。

関連タスク
9 ページの『ソース・リストのヘッダーの変更』
『プログラムを再帰的として識別する』
『収容プログラムによってプログラムに呼び出し可能のマークを付ける』
9 ページの『プログラムを初期状態に設定する』

関連参照
コンパイラー限界値 (*Enterprise COBOL 言語解説書*)
プログラム名の規則 (*Enterprise COBOL 言語解説書*)

プログラムを再帰的として識別する

前の呼び出しがまだアクティブである間にプログラムに再帰的に再入できるようにするには、PROGRAM-ID 文節に RECURSIVE 属性をコーディングしてください。

RECURSIVE は、コンパイル単位の最外部のプログラムにのみコーディングすることができます。ネストされたサブプログラムも、ネストされたサブプログラムを含むプログラムも、再帰的にすることはできません。THREAD オプションを指定してコンパイルするプログラムに RECURSIVE をコーディングする必要があります。

関連タスク
20 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』
451 ページの『再帰呼び出しの作成』

収容プログラムによってプログラムに呼び出し可能のマークを付ける

プログラムを、収容プログラムまたは収容プログラム内の任意のプログラムによって呼び出せることを指定するには、PROGRAM-ID 文節で COMMON 属性を使用してください。ただし、COMMON プログラムは、それ自体に含まれているプログラムによって呼び出すことはできません。

含まれているプログラムだけが COMMON 属性を持つことができます。

関連概念
448 ページの『ネストされたプログラム』

プログラムを初期状態に設定する

プログラムを呼び出すたびにプログラムを初期状態にすることを指定するには、INITIAL 属性を使用します。そのプログラムにプログラムが含まれていると、含まれているプログラムも初期状態に置かれます。

プログラムが初期状態になるのは、以下の状況が発生した場合です。

- VALUE 文節を持つデータ項目が、指定された値に設定された。
- 変更された GO TO ステートメントおよび PERFORM ステートメントが、それぞれ初期状態に設定された。
- 非 EXTERNAL ファイルがクローズされた。

ソース・リストのヘッダーの変更

ソース・ステートメント・リストの最初のページのヘッダーには、コンパイラーおよび現行リリース・レベルの識別、コンパイルの日時、およびページ番号が入れられます。たとえば、次のようになります。

```
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0   Date 09/30/2001  Time 15:05:19   Page     1
```

コンパイラー指示 TITLE ステートメントを使用すれば、リストの後続のページのヘッダーをカスタマイズすることができます。

関連参照

TITLE ステートメント (*Enterprise COBOL 言語解説書*)

コンピューター環境の記述

ENVIRONMENT DIVISION (環境部) では、コンピューター環境に依存するプログラムの局面について記述します。

CONFIGURATION SECTION を使用して、以下の項目を指定してください。

- プログラムをコンパイルするコンピューター (SOURCE-COMPUTER 段落)
- プログラムを実行するコンピューター (OBJECT-COMPUTER 段落)
- 通貨記号やシンボリック文字などの特殊な項目 (SPECIAL-NAMES 段落)
- ユーザー定義のクラス (REPOSITORY 段落)

INPUT-OUTPUT SECTION の FILE-CONTROL および I-O-CONTROL 段落は、以下の目的に使用します。

- プログラム・ファイルの特性を識別および記述する。
- ファイルを、物理的に存在している外部 QSAM、VSAM、または HFS (階層ファイル・システム) データ・セットに関連付ける。

  COBOL 用語の *ファイル* という用語と、オペレーティング・システム (OS) 用語のデータ・セット または *HFS ファイル* という用語は、本質的に同じ意味を持ち、本書では区別なく使用されます。

  顧客情報管理システム (CICS) およびオンライン情報管理システム (IMS) のメッセージ処理プログラム (MPP) については、ENVIRONMENT DIVISION ヘッダーと、オプションとして、CONFIGURATION SECTION だけをコーディングしてください。

CICS では、COBOL によるファイルの定義は許可されません。IMS では、バッチ・プログラムについてのみ、COBOL によるファイルの定義が許可されます。

- プログラムと外部メディア間でのデータ・レコードの効率的な伝送を制御するために情報を提供する。

『例: FILE-CONTROL の記入項目』

関連タスク

『照合シーケンスの指定』
12 ページの『シンボリック文字を定義する』
12 ページの『ユーザー定義のクラスを定義する』
12 ページの『オペレーティング・システムに対してファイルを定義する』

関連参照

セクションおよび段落 (*Enterprise COBOL 言語解説書*)

例: FILE-CONTROL の記入項目

次の表に示す FILE-CONTROL 記入項目は、QSAM 順次ファイル、VSAM 索引付きファイル、および行順次ファイル用です。

QSAM ファイル	VSAM ファイル	行順次ファイル
SELECT PRINTFILE[1] ASSIGN TO UPDPRINT[2] ORGANIZATION IS SEQUENTIAL[3] ACCESS IS SEQUENTIAL.[4]	SELECT COMMUTER-FILE[1] ASSIGN TO COMMUTER[2] ORGANIZATION IS INDEXED[3] ACCESS IS RANDOM[4] RECORD KEY IS COMMUTER-KEY[5] FILE STATUS IS[5] COMMUTER-FILE-STATUS COMMUTER-VSAM-STATUS.	SELECT PRINTFILE[1] ASSIGN TO UPDPRINT[2] ORGANIZATION IS LINE SEQUENTIAL[3] ACCESS IS SEQUENTIAL.[4]

1. SELECT 文節は、外部データ・セットと関連付けられる、COBOL プログラム内のファイルを選択します。
2. ASSIGN 文節は、プログラムのファイル名を実際のデータ・ファイルの外部名と関連付けます。外部名は、DD ステートメントまたは環境変数を使用して定義することができます。
3. ORGANIZATION 文節は、ファイルの編成を記述します。QSAM ファイルの場合、ORGANIZATION 文節は任意指定です。
4. ACCESS MODE 文節は、レコードを処理する方式 (順次、ランダム、または動的) を定義します。QSAM ファイルおよび行順次ファイルの場合、ACCESS MODE 文節はオプショナルです。これらのファイルの編成は常に順次です。
5. VSAM ファイルの場合、使用する VSAM ファイルのタイプによって、FILE-CONTROL 段落に追加のステートメントを指定することができます。

関連タスク

133 ページの『第 9 章 QSAM ファイルの処理』
165 ページの『第 10 章 VSAM ファイルの処理』
195 ページの『第 11 章 行順次ファイルの処理』
9 ページの『コンピューター環境の記述』

照合シーケンスの指定

以下の操作で使用する照合シーケンスを確立するには、SPECIAL-NAMES 段落の PROGRAM COLLATING SEQUENCE 文節と ALPHABET 文節を使用します。

- 比較条件および条件名条件で明示的に指定された非数値比較
- HIGH-VALUE および LOW-VALUE の設定
- SEARCH ALL
- SORT および MERGE (SORT または MERGE ステートメントの COLLATING SEQUENCE 句でオーバーライドされていない場合)

『例: 照合シーケンスの指定』

以下のいずれかのアルファベットを基に、使用するシーケンスを選択できます。

- EBCDIC (固有文字セットが EBCDIC の場合は、NATIVE を使用してください)。これは、ALPHABET 文節を省略した場合のデフォルトです。
- ASCII (STANDARD-1 を使用してください)。
- ISO 7 ビット・コード、International Reference バージョン (STANDARD-2 を使用してください)。
- SPECIAL-NAMES 段落で定義された EBCDIC シーケンスの代替。

PROGRAM COLLATING SEQUENCE 文節は国別オペランドを含む比較に影響を及ぼしません。

関連タスク

214 ページの『代替照合シーケンスの選択』
121 ページの『国別データ項目の比較』

例: 照合シーケンスの指定

次の例で示している ENVIRONMENT DIVISION コーディングは、比較およびソート / マージの場合に大文字と小文字が同様に処理される照合シーケンスを指定するものです。SPECIAL-NAMES 段落の EBCDIC シーケンスを変更すると、SPECIAL-NAMES 段落に含まれている文字の照合シーケンスだけでなく、全体の照合シーケンスが影響を受けます。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.

   Source-Computer. IBM-390.
   Object-Computer. IBM-390.


      Program Collating Sequence Special-Sequence.
   Special-Names.
      Alphabet Special-Sequence Is
          "A" Also "a"
          "B" Also "b"
          "C" Also "c"
          "D" Also "d"
          "E" Also "e"
          "F" Also "f"
          "G" Also "g"
          "H" Also "h"
          "I" Also "i"
          "J" Also "j"
          "K" Also "k"
          "L" Also "l"
          "M" Also "m"
          "N" Also "n"
```

```
"O" Also "o"
"P" Also "p"
"Q" Also "q"
"R" Also "r"
"S" Also "s"
"T" Also "t"
"U" Also "u"
"V" Also "v"
"W" Also "w"
"X" Also "x"
"Y" Also "y"
"Z" Also "z".
```

関連タスク

10 ページの『照合シーケンスの指定』

シンボリック文字を定義する

SYMBOLIC CHARACTER 文節を使用すると、指定したアルファベットの任意の文字に記号名を与えることができます。序数位置を用いて文字を識別してください。位置 1 は文字 X'00' に対応します。たとえば、バックスペース文字 (EBCDIC アルファベットでは X'16') に名前を与えるには、次のようにコーディングします。

```
SYMBOLIC CHARACTERS BACKSPACE IS 23
```

ユーザー定義のクラスを定義する

CLASS 文節は、文節内にリストされている文字のセットに名前を与えるために使用します。たとえば、数字のセットに名前を与えるには、次のようにコーディングします。

```
CLASS DIGIT IS "0" THROUGH "9"
```

このクラス名は、クラス条件の中でのみ参照することができます。このユーザー定義クラスは、オブジェクト指向クラスと同じ概念ではありません。

オペレーティング・システムに対してファイルを定義する

COBOL プログラムで処理するすべてのファイルについて、適切なシステム・データ定義を使用し、ファイルをオペレーティング・システム (OS) に対して定義する必要があります。

- z/OS JCL の場合は DD ステートメント。
- TSO のもとでは、ALLOCATE コマンド。
- z/OS または UNIX の場合は、環境変数。この内容は、MVS データ・セットまたは HFS (階層ファイル・システム) 内のファイルのいずれかを定義することができます。

以下に、FILE-CONTROL 記入項目の、システム・データ定義との関係、および FILE SECTION の FD 記入項目との関係を示します。

- JCL DD ステートメント:

```
    (1)
//OUTFILE  DD  DSNAME=MY.OUT171,UNIT=SYSDA,SPACE=(TRK,(50,5)),
//             DCB=(BLKSIZE=400)
/*
. . .
```

- 環境変数(export コマンド):

```
          (1)
export OUTFILE=DSNAME(MY.OUT171),UNIT(SYSDA),SPACE(TRK,(50,5))
. . .
```

- COBOL コード:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARPOOL
      ASSIGN TO OUTFILE (1)
      ORGANIZATION IS SEQUENTIAL.
. . .
DATA DIVISION.
FILE SECTION.
FD CARPOOL       (2)
    LABEL RECORD STANDARD
    BLOCK CONTAINS 0 CHARACTERS
    RECORD CONTAINS 80 CHARACTERS
```

(1) DD ステートメントの *ddname*、または export コマンドの環境変数名は、ASSIGN 文節の *assignment-name* に対応します。

- //OUTFILE DD DSNAME=OUT171 . . ., または
- export OUTFILE= . . .

この *assignment-name* は、DD ステートメントの *ddname* OUTFILE、または export コマンドの環境変数名 OUTFILE を指します。

```
ASSIGN TO OUTFILE
```

(2) COBOL FILE-CONTROL 記入項目にファイルを指定する場合は、そのファイルを FD 記入項目で *file-name* として記述しなければなりません。

```
SELECT CARPOOL
. . .
FD CARPOOL
```

関連タスク

14 ページの『バッファーおよび装置スペースの最適化』

実行時の入出力ファイルの変更

SELECT 文節でコーディングした *file-name* は、COBOL プログラム全体にわたって定数として使用されますが、DD ステートメントまたは export コマンド上のファイルの名前を、実行時に別のファイルと関連付けることもできます。

COBOL プログラム内の *file-name* を変更するには、入力ステートメントと出力ステートメントを変更し、プログラムを再コンパイルしなければなりません。あるいは、DD ステートメントの *dsname* を変更するか、export コマンドの *dsname path-name* を変更します。

異なるファイルを使用するための規則: ASSIGN 文節の *assignment-name* で使用する名前は、DD ステートメントの *ddname* または export コマンドの環境変数と同じでなければなりません。実際のファイルを変更するには、DD ステートメントの DSNAME、または環境変数の DSNAME またはパス名を使用します。

SELECT 文節で使用する *file-name* (SELECT MASTER など) は、FD *file-name* 記入項目で使用するファイル名と同じでなければなりません。

2 つのファイルが SELECT 文節でそれぞれ同じ *ddname* または環境変数名を使用することはできません。もし使用した場合、結果は予測できません。たとえば、DISPLAY の出力先が SYSOUT になっている場合、ファイルに対する SELECT 文節で *ddname* または環境変数として SYSOUT を使用しないでください。

『例: さまざまな入力ファイルの使用』

例: さまざまな入力ファイルの使用: 例として、複数の異なるマスター・ファイルに対してまったく同じ方法で使用される COBOL プログラムを想定してください。このプログラムには、次の SELECT 文節が含まれています。

```
SELECT MASTER
    ASSIGN TO DA-3330-S-MASTERA
```

3 つのファイル、MASTER1、MASTER2、および MASTER3 が入力ファイルとなる可能性があるとします。プログラム実行を要求するジョブ・ステップに、次の DD ステートメントの 1 つをコーディングするか、または、プログラムの実行に先立って、プログラムを実行する同じシェルから、次の export コマンドの 1 つを出さなければなりません。

```
//MASTERA     DD   DSNAME=MY.MASTER1,. . .
export MASTERA=DSN(MY.MASTER1),. . .

//MASTERA     DD   DSNAME=MY.MASTER2,. . .
export MASTERA=DSN(MY.MASTER2),. . .

//MASTERA     DD   DSNAME=MY.MASTER3,. . .
export MASTERA=DSN(MY.MASTER3),. . .
```

したがって、プログラムで MASTERA を参照すると、それは現在 DD 名または環境変数名 MASTERA に割り当てられているファイルへの参照になります。

この例では、PATH(*path*) 形式の export コマンドを使用して HFS の行順次ファイルを参照できないことに注意してください。行順次ファイルでは編成フィールド (S- または AS-) を指定することができないためです。

バッファーおよび装置スペースの最適化

APPLY WRITE-ONLY 文節を使用すると、ブロック化可変長レコードの順次ファイルを作成する際に、バッファーおよび装置スペースを最適に使用することができます。APPLY WRITE-ONLY を指定すると、バッファーは、次のレコードがバッファーの未使用部分に収まらない場合のみ切り捨てられます。APPLY WRITE-ONLY を指定しない場合は、最大サイズのレコードを収容できる十分なスペースが残っていないと、バッファーは切り捨てられます。

APPLY WRITE-ONLY 文節が意味を持つのは、ブロック化可変長レコードの順次ファイルの場合だけです。

AWO コンパイラー・オプションは、すべての適格ファイルに暗黙の APPLY WRITE-ONLY 文節を適用します。NOAWO コンパイラー・オプションが指定されると、ファイルに APPLY WRITE-ONLY 文節が指定されていても、効力はありません。APPLY WRITE-ONLY 文節の方が、NOAWO コンパイラー・オプションに優先します。

APPLY-WRITE ONLY 文節を使用すると、入力ファイルがバッファー内のデータを処理せずに、レコード域を使用することがあります。これは、入力ファイルと出力ファイルの両方の処理に影響を与える可能性があります。

関連参照
316 ページの『AWO』

データの記述

データの特性を定義し、データ定義をグループ化して、DATA DIVISION のセクションの 1 つに入れなければなりません。

- 入出力操作で使用するデータを定義する (FILE SECTION)。
- 内部処理用に作成するデータを定義する。
  - ストレージを静的に割り振り、実行単位の存続期間中存在させる場合 (WORKING-STORAGE SECTION)。
  - プログラムの呼び出しのたびにストレージを割り振り、プログラムの終了時に割り振り解除させる場合 (LOCAL-STORAGE SECTION)。
- 別のプログラムからのデータを記述する (LINKAGE SECTION)。

Enterprise COBOL コンパイラーでは、DATA DIVISION エレメントの最大サイズの制限があります。

関連概念
18 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

関連タスク
『入出力操作でのデータの使用』
19 ページの『別のプログラムからのデータの使用』

関連参照
コンパイラー限界値 (*Enterprise COBOL 言語解説書*)

入出力操作でのデータの使用

入出力操作で使用するデータは、FILE SECTION で定義します。

- プログラムが使用する入出力ファイルの名前を指定します。FD 記入項目を使用して、PROCEDURE DIVISION の入出力ステートメントで参照できるファイルに名前を与えます。

  FILE SECTION 内で定義されたデータ項目は、ファイルが正常にオープンされるまでは PROCEDURE DIVISION のステートメントにとって使用可能ではありません。
- FD 記入項目の後のレコード記述で、ファイル内のレコードのフィールドを記述します。
  - レコード全体をレベル 01 記述でコーディングし、次に WORKING-STORAGE SECTION で、レコードのフィールドをより詳しく記述する作業用コピーをコーディングすることができます。レコードを WORKING-STORAGE に入れるには、READ-INTO ステートメントを使用します。処理は、WORKING-STORAGE のデータのコピーに対して行われます。処理されたデータは、WRITE FROM ステートメントによって、FILE SECTION で定義されたレコード域に書き込まれます。
  - 設定されるレコード名は、WRITE および REWRITE ステートメントの対象となります。

- QSAM ファイルの場合のみ、RECORDING MODE 文節でレコード形式を設定することができます。RECORDING MODE 文節を省くと、コンパイラーは RECORD 文節およびレベル 01 レコード記述に基づいてレコード形式を判別します。
- QSAM ファイルの場合は、BLOCK CONTAINS 文節でファイルについてのブロック化因数を設定することができます。BLOCK CONTAINS 文節を省略すると、デフォルトとしてファイルはブロック化されません。しかし、z/OS のデータ管理機能 (DD ファイル・ジョブ制御ステートメントを含む) を使用すると、これをオーバーライドできます。
- 行順次ファイルの場合は、BLOCK CONTAINS 文節でファイルについてのブロック化因数を設定することができます。BLOCK CONTAINS 1 RECORDS または BLOCK CONTAINS *n* CHARACTERS (*n* は 1 つの論理レコードの長さ) をコーディングした場合は、WRITE ステートメントを使用すると、レコードはバッファーに入れられるのではなく、直ちにファイルに転送されます。この手法は、各レコードを直ちに (エラー・ログなどに) 書き込みたい場合に有用です。

同じ実行単位内のプログラムは、共通ファイルを共用したり、共通ファイルにアクセスすることができます。これを行う方法は、プログラムがネストされた構造 (含まれている構造) の一部であるか、個別にコンパイルされる (バッチ・シーケンスの一部としてコンパイルされるプログラムを含む) かによって異なります。

別個にコンパイルされたプログラムには EXTERNAL 文節を使用することができます。EXTERNAL として定義されたファイルは、実行単位の中でそのファイルを記述する任意のプログラムによって参照することができます。

ネストされた構造、すなわち含まれている構造の中のプログラムには、GLOBAL 文節を使用できます。あるプログラムが別のプログラムを (直接または間接的に) 含む場合には、どちらのプログラムも GLOBAL ファイル名を参照することによって共通のファイルにアクセスすることができます。

関連概念
448 ページの『ネストされたプログラム』

関連タスク
464 ページの『プログラム間でのファイルの共用 (外部ファイル)』

関連参照
『FILE SECTION の記入項目』

FILE SECTION の記入項目

文節	定義対象	注
FD	PROCEDURE DIVISION の入出力ステートメント (OPEN、CLOSE、READ、さらに、VSAM の場合は START および DELETE) 内で参照される *file-name*	SELECT 文節内の *file-name* と一致しなければなりません。*file-name* は、*assignment-name* を介して *ddname* と関連付けられます。

文節	定義対象	注
BLOCK CONTAINS	物理レコードのサイズ	QSAM: 指定する場合は、JCL またはデータ・セット・ラベルの情報と一致していなければなりません。指定されないと、最適ブロック・サイズはシステムによって決定されます。 行順次: WRITE ステートメントのバッファリングを制御するために指定できます。 VSAM: コメントとして処理されます。
RECORD CONTAINS *n*	論理レコード (固定長) のサイズ	指定される場合には、JCL またはデータ・セット・ラベル上の情報と一致しなければなりません。*n* が 0 である場合には、JCL またはデータ・セット・ラベルに LRECL をコーディングしなければなりません。
RECORD IS VARYING	論理レコード (可変長) のサイズ	指定される場合には、JCL またはデータ・セット・ラベル上の情報と一致しなければなりません。コンパイラーがレコード記述との一致を検査します。
RECORD CONTAINS *n* TO *m*	論理レコード (可変長) のサイズ	指定される場合には、JCL またはデータ・セット・ラベル上の情報と一致しなければなりません。コンパイラーがレコード記述との一致を検査します。
LABEL RECORDS	QSAM ファイルのラベル	VSAM: コメントとして処理されます
STANDARD	ラベルが存在する	QSAM: コメントとして処理されます
OMITTED	ラベルが存在しない	QSAM: コメントとして処理されます
data-name	ユーザーによって定義されたラベル	QSAM: (オプショナルの) テープまたはディスクに許可されます
VALUE OF	ファイルに関連付けられているラベル・レコードの項目	コメントのみ
DATA RECORDS	ファイルに関連付けられているレコードの名前	コメントのみ
LINAGE	論理ページの深さ	QSAM のみ
CODE-SET	ASCII または EBCDIC ファイル	QSAM のみ。 ASCII ファイルが CODE-SET 文節で識別される場合は、対応する DD ステートメントに、DCB=(OPTCD=Q. . .) または DCB=(RECFM=D. . .) をコーディングすることが必要になる場合があります (ファイルが VS COBOL II、COBOL (OS/390 および VM 版)、または IBM Enterprise COBOL for z/OS and OS/390 を使用して作成されなかった場合)。
RECORDING MODE	物理レコード記述	QSAM のみ

関連概念
158 ページの『QSAM ファイルのラベル』

WORKING-STORAGE と LOCAL-STORAGE の比較

WORKING-STORAGE は、実行単位の開始時に割り振られます。VALUE 文節を持つすべてのデータ項目は、そのときに適切な値に初期化されます。その実行単位の存続期間中、WORKING-STORAGE 項目はそれぞれ最後に使われた状態を維持します。例外は次のとおりです。

- PROGRAM-ID 段落に INITIAL が指定されたプログラム。

  この場合、WORKING-STORAGE データ項目は、プログラムに入るたびに再初期化されます。
- 動的に呼び出されてから取り消されるサブプログラム。

  この場合、WORKING-STORAGE データ項目は、CANCEL 後のプログラムへの最初の再入時に再初期化されます。

WORKING-STORAGE は、実行単位の終了時に割り振り解除されます。

LOCAL-STORAGE は、プログラムが呼び出されるたびに割り振られ、EXIT PROGRAM、GOBACK、または STOP RUN によってプログラムが戻るときに割り振り解除されます。しかし、ネストされたプログラムの場合、LOCAL-STORAGE は、収容している最外部プログラムに入るときに割り振られ、そこから出るときに割り振り解除されます。ネストされたプログラム内の VALUE 文節を持つ LOCAL-STORAGE データ項目は、ネストされたプログラムが呼び出されるたびに適切な値に初期化されます。

スレッド化: 複数のスレッドで同時に実行されるプログラムは、それぞれの起動先が WORKING-STORAGE データの単一コピーへのアクセスを共用します。
LOCAL-STORAGE データは起動先ごとに個別のコピーを使用します。

関連概念
『例: ストレージ・セクション』

関連タスク
439 ページの『メインプログラムまたはサブプログラムの終了と再入』
481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照
Working-Storage セクション (*Enterprise COBOL 言語解説書*)
Local-Storage セクション (*Enterprise COBOL 言語解説書*)

例: ストレージ・セクション

以下に、WORKING-STORAGE と LOCAL-STORAGE の両方を使用する再帰的プログラムの例を示します。

```
CBL nocmpr2,pgmn(lu)
*********************************
* Recursive Program - Factorials
*********************************
 IDENTIFICATION DIVISION.
 Program-Id. factorial recursive.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 Working-Storage Section.
```

```
 01  numb  pic 9(4)  value 5.
 01  fact  pic 9(8)  value 0.
 Local-Storage Section.
 01  num  pic 9(4).
 PROCEDURE DIVISION.
     move numb to num.

     if numb = 0
        move 1 to fact
     else
        subtract 1 from numb
        call 'factorial'
        multiply num by fact
     end-if.

     display num '! = ' fact.
     goback.
 End Program factorial.
```

次の表は、プログラムの連続する再帰呼び出し (CALL) および結果として起こる戻り (GOBACK) における、LOCAL-STORAGE (L-S) および WORKING-STORAGE (W-S) 内のデータ項目の値の変化を示しています。戻り時、fact は 5! の値を次々に累算します (5 階乗)。

Recursive CALLs:		Main	1	2	3	4	5
L-S	num	5	4	3	2	1	0
W-S	numb	5	4	3	2	1	0
	fact	0	0	0	0	0	0

Recursive GOBACKs:		5	4	3	2	1	Main
L-S	num	0	1	2	3	4	5
W-S	numb	0	0	0	0	0	0
	fact	1	1	2	6	24	120

関連概念

18 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

別のプログラムからのデータの使用

データを共用する方法は、プログラムのタイプによって異なります。別個にコンパイルされたプログラムでは、ネストされたプログラムや、再帰的またはマルチスレッド化されたプログラムの場合とは異なる方法でデータを共用します。

関連タスク

『別個にコンパイルされたプログラムでのデータの共用』

20 ページの『ネストされたプログラムでのデータの共用』

別個にコンパイルされたプログラムでのデータの共用

多くのアプリケーションは、相互に呼び出し、データを受け渡しする、別個にコンパイルされたプログラムから構成されます。呼び出されるプログラム内の LINKAGE

SECTION を用いて、別のプログラムから渡されるデータを記述します。呼び出しプログラムでは、CALL ... USING または INVOKE ... USING ステートメントを使ってデータを受け渡します。

関連タスク
455 ページの『データの受け渡し』

ネストされたプログラムでのデータの共用

アプリケーションの中には、ネストされたプログラム (他のプログラムに含まれているプログラム) から構成されるものもあります。LINKAGE SECTION のレベル 01 データ項目は、GLOBAL 属性を含むことができます。この属性を使用すると、宣言を含むネストされたプログラムがこれらの LINKAGE SECTION データ項目にアクセスできるようになります。

ネストされたプログラムは、COMMON 属性で定義された兄弟プログラム(同一の含まれているプログラム内で同じネスト・レベルにあるプログラム) のデータ項目にもアクセスできます。

関連概念
448 ページの『ネストされたプログラム』

再帰的またはマルチスレッド化されたプログラムでのデータの共用

プログラムを RECURSIVE としてコンパイルするか、または THREAD コンパイラー・オプションを指定してコンパイルする場合、LINKAGE SECTION で定義されたデータに、プログラムの後続の呼び出し先でアクセスすることはできません。

LINKAGE SECTION のレコードをアドレッシングするには、以下のいずれかの技法を使用します。

- プログラムに引き数を渡し、プログラムの USING 句に適切な位置のレコードを指定する。
- フォーマット-5 の SET ステートメントを使用する。

プログラムを RECURSIVE としてコンパイルするか、または THREAD コンパイラー・オプションを指定してコンパイルする場合、レコードのアドレスはプログラム起動の特定のインスタンスについて有効です。同じプログラムの別の実行インスタンスにあるレコードのアドレスは、その実行インスタンスに対して再設定する必要があります。アドレスが設定されていないデータ項目を参照すると、予測できない結果が生じます。

関連概念
482 ページの『マルチスレッド化』

関連タスク
451 ページの『再帰呼び出しの作成』
485 ページの『マルチスレッド化によるファイルの処理』

関連参照
SET ステートメント (*Enterprise COBOL 言語解説書*)
352 ページの『THREAD』

データの処理

プログラムの PROCEDURE DIVISION (手続き部) では、他の部で定義したデータを処理する実行可能なステートメントをコーディングします。PROCEDURE DIVISION には、1 つまたは 2 つのヘッダーと、プログラムのロジックが入れられます。

PROCEDURE DIVISION は、部のヘッダーとプロシージャー名のヘッダーで始まります。プログラムの部のヘッダーは、単に次のようにすることができます。

```
PROCEDURE DIVISION.
```

あるいは、USING 句を使用してパラメーターを受け取ったり、RETURNING 句を使用して値を戻すような部のヘッダーをコーディングすることができます。

参照によって (デフォルト) あるいは内容によって渡された引き数を受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING dataname
PROCEDURE DIVISION USING BY REFERENCE dataname
```

dataname は、DATA DIVISION の LINKAGE SECTION で定義しなければなりません。

値によって渡されたパラメーターを受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING BY VALUE dataname
```

結果として値を戻すためには、部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION RETURNING dataname2
```

また、PROCEDURE DIVISION のヘッダーの中で USING と RETURNING を組み合わせることもできます。

```
PROCEDURE DIVISION USING dataname RETURNING dataname2
```

dataname および *dataname2* は、DATA DIVISION の LINKAGE SECTION で定義しなければなりません。

関連概念
『PROCEDURE DIVISION 内でロジックが分割される方法』

関連タスク
609 ページの『反復コーディングの除去』

PROCEDURE DIVISION 内でロジックが分割される方法

プログラムの PROCEDURE DIVISION は、セクションと段落に分割されます。セクションおよび段落には、文およびステートメントが含まれています。

セクション

処理ロジックの論理的な副区分。

セクションにはセクション・ヘッダーがあり、オプションとして後ろに 1 つまたは複数の段落が続きます。

セクションは、PERFORM ステートメントのサブジェクトにすることができます。セクションのタイプの 1 つは宣言用です。

段落 セクション、プロシージャー、またはプログラムの再分割。

段落は、後ろにピリオドが付いた名前を持ち、その後に文が続く場合と続かない場合があります。

段落は、ステートメントのサブジェクトにすることができます。

文 1 つのまたは複数の COBOL ステートメントの並びであって、ピリオドで終わるもの。

多くの構造化プログラムは、独立した文を持ちません。各段落には、1 つの文を含めることができます。

ステートメント
2 つの数値の加算など、COBOL 処理の定義されたステップを実行します。

ステートメントは、COBOL 動詞で始まる、ワードの有効な組み合わせです。ステートメントには、命令ステートメント (無条件アクションを示す)、条件ステートメント、およびコンパイラー指示ステートメントがあります。ステートメントの論理的な終わりを示すためには、ピリオドではなく明示的範囲終了符号を使用することをお勧めします。

句 ステートメントの再分割。

関連概念
24 ページの『コンパイラー指示ステートメント』
24 ページの『範囲終了符号』
『命令ステートメント』
23 ページの『条件ステートメント』
25 ページの『宣言』

関連参照
PROCEDURE DIVISION 構造 (*Enterprise COBOL 言語解説書*)

命令ステートメント

命令ステートメントは、取るべき無条件アクション (たとえば、ADD、MOVE、INVOKE、または CLOSE など) を指示します。

命令ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。

明示的範囲終了符号で終わる条件ステートメントは、*範囲区切りステートメント* と呼ばれる命令ステートメントになります。命令ステートメント (または範囲区切りステートメント) のみをネストすることができます。

関連概念
23 ページの『条件ステートメント』
24 ページの『範囲終了符号』

条件ステートメント

条件ステートメントには、単純な条件ステートメント (IF、EVALUATE、SEARCH) と、条件句またはオプションを含む命令ステートメントから構成された条件ステートメントの 2 種類があります。

条件ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。条件ステートメントを明示的に終わらせると、それは範囲区切りステートメント (命令ステートメント) になります。

範囲区切りステートメントは、次のような方法で使用できます。

- COBOL 条件ステートメントの操作の範囲を区切り、ネストのレベルを明示的に指示する場合。

  たとえば、ネストされた IF の中の IF ステートメントの範囲を終了させるために、ピリオドではなく END-IF ステートメントを使用します。
- COBOL 構文が命令ステートメントを必要とする条件ステートメントをコーディングする場合。

  たとえば、インライン PERFORM のオブジェクトとして条件ステートメントをコーディングします。

```
PERFORM UNTIL TRANSACTION-EOF
    PERFORM 200-EDIT-UPDATE-TRANSACTION
    IF NO-ERRORS
        PERFORM 300-UPDATE-COMMUTER-RECORD
    ELSE
        PERFORM 400-PRINT-TRANSACTION-ERRORS
    END-IF
    READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD
        AT END
            SET TRANSACTION-EOF TO TRUE
    END-READ
END-PERFORM
```

  インライン PERFORM ステートメントには、明示的範囲終了符号が必須ですが、これはライン外の PERFORM ステートメントについては無効です。

追加のプログラム制御として、条件ステートメントと一緒に NOT 句を使用することもできます。たとえば、NOT ON SIZE ERROR のように、特定の例外が発生しない場合に実行される命令を指定することができます。NOT 句は、CALL ステートメントの ON OVERFLOW 句とは一緒に使用できませんが、ON EXCEPTION 句とは一緒に使用できます。

条件ステートメントをネストしてはなりません。ネストされるステートメントは、命令ステートメント (または範囲区切りステートメント) でなければならず、命令ステートメントの規則に従っていなければなりません。

以下に、範囲終了符号なしでコーディングされる場合の条件ステートメントの例を示します。

- ON SIZE ERROR が指定された算術ステートメント。
- ON OVERFLOW が指定されたデータ操作ステートメント。
- ON OVERFLOW が指定された CALL ステートメント。
- INVALID KEY、AT END、または AT END-OF-PAGEが指定された I/O ステートメント。

- AT END が指定された RETURN。

関連概念
22 ページの『命令ステートメント』
『範囲終了符号』

関連タスク
83 ページの『プログラム・アクションの選択』

関連参照
条件ステートメント (*Enterprise COBOL 言語解説書*)

コンパイラー指示ステートメント

コンパイラー指示ステートメントは、プログラム・ロジックの一部ではありません。コンパイラー指示ステートメントは、コンパイラーに、プログラム構造、COPY 処理、リスト制御、または制御フローについて特定のアクションを行わせます。

関連参照
360 ページの『コンパイラー指示ステートメント』
コンパイラー指示ステートメント (*Enterprise COBOL 言語解説書*)

範囲終了符号

範囲終了符号は、明示的にも暗黙的にも使用できます。明示的範囲終了符号は、文を終了させることなく、動詞を終了させます。これは、END の後にハイフンと、終了させる動詞の名前を続けたものから構成されます (たとえば、END-IF)。暗黙の範囲終了符号はピリオド (.) で、これはまだ終了されていないすべての先行ステートメントの範囲を終了させます。

次のプログラムの断片における 2 つのピリオドは、それぞれ IF ステートメントを終了させます。そのため、このコードは、明示的範囲終了符号を持つ以下の例と同等です。

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM.
IF ITEM = "B"
    ADD 2 TO TOTAL.
```

```
IF ITEM = "A"
    DISPLAY "THE VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL
    MOVE "C" TO ITEM
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM
END-IF
IF ITEM = "B"
    ADD 2 TO TOTAL
END-IF
```

暗黙の範囲終了符号を使用すると、ステートメントの終わる場所が不明確になることがあります。結果として、ステートメントを意図に反して終了させ、プログラムのロジックが変わる可能性があります。明示的範囲終了符号を使用すると、プログラムが理解しやすくなり、ステートメントを意図に反して終了させることがなくなります。たとえば、次のプログラム断片で、最初の暗黙の範囲例の最初のピリオドの位置を変更すると、コードの意味が変更されます。

```
IF ITEM = "A"
    DISPLAY "VALUE OF ITEM IS " ITEM
    ADD 1 TO TOTAL.
    MOVE "C" TO ITEM
    DISPLAY " VALUE OF ITEM IS NOW " ITEM
IF ITEM = "B"
    ADD 2 TO TOTAL.
```

最初のピリオドが IF ステートメントを終わらせるため、その後の MOVE ステートメントおよび DISPLAY ステートメントは、字下げの意味を無視し、ITEM の値に関係なく実行されます。

プログラムをより読みやすくし、ステートメントの意図しない終了を防ぐために、特に段落内では、明示的範囲終了符号を使用するようにすべきです。暗黙の範囲終了符号は、段落の終わりまたはプログラムの終わりでのみ使用してください。

条件ステートメント内にネストされている命令ステートメントについての明示的範囲終了符号をコーディングする際には、注意が必要です。範囲終了符号が、それが意図されたステートメントと対にされるようにしてください。次の例では、範囲終了符号は最初の READ ステートメントと対になるように意図されましたが、実際には 2 つ目と対にされます。

```
READ FILE1
  AT END
    MOVE A TO B
    READ FILE2
END-READ
```

明示的範囲終了符号が意図されたステートメントと対にされるようにするために、上記の例を次のようにコーディングし直すことができます。

```
READ FILE1
  AT END
    MOVE A TO B
    READ FILE2
    END-READ
END-READ
```

関連概念

23 ページの『条件ステートメント』
22 ページの『命令ステートメント』

宣言

宣言は、例外条件が起こったときに実行される 1 つまたは複数の特殊目的セクションを提供します。

各宣言セクションは、そのセクションの機能を識別する USE ステートメントで始まります。プロシージャーの中に、条件が起こった場合に取るべきアクションを指定します。

関連タスク

367 ページの『入出力エラーの検出および処理』

関連参照

宣言部分 (*Enterprise COBOL 言語解説書*)

第 2 章 データの使用

この節は、非 COBOL プログラマーが他のプログラム言語で使用されている用語を COBOL 用語に関連付けるのに役立ちます。以下のものに対応する COBOL の基本事項を紹介します。

- 変数、構造、リテラル、および定数
- 値の割り当てと表示
- 組み込み関数
- テーブル (配列) とポインター

関連概念
38 ページの『ストレージとそのアドレス可能度』

関連タスク
『変数、構造、リテラル、および定数の使用』
29 ページの『データ項目への値の割り当て』
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
37 ページの『組み込み関数の使用 (組み込み関数)』
38 ページの『テーブル (配列) とポインターの使用』
113 ページの『第 7 章 実行時に各国語を使用するためのコーディング』

変数、構造、リテラル、および定数の使用

大部分の高水準プログラミング言語では、変数、構造、リテラル、および定数として表されるデータの概念は共通しています。データ定義はすべて、プログラムの DATA DIVISION に入れます。COBOL プログラム内で使用されるデータは、英字、英数字、または数字です。

変数

変数 とは、プログラムの実行中にその値が変わる可能性があるデータ項目です。ただし、値は、変数に名前と長さを与えるときに定義されたデータ型に制限されます。たとえば、顧客名がプログラムの変数である場合は、次のようにコーディングします。

```
Data Division.
. . .
01  Customer-Name               Pic X(20).
01  Original-Customer-Name      Pic X(20).
. . .
Procedure Division.
    . . .
    Move Customer-Name to Original-Customer-Name
    . . .
```

データ構造: データ項目とグループ項目

関連データ項目は、階層データ構造の一部となることができます。従属項目を持たないデータ項目は、基本 データ項目と呼ばれます。従属データ項目から構成される

© Copyright IBM Corp. 1991, 2001

データ項目は、グループ 項目と呼ばれます。レコードは、1 つの基本データ項目であることも、データ項目のグループであることもあります。

以下の例では、Customer-Record は 2 つのグループ項目 (Customer-Name と Part-Order) から構成されるグループ項目です。この 2 つのグループ項目には、基本データ項目が含まれています。PROCEDURE DIVISION の MOVE ステートメントが示しているように、グループ項目全体を参照することもできるし、グループ項目の一部を参照することもできます。

```
Data Division.
File Section.
FD  Customer-File
    Record Contains 45 Characters.
01  Customer-Record.
    05  Customer-Name.
        10 Last-Name            Pic x(17).
        10 Filler               Pic x.
        10 Initials             Pic xx.
    05  Part-Order.
        10 Part-Name            Pic x(15).
        10 Part-Color           Pic x(10).
Working-Storage Section.
01  Orig-Customer-Name.
    05  Surname                Pic x(17).
    05  Initials               Pic x(3).
01  Inventory-Part-Name        Pic x(15).
. . .
Procedure Division.
    . . .
    Move Customer-Name to Orig-Customer-Name
    Move Part-Name to Inventory-Part-Name
    . . .
```

リテラル

リテラル とは、値が文字それ自体によって与えられる文字ストリングのことです。データ項目に使用したい値がわかっている場合には、PROCEDURE DIVISION でデータ値のリテラル表記を使用してください。それを定義したり、データ名を参照したりする必要はありません。たとえば、次のようにして、出力ファイルについてのエラー・メッセージを準備することができます。

```
Move "Name is not valid" To Customer-Name
```

次のように、データ項目をある数値と比較することができます。

```
01  Part-number                Pic 9(5).
. . .
    If Part-number = 03519 then display "Part number was found"
```

これらの例では、"Name is not valid" は英数字リテラルであり、03519 は数字リテラルです。

定数

定数 は、1 つの値しか持たないデータ項目です。COBOL では、定数を表す構造を特に定義していません。しかし、大部分の COBOL プログラマーは、初期 VALUE を持つデータ項目を定義します (それに対して、変数は INITIALIZE ステートメントを使用して初期化します)。たとえば、次のように記述します。

```
Data Division.
. . .
01  Report-Header              pic x(50)  value "Company Sales Report".
. . .
01  Interest                   pic 9v9999 value 1.0265.
```

形象定数

一般に使用される特定の定数およびリテラルは、*形象定数* と呼ばれる予約語として提供されます。すなわち、ZERO、SPACE、HIGH-VALUE、LOW-VALUE、QUOTE、NULL、および ALL です。これらは固定値を表すため、形象定数はデータ定義を必要としません。たとえば、次のように記述します。

```
Move Spaces To Report-Header
```

関連参照

PICTURE 文節 (*Enterprise COBOL 言語解説書*)
リテラル (*Enterprise COBOL 言語解説書*)
形象定数 (*Enterprise COBOL 言語解説書*)

データ項目への値の割り当て

データ項目を定義した後、いつでもそれに値を割り当てることができます。COBOL における割り当ては、その背後にある目的によって多くの形式を取ります。

目的	方法
データ項目または大きいデータ域に値を割り当てる	以下のいずれかの方法を使用する。 • INITIALIZE ステートメント • MOVE ステートメント • STRING または UNSTRING ステートメント • VALUE 文節 (データ項目を、プログラムが初期状態にあるときにそれに与えたい値に設定する場合)
算術の結果を割り当てる	COMPUTE ステートメントを使用する。
データ項目内の文字または文字のグループを置き換える	INSPECT ステートメントを使用する。
ファイルから値を受け取る	READ (または READ INTO) ステートメントを使用する。
画面またはファイルから値を受け取る	ACCEPT ステートメントを使用する。
画面上またはファイル内に値を表示する	DISPLAY ステートメントを使用する。
定数を設定する[1]	データ項目の定義の中で VALUE 文節を使用する。[2]

1. 定数は COBOL のフィーチャーではありません。
2. 最適化コードの場合のみ: 最適化プログラムはインバリアントの VALUE 項目を認識し、それを定数として扱います。

30 ページの『例: 変数の初期化』

関連タスク

31 ページの『構造の初期化 (INITIALIZE)』

32 ページの『変数または構造への値の割り当て (MOVE)』
95 ページの『データ項目の結合 (STRING)』
97 ページの『データ項目の分割 (UNSTRING)』
32 ページの『算術結果の割り当て (MOVE または COMPUTE)』
104 ページの『データ項目の計算および置換 (INSPECT)』
33 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
113 ページの『第 7 章 実行時に各国語を使用するためのコーディング』

例: 変数の初期化

変数のブランクまたはゼロへの初期化:

```
INITIALIZE identifier-1
```

IDENTIFIER-1 **PICTURE**	*IDENTIFIER-1* **(初期化前)**	*IDENTIFIER-1* **(初期化後)**
9(5)	12345	00000
X(5)	AB123	*bbbbb*[1]
99XX9	12AB3	*bbbbb*[1]
XXBX/XX	ABbC/DE	*bbbb/bb*[1]
**99.9CR	1234.5CR	**00.0*bb*[1]
A(5)	ABCDE	*bbbbb*[1]
+99.99E+99	+12.34E+02	+00.00E+00
1. 記号 *b* は、ブランク・スペースを表します。		

右寄せフィールドの初期化:

```
01  ANJUST                 PIC X(8)  JUSTIFIED RIGHT.
01  ALPHABETIC-1           PIC A(4)  VALUE "ABCD".
. . .
    INITIALIZE ANJUST
        REPLACING ALPHANUMERIC DATA BY ALPHABETIC-1
```

ALPHABETIC-1	**ANJUST (初期化前)**	**ANJUST (初期化後)**
ABCD	*bbbbbbbb*[1]	*bbbb*ABCD[1]
1. 記号 *b* は、ブランク・スペースを表します。		

英数字変数の初期化:

```
01  ALPHANUMERIC-1         PIC X.
01  ALPHANUMERIC-3         PIC X(1) VALUE "A".
. . .
    INITIALIZE ALPHANUMERIC-1
        REPLACING ALPHANUMERIC DATA BY ALPHANUMERIC-3
```

ALPHANUMERIC-3	**ALPHANUMERIC-1 (初期化前)**	**ALPHANUMERIC-1 (初期化後)**
A	y	A

数値変数の初期化:

```
01  NUMERIC-1              PIC 9(8).
01  NUM-INT-CMPT-3         PIC 9(7) COMP VALUE 1234567.
. . .
    INITIALIZE NUMERIC-1
        REPLACING NUMERIC DATA BY NUM-INT-CMPT-3
```

NUM-INT-CMPT-3	NUMERIC-1 (初期化前)	NUMERIC-1 (初期化後)
1234567	98765432	01234567

編集済み英数字変数の初期化:

```
01  ALPHANUM-EDIT-1        PIC XXBX/XXX.
01  ALPHANUM-EDIT-3        PIC X/BB VALUE "M/bbb".
. . .
    INITIALIZE ALPHANUM-EDIT-1
        REPLACING ALPHANUMERIC-EDITED DATA BY ALPHANUM-EDIT-3
```

ALPHANUM-EDIT-3	ALPHANUM-EDIT-1 (初期化前)	ALPHANUM-EDIT-1 (初期化後)
M/*bb*[1]	AB*b*C/DEF[1]	M/*bb*/*bbb*[1]
1. 記号 *b* は、ブランク・スペースを表します。		

関連タスク

『構造の初期化 (INITIALIZE)』

構造の初期化 (INITIALIZE)

INITIALIZE ステートメントをグループ項目に適用することによって、グループ内のすべての従属項目の値を初期化することができます。ただし、グループ内のすべての項目を初期化することが必要な場合を除き、グループ全体を初期化することは非効率的です。

以下の例では、プログラムによって作成されたトランザクション・レコードの中のフィールドをスペースまたは 0 に初期化する方法を示します。フィールドは、作成される各レコードで同一というわけではありません。

```
01  TRANSACTION-OUT.
    05  TRANSACTION-CODE              PIC X.
    05  PART-NUMBER                   PIC 9(6).
    05  TRANSACTION-QUANTITY          PIC 9(5).
    05  PRICE-FIELDS.
        10  UNIT-PRICE                PIC 9(5)V9(2).
        10  DISCOUNT                  PIC V9(2).
        10  SALES-PRICE               PIC 9(5)V9(2).
. . .
    INITIALIZE TRANSACTION-OUT
```

レコード	TRANSACTION-OUT (初期化前)	TRANSACTION-OUT (初期化後)
1	R00138300024000000000000000	*b*0000000000000000000000000[1]
2	R00139000048000000000000000	*b*0000000000000000000000000[1]
3	S00141000012000000000000000	*b*0000000000000000000000000[1]
4	C00138300000000425000000000	*b*0000000000000000000000000[1]
5	C00201000000000000100000000	*b*0000000000000000000000000[1]
1. 記号 *b* は、ブランク・スペースを表します。		

変数または構造への値の割り当て (MOVE)

変数または構造に値を割り当てるには、MOVE ステートメントを使用してください。

たとえば、次のステートメントは、変数 Customer-Name の内容を変数 Orig-Customer-Nameに割り当てます。

```
Move Customer-Name to Orig-Customer-Name
```

Customer-Name が Orig-Customer-Name より長い場合は、右側で切り捨てが起こります。短い場合には、右側の余分な文字位置がスペースで埋められます。

グループ項目を別のグループ項目に移動するときには、従属データ記述に互換性があることを確認してください。コンパイラーは、項目が適合するかどうかに関係なく (実行時に破壊的なオーバーラップが起こる可能性がある場合でも)、すべての MOVE ステートメントを実行します。

数値を含む変数の場合、数値には複数の表現方法があるため、移動がより複雑になることがあります。一般に、可能であれば、数値の代数的な値が移動されます (文字データに関して実行される桁単位の移動ではなく)。

```
01  Item-x            Pic 999v9.
. . .
    Move 3.06 to Item-x
```

この移動の結果、Item-x には、0030 によって表される値 3.0 が入ります。

英数字項目または整数項目を、国別変数に移動できます。このとき、項目は変換されます。国別項目は、別の国別変数にのみ移動できます。いずれの場合も埋め込みまたは切り捨てが発生する可能性がありますが、切り捨てが 1 つの文字内で発生しないようにしてください。

以下の例は、ギリシャ語のデータ項目を国別データ項目に移動する場合を示しています。

```
 CBL CODEPAGE(00875)
. . .
01 Data-in-Unicode   Pic N(100) usage national.
01 Data-in-Greek     Pic X(100).
. . .
    Read Greek-file into Data-in-Greek
    Move Data-in-Greek to Data-in-Unicode
```

関連概念

115 ページの『Unicode および言語文字のエンコード』

関連参照

MOVE ステートメント (*Enterprise COBOL 言語解説書*)

318 ページの『CODEPAGE』
118 ページの『国別データの変換』

算術結果の割り当て (MOVE または COMPUTE)

変数に数値を割り当てるときには、MOVE ステートメントではなく COMPUTE ステートメントを使用することを考慮してください。たとえば、次の 2 つのステートメントは、ほとんどの場合は同じことを行います。

```
Move w to z
Compute z = w
```

MOVE ステートメントは、切り捨てを伴う割り当てを実行します。ただし、DIAGTRUNC コンパイラー・オプションを指定して、数値受け取り側で切り捨てが起こる可能性のある MOVE ステートメントについてコンパイラーが警告診断を出すように要求することができます。

ただし、実行時に左側の有効数字が失われる場合、COMPUTE ステートメントを使用するとこの条件を検出し、それに対処することができます。

COMPUTE ステートメントの ON SIZE ERROR 句を使用すると、コンパイラーはサイズ・オーバーフロー条件を検出するコードを生成します。条件が起こると、ON SIZE ERROR 句内のコードが実行され、z の内容は未変更のままになります。ON SIZE ERROR 句が指定されていない場合には、割り当ては切り捨てを伴って実行されます。MOVE ステートメントの ON SIZE ERROR サポートはありません。

COMPUTE ステートメントを使用して、算術式 (または組み込み関数) の結果を変数に割り当てることもできます。たとえば、次のように記述します。

```
Compute z = y + (x ** 3)
Compute x = Function Max(x y z)
```

言語環境プログラム呼び出し可能サービスを使用すれば、日付、時刻、および数値計算、そしてその他の演算の結果をデータ項目に割り当てることができます。これらの言語環境プログラム・サービスは、標準の COBOL CALL ステートメントを介して使用することができ、それらが戻す値は CALL ステートメントのパラメーターに入れて渡されます。たとえば、次のステートメントを使用して、変数の絶対値を見つけるために言語環境プログラム・サービス CEESIABS を呼び出すことができます。

```
Call 'CEESIABS' Using Arg, Feedback-code, Result.
```

この呼び出しの結果、変数 Result には、変数 Arg の中にある値の絶対値が割り当てられます。変数 Feedback-code には、サービスが正常に完了したかどうかを示す戻りコードが入ります。使用する呼び出し可能サービスの要件に従って、正しい記述を使用してすべての変数を DATA DIVISION で定義しなければなりません。上記の例の場合は、変数を次のように定義することができます。

```
77 Arg                Pic s9(9)  Binary.
77 Feedback-code      Pic x(12)  Display.
77 Result             Pic s9(9)  Binary.
```

関連参照

324 ページの『DIAGTRUNC』
組み込み関数 (*Enterprise COBOL 言語解説書*)
呼び出し可能サービス (*言語環境プログラム プログラミング・リファレンス*)

画面またはファイルからの入力の割り当て (ACCEPT)

変数に値を割り当てる方法の 1 つとして、画面またはファイルから値を読み取ることができます。画面からデータを入力するには、最初にモニターを SPECIAL-NAMES 段落内の簡略名と関連付けます。次に、ACCEPT を使用して、画面から入力された入力行を変数に割り当てます。

たとえば、次のように記述します。

```
Environment Division.
Configuration Section.
Special-Names.
    Console is Names-Input.
. . .
    Accept Customer-Name From Names-Input
```

画面ではなくファイルから読み取る場合は、次のように変更します。

- Console を *device* に変更します (ここで、*device* は任意の有効なシステム装置 (たとえば、SYSIN) です)。たとえば、次のように記述します。

  ```
  SYSIN is Names-Input
  ```

  device は、階層ファイル・システム (HFS) のパスを参照する DD 名にすることもできます。この DD が定義されていない場合、プログラムが UNIX 環境で実行されていれば、stdin が入力ソースになります。この DD が定義されておらず、プログラムが UNIX 環境で実行されていない場合には、ACCEPT ステートメントは失敗します。

値を国別データ項目に割り当てるときは、コンソールからの入力データは CODEPAGE オプションの値に基づいて、EBCDIC から Unicode 表現 (UTF-16) に変換されます。ACCEPT ステートメントの使用時に国別データの変換が実行されるのはこの場合だけです。これは、入力が画面からのものであることが認識されるためです。

入力データが他の装置からのものであるときに変換を実行したい場合は、NATIONAL-OF 組み込み関数を使用します。

関連概念
115 ページの『Unicode および言語文字のエンコード』

関連参照
318 ページの『CODEPAGE』
SPECIAL-NAMES 段落 (*Enterprise COBOL 言語解説書*)

画面上またはファイル内での値の表示 (DISPLAY)

DISPLAY ステートメントを使用すると、変数の値を画面に表示したり、ファイルに書き込むことができます。たとえば、次のように記述します。

```
Display "No entry for surname '" Customer-Name
    "' found in the file.".
```

変数 *Customer-Name* の内容が JOHNSON の場合、上記のステートメントは、次のメッセージをシステム論理出力装置に表示します。

```
No entry for surname 'JOHNSON' found in the file.
```

データをシステム論理出力装置以外の宛先に書き込みたい場合には、SYSOUT 以外の宛先を指定した UPON 文節を使用してください。たとえば、次のステートメントは、SYSPUNCH DD ステートメントで指定されたファイルに書き込みを行います。

```
Display "Hello" upon syspunch.
```

この DD 名に階層ファイル・システム (HFS) のファイルを指定できます。たとえば、次の宛先を使用すると、DISPLAY 出力は HFS ファイル /u/userid/cobol/demo.lst に書き込まれます。

```
//SYSPUNCH DD PATH='/u/userid/cobol/demo.lst',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

次のステートメントは、ジョブ・ログまたはコンソール、および TSO 画面 (TSO のもとで実行している場合) に書き込みを行います。

```
Display "Hello" upon console.
```

国別データ項目の値をコンソールに表示するときは、値は CODEPAGE オプションの値に基づいて、Unicode 表現 (UTF-16) から EBCDIC に変換されます。DISPLAY ステートメントの使用時に国別データの変換が実行されるのはこの場合だけです。これは、出力の宛先が画面であることが認識されるためです。

出力の宛先が別の装置であるときに国別データ項目を変換するには、DISPLAY-OF 組み込み関数 (以下の例を参照) を使用します。

```
01 Data-in-Unicode pic N(10) usage national.
. . .
    Display function Display-of(Data-in-Unicode, 00037)
```

関連概念

115 ページの『Unicode および言語文字のエンコード』

システム論理出力装置上でのデータの表示

データをシステム論理出力装置に書き込むには、UPON 文節を省略するか、または宛先 SYSOUT を指定した UPON 文節を使用してください。たとえば、次のように記述します。

```
Display "Hello" upon sysout.
```

出力は、OUTDD コンパイラー・オプションで指定された DD 名に送られます。この DD 名に階層ファイル・システム (HFS) のファイルを指定できることに注意してください。

OUTDD という DD 名が割り振られておらず、プログラムが UNIX 環境で実行されていない場合には、SYSOUT=* というデフォルトの DD が割り振られます。

OUTDD という DD 名が割り振られていないときに、プログラムが UNIX 環境で実行されていれば、_IGZ_SYSOUT 環境変数が次のように使用されます。

未定義または stdout に設定されている
: 出力は stdout に送られます (ファイル記述子 1)。

stderr に設定されている
: 出力は stderr に送られます (ファイル記述子 2)。

それ以外 (stdout または stderr 以外に設定されている)
: DISPLAY ステートメントは失敗します。重大度 3 の言語環境プログラム条件が発生します。

DISPLAY 出力が stdout または stderr に経路指定された場合、出力はレコード単位に分割されません。それどころか、出力は改行のない単一の文字ストリームとして書き込まれます。

OUTDD と言語環境プログラムのランタイム・オプションの MSGFILE が両方とも同じ DD 名を指定していると、DISPLAY 出力と言語環境プログラム実行時診断の両方が言語環境プログラムのメッセージ・ファイルに経路指定されます。

WITH NO ADVANCING の使用

WITH NO ADVANCING 句を指定し、出力が DD 名に送られる場合は、印刷制御文字 + (正符号) が次の DISPLAY ステートメントの最初の出力位置に入れられます。+ は ANSI 定義の印刷制御文字であり、これはレコード印刷前の行送りを抑止します。

WITH NO ADVANCING 句を指定した場合に、出力が stdout または stderr に送られるときは、ストリームの終わりに改行文字が追加されません。その後の DISPLAY ステートメントがストリームの終わりに文字を追加する場合があります。

WITH NO ADVANCING を指定せず、出力が DD 名に送られる場合は、印刷制御文字 ’ ’ (スペース) が次の DISPLAY ステートメントの最初の出力位置に入れられます (これはシングル・スペース出力を示します)。

たとえば、次のように記述します。

```
DISPLAY "ABC"
DISPLAY "CDEF" WITH NO ADVANCING
DISPLAY "GHIJK" WITH NO ADVANCING
DISPLAY "LMNOPQ"
DISPLAY "RSTUVWX"
```

上記のステートメントを使用した場合は、以下の結果が出力装置に送られます。

```
 ABC
 CDEF
+GHIJK
+LMNOPQ
 RSTUVMX
```

印刷される出力は、出力装置が印刷制御文字をどのように解釈するかによって異なります。

WITH NO ADVANCING 句を指定しなかった場合、出力が stdout または stderr に送られるときは、ストリームの終わりに改行文字が追加されます。

関連タスク

429 ページの『環境変数の設定およびアクセス』
408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
118 ページの『国別データの変換』

関連参照

DISPLAY ステートメント (*Enterprise COBOL 言語解説書*)
340 ページの『OUTDD』
318 ページの『CODEPAGE』

組み込み関数の使用 (組み込み関数)

一部の高水準プログラム言語には組み込み関数があります。組み込み関数は、プログラム内で、定義済み属性および事前定義値を持つ変数であるかのように参照することができます。COBOL では、これらは *組み込み関数* (intrinsic functions) と呼ばれます。これらの関数はストリングと数値を操作する機能を提供します。

組み込み関数の値は参照時に自動的に導かれるため、関数を DATA DIVISION で定義する必要はありません。引き数として使用する非リテラル・データ項目だけを定義してください。形象定数を引き数として使用することはできません。

関数 ID は、COBOL 予約語 FUNCTION と、それに続く関数名 (Max など) と、それに続く関数の評価に使用される任意の引き数 (x、y、z など) の組み合わせです。

以下の例で強調されているワードのグループは、*関数 ID* と呼ばれます。

```
Unstring Function Upper-case(Name) Delimited By Space Into Fname Lname
Compute A = 1 + Function Log10(x)
Compute M = Function Max(x y z)
```

関数 ID は、関数の呼び出しと、関数によって戻されるデータ値の両方を表します。関数 ID は、実際にデータ項目を表すため、戻される値の属性を持つデータ項目が使用できるところならば、PROCEDURE DIVISION のほとんどの場所で使用することができます。

COBOL ワード function は予約語ですが、関数名は予約されていません。このため、関数名を他のコンテキスト (変数の名前など) で使用することができます。たとえば、Sqrt は、組み込み関数を呼び出し、プログラム内の変数を指定するために使用できます。

```
Working-Storage Section.
01  x                 Pic 99  value 2.
01  y                 Pic 99  value 4.
01  z                 Pic 99  value 0.
01  Sqrt              Pic 99  value 0.
. . .
    Compute Sqrt = 16 ** .5
    Compute z = x + Function Sqrt(y)
    . . .
```

組み込み関数の型

関数 ID は、関数の型によって、文字ストリング (英数字データ・クラス) または数値 (数値データ・クラス) のいずれかである値を表します。英数字関数の関数 ID には、サブストリング指定 (参照修飾子) を組み込むことができます。数値組み込み関数は、それらが戻す数値のタイプに応じてさらに分類されます。

関数 MAX、MIN、DATEVAL、および UNDATE は、指定された引き数の型に応じていずれかのタイプの値を戻します。

関数 DATEVAL、UNDATE、および YEARWINDOW は、ウィンドウ化日付フィールドの操作および変換を援助するために、2000 年言語拡張として提供されています。

関数のネスト

関数は、ネストされた関数の結果が外側の関数の引き数についての要件を満たす限り、引き数として他の関数を参照することができます。たとえば、次のように記述します。

```
Compute x = Function Max((Function Sqrt(5)) 2.5 3.5)
```

この場合、Function Sqrt(5) は数値を戻します。したがって、MAX 関数への 3 つの引き数はすべて、この関数についての許容される引き数型である数値になります。

関連タスク

80 ページの『組み込み関数を使用したテーブル項目の処理』
105 ページの『データ項目の変換 (組み込み関数)』
107 ページの『データ項目の評価 (組み込み関数)』

テーブル (配列) とポインターの使用

COBOL では、配列はテーブルと呼ばれます。テーブルは、OCCURS 文節を使用して DATA DIVISION に定義される論理的に連続するデータ項目の集合です。

ポインターは、仮想記憶アドレスを含むデータ項目です。ポインターは、USAGE IS POINTER 文節を使用して DATA DIVISION の中に明示的に定義するか、または ADDRESS OF 特殊レジスターとして暗黙的に定義します。

ポインター・データ項目には以下の操作を実行することができます。

- それらを CALL . . BY REFERENCE ステートメントを使用してプログラム間で受け渡す。
- SET ステートメントを使用してそれらを他のポインターへ移動する。
- 比較条件を使って他のポインターと比較し、等しいかどうかを調べる。
- VALUE IS NULL を使用して、それらが無効なアドレスを含むように初期化する。

ポインター・データ項目は、以下のことを行うために使用してください。

- 限定された基底アドレッシングの実施。特に、レコード域 (OCCURS DEPENDING ON で定義される、したがって可変位置の) のアドレスを受け渡したい場合。
- チェーン・リストの処理。

関連タスク

65 ページの『テーブルの定義 (OCCURS)』
452 ページの『プロシージャー・ポインターと関数ポインターの使用』

ストレージとそのアドレス可能度

COBOL プログラムを実行するときは、プログラムおよびプログラムで使用されるデータは仮想記憶域にあります。COBOL で使用するストレージは、16MB 境界より下でも 16MB 境界より上でも構いませんが、2GB の境界を超えないようにしてください。このストレージをアドレッシングするときは、24 ビットおよび 31 ビットの 2 つのアドレッシング・モードが使用可能です。

24 ビット・アドレッシングを使用して、16MB 境界より下のストレージをアドレッシングすることができます (ただし、16MB 境界より上のアドレッシングはできません)。31 ビット・アドレッシングを使用すると、16MB 境界より上と 16MB 境界より下のどちらのストレージでもアドレッシングできます。31 ビット・アドレッシングを使用して、*無制限のストレージ* をアドレッシングできます。したがって、16MB 境界より上と 16MB 境界より下の両方を含む、プログラムで使用可能なすべてのストレージをアドレッシングできます。

Enterprise COBOL は、z/OS の 64 ビットの仮想アドレッシング機能を直接には利用しませんが、31 ビット・アドレッシング・モードまたは 24 ビット・アドレッシング・モードで実行している COBOL アプリケーションは、64 ビット z/OS システムで完全にサポートされます。

アドレッシング・モード (AMODE) は、ユーザーのプログラムによってどのハードウェア・アドレッシング・モードがサポートされるかを示す属性です。24 ビット・アドレッシング、31 ビット・アドレッシング、24 ビットまたは 31 ビット・アドレッシングのいずれかを示す属性があります。この属性はそれぞれ AMODE 24、AMODE 31、または AMODE ANY になります。オブジェクト・プログラム、ロード・モジュール、および実行プログラムにはそれぞれ AMODE 属性があります。Enterprise COBOL オブジェクト・プログラムはすべて AMODE ANY です。

常駐モード (RMODE) は、仮想記憶域のどこにプログラムが常駐するかを示すプログラム・ロード・モジュールの属性です。16MB 境界の下、16MB 境界の下または上の 2 つの属性があります。この属性は RMODE 24 または RMODE ANY です。

Enterprise COBOL は、実行時に使用されるストレージを制御する言語環境プログラム・サービスを使用します。したがって、COBOL コンパイラー・オプションおよび言語環境プログラムのランタイム・オプションは、単独でまたは組み合わせによって、プログラムとデータの AMODE 属性および RMODE 属性に影響を与えます。

DATA 作業用ストレージ・データ、入出力バッファー、およびパラメーター・リスト (PLIST) 用ストレージのロケーションに影響を及ぼすコンパイラー・オプション。

RMODE 常駐モード (RMODE) に影響を及ぼすコンパイラー・オプション。

RENT 再入可能プログラムを生成するコンパイラー・オプションで、ストレージのロケーション、およびプログラムの常駐モードに影響を及ぼす。

HEAP 実行時ヒープ用のストレージを制御するランタイム・オプション。たとえば、COBOL 作業用ストレージはヒープ・ストレージから割り振られる。

STACK 実行時スタック用のストレージを制御するランタイム・オプション。たとえば、COBOL ローカル・ストレージはスタック・ストレージから割り振られる。

ALL31 アプリケーションを完全に AMODE 31 で実行できるかどうかを指定するランタイム・オプション。

RMODE の設定

RMODE オプションと RENT オプションは、プログラムの RMODE 属性を決定します。

RMODE コンパイラー・オプション	RENT コンパイラー・オプション	RMODE 属性
RMODE(AUTO)	NORENT	RMODE 24
RMODE(AUTO)	RENT	RMODE ANY
RMODE(24)	RENT または NORENT	RMODE 24
RMODE(ANY)	RENT または NORENT	RMODE ANY

データの受け渡しに関するストレージ制限

16MB 境界より上のストレージで割り振られたパラメーターを AMODE 24 サブプログラムに渡してはなりません。31 ビット・アドレッシング・モードで実行しているプログラムの場合は、WORKING-STORAGE データおよびパラメーター・リストを強制的に境界より下に配置して、AMODE 24 で実行しているプログラムにデータを渡してください。

- DATA(24) を指定して、再入可能プログラム (RENT) をコンパイルする。
- RMODE(24) または AMODE(AUTO)を指定して、再入不可プログラム (NORENT) をコンパイルする。
- RMODE(ANY) を指定してコンパイルされた再入不可プログラム (NORENT) は、RMODE 24 を指定してリンク・エディットする必要があります。NORENT プログラムのデータ域は、DATA(24) が指定されている場合でも、プログラムのロード先に応じて、16MB 境界より上または下になります。DATA オプションは、NORENT を指定してコンパイルされたプログラムに影響を与えません。

データ域のロケーション

再入可能プログラムの場合は、DATA コンパイラー・オプションおよび HEAP ランタイム・オプションによって、データ域のストレージ (WORKING-STORAGE SECTION や FD レコード域など) を 16MB 境界より下から獲得するのか、制限のないストレージから獲得するのかを制御します。 プログラムが 16MB 境界より上の仮想記憶域アドレスで 31 ビット・アドレス方式で実行される場合は、プログラムを RENT または RMODE(ANY) でコンパイルしなければなりません。DATA オプションは、NORENT を指定してコンパイルされたプログラムに影響を与えません。

ランタイム・オプション HEAP(,,BELOW) を指定すると、DATA コンパイラー・オプションは効力を持ちません。LOCAL-STORAGE SECTION データ域のストレージは、16MB 境界より下から割り振られます。しかし、HEAP(,,ANYWHERE) をランタイム・オプションとして指定すると、データ域のストレージは、プログラムを DATA(24) コンパイラー・オプションを使用してコンパイルした場合は 16MB 境界より下から割り振られ、DATA(31) コンパイラー・オプションを使用してコンパイルした場合は、制限のないストレージから割り振られます。

LOCAL-STORAGE データ項目のロケーションは、STACK ランタイム・オプションおよびプログラムの AMODE によって制御されます。LOCAL-STORAGE データ項目は、STACK(,,ANYWHERE) ランタイム・オプションが有効で、プログラムが AMODE 31 で実行されているときは、制限のないストレージで獲得されます。それ以外の場合、

LOCAL-STORAGE は 16MB 境界より下で獲得されます。DATA コンパイラー・オプションは、LOCAL-STORAGE データのロケーションに影響を与えません。

外部データ用のストレージ

DATA コンパイラー・オプションは、動的データ域 (WORKING-STORAGE、FD レコード域、およびパラメーター・リスト) のストレージ獲得方法に影響を与えるだけでなく、EXTERNAL データ用のストレージをどこから獲得するかにも影響を与えることがあります。EXTERNAL データに必要なストレージは、次の条件が満たされる場合には、制限のないストレージから獲得されます。

- プログラムが DATA(31) および RENT でコンパイルされているか、またはプログラムが RMODE(ANY) および NORENT コンパイラー・オプションでコンパイルされている。
- HEAP(,,ANYWHERE) ランタイム・オプションが有効である。
- ALL31(ON) ランタイム・オプションが有効である。

これ以外の場合はすべて、EXTERNAL データ用のストレージは 16MB 境界より下から獲得されます。ALL31(ON) ランタイム・オプションを指定するためには、実行単位内のすべてのプログラムが 31 ビット・アドレッシング・モードで実行可能でなければなりません。

QSAM 入出力バッファー用のストレージ

DATA コンパイラー・オプションは、QSAM ファイルの入出力バッファーをどこに獲得するかにも影響する場合があります。QSAM ファイルのバッファーの割り振りおよび DATA コンパイラー・オプションについては、以下の関連参照情報を参照してください。

関連概念

ヒープ・ストレージの AMODE 考慮事項 (*言語環境プログラム プログラミング・ガイド*)

関連タスク

437 ページの『第 24 章 サブプログラムの使用』
455 ページの『第 25 章 データの共用』

関連参照

157 ページの『QSAM ファイル用のバッファーの割り振り』
321 ページの『DATA』
343 ページの『RENT』
344 ページの『RMODE』
603 ページの『パフォーマンスに関連するコンパイラー・オプション』
HEAP (*言語環境プログラム プログラミング・リファレンス*)
STACK (*言語環境プログラム プログラミング・リファレンス*)
ALL31 (*言語環境プログラム プログラミング・リファレンス*)
z/OS DFSMS: Program Management

第 3 章 数値および算術演算

一般的に、COBOL 数値データは、一連の 10 進数字の桁として表示することができます。ただし、数値項目は、算術符号や通貨記号などの特殊な特性を持つこともできます。

この節では、算術演算を効率よく実行できるように、数値データを定義、表示、および格納する方法について説明します。

- 数値データを定義するには、PICTURE 文節と、文字 9、+、-、P、S、および V を使用します。
- 数値データを表示するには、PICTURE 文節と、MOVE および DISPLAY ステートメントと一緒に編集文字 (Z、コンマ、ピリオドなど) を使用します。
- 数値データの格納方法を制御するには、さまざまな形式を指定した USAGE 文節を使用します。
- データ値が適切であるかどうかを妥当性検査するには、数値のクラス・テストを使用します。
- 算術を実行するには、ADD、SUBTRACT、MULTIPLY、DIVIDE、および COMPUTE ステートメントを使用します。
- 必要な通貨記号を指定するには、CURRENCY SIGN 文節と適切な PICTURE 文字を使用します。

関連タスク
『数値データの定義』
44 ページの『数値データの表示』
45 ページの『数値データの保管方法の制御』
53 ページの『非互換データの検査 (数値のクラス・テスト)』
54 ページの『算術の実行』
63 ページの『通貨記号の使用』

数値データの定義

数値項目を定義するには、数値の 10 進数の桁数を表すためにデータ記述で文字 9 を指定した PICTURE 文節を使用します。X を使用してはなりません。これは英数字項目用です。

```
05  Count-y           Pic 9(4)   Value 25.
05  Customer-name     Pic X(20)  Value "Johnson".
```

PICTURE 文節には最大 18 桁までコーディングすることができます (デフォルト・コンパイラー・オプション ARITH(COMPAT) (*互換モード* と呼ばれる) を使用してコンパイルする場合)。 ARITH(EXTEND) (*拡張モード* と呼ばれる) を使用してコンパイルする場合は、PICTURE 文節には最大 31 桁までコーディングすることができます。

それ以外にコーディングできる特殊な意味を持つ文字は、次のとおりです。

P　　先行ゼロまたは後続ゼロを示します。

© Copyright IBM Corp. 1991, 2001

S　　正または負の符号を示します。

V　　小数点を暗黙指定します。

次の例の s は、値が符号付きであることを意味します。

```
05  Price             Pic s99v99.
```

したがって、このフィールドには、正または負の値を格納することができます。v は、暗黙の小数点の位置を示しますが、ストレージ上の位置を占めないので、項目のサイズには含まれません。s も、デフォルトではストレージの位置を占めないので、数値項目のサイズには含まれません。

しかし、プログラムまたはデータを異なるマシンに移植する予定がある場合には、ストレージ上に独立した場所を占めるように符号をコーディングすることができます。この場合、符号は 1 バイトを占めます。

```
05  Price             Pic s99V99   Sign Is Leading, Separate.
```

このようにすれば、現在使用中のマシンとは非分離符号を格納するための規則が異なるマシンを使用する場合に、予測外の結果が生じることがなくなります。

印刷または表示が予定されるデータ項目の場合にも、分離符号の方が望ましくなります。

PICTURE 文節に内部浮動小数点データ (COMP-1 または COMP-2) を指定することはできません。ただし、VALUE 文節を使用すれば、浮動小数点リテラルの初期値を与えることができます。

```
05  Compute-result    Usage Comp-2    Value 06.23E-24.
```

49 ページの『例: 数値データおよび内部表現』

関連概念

619 ページの『付録 A. 中間結果と算術精度』

関連タスク

『数値データの表示』

45 ページの『数値データの保管方法の制御』
54 ページの『算術の実行』

関連参照

52 ページの『符号表現および処理』
316 ページの『ARITH』
336 ページの『NUMPROC』

数値データの表示

数値項目を特定の編集記号 (小数点、コンマ、ドル記号、デビット記号、クレジット記号など) を付けて定義すると、データの表示または印刷時に、データをより見やすく理解しやすいようにすることができます。たとえば、以下のコードの中の Edited-price は数字編集項目です。

```
05  Price          Pic     9(5)v99.
05  Edited-price   Pic  $zz,zz9.99.
. . .
Move Price To Edited-price
Display Edited-price
```

Price の内容が 0150099 (値 1,500.99 を表す) であった場合には、コードを実行すると、$ 1,500.99 が表示されます。 Edited-price の PICTURE 文節内の z は、先行ゼロの抑止を示します。

数字編集項目は、算術式または ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントでオペランドとして使用することはできません。数字編集項目は、主として、数値データの表示または印刷のために使用されます。(数字編集項目に文節 USAGE IS DISPLAY を指定することができますが、これは暗黙の指定です。これは、項目が文字形式で保管されることを意味します。)

数字編集項目は、数値項目または数字編集項目に移動することができます。次の例では、数字編集項目の値が数値項目に移動されます。

```
Move Edited-price to Price
Display Price
```

これらの 2 つのステートメントを、前の例で示されているステートメントの直後に続けた場合には、Price は 0150099 (値 1,500.99 を表す) として表示されます。

49 ページの『例: 数値データおよび内部表現』

関連タスク

『数値データの保管方法の制御』
43 ページの『数値データの定義』
54 ページの『算術の実行』

関連参照

MOVE ステートメント (*Enterprise COBOL 言語解説書*)

数値データの保管方法の制御

データ記述記入項目に USAGE 文節をコーディングすることによって、コンピューターが数値データを保管する方法を制御することができます。次のような理由から、形式を制御したいと思う場合があります。

- 計算データ型での算術の方が、USAGE DISPLAY データ型での算術より効率的である。
- パック 10 進数形式の方が、USAGE DISPLAY データ型に比べ、1 桁当たりのストレージが少なくて済む。
- パック 10 進数形式の方が 2 進数形式の場合よりも、DISPLAY 形式との変換が効率的である。
- 浮動小数点形式は、位取りが大きく変化するような算術オペランドおよび結果を格納するのに最適であり、最大数の有効数字が維持される。
- データをあるマシンから別のマシンに移すときに、データ形式を保持する必要がない。

プログラム内で使用する数値データは、COBOL で使用可能な以下の形式のいずれかです。

- 外部 10 進数 (USAGE DISPLAY)
- 外部浮動小数点 (USAGE DISPLAY)
- 内部 10 進数 (USAGE PACKED-DECIMAL)
- 2 進数 (USAGE BINARY)
- 固有 2 進数 (USAGE COMP-5)
- 内部浮動小数点 (USAGE COMP-1、USAGE COMP-2)

COMP および COMP-4 は BINARY と同義であり、COMP-3 は PACKED-DECIMAL と同義です。

コンパイラーは、表示可能な数値をそれらの数値の内部表現に変換してから、それらを算術演算で使用します。このため、データ項目を DISPLAY ではなく BINARY または PACKED-DECIMAL として定義した方がより効率的な場合が多くなります。たとえば、次のように記述します。

```
05  Initial-count   Pic S9(4)  Usage Binary   Value 1000.
```

値の内部表現の制御にどの USAGE 文節を使用しても、使用する PICTURE 文節の規則および VALUE 文節内の 10 進値は同じです (ただし浮動小数点データを除く。浮動小数点には PICTURE 文節は使用できません)。

49 ページの『例: 数値データおよび内部表現』

関連概念

『数値データの形式』
50 ページの『データ形式の変換』
619 ページの『付録 A. 中間結果と算術精度』

関連タスク

43 ページの『数値データの定義』
44 ページの『数値データの表示』
54 ページの『算術の実行』

関連参照

51 ページの『変換および精度』
52 ページの『符号表現および処理』

数値データの形式

数値データに使用できる形式は、次のとおりです。

外部 10 進数 (DISPLAY) 項目

データ項目に対して USAGE DISPLAY が有効であると (USAGE DISPLAY を明示的にコーディングしたため、またはデフォルトによって)、ストレージのそれぞれの位置 (バイト) には 1 つの 10 進数字が入れられます。すなわち、項目は表示可能な形式で保管されます。

外部 10 進数 (ゾーン *10 進数* とも呼ばれます) 項目は、主として、プログラムとファイル、端末、またはプリンターとの間で数値を送受信することを目的としています。外部 10 進数項目は、算術処理で、オペランドおよび受け取り側として使用することもできます。ただし、プログラムで多数の算術計算を集中的に実行し、効率を優先させるのであれば、算術計算で使用するデータ項目には、COBOL の計算数値タイプを使用した方がよい場合もあります。

外部浮動小数点 (DISPLAY) 項目

浮動小数点形式でコーディングされた表示可能な数値は、*外部浮動小数点項目* と呼ばれます。外部浮動小数点項目は、外部 10 進数項目と同様に、USAGE DISPLAY をコーディングして明示的に定義するか、USAGE 文節を省略することによって暗黙的に定義します。外部浮動小数点項目に VALUE 文節を使用することはできません。

次の例では、Compute-Result が外部浮動小数点項目として暗黙的に定義されます。ストレージの各バイトには 1 つの文字 (v を除く) が入ります。

```
05  Compute-Result     Pic -9v9(9)E-99.
```

負符号 (-) は、仮数および指数が必ず負の数値でなければならないことを意味するものではありません。負符号は、数値が表示されるときに、正数であれば符号がブランクとなり、負数であれば符号が負符号となることを意味しています。正符号 (+) をコーディングすると、符号は、正数であれば正符号となり、負数であれば負符号となります。

浮動小数点数は、外部 10 進数と同様、(コンパイラーによって) 数値の内部表現に変換しなければ、算術演算で使用することはできません。デフォルト・オプションの ARITH (COMPAT) を使用してコンパイルした場合、外部浮動小数点数は長精度 (64 ビット) の浮動小数点形式に変換されます。ただし、ARITH (EXTEND) を使用してコンパイルすれば、外部浮動小数点数は拡張精度 (128 ビット) の浮動小数点形式に変換されます。

2 進数 (COMP) 項目

BINARY、COMP、および COMP-4 は、どのプラットフォームにおいても同義語です。

2 進数形式の数値は、2、4、または 8 バイトのストレージを占めます。PICTURE 文節で項目が符号付きであることが指定されている場合は、左端のビットが演算符号として使用されます。

2 進数は、付随する PICTURE 記述が 4 個以下の 10 進数字であれば 2 バイトを占め、5 個から 9 個の 10 進数字であれば 4 バイトを占め、10 個から 18 個の 10 進数字であれば 8 バイトを占めます。9 桁以上の 2 進数項目には、コンパイラーによる余分な処理が必要となります。それらを SIZE ERROR 条件についてテストしたり、丸めたりするのは、他のタイプに比べて非効率的です。

2 進数項目には、たとえば、指標、添え字、スイッチ、および算術オペランドや結果を入れることができます。

2 進データ (BINARY、COMP、または COMP-4) の切り捨て方法を指定するには、TRUNC(STD|OPT|BIN) コンパイラー・オプションを使用してください。

固有 2 進数 (COMP-5) 項目

COMP-5 は、X/OPEN COBOL 仕様を基にした USAGE タイプです。

USAGE COMP-5 として宣言したデータ項目は、ストレージ内では 2 進データとして表されます。しかし、これらは、USAGE COMP 項目とは異なり、PICTURE 文節の 9 の数で暗黙指定される値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を含むことができます。

数値データを COMP-5 項目に移動または保管すると、COBOL PICTURE サイズ制限ではなく、2 進数フィールド・サイズで切り捨てが行われます。COMP-5 項目を参照する場合、演算では完全な 2 進数フィールド・サイズが使用されます。

したがって、COMP-5 は、データが COBOL PICTURE 文節に適合しない可能性のある非 COBOL プログラムから生じる 2 進データ項目の場合に特に有用です。

次の表は、COMP-5 データ項目の可能な値の範囲を示しています。

PICTURE	ストレージ表現	数値
S9(1) ～ S9(4)	2 進数ハーフワード (2 バイト)	-32768 ～ +32767
S9(5) ～ S9(9)	2 進数フルワード (4 バイト)	-2,147,483,648 ～ +2,147,483,647
S9(10) ～ S9(18)	2 進数ダブルワード (8 バイト)	-9,223,372,036,854,775,808 ～ +9,223,372,036,854,775,807
9(1) ～ 9(4)	2 進数ハーフワード (2 バイト)	0 ～ 65535
9(5) ～ 9(9)	2 進数フルワード (4 バイト)	0 ～ 4,294,967,295
9(10) ～ 9(18)	2 進数ダブルワード (8 バイト)	0 ～ 18,446,744,073,709,551,615

COMP-5 項目の PICTURE 文節に、スケーリング (すなわち、小数部の桁数または暗黙の整数桁数) を指定することができます。その場合は、上記にリストされた最大容量とほぼ同じ大きさをスケーリングする必要があります。たとえば、PICTURE S99V99 COMP-5 として記述したデータ項目は、ストレージ内では 2 進数ハーフワードとして表され、-327.68 ～ +327.67 までの範囲の値をサポートします。

VALUE 文節のラージ・リテラル: COMP-5 項目の VALUE 文節に指定されたリテラルは、一部の例外を除いて、固有 2 進数表現の容量までの大きさの値を含むことができます。例外については *Enterprise COBOL 言語解説書* を参照してください。

TRUNC コンパイラー・オプションの設定に関係なく、COMP-5 データ項目は、TRUNC(BIN) でコンパイルされたプログラムでは、2 進データのように動作します。

パック 10 進数 (COMP-3) 項目

PACKED-DECIMAL および COMP-3 は、どのプラットフォームにおいても同義語です。

パック 10 進数項目は、PICTURE 記述でコーディングされる 2 つの 10 進数字ごとに 1 バイトのストレージを占めます。ただし、右端のバイトだけが例外で、右端の

バイトには 1 つの数字と符号が入ります。この形式が最も効率的に使用されるのは、PICTURE 記述で奇数の桁をコーディングして、左端のバイトが完全に使用されるようにするときです。パック 10 進数項目は、算術演算の目的では固定小数点数として扱われます。

浮動小数点 (COMP-1 および COMP-2) 項目

COMP-1 は短精度浮動小数点形式を指し、COMP-2 は長精度浮動小数点形式を指します。これらの形式は、それぞれ、4 バイトと 8 バイトのストレージを占めます。左端のビットには符号が入り、次の 7 つのビットには指数が入り、残りの 3 または 7 バイトには仮数が入ります。

COMP-1 および COMP-2 データ項目は、z900 の 16 進形式で保管されます。

関連概念
619 ページの『付録 A. 中間結果と算術精度』

関連参照
VALUE 文節 (*Enterprise COBOL 言語解説書*)
353 ページの『TRUNC』

例: 数値データおよび内部表現

次の表は、数値項目の内部表現を示しています。

数値タイプ	PICTURE および USAGE 文節とオプションの SIGN 文節	値	内部表現
外部 10 進数	PIC S9999 DISPLAY	+ 1234 - 1234 1234	F1 F2 F3 C4 F1 F2 F3 D4 F1 F2 F3 C4
	PIC 9999 DISPLAY	1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+ 1234 - 1234	C1 F2 F3 F4 D1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING SEPARATE PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+ 1234 - 1234 + 1234 - 1234	4E F1 F2 F3 F4 60 F1 F2 F3 F4 F1 F2 F3 F4 4E F1 F2 F3 F4 60
2 進数	PIC S9999 BINARY COMP COMP-4	+ 1234 - 1234	04 D2 FB 2E
	COMP-5	+ 12345[1] - 12345[1]	30 39 CF C7
	PIC 9999 BINARY COMP COMP-4	1234	04 D2
	COMP-5	60000[1]	EA 60

数値タイプ	PICTURE および USAGE 文節とオプションの SIGN 文節	値	内部表現
内部 10 進数	PIC S9999 PACKED-DECIMAL COMP-3	+ 1234 - 1234	01 23 4C 01 23 4D
	PIC 9999 PACKED-DECIMAL COMP-3	+ 1234 - 1234	01 23 4F 01 23 4F
内部 浮動 小数点	COMP-1	+ 1234	43 4D 20 00
内部 浮動 小数点	COMP-2	+ 1234 - 1234	43 4D 20 00 00 00 00 00 C3 4D 20 00 00 00 00 00
外部 浮動 小数点	PIC +9(2).9(2)E+99 DISPLAY	+ 1234	4E F1 F2 4B F3 F4 C5 4E F0 F2
		- 1234	60 F1 F2 4B F3 F4 C5 4E F0 F2

1. この例では、COMP-5 データ項目に含めることのできる値が、PICTURE 文節の 9 の数によって暗黙指定された値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を入れることができることを示しています。

データ形式の変換

異なるデータ形式の項目との相互作用を含むプログラムをコーディングすると、コンパイラーがそれらの項目を次のように変換します。

- 一時的 (比較および算術演算の場合)。
- 永続的 (MOVE または COMPUTE ステートメント内の受け取り側への割り当ての場合)。

変換とは、実際には、値をあるデータ項目から別のデータ項目に移動することです。コンパイラーは、MOVE および COMPUTE ステートメントについて使用されるのと同じ規則を使用して、比較および算術の実行時に必要とされる変換を実行します。

可能であれば、コンパイラーは、直接的な 1 桁ずつの移動ではなく、数値を保持する移動を実行します。

変換には、通常、追加のストレージと処理時間が必要とされます。これは、演算が実行される前に、データが内部作業域に移動され、変換されるためです。さらに、結果を作業域に戻し、再度変換することが必要な場合もあります。

固定小数点データ形式 (外部 10 進数、パック 10 進数、または 2 進数) 間の変換は、ターゲット・フィールドがソース・オペランドのすべての桁を含むことができれば、精度が失われることなく完了します。

固定小数点データ形式と浮動小数点データ形式 (短精度浮動小数点、長精度浮動小数点、または外部浮動小数点) 間の変換では、精度が失われる可能性があります。このような変換は、固定小数点と浮動小数点の両方のオペランドが混在する算術計算時に起こります。

関連参照
『変換および精度』
52 ページの『符号表現および処理』

変換および精度

固定小数点項目と外部浮動小数点項目は、どちらも 10 進数の特性を持つため、以下の例での固定小数点項目への参照は、特に明記しない限り、外部浮動小数点項目への参照も含みます。

コンパイラーが固定小数点形式を内部浮動小数点形式に変換するときには、基数 10 の固定小数点数は、内部で使用される数体系に変換されます。

コンパイラーが比較のために短精度形式を長精度形式に変換するときには、短精度数値の埋め込みにはゼロが使用されます。

USAGE COMP-1 データ項目が 9 桁を超える固定小数点データ項目に移動されるときには、固定小数点データ項目は有効数字を 9 個だけ受け取り、残りの桁は 0 になります。

USAGE COMP-2 データ項目が 18 桁を超える固定小数点データ項目に移動されるときには、固定小数点データ項目は有効数字を 18 個だけ受け取り、残りの桁は 0 になります。

精度を保つ変換

6 桁以下の固定小数点データ項目が USAGE COMP-1 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値が復元されます。

USAGE COMP-1 データ項目が 9 桁以上の固定小数点データ項目に移動され、その後で USAGE COMP-1 データ項目に戻される場合は、元の値が復元されます。

15 桁以下の固定小数点データ項目が USAGE COMP-2 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値が復元されます。

USAGE COMP-2 データ項目が 18 桁以上の固定小数点 (外部浮動小数点ではなく) データ項目に移動され、その後で USAGE COMP-2 データ項目に戻される場合は、元の値が復元されます。

丸めを生じさせる変換

USAGE COMP-1 データ項目、USAGE COMP-2 データ項目、外部浮動小数点データ項目、または浮動小数点リテラルが固定小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

USAGE COMP-2 データ項目が USAGE COMP-1 データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

固定小数点データ項目の PICTURE に外部浮動小数点データ項目の PICTURE よりも多くの桁位置が含まれている場合に、固定小数点データ項目が外部浮動小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

関連概念
619 ページの『付録 A. 中間結果と算術精度』

符号表現および処理

符号表現は、数値データの処理および相互作用に影響を及ぼします。

X'*sd*' (*s* は符号表現であり、*d* は数字を表す) の場合、外部 10 進数 (SIGN IS SEPARATE 文節を指定しない USAGE DISPLAY) についての有効な符号表現は、以下のとおりです。

正: C、A、E、および F

負: D および B

COBOL NUMPROC コンパイラー・オプションは、外部 10 進数データおよび内部 10 進数データの符号処理に影響を与えます。NUMPROC は、2 進数データまたは浮動小数点データには影響を与えません。

NUMPROC(PFD)

X'*sd*' (*s* は符号表現であり、*d* は数字を表す) の場合に、NUMPROC(PFD) を使用すると、コンパイラーは、データ内の符号が 3 つの優先符号の 1 つであると想定します。

符号付き正数または 0:
X'C'

符号付き負数:
X'D'

符号なしまたは英数字:
X'F'

この想定に基づいて、コンパイラーは与えられるすべての符号を使用してデータを処理します。優先符号は、必要な場合にのみ生成されます (たとえば、符号なしデータが符号付きデータに移動される場合)。NUMPROC(PFD) オプションを使用すると、処理時間を節約することができますが、正しい処理を行うためには、データに優先符号を使用しなければなりません。

NUMPROC(NOPFD)

NUMPROC(NOPFD) コンパイラー・オプションが有効であると、コンパイラーはすべての有効な符号構成を受け入れます。受け取り側では、常に、優先符号が生成されます。NUMPROC(NOPFD) を使用すると、NUMPROC(PFD) より効率が低下しますが、優先符号を使用しないデータが存在する可能性があるときには NUMPROC(NOPFD) を使用しなければなりません。

符号なしの外部 10 進数の送り出し側が英数字の受け取り側に移動される場合には、符号は変更されないままです (NUMPROC(NOPFD) が指定された場合でも)。

NUMPROC(MIG)

NUMPROC(MIG) が有効であると、コンパイラーは、OS/VS COBOL によって作成されるコードに似たコードを生成します。このオプションは、特に OS/VS COBOL プログラムを IBM Enterprise COBOL for z/OS and OS/390 に移行する場合に有用です。

関連参照
336 ページの『NUMPROC』

非互換データの検査 (数値のクラス・テスト)

コンパイラーは、データ項目に指定された値がその項目の PICTURE および USAGE 文節に対して有効であると想定し、指定された値を妥当性検査を行わずに割り当てます。項目にそれのデータ記述と互換性のない値が与えられると、PROCEDURE DIVISION 内でのその項目への参照が未定義になり、結果が予測できなくなります。

値がプログラムに渡され、それらの値に対する互換性のないデータ記述を持つ項目に割り当てられることがよくあります。たとえば、非数値データが、数値として定義されたフィールドに移動されたり、渡されたりすることがあります。あるいは、符号付き数値が、符号なしとして定義されたフィールドに渡されることがあります。いずれの場合にも、受け取りフィールドには無効なデータが入れられます。データ項目を後続の処理ステップで使用する前に、そのデータ項目の内容がそれの PICTURE および USAGE 文節に適合していることを確認してください。

数値のクラス・テストを使用して、データ妥当性検査を行うことができます。たとえば、次のように記述します。

```
Linkage Section.
01  Count-x      Pic 999.
. . .
Procedure Division Using Count-x.
    If Count-x is numeric then display "Data is good"
```

数値のクラス・テストでは、データ項目の内容が、そのデータ項目の特定の PICTURE および USAGE にとって有効な値のセットと照合されます。たとえば、パック 10 進数項目は、数字位置に 16 進値 X'0' ～ X'9' があり、符号位置 (分離または非分離) に有効な符号値があるかどうか検査されます。

外部 10 進数、外部浮動小数点、およびパック 10 進数の場合、数値のクラス・テストは、NUMPROC コンパイラー・オプションおよび NUMCLS オプション (インストール時に設定) の影響を受けます。インストールの際に使用された NUMCLS 設定を確認するには、システム・プログラマーに問い合わせてください。

NUMCLS(PRIM) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

	NUMPROC(NOPFD)	NUMPROC(PFD)	NUMPROC(MIG)
符号付き	C、D、F	C、D、+0 (正のゼロ)	C、D、F
符号なし	F	F	F
分離符号	+、-	+、-、+0 (正のゼロ)	+、-

NUMCLS(ALT) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

	NUMPROC(NOPFD)	NUMPROC(PFD)	NUMPROC(MIG)
符号付き	A ～ F	C、D、+0 (正のゼロ)	A ～ F
符号なし	F	F	F
分離符号	+、-	+、-、+0 (正のゼロ)	+、-

関連参照
336 ページの『NUMPROC』

算術の実行

複数ある COBOL 言語機能のいずれを使用しても算術を実行することができます。

- COMPUTE、ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメント
- 算術式
- 組み込み関数
- 言語環境プログラム呼び出し可能サービス

COMPUTE およびその他の算術ステートメント

ほとんどの算術計算では、ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメントではなく、COMPUTE ステートメントが使用されます。個々の算術ステートメントを複数コーディングする代わりに、1 つの COMPUTE ステートメントでコーディングできる場合がしばしばあります。

COMPUTE ステートメントは、算術式の結果を 1 つまたは複数のデータ項目に割り当てます。

```
Compute z     = a + b / c ** d - e
Compute x y z = a + b / c ** d - e
```

COMPUTE 以外の算術ステートメントを使用した方が、より直観的にわかりやすくなる算術計算もあります。たとえば、次のように記述します。

COMPUTE	等価の算術ステートメント
`Compute Increment =` `  Increment + 1`	`Add 1 to Increment`
`Compute Balance =` `  Balance - Overdraft`	`Subtract Overdraft` `  from Balance`

`Compute IncrementOne =`
`  IncrementOne + 1`
`Compute IncrementTwo =`
`  IncrementTwo + 1`
`Compute IncrementThree =`
`  IncrementThree + 1`
	`Add 1 to IncrementOne,`
	`  IncrementTwo,`
	`  IncrementThree`

さらに、剰余を処理したい除算については、DIVIDE ステートメント (REMAINDER 句を指定した) を使用したい場合もあります。REM 組み込み関数も、剰余を処理する機能を提供します。

算術式

数値データ項目が許可されているステートメント内の多くの場所で、算術式を使用できます (ただし、どの場所でも使用できるわけではありません)。たとえば、算術式を比較条件の被比較数として使用することができます。

```
If (a + b) > (c - d + 5) Then. . .
```

算術式は、単一の数字リテラル、単一の数値データ項目、または単一の組み込み関数参照で構成することができます。また、これらの項目のいくつかを算術演算子で結合して構成することもできます。 算術演算子は、次の優先順位に従って計算されます。

演算子	意味	評価の順序
単項 + または -	代数符号	1 番目
**	べき乗計算	2 番目
/ または *	除算または乗算	3 番目
2 項 + または -	加算または減算	最後

優先順位が同じレベルの演算子は、左から右へと評価されます。ただし、演算子とともに括弧を使用して、それらが評価される順序を変更することができます。括弧の中の式は、個々の演算子が評価される前に評価されます。必要であるかどうかにかかわらず、括弧を使用するとプログラムが読みやすくなります。

数字組み込み関数

数字組み込み関数は、数式を使用できる場所でのみ使用することができます。これらの関数を使用すると、時間の節約に役立ちます。これは、これらの関数が取り扱う多種類の一般的な計算をコーディングする必要がなくなるためです。

数字組み込み関数は、符号付き数値を戻します。これらは一時数値データ項目として扱われます。

数字組み込み関数の機能の多くは、言語環境プログラム呼び出し可能サービスによっても提供されます。

数字関数は、以下のカテゴリーに分類されます。

整数　整数を戻すもの。

浮動小数点
長精度 (64 ビット) または拡張精度 (128 ビット) の浮動小数点値を戻すもの (これは、デフォルト・オプション ARITH(COMPAT) を使用してコンパイルするか、ARITH(EXTEND) を使用してコンパイルするかによって決まります)。

混合　引き数によって、整数、浮動小数点値、または小数部の桁がある固定小数点数を戻すもの。

組み込み関数を使用すると、次の表に概説されているようなさまざまな種類の算術演算を実行することができます。

数値処理	日付および時刻	金融	数学	統計
LENGTH	CURRENT-DATE	ANNUITY	ACOS	MEAN
MAX	DATE-OF-INTEGER	PRESENT-VALUE	ASIN	MEDIAN
MIN	DATE-TO-YYYYMMDD		ATAN	MIDRANGE
NUMVAL	DATEVAL		COS	RANDOM
NUMVAL-C	DAY-OF-INTEGER		FACTORIAL	RANGE
ORD-MAX	DAY-TO-YYYYDDD		INTEGER	STANDARD-DEVIATION
ORD-MIN	INTEGER-OF-DATE		INTEGER-PART	VARIANCE
	INTEGER-OF-DAY		LOG	
	UNDATE		LOG10	
	WHEN-COMPILED		MOD	
	YEAR-TO-YYYY		REM	
	YEARWINDOW		SIN	
			SQRT	
			SUM	
			TAN	

58 ページの『例: 数字組み込み関数』

関数および算術式のネスト

ある関数を別の関数の引き数として参照することができます。ネストされた関数は、外側の関数からは独立して評価されます。ただし、混合関数を固定小数点命令と浮動小数点命令のどちらを使って評価すべきかを判別するときは例外です。

算術式を数字関数への引き数としてネストすることもできます。

```
Compute x = Function Sum(a b (c / d))
```

この例では、関数の引き数は a、b、および 算術式 (c / d) の 3 つだけです。

ALL 添え字付けおよび特殊レジスター

ALL 添え字を使用すると、あるテーブル (または配列) のすべてのエレメントを関数の引き数として参照することができます。

整数タイプの特殊レジスターは、整数の引き数が使用できるところであればどこでも引き数として使用することができます。

数学および日付の言語環境プログラム・サービス

COBOL 組み込み関数の機能の多くは、言語環境プログラム呼び出し可能サービスによっても提供されます。言語環境プログラム呼び出し可能サービスは、算術結果をデータ項目に割り当てる手段を提供します。これらのサービスには、数学関数、日付および時刻操作も含まれます。

数学用の呼び出し可能サービス

ほとんどの COBOL 組み込み関数には、次の表に示すように、対応する数学用の呼び出し可能サービスがあり、それらを使用して同じ結果を作成することができます。デフォルト・オプション ARITH(COMPAT) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数は長精度 (64 ビット) の結果を戻します。オプシ

ョン ARITH(EXTEND) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数 (RANDOM は例外) は拡張精度 (128 ビット) の結果を戻します。

たとえば (表の最初の行を例にとると)、ARITH(COMPAT) を使用してコンパイルした場合は、CEESDACS が ACOS と同じ結果を戻します。ARITH(EXTEND) を使用してコンパイルした場合は、CEESQACS が ACOS と同じ結果を戻します。

COBOL 組み込み関数	対応する言語環境プログラム呼び出し可能サービス		組み込み関数と呼び出し可能サービスで結果が同じか
	長精度	拡張精度	
ACOS	CEESDACS	CEESQACS	はい
ASIN	CEESDASN	CEESQASN	はい
ATAN	CEESDATN	CEESQATN	はい
COS	CEESDCOS	CEESQCOS	はい
LOG	CEESDLOG	CEESQLOG	はい
LOG10	CEESDLG1	CEESQLG1	はい
RANDOM[1]	CEERAN0	なし	いいえ
REM	CEESDMOD	CEESQMOD	はい
SIN	CEESDSIN	CEESQSIN	はい
SQRT	CEESDSQT	CEESQSQT	はい
TAN	CEESDTAN	CEESQTAN	はい

1. RANDOM は、オプション ARITH(EXTEND) を使用してコンパイルし、31 ビットの引き数を渡した場合でも、長精度 (64 ビット) 浮動小数点結果を戻します。

RANDOM 組み込み関数と CEERAN0 サービスはともに、0 ～ 1 の範囲の乱数を生成します。ただし、それぞれが独自のアルゴリズムを使用するため、RANDOM と CEERAN0 は同じ数から異なる乱数を生成します。

同じ結果をもたらす関数であっても、組み込み関数と言語環境プログラム呼び出し可能サービスの使用法は異なります。組み込み関数の引き数に要求されるデータ型に関する規則の方が、制限が緩くなっています。数字組み込み関数の場合には、任意の数値データ型の引き数を使用することができます。しかし、CALL ステートメントを使用して言語環境プログラム呼び出し可能サービスを呼び出す場合は、パラメーターをそのサービスで必要な数値データ型 (通常は COMP-1 または COMP-2) と一致させなければなりません。

組み込み関数と言語環境プログラム呼び出し可能サービスのエラー処理は異なる場合があります。言語環境プログラムの数学サービスを呼び出すときに明示的なフィードバック・トークンを渡す場合は、それぞれの呼び出しの後でそのフィードバック・コードを検査し、エラーを処理するための明示的なアクションを取らなければなりません。しかし、フィードバック・トークンを使用して明示的に OMITTED を呼び出した場合は、トークンを検査する必要はありません。エラーがあれば、言語環境プログラムが自動的にエラーを通知します。

日付呼び出し可能サービス

COBOL の日付組み込み関数と言語環境プログラムの日付呼び出し可能サービスは両方とも、グレゴリオ暦を基本とします。ただし、開始日付は、INTDATE コンパイラー・オプションの設定値によって異なることがあります。デフォルト設定である

INTDATE(ANSI) が有効な場合、COBOL は第 1 日として 1601 年 1 月 1 日を使用します。INTDATE(LILIAN) が有効な場合、COBOL は第 1 日として 1582 年 10 月 15 日を使用します。言語環境プログラムは常に、第 1 日として 1582 年 10 月 15 日を使用します。

これは、INTDATE(LILIAN) を使用すると、COBOL 組み込み関数と言語環境プログラムの呼び出し可能日付サービスから同じ結果が得られることを意味します。

次の表は、INTDATE(ANSI) が有効な場合の結果を示しています。

COBOL 組み込み関数	言語環境プログラム呼び出し可能サービス	結果
INTEGER-OF-DATE	CEECBLDY	互換性あり
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYMMDD 付き)	非互換
DAY-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYDDD 付き)	非互換
INTEGER-OF-DATE	CEEDAYS	非互換

次の表は、INTDATE(LILIAN) が有効な場合の結果を示しています。

COBOL 組み込み関数	言語環境プログラム呼び出し可能サービス	結果
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYMMDD 付き)	互換性あり
DAY-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYDDD 付き)	互換性あり
INTEGER-OF-DATE	CEEDAYS	互換性あり
INTEGER-OF-DATE	CEECBLDY	非互換

関連概念
60 ページの『固定小数点演算と浮動小数点演算の対比』
619 ページの『付録 A. 中間結果と算術精度』

関連タスク
611 ページの『言語環境プログラム呼び出し可能サービスの使用』

関連参照
316 ページの『ARITH』

例: 数字組み込み関数

以下の例と付随する説明では、それぞれのカテゴリーごとに組み込み関数を示します。

一般数値処理

2 つの価格 (ドル記号の付いた英数字項目として表される) の最大値を調べ、この値を出力レコードの数値フィールドに入れ、出力レコードの長さを判別したいとします。これを行うには、NUMVAL-C (英数字ストリングの数値を戻す関数)、MAX、および LENGTH 関数を使用することができます。

```
01  X                   Pic 9(2).
01  Price1              Pic x(8)   Value "$8000".
01  Price2              Pic x(8)   Value "$2000".
01  Output-Record.
    05  Product-Name    Pic x(20).
    05  Product-Number  Pic 9(9).
    05  Product-Price   Pic 9(6).
. . .
Procedure Division.
    Compute Product-Price =
      Function Max (Function Numval-C(Price1) Function Numval-C(Price2))
    Compute X = Function Length(Output-Record)
```

さらに、Product-Name の内容が大文字になるようにするために、次のステートメントを使用することができます。

```
Move Function Upper-case (Product-Name) to Product-Name
```

日付および時刻

次の例は、今日から 90 日後の満期日を計算する方法を示しています。
CURRENT-DATE 関数から戻される最初の 8 文字は、日付を 4 桁の年、2 桁の月、および 2 桁の日という形式 (YYYYMMDD) で表します。この日付がその整数値に変換されます。そのあと、この値に 90 が追加され、整数が YYYYMMDD 形式に再度変換されます。

```
01  YYYYMMDD         Pic 9(8).
01  Integer-Form     Pic S9(9).
. . .
    Move Function Current-Date(1:8) to YYYYMMDD
    Compute Integer-Form = Function Integer-of-Date(YYYYMMDD)
    Add 90 to Integer-Form
    Compute YYYYMMDD = Function Date-of-Integer(Integer-Form)
    Display 'Due Date: ' YYYYMMDD
```

金融

ビジネス投資の判断では、計画された投資の利益率を評価するために、予期される将来の現金流入の現在価格を計算することがしばしば必要になります。将来の特定の時期に受け取ることが期待される金額の現在価格は、今日特定の利率で投資された場合に、累積されてその将来の金額になるであろう金額です。

たとえば、計画された $1,000 の投資で、次の 3 年間にわたり、それぞれ年 1 回の支払いで $100、$200、および $300 の支払いの流れになるとします。次の COBOL ステートメントは、10% の利率でこれらの現金流入の現在の値を計算する方法を示しています。

```
01  Series-Amt1      Pic 9(9)V99       Value 100.
01  Series-Amt2      Pic 9(9)V99       Value 200.
01  Series-Amt3      Pic 9(9)V99       Value 300.
01  Discount-Rate    Pic S9(2)V9(6)    Value .10.
01  Todays-Value     Pic 9(9)V99.
. . .
```

```
    Compute Todays-Value =
      Function
        Present-Value(Discount-Rate Series-Amt1 Series-Amt2 Series-Amt3)
```

ANNUITY 関数は、ローンの元金および利息を返済するために必要な分割払いの支払金 (年賦金) の金額を判断することが必要とされるビジネス問題で使用できます。一連の支払いの特徴は、各期間の長さと、期間ごとの金額および利率が一定であるということです。次の例は、$15,000 のローンを 12% の年利で 3 年間で返済するのに必要な月賦金額を計算する方法を示しています (36 か月払い、1 か月当たりの利率 = .12/12)。

```
01  Loan             Pic 9(9)V99.
01  Payment          Pic 9(9)V99.
01  Interest         Pic 9(9)V99.
01  Number-Periods   Pic 99.
. . .
    Compute Loan = 15000
    Compute Interest = .12
    Compute Number-Periods = 36
    Compute Payment =
      Loan * Function Annuity((Interest / 12) Number-Periods)
```

数学

次の COBOL ステートメントでは、組み込み関数をネストし、算術式を引き数として使用し、前の複雑な計算を簡単に行う方法を示しています。

```
Compute Z = Function Log(Function Sqrt (2 * X + 1)) + Function Rem(X 2)
```

ここでは、組み込み関数 REM (REMAINDER 文節を指定した DIVIDE ステートメントではなく) が、X を 2 で割った剰余を加数に戻します。

統計

組み込み関数は、統計情報を計算しやすくします。さまざまな市民税を分析していて、平均値、中央値、および範囲 (最高税額と最低税額の差) を計算したいとします。

```
01  Tax-S            Pic 99v999 value .045.
01  Tax-T            Pic 99v999 value .02.
01  Tax-W            Pic 99v999 value .035.
01  Tax-B            Pic 99v999 value .03.
01  Ave-Tax          Pic 99v999.
01  Median-Tax       Pic 99v999.
01  Tax-Range        Pic 99v999.
. . .
    Compute Ave-Tax    = Function Mean   (Tax-S Tax-T Tax-W Tax-B)
    Compute Median-Tax = Function Median (Tax-S Tax-T Tax-W Tax-B)
    Compute Tax-Range  = Function Range  (Tax-S Tax-T Tax-W Tax-B)
```

固定小数点演算と浮動小数点演算の対比

プログラム内の多くのステートメントには、算術計算が伴うことがあります。たとえば、以下のそれぞれの COBOL ステートメントには、ある種の算術計算が必要です。

- 一般算術計算

```
compute report-matrix-col = (emp-count ** .5) + 1
add report-matrix-min to report-matrix-max giving report-matrix-tot
```

- 式および関数

```
compute report-matrix-col = function sqrt(emp-count) + 1
compute whole-hours       = function integer-part((average-hours) + 1)
```

- 算術比較

```
if report-matrix-col <     function sqrt(emp-count) + 1
if whole-hours not = function integer-part((average-hours) + 1)
```

プログラム内のそれぞれの算術計算 (それが算術ステートメント、組み込み関数、式、またはこれらの相互にネストされた組み合わせのいずれであっても) では、算術計算をコーディングする方法によって、それが浮動小数点演算になるか固定小数点演算になるかが決まります。

浮動小数点計算

通常、算術計算に以下のいずれかの特性がある場合、それは浮動小数点演算で計算されます。

- オペランドまたは結果フィールドが浮動小数点である。

  オペランドは、浮動小数点リテラルとしてコーディングするか、あるいは USAGE COMP-1、USAGE COMP-2、または外部浮動小数点 (浮動小数点 PICTURE を指定した USAGE DISPLAY) として定義されたデータ項目としてコーディングした場合に浮動小数点になります。

  ネストされた算術式であるか、または数字組み込み関数への参照であるオペランドは、次の場合に浮動小数点演算になります。

  – 算術式内の引き数が浮動小数点になる。
  – 関数が浮動小数点関数である。
  – 関数が 1 つまたは複数の浮動小数点引き数を持つ混合関数である。

- 指数に小数部の桁が含まれている。

  これは、小数部の桁が含まれるリテラルを使用するか、小数部の桁が含まれる PICTURE を項目に与えるか、あるいは結果が小数部の桁を持つ算術式または関数を使用する場合に当てはまります。

算術式または数字関数は、オペランドまたは引き数 (除数および指数を除く) に小数部の桁がある場合に、小数部の桁がある結果をもたらします。

固定小数点計算

通常、算術演算に上記の浮動小数点についての特性がない場合、コンパイラーはそれを固定小数点演算で計算します。すなわち、算術計算が固定小数点として処理されるのは、すべてのオペランドが固定小数点で、結果フィールドが固定小数点と定義されており、しかもどの指数も小数部の桁がある値を表さない場合だけです。また、ネストされた算術式および関数参照も、固定小数点値を表さなければなりません。

算術比較 (比較条件)

関係演算子を使用して数式を比較する場合、数式 (それらがデータ項目、算術式、関数参照、またはこれらの組み合わせのいずれであっても) は、計算全体のコンテキストでは、被比較数です。すなわち、それぞれの属性が互いの計算に影響を与える可能性があり、両方の式が固定小数点で計算されるか、または両方の式が浮動小

数点で計算されることになります。これは、簡略比較にも当てはまります (比較の中で一方の被比較数が明示的に指定されない場合でも)。たとえば、次のように記述します。

```
if (a + d) = (b + e) and c
```

このステートメントには、(a + d) = (b + e) と (a + d) = c の 2 つの比較があります。(a + d) は、2 番目の比較では明示的に指定されていませんが、その比較の被比較数です。したがって、c の属性が (a + d) の計算に影響を与える可能性があります。

比較演算 (および比較の中にネストされた算術式の評価) は、一方の被比較数が浮動小数点値であるかまたは結果が浮動小数点値になる場合、コンパイラーによって浮動小数点演算として処理されます。

比較演算 (および比較の中にネストされた算術式の評価) は、両方の被比較数が固定小数点値であるかまたは結果が固定小数点値になる場合、コンパイラーによって固定小数点演算として処理されます。

暗黙の比較 (関係演算子が使用されない) は、単位として扱われません。しかし、2 つの被比較数は、浮動小数点演算または固定小数点演算における評価に関しては別々に扱われます。以下の例では、実際には、それぞれの属性に関係なく評価され、その後で相互に比較される 5 つの算術式があります。

```
evaluate (a + d)
    when (b + e) thru c
    when (f / g) thru (h * i)
    . . .
end-evaluate
```

『例: 固定小数点計算および浮動小数点計算』

関連参照

628 ページの『非算術ステートメントでの算術式』

例: 固定小数点計算および浮動小数点計算

従業員表のデータ項目を次のように定義すると仮定します。

```
01  employee-table.
    05  emp-count         pic  9(4).
    05  employee-record occurs 1 to 1000 times
            depending on emp-count.
        10 hours           pic +9(5)e+99.
. . .
01  report-matrix-col     pic  9(3).
01  report-matrix-min     pic  9(3).
01  report-matrix-max     pic  9(3).
01  report-matrix-tot     pic  9(3).
01  average-hours         pic  9(3)v9.
01  whole-hours           pic  9(4).
```

以下のステートメントは、浮動小数点演算を使用して計算されます。

```
compute report-matrix-col = (emp-count ** .5) + 1
compute report-matrix-col = function sqrt(emp-count) + 1
if report-matrix-tot < function sqrt(emp-count) + 1
```

以下のステートメントは、固定小数点演算を使用して計算されます。

```
add report-matrix-min to report-matrix-max giving report-matrix-tot
compute report-matrix-max =
    function max(report-matrix-max report-matrix-tot)
if whole-hours not = function integer-part((average-hours) + 1)
```

通貨記号の使用

多くのプログラムでは金融情報を処理する必要があり、それらの情報を適切な通貨記号で出力表示する必要があります。COBOL 通貨サポート (および使用しているプリンターまたはディスプレイ装置の適切なコード・ページ) により、以下の 1 つまたは複数の符号をプログラムで使用することができます。

- ドル記号 ($) のような記号
- 複数文字からなる通貨記号 (USD、DEM、EUR など)
- 欧州経済通貨同盟 (EMU) によって確立されているユーロ記号

金融情報を表示するための記号を指定するには、それらの記号に関連付けられる PICTURE 文字を指定した CURRENCY SIGN 文節を (CONFIGURATION SECTION の SPECIAL-NAMES 段落の中で) 使用してください。次の例で、PICTURE 文字 $ は、通貨記号として $US を使用することを示しています。

```
    Currency Sign is "$US" with Picture Symbol "$".
    . . .
77  Invoice-Amount      Pic $$,$$9.99.
. . .
    Display "Invoice amount is " Invoice-Amount.
```

この例で、Invoice-Amount に 1500.00 が含まれている場合は、次のように出力されます。

```
Invoice amount is  $US1,500.00
```

プログラム内で複数の CURRENCY SIGN 文節を使用することにより、複数の通貨記号を表示することができます。

16 進リテラルを使用して通貨記号の値を表すことができます。この方法は、ソース・プログラムのデータ入力において特定の文字を簡単には入力できない場合に便利です。次の例は、16 進値 X'9F' を通貨記号として使用する方法を示しています。

```
    Currency Sign X'9F' with Picture Symbol 'U'.
    . . .
01  Deposit-Amount       Pic UUUUU9.99.
```

キーボード上にユーロ記号に相当する文字がない場合は、それを CURRENCY SIGN 文節で 16 進値として指定する必要があります。ユーロ記号の 16 進値は、次の表に示すように、使用されているコード・ページに応じて、X'9F' または X'5A' のいずれかです。

コード・ページ	適用される国	変更元	ユーロ記号
IBM-1140	米国、カナダ、オランダ、ポルトガル、オーストラリア、ニュージーランド	IBM-037	X'9F'
IBM-1141	オーストリア、ドイツ	IBM-273	X'9F'
IBM-1142	デンマーク、ノルウェー	IBM-277	X'5A'
IBM-1143	フィンランド、スウェーデン	IBM-278	X'5A'

コード・ページ	適用される国	変更元	ユーロ記号
IBM-1144	イタリア	IBM-280	X'9F'
IBM-1145	スペイン、ラテンアメリカ - スペイン語圏	IBM-284	X'9F'
IBM-1146	英国	IBM-285	X'9F'
IBM-1147	フランス	IBM-297	X'9F'
IBM-1148	ベルギー、カナダ、スイス	IBM-500	X'9F'
IBM-1149	アイスランド	IBM-871	X'9F'

『例: 複数の通貨記号』

例: 複数の通貨記号

次の例は、ユーロ通貨 (EUR) とフランスのフラン (FRF) の両方で値を表示する方法を示すものです。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EuroExample.
Environment Division.
Configuration Section.
Special-Names.
    Currency Sign is "FRF "  with Picture Symbol "F"
    Currency Sign is "EUR "  with Picture Symbol "U".
Data Division.
Working-Storage Section.
01  Deposit-in-Euro        Pic S9999V99 Value 8000.00.
01  Deposit-in-FRF         Pic S99999V99.
01  Deposit-Report.
    02  Report-in-Franc    Pic -FFFFF9.99.
    02  Report-in-Euro     Pic -UUUUU9.99.
. . .
01  EUR-to-FRF-Conv-Rate   Pic 9V99999  Value 6.78901.
. . .
PROCEDURE DIVISION.
Report-Deposit-in-FRF-and-EUR.
    Move Deposit-in-Euro to Report-in-Euro
    . . .
    Compute Deposit-in-FRF Rounded
          = Deposit-in-Euro * EUR-to-FRF-Conv-Rate
      On Size Error
        Perform Conversion-Error
      Not On Size Error
        Move Deposit-in-FRF to Report-in-Franc
        Display "Deposit in Euro  = " Report-in-Euro
        Display "Deposit in Franc = " Report-in-Franc
    End-Compute
    . . .
    Goback.
Conversion-Error.
           Display "Conversion error from EUR to FRF"
           Display "Euro value: " Report-in-Euro.
```

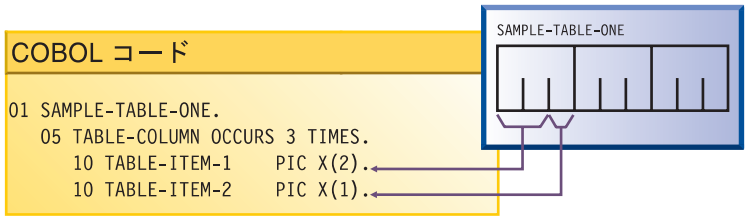
上記の例は、次のような表示出力を作成します。

```
Deposit in Euro  =  EUR 8000.00
Deposit in Franc = FRF 54312.08
```

この例で使用されている交換レートは例として使われているだけです。

第 4 章 テーブルの処理

テーブル は、合計額や月平均額のような同じデータ記述を持つデータ項目の集合です。テーブルは、エレメントからなる配列に相当する COBOL 用語です。テーブル名と、テーブル・エレメント と呼ばれる従属項目から構成されます。



上記の例では、SAMPLE-TABLE-ONE が、テーブルを含むグループ項目です。TABLE-COLUMN は、3 回出現する 1 次元テーブルのテーブル・エレメントを指しています。

反復項目を別個の連続する項目として DATA DIVISION に定義するのではなく、DATA DIVISION 記入項目の OCCURS 文節を使用してテーブルを定義することができます。この方法には、次のような利点があります。

- コードは、項目 (テーブル・エレメント) の単位を明確に示します。
- 添え字と指標を使用してテーブル・エレメントを参照することができます。
- データ項目の反復が容易です。

テーブルは、プログラムの処理速度、特にレコードを探索する速度を高めるうえで大切なものです。

関連タスク

『テーブルの定義 (OCCURS)』
68 ページの『テーブルでの項目の参照』
70 ページの『テーブルに値を入れる方法』
66 ページの『テーブルのネスト』
74 ページの『可変長テーブルの作成 (DEPENDING ON)』
77 ページの『テーブルの探索』
80 ページの『組み込み関数を使用したテーブル項目の処理』
595 ページの『テーブルの効率的な処理』

テーブルの定義 (OCCURS)

テーブルをコーディングするには、テーブルにグループ名を与え、*n* 回繰り返される従属項目 (テーブル・エレメント) を定義します。次の例では、table-name がグループ名です。

```
01  table-name.
    05 element-name OCCURS n TIMES.
    . . . (subordinate items of the table element might follow)
```

© Copyright IBM Corp. 1991, 2001

テーブル・エレメント定義 (OCCURS 文節が組み込まれている) は、テーブルを含むグループ項目に従属しています。OCCURS 文節をレベル 01 記述で指定することはできません。

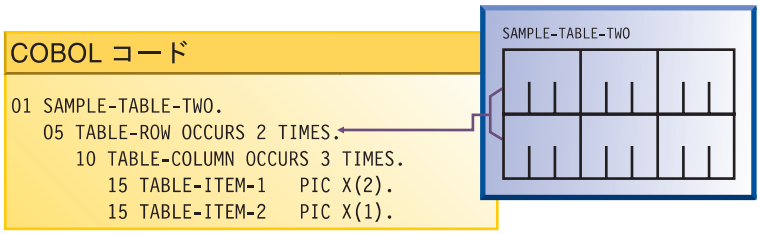
2 次元から 7 次元までのテーブルを作成するには、ネストされた OCCURS 文節を使用してください。

関連タスク
74 ページの『可変長テーブルの作成 (DEPENDING ON)』
『テーブルのネスト』
70 ページの『テーブルに値を入れる方法』
68 ページの『テーブルでの項目の参照』
77 ページの『テーブルの探索』

関連参照
OCCURS 文節 (*Enterprise COBOL 言語解説書*)

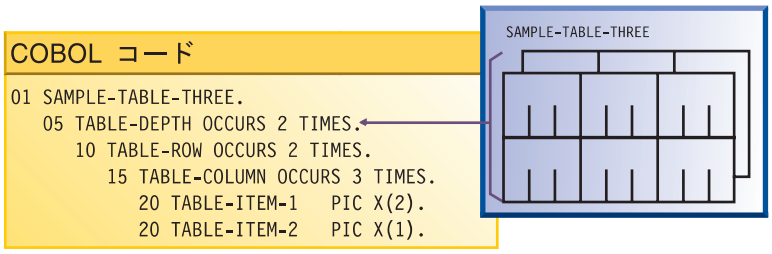
テーブルのネスト

2 次元テーブルを作成するには、ある 1 次元テーブルのそれぞれのオカレンス (出現) の中で別の 1 次元テーブルを定義します。たとえば、次のように記述します。



SAMPLE-TABLE-TWO の TABLE-ROW は 1 次元テーブルのエレメントで、2 回出現します。TABLE-COLUMN は、2 次元テーブルのエレメントで、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

3 次元テーブルを作成するには、ある 1 次元テーブル (それ自体が別の 1 次元テーブルのそれぞれのオカレンスに含まれている) のそれぞれのオカレンスの中で別の 1 次元テーブルを定義します。たとえば、次のようになります。



SAMPLE-TABLE-THREE の TABLE-DEPTH は 1 次元テーブルのエレメントで、2 回出現します。TABLE-ROW は 2 次元テーブルのエレメントで、TABLE-DEPTH のそれぞれの

オカレンスで 2 回出現します。TABLE-COLUMN は 3 次元テーブルのエレメントで、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

添え字付け

2 次元テーブルでは、2 つの添え字が行番号と列番号に対応します。3 次元テーブルでは、3 つの添え字が深さ番号、列番号、および行番号に対応します。

以下の SAMPLE-TABLE-THREE への有効参照では、リテラル添え字を使用しています。2 番目の例では、スペースは必須です。

```
TABLE-COLUMN (2, 2, 1)
TABLE-COLUMN (2 2 1)
```

いずれのテーブル参照でも、最初の値 (2) は TABLE-DEPTH 内の 2 番目のオカレンスを参照し、2 つ目の値 (2) は TABLE-ROW 内の 2 番目のオカレンスを参照し、3 つ目の値 (1) は TABLE-COLUMN 内の 1 番目のオカレンスを参照します。

次の SAMPLE-TABLE-TWO への参照では、変数添え字が使用されています。SUB1 と SUB2 が、テーブルの範囲内の正の整数値を含むデータ名であれば、この参照は有効になります。

```
TABLE-COLUMN (SUB1 SUB2)
```

指標付け

次の 3 次元テーブル SAMPLE-TABLE-FOUR を考えてみてください。

```
01  SAMPLE-TABLE-FOUR
    05 TABLE-DEPTH OCCURS 3 TIMES INDEXED BY INX-A.
       10 TABLE-ROW OCCURS 4 TIMES INDEXED BY INX-B.
          15 TABLE-COLUMN OCCURS 8 TIMES INDEXED BY INX-C  PIC X(8).
```

SAMPLE-TABLE-FOUR に対して次の相対指標付け参照をコーディングするとします。

```
TABLE-COLUMN (INX-A + 1, INX-B + 2, INX-C - 1)
```

この参照によって、TABLE-COLUMN への変位が次のように計算されます。

```
  (contents of INX-A) + (256 * 1)
+ (contents of INX-B) + (64 * 2)
+ (contents of INX-C) - (8 * 1)
```

この計算は、次のエレメント長に基づいています。

- TABLE-DEPTH のそれぞれのオカレンスは 256 バイトの長さです (4 * 8 * 8)。
- TABLE-ROW のそれぞれのオカレンスは 64 バイトの長さです (8 * 8)。
- TABLE-COLUMN のそれぞれのオカレンスは 8 バイトの長さです。

関連タスク

65 ページの『テーブルの定義 (OCCURS)』
68 ページの『テーブルでの項目の参照』
70 ページの『テーブルに値を入れる方法』
74 ページの『可変長テーブルの作成 (DEPENDING ON)』
77 ページの『テーブルの探索』
80 ページの『組み込み関数を使用したテーブル項目の処理』
595 ページの『テーブルの効率的な処理』

関連参照
OCCURS 文節 (*Enterprise COBOL 言語解説書*)

テーブルでの項目の参照

テーブル・エレメントは集合名を持ちますが、その中の個々の項目は固有のデータ名を持っていません。

項目を参照するには、次の 3 つの方法のいずれかを使用できます。

- テーブル・エレメントのデータ名と一緒に、そのオカレンス番号 (*添え字* と呼ばれる) を括弧で囲んで使用する。この手法は、添え字付けと呼ばれます。
- テーブル・エレメントのデータ名と一緒に、項目を位置指定するために (テーブルの先頭からの変位) テーブルのアドレスに追加される値 (*指標* と呼ばれる) を使用する。この手法は、指標付け、または指標名を使用する添え字付けと呼ばれます。
- 添え字と指標の両方を使用する。

関連タスク
69 ページの『指標付け』
『添え字付け』

添え字付け

可能な最小の添え字値は 1 であり、これはテーブル・エレメントの最初のオカレンスを指します。1 次元テーブルでは、添え字は行番号に対応します。

添え字にはデータ名またはリテラルを使用することができます。

リテラルの添え字を持つデータ項目が固定長である場合は、コンパイラーがそのデータ項目の位置を解決します。

変数の添え字であるデータ名を使用する場合は、そのデータ名を基本数値整数として記述しなければなりません。最も効率的な形式は、PICTURE サイズが 5 桁よりも少ない COMPUTATIONAL (COMP) です。 添え字として使用されるデータ名に添え字を付けることはできません。

アプリケーションのために生成されるコードが、実行時に変数添え字の位置を解決します。

リテラルまたは変数の添え字を、指定した整数ぶんだけ増分または減分することができます。たとえば、次のように記述します。

```
TABLE-COLUMN (SUB1 - 1, SUB2 + 3)
```

テーブル・エレメント全体ではなく、その一部を変更することができます。単に、添え字付きエレメント内で変更するサブストリングの文字位置と長さを参照すればよいだけです。たとえば、次のように記述します。

```
01  ANY-TABLE.
    05 TABLE-ELEMENT         PIC X(10)
        OCCURS 3 TIMES
        VALUE "ABCDEFGHIJ".
. . .
    MOVE "??" TO TABLE-ELEMENT (1) (3 : 2).
```

この MOVE ステートメントは、値 ?? を、文字位置 3 から始めて 2 の長さだけ、テーブル・エレメント 1 に移動します。

関連タスク
『指標付け』
70 ページの『テーブルに値を入れる方法』
77 ページの『テーブルの探索』
595 ページの『テーブルの効率的な処理』

指標付け

指標は、個々のテーブルと一緒に作成する (OCCURS INDEXED BY を使用) ことも、または別個に作成する (USAGE IS INDEX を使用) こともできます。

たとえば、次のように記述します。

```
05 TABLE-ITEM PIC X(8)
     OCCURS 10 INDEXED BY INX-A.
```

コンパイラーは、指標に含まれる値を、オカレンス番号 (添え字) から 1 引いた値にテーブル・エレメントの長さを掛けた値として計算します。したがって、TABLE-ITEM の 5 回目のオカレンスの場合、INX-A に含まれる値は、(5 - 1) * 8、すなわち 32 です。

この指標を使用して別のテーブルに指標を付けることができるのは、両方のテーブル記述が同じ数のテーブル・エレメントを持ち、テーブル・エレメントが同じ長さである場合だけです。

USAGE IS INDEX を使用して指標を作成する場合は、どのテーブルでも指標データ項目を使用することができます。たとえば、次のように記述します。

```
77  INX-B  USAGE IS INDEX.
. . .
    PERFORM VARYING INX-B FROM 1 BY 1 UNTIL INX-B > 10
        DISPLAY TABLE-ITEM (INX-B)
        . . .
    END-PERFORM.
```

INX-B は、上記のテーブル TABLE-ITEM を走査するために使用されるものですが、他のテーブルの走査にも使用することができます。

符号なし数値リテラルによって指標名を増分または減分することができます。このリテラルはオカレンス番号と見なされ、指標名に対して加算または減算される前に、指標値に変換されます。

指標名は、SET、PERFORM VARYING、または SEARCH ALL ステートメントを使用して初期化してください。そのあと、指標名を SEARCH ステートメントまたは関係条件

ステートメントでも使用できるようになります。値を変更するには、PERFORM、SEARCH、または SET ステートメントを使用してください。

SET ステートメントを使用して、USAGE IS INDEX によって定義した指標データ項目に保管した値を指標に割り当てます。たとえば、可変長テーブルにレコードをロードする場合は、読み取った最後のレコードの指標値を USAGE IS INDEX として定義したデータ項目に保管することができます。そのあと、現行の指標値を最後のレコードの指標値と比較することによって、テーブルの終わりをテストすることができます。この手法は、テーブルを初めから終わりまで検索したり、テーブルを処理したりする場合に有用です。

物理的な変位を比較することになるため、指標データ項目は、SEARCH および SET ステートメントの中でのみ、あるいは指標または他の指標データ項目との比較においてのみ使用することができます。指標データ項目を添え字または指標として使用することはできません。

関連タスク
68 ページの『添え字付け』
『テーブルに値を入れる方法』
77 ページの『テーブルの探索』
80 ページの『組み込み関数を使用したテーブル項目の処理』
595 ページの『テーブルの効率的な処理』

関連参照
INDEXED BY 句 (*Enterprise COBOL 言語解説書*)
INDEX 句 (*Enterprise COBOL 言語解説書*)

テーブルに値を入れる方法

テーブルに値を入れるには、次のいずれかの方法を使用してください。

- テーブルを動的にロードする。
- テーブルを初期化する (INITIALIZE ステートメント)。
- テーブルを定義するときに値を割り当てる (VALUE 文節)。

関連タスク
『テーブルの動的なロード』
76 ページの『可変長テーブルのロード』
71 ページの『テーブルの初期化 (INITIALIZE)』
71 ページの『テーブルの定義時の値の割り当て (VALUE)』
77 ページの『可変長テーブルへの値の割り当て』

テーブルの動的なロード

テーブルの初期値がプログラムの実行のたびに異なる場合は、初期値を指定せずにテーブルを定義することができます。そのあと、プログラムでテーブルを参照する前に、変更済みの値をテーブルに読み込むことができます。

テーブルをロードするには、PERFORM ステートメントと添え字付けまたは指標付けのいずれかを使用してください。

データを読み取ってテーブルをロードするときは、データがテーブルに割り振られているスペースを超えないようにしてください。項目カウントには、名前付きの値 (リテラルではなく) を使用してください。そうすれば、テーブルをより大きくする場合に、リテラルへのすべての参照を変更する代わりに、1 つの値を変更するだけで済みます。

72 ページの『例: PERFORM と添え字付け』
73 ページの『例: PERFORM と指標付け』

関連参照
VARYING 句を持つ PERFORM (*Enterprise COBOL 言語解説書*)

テーブルの初期化 (INITIALIZE)

INITIALIZE ステートメントを使用すれば、実行中にテーブルに値をロードすることができます。たとえば、テーブルを 3 で埋めるには、次のように指定します。

```
INITIALIZE TABLE-ONE REPLACING NUMERIC DATA BY 3.
```

INITIALIZE ステートメントで、可変長テーブル (OCCURS DEPENDING ON を使用して定義されたもの) をロードすることはできません。

関連参照
INITIALIZE ステートメント (*Enterprise COBOL 言語解説書*)

テーブルの定義時の値の割り当て (VALUE)

テーブルに持続性のある値 (年の日や月など) が含まれる場合は、テーブルを定義するときに、テーブルに入れる特定の値を設定してください。

静的値は、次のいずれかの方法で WORKING-STORAGE に定義します。

- 各テーブル項目を個別に初期化する。
- テーブル全体を 01 レベルで初期化する。
- 特定のテーブル・エレメントのすべてのオカレンスを同じ値に初期化する。

それぞれのテーブル項目の個別の初期化

テーブルが小さい場合に、この手法を使用することができます。

1. テーブルにあるものと同じ項目を含むレコードを宣言します。
2. 各項目の初期値を VALUE 文節で設定します。
3. そのレコードをテーブルに入れるための REDEFINES 記入項目をコーディングします。

たとえば、次のように記述します。

```
************************************************************
***          E R R O R   F L A G   T A B L E          ***
************************************************************
 01  Error-Flag-Table                  Value Spaces.
   88 No-Errors                       Value Spaces.
     05 Type-Error                    Pic X.
     05 Shift-Error                   Pic X.
     05 Home-Code-Error               Pic X.
     05 Work-Code-Error               Pic X.
     05 Name-Error                    Pic X.
```

```
    05 Initials-Error                  Pic X.
    05 Duplicate-Error                 Pic X.
    05 Not-Found-Error                 Pic X.
 01  Filler Redefines Error-Flag-Table.
    05 Error-Flag Occurs 8 Times
         Indexed By Flag-Index         Pic X.
```

(この例では、各項目が同じ値に初期化されているので、すべての項目を 01 レベルで 1 つの VALUE 文節で初期化することができます。)

もっと大きなテーブルを初期化する場合は、MOVE、PERFORM、または INITIALIZE ステートメントを使用します。

01 レベルでのテーブルの初期化

レベル 01 のレコードをコーディングし、VALUE 文節を使用して、そのレコードにテーブル全体の内容を割り当てます。次に、従属レベルのデータ項目で、OCCURS 文節を使用して個々のテーブル項目を定義します。

たとえば、次のように記述します。

```
01  TABLE-ONE                      VALUE "1234".
    05 TABLE-TWO OCCURS 4 TIMES    PIC X.
```

テーブル・エレメントのすべてのオカレンスの初期化

テーブル・エレメントについての VALUE 文節を使用して、そのエレメントを指定の値に初期化することができます。たとえば、次のように記述します。

```
01  T2.
    05 T-OBJ                    PIC 9   VALUE 3.
    05 T OCCURS 5 TIMES
          DEPENDING ON T-OBJ.
       10 X                     PIC XX  VALUE "AA".
       10 Y                     PIC 99  VALUE 19.
       10 Z                     PIC XX  VALUE "BB".
```

この場合、すべての X エレメント (1 ～ 5) が AA に初期化され、すべての Y エレメント (1 ～ 5) が 19 に初期化され、すべての Z エレメント (1 ～ 5) が BB に初期化されます。その後、T-OBJ が 3 に設定されます。

関連参照

REDEFINES 文節 (*Enterprise COBOL 言語解説書*)
PERFORM ステートメント (*Enterprise COBOL 言語解説書*)
INITIALIZE ステートメント (*Enterprise COBOL 言語解説書*)
OCCURS 文節 (*Enterprise COBOL 言語解説書*)

例: PERFORM と添え字付け

この例では、設定されたエラー・コードが見つかるまで、添え字付けを使用してエラー・フラグ・テーブルを検索します。エラー・コードが見つかると、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```
. . .
************************************************************
***          E R R O R   F L A G   T A B L E          ***
************************************************************
 01  Error-Flag-Table                  Value Spaces.
   88 No-Errors                       Value Spaces.
     05 Type-Error                    Pic X.
```

```
     05 Shift-Error                      Pic X.
     05 Home-Code-Error                  Pic X.
     05 Work-Code-Error                  Pic X.
     05 Name-Error                       Pic X.
     05 Initials-Error                   Pic X.
     05 Duplicate-Error                  Pic X.
     05 Not-Found-Error                  Pic X.
 01  Filler Redefines Error-Flag-Table.
     05 Error-Flag Occurs 8 Times
 77  ERROR-ON                           Pic X  Value "E".
          Indexed By Flag-Index          Pic X.
************************************************************
***         E R R O R   M E S S A G E   T A B L E       ***
************************************************************
 01  Error-Message-Table.
     05  Filler                          Pic X(25) Value
          "Transaction Type Invalid".
     05  Filler                          Pic X(25) Value
          "Shift Code Invalid".
     05  Filler                          Pic X(25) Value
          "Home Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Work Location Code Inval.".
     05  Filler                          Pic X(25) Value
          "Last Name - Blanks".
     05  Filler                          Pic X(25) Value
          "Initials - Blanks".
     05  Filler                          Pic X(25) Value
          "Duplicate Record Found".
     05  Filler                          Pic X(25) Value
          "Commuter Record Not Found".
 01  Filler Redefines Error-Message-Table.
     05  Error-Message Occurs 8 Times
            Indexed By Message-Index      Pic X(25).
 . . .
 PROCEDURE DIVISION.
     . . .
     Perform
          Varying Sub From 1 By 1
          Until No-Errors
        If Error-Flag (Sub) = Error-On
          Move Space To Error-Flag (Sub)
          Move Error-Message (Sub) To Print-Message
          Perform 260-Print-Report
        End-If
     End-Perform
     . . .
```

例: PERFORM と指標付け

この例では、設定されたエラー・コードが見つかるまで、指標付けを使用してエラー・フラグ・テーブルを検索します。エラー・コードが見つかると、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```
. . .
************************************************************
***           E R R O R   F L A G   T A B L E           ***
************************************************************
 01  Error-Flag-Table                   Value Spaces.
   88 No-Errors                         Value Spaces.
     05 Type-Error                      Pic X.
     05 Shift-Error                     Pic X.
     05 Home-Code-Error                 Pic X.
     05 Work-Code-Error                 Pic X.
     05 Name-Error                      Pic X.
     05 Initials-Error                  Pic X.
```

```
     05 Duplicate-Error                  Pic X.
     05 Not-Found-Error                  Pic X.
 01  Filler Redefines Error-Flag-Table.
     05 Error-Flag Occurs 8 Times
          Indexed By Flag-Index          Pic X.
************************************************************
***         E R R O R   M E S S A G E   T A B L E        ***
************************************************************
 01  Error-Message-Table.
     05  Filler                           Pic X(25) Value
          "Transaction Type Invalid".
     05  Filler                           Pic X(25) Value
          "Shift Code Invalid".
     05  Filler                           Pic X(25) Value
          "Home Location Code Inval.".
     05  Filler                           Pic X(25) Value
          "Work Location Code Inval.".
     05  Filler                           Pic X(25) Value
          "Last Name - Blanks".
     05  Filler                           Pic X(25) Value
          "Initials - Blanks".
     05  Filler                           Pic X(25) Value
          "Duplicate Record Found".
     05  Filler                           Pic X(25) Value
          "Commuter Record Not Found".
 01  Filler Redefines Error-Message-Table.
     05  Error-Message Occurs 8 Times
            Indexed By Message-Index       Pic X(25).
 . . .
 PROCEDURE DIVISION.
     . . .
     Set Flag-Index To 1
     Perform Until No-Errors
       Search Error-Flag
         When Error-Flag (Flag-Index) = Error-On
           Move Space To Error-Flag (Flag-Index)
           Set Message-Index To Flag-Index
           Move Error-Message (Message-Index) To
             Print-Message
           Perform 260-Print-Report
       End-Search
     End-Perform
     . . .
```

可変長テーブルの作成 (DEPENDING ON)

テーブル・エレメントが出現する回数が実行前にわからない場合には、可変長テーブル定義をセットアップしなければなりません。そのためには、OCCURS DEPENDING ON (ODO) 文節を使用します。たとえば、次のように記述します。

```
X OCCURS 1 TO 10 TIMES DEPENDING ON Y
```

この例では、X が *ODO* のサブジェクト であり、Y が *ODO* のオブジェクト です。

可変長レコードを正しく操作するためには、次の 2 つの要因が影響します。

- レコード長を正しく計算すること

  グループ項目の可変部分の長さは、DEPENDING ON オプションのオブジェクトと OCCURS 文節のサブジェクトの長さとの積です。
- OCCURS DEPENDING ON 文節のオブジェクトにおけるデータの PICTURE 文節との適合性。

ODO オブジェクトの内容がそれの PICTURE 文節と一致していない場合には、プログラムが異常終了することがあります。ODO オブジェクトに、テーブル・エレメントの現在のオカレンス回数を正しく指定するようにしてください。

次の例は、OCCURS DEPENDING ON 文節のサブジェクトとオブジェクトの両方を含むグループ項目 (REC-1) を示しています。グループ項目の長さがどのようにして決定されるかは、それがデータを送り出しているのか、データを受け取っているのかによって異なります。

```
WORKING-STORAGE SECTION.
01  MAIN-AREA.
    03 REC-1.
      05 FIELD-1                        PIC 9.
      05 FIELD-2 OCCURS 1 TO 5 TIMES
           DEPENDING ON FIELD-1        PIC X(05).
01  REC-2.
    03 REC-2-DATA                       PIC X(50).
```

REC-1 (この場合は送り出し項目) を REC-2 に移動したい場合、REC-1 の長さは、FIELD-1 の現行値を使用して、移動の直前に決定されます。FIELD-1 の内容がその PICTURE に適合する (すなわち、FIELD-1 に外部 10 進数項目が含まれている) 場合は、REC-1 の実際の長さに基づいて移動を行うことができます。さもないと、結果は予測できません。移動を開始する前に、ODO オブジェクトに正しい値が含まれていることを確認してください。

REC-1 (この場合は受け取り項目) に移動を行う場合、REC-1 の長さは、最大のオカレンス回数を使用して決定されます。この例では、FIELD-2 の 5 回のオカレンスと FIELD-1 の合計で 26 バイトの長さになります。

この場合、REC-1 を受け取り項目として参照する前に、ODO オブジェクト (FIELD-1) を設定する必要はありません。ただし、移動によって受け取りフィールドの ODO オブジェクトを有効に設定するようにするためには、送り出しフィールドの ODO オブジェクトを 1 ～ 5 の間の有効な数値に設定しなければなりません。

しかし、REC-1 の後に可変位置グループ (*複合 ODO*) が続いているような REC-1 (この場合も受け取り項目) に移動を行う場合は、REC-1 の実際の長さは、移動の直前に計算されます。次の例では、REC-1 と REC-2 は同じレコードにありますが、REC-2 は REC-1 に従属していないため、可変位置項目です。

```
01  MAIN-AREA
    03 REC-1.
       05 FIELD-1                       PIC 9.
       05 FIELD-3                       PIC 9.
       05 FIELD-2 OCCURS 1 TO 5 TIMES
            DEPENDING ON FIELD-1       PIC X(05).
    03 REC-2.
       05 FIELD-4 OCCURS 1 TO 5 TIMES
            DEPENDING ON FIELD-3       PIC X(05).
```

この場合、REC-1 への MOVE を行うと、REC-1 の実際の長さは、ODO オブジェクト (FIELD-1) の現行値を使用して、移動の直前に計算されます。コンパイラーは、実際の長さが使用されたことを知らせるメッセージを出します。この場合には、グループ項目を受け取りフィールドとして使用する前に、ODO オブジェクトの値を設定することが必要となります。

次の例は、ODO オブジェクト (ここでは LOCATION-TABLE-LENGTH) がグループの外側にあるときの、可変長テーブルの定義方法を示します。

```
 DATA DIVISION.
 FILE SECTION.
 FD  LOCATION-FILE
     RECORDING MODE F
     BLOCK 0 RECORDS
     RECORD 80 CHARACTERS
     LABEL RECORD STANDARD.
 01  LOCATION-RECORD.
     05  LOC-CODE                 PIC XX.
     05  LOC-DESCRIPTION          PIC X(20).
     05  FILLER                   PIC X(58).
 . . .
 WORKING-STORAGE SECTION.
 01  FLAGS.
     05 LOCATION-EOF-FLAG         PIC X(5) VALUE SPACE.
        88 LOCATION-EOF              VALUE "FALSE".
 01  MISC-VALUES.
     05 LOCATION-TABLE-LENGTH     PIC 9(3) VALUE ZERO.
     05 LOCATION-TABLE-MAX        PIC 9(3) VALUE 100.
*****************************************************************
***               L O C A T I O N   T A B L E                 ***
***               FILE CONTAINS LOCATION CODES.               ***
*****************************************************************
 01  LOCATION-TABLE.
     05 LOCATION-CODE OCCURS 1 TO 100 TIMES
          DEPENDING ON LOCATION-TABLE-LENGTH   PIC X(80).
```

関連概念

631 ページの『付録 B. 複合 OCCURS DEPENDING ON』

関連タスク

77 ページの『可変長テーブルへの値の割り当て』

『可変長テーブルのロード』

Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド

関連参照

OCCURS DEPENDING ON 文節 (*Enterprise COBOL 言語解説書*)

可変長テーブルのロード

do-until 構造 (TEST AFTER ループ) を使用して、可変長テーブルのロードを制御することができます。たとえば、次のコードが実行されると、LOCATION-TABLE-LENGTH には、テーブルの最後の項目の添え字が入れられます。

```
 DATA DIVISION.
 FILE SECTION.
 FD  LOCATION-FILE
     RECORDING MODE F
     BLOCK 0 RECORDS
     RECORD 80 CHARACTERS
     LABEL RECORD STANDARD.
 01  LOCATION-RECORD.
     05  LOC-CODE                 PIC XX.
     05  LOC-DESCRIPTION          PIC X(20).
     05  FILLER                   PIC X(58).
 . . .
 WORKING-STORAGE SECTION.
 01  FLAGS.
     05 LOCATION-EOF-FLAG         PIC X(5) VALUE SPACE.
        88 LOCATION-EOF                    VALUE "YES".
```

```
 01  MISC-VALUES.
     05 LOCATION-TABLE-LENGTH      PIC 9(3) VALUE ZERO.
     05 LOCATION-TABLE-MAX         PIC 9(3) VALUE 100.
*****************************************************************
***               L O C A T I O N   T A B L E                 ***
***               FILE CONTAINS LOCATION CODES.               ***
*****************************************************************
 01  LOCATION-TABLE.
     05 LOCATION-CODE OCCURS 1 TO 100 TIMES
          DEPENDING ON LOCATION-TABLE-LENGTH   PIC X(80).
 . . .
 PROCEDURE DIVISION.
     . . .
     Perform Test After
         Varying Location-Table-Length From 1 By 1
           Until Location-EOF
           Or Location-Table-Length = Location-Table-Max
       Move Location-Record To
           Location-Code (Location-Table-Length)
       Read Location-File
           At End Set Location-EOF To True
       End-Read
     End-Perform
```

可変長テーブルへの値の割り当て

DEPENDING ON オプションを指定した OCCURS 文節を含むグループ項目に、VALUE 文節を使用することができます。DEPENDING ON オプションを含むそれぞれの従属構造は、最大オカレンス回数を使用して初期化されます。DEPENDING ON オプションを使用してテーブル全体を定義した場合、すべてのエレメントは、DEPENDING ON のオブジェクトの最大定義値を使用して初期化されます。

ODO のオブジェクト に VALUE 文節がある場合、これは論理的には、*ODO* のサブジェクト が初期化された後で初期化されます。たとえば、次のコーディングでは、

```
01  TABLE-THREE             VALUE "3ABCDE".
    05 X                     PIC 9.
    05 Y OCCURS 5 TIMES
          DEPENDING ON X    PIC X.
```

ODO のサブジェクト Y(1) が A に初期化され、Y(2) が B に、と順次初期化されていき、Y(5) が E に初期化されると、最後に ODO のオブジェクト X が 3 に初期化されます。その後の TABLE-THREE (DISPLAY ステートメントなどでの) への参照はすべて、最初の 3 つのエレメント Y(1) ～ Y(3) を参照します。

関連タスク

Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド

関連参照

OCCURS DEPENDING ON 文節 (*Enterprise COBOL 言語解説書*)

テーブルの探索

COBOL には、*シリアル探索* と*二分探索* (バイナリー・サーチ) の 2 つのテーブル探索技法があります。

シリアル検索を行うには、SEARCH と指標付けを使用します。可変長テーブルの場合は、PERFORM と添え字付けまたは指標付けを使用することができます。

二分探索を行うには、SEARCH ALL と指標付けを使用します。

二分探索の方が、シリアル検索よりも効率が相当よくなる可能性があります。シリアル検索の場合、比較の数は、*n* の次数、すなわちテーブルの項目の数です。二分探索の場合、比較の数は、*n* の対数 (基底 2) の次数にすぎません。ただし、二分探索を行うには、テーブル項目をあらかじめソートしておく必要があります。

関連タスク
『シリアル検索 (SEARCH)』
79 ページの『二分探索 (SEARCH ALL)』

シリアル検索 (SEARCH)

SEARCH ステートメントを使用すると、現在の指標設定値からシリアル検索を行うことができます。指標設定値を変更するには、SET ステートメントを使用してください。

WHEN オプション内の条件は、それらが指定された順番に評価されます。

- どの条件も満たされない場合には、指標が次のテーブル・エレメントに対応するように増加され、WHEN 条件が再度評価されます。
- WHEN 条件の 1 つが満たされると、探索は終了します。指標は条件を満たしたテーブル・エレメントを指したままになります。
- テーブル全体が探索され、どの条件も満たされなかった場合には、AT END 命令ステートメントが実行されます (もしあれば)。AT END を使用しない場合、制御はプログラム内の次のステートメントに渡ります。

それぞれの SEARCH ステートメントでは、テーブルの 1 つのレベル (1 つのテーブル・エレメント) だけを参照することができます。テーブルの複数のレベルを探索するには、ネストされた SEARCH ステートメントを使用してください。ネストされたそれぞれの SEARCH ステートメントは、END-SEARCH で区切らなければなりません。

検出された条件がテーブル内のある中間点より後にくる場合には、探索を速めることができます。SET ステートメントを使用して、そのポイントより後から探索を開始するよう指標を設定してください。

さらに、最も頻繁に使用されるデータが先頭になるようにテーブルを配置すると、シリアル検索をより効率的に行うことができます。テーブルが大きくて、事前にソートされている場合には、二分探索の方が効率的です。

79 ページの『例: シリアル検索』

関連参照
SEARCH ステートメント (*Enterprise COBOL 言語解説書*)

例: シリアル検索

3 次元のテーブルがあり、それぞれが独自の指標を持っているとします (それぞれ、1、4、および 1 に設定されています)。最も内部のテーブル (TABLE-ENTRY3) には昇順キーがあります。探索の目的は、最も内部にあるテーブルの中の特定のストリングを見つけることです。

```
01  TABLE-ONE.
    05 TABLE-ENTRY1 OCCURS 10 TIMES
          INDEXED BY TE1-INDEX.
       10 TABLE-ENTRY2 OCCURS 10 TIMES
             INDEXED BY TE2-INDEX.
          15 TABLE-ENTRY3 OCCURS 5 TIMES
                ASCENDING KEY IS KEY1
                INDEXED BY TE3-INDEX.
             20 KEY1                 PIC X(5).
             20 KEY2                 PIC X(10).
. . .
PROCEDURE DIVISION.
    . . .
    SET TE1-INDEX TO 1
    SET TE2-INDEX TO 4
    SET TE3-INDEX TO 1
    MOVE "A1234" TO KEY1 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
    MOVE "AAAAAAAA00" TO KEY2 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
    . . .
    SEARCH TABLE-ENTRY3
      AT END
        MOVE 4 TO RETURN-CODE
      WHEN TABLE-ENTRY3(TE1-INDEX, TE2-INDEX, TE3-INDEX)
          = "A1234AAAAAAAA00"
        MOVE 0 TO RETURN-CODE
    END-SEARCH
```

実行後の値:

```
TE1-INDEX = 1
TE2-INDEX = 4
TE3-INDEX points to the TABLE-ENTRY3 item
          that equals "A1234AAAAAAAA00"
RETURN-CODE = 0
```

二分探索 (SEARCH ALL)

SEARCH ALL を使用して二分探索を行う場合、開始する前に指標を設定する必要はありません。使用される指標は常に、OCCURS 文節内の最初の指標名と関連付けられるものです。この指標は、探索の効率を最大にするために、実行中に変わります。

SEARCH ALL ステートメントを使用するためには、テーブルが、OCCURS 文節でコーディングされた 1 つまたは複数のキーに基づいて順序付けられていなければなりません。WHEN 条件内では任意のキーを使用できますが、先行するデータ名があれば、それらを KEY オプションでテストしなければなりません。テストは等号条件でなければならず、KEY *data-name* は、条件のサブジェクト、またはテストされる条件名と関連付けられている条件変数の名前でなければなりません。WHEN 条件は、AND を唯一の論理連結語とする単純条件から構成される複合条件であっても構いません。比較のキーとその対象 (オブジェクト) には互換性がなければなりません。

80 ページの『例: 二分探索』

関連参照

SEARCH ステートメント (*Enterprise COBOL 言語解説書*)

例: 二分探索

テーブルにそれぞれが 40 バイトの 90 個のエレメントが含まれており、3 つのキーがあるとします。1 次キーと 2 次キー (KEY-1 と KEY-2) は昇順ですが、最も重要でないキー (KEY-3) は降順です。

```
01  TABLE-A.
    05 TABLE-ENTRY OCCURS 90 TIMES
           ASCENDING KEY-1, KEY-2
           DESCENDING KEY-3
           INDEXED BY INDX-1.
       10 PART-1       PIC 99.
       10 KEY-1        PIC 9(5).
       10 PART-2       PIC 9(6).
       10 KEY-2        PIC 9(4).
       10 PART-3       PIC 9(18).
       10 KEY-3        PIC 9(5).
```

このテーブルは、次の命令を使用して探索することができます。

```
SEARCH ALL TABLE-ENTRY
  AT END
    PERFORM NOENTRY
  WHEN KEY-1 (INDX-1) = VALUE-1 AND
       KEY-2 (INDX-1) = VALUE-2 AND
       KEY-3 (INDX-1) = VALUE-3
    MOVE PART-1 (INDX-1) TO OUTPUT-AREA
END-SEARCH
```

3 つのキーが指定された値 (VALUE-1、VALUE-2、および VALUE-3) に等しい記入項目が見つかると、その記入項目の PART-1 が OUTPUT-AREA に移動されます。TABLE-A 内のどの記入項目でも一致するキーが検出されない場合には、NOENTRY ルーチンが実行されます。

組み込み関数を使用したテーブル項目の処理

組み込み関数を使用して、英数字または数値のテーブル項目を処理することができます。ただし、そのテーブル項目のデータ記述が、その関数の引き数要件と互換性がなければなりません。

個々のデータを関数引き数として参照するには、添え字または指標を使用してください。Table-One が 3x3 の数値項目の配列であるとすると、次のようなステートメントを使用して中間エレメントの平方根を求めることができます。

```
Compute X = Function Sqrt(Table-One(2,2))
```

テーブル内のデータを繰り返し処理することが必要な場合もあります。複数の引き数を受け入れる組み込み関数の場合は、ALL 添え字を使用して、テーブル内のすべての項目またはテーブルの 1 つの次元を参照することができます。反復は自動的に処理されるため、コードがより短く、単純になります。

複数の引き数を受け入れる関数の場合は、スカラーと配列引き数を混在させることができます。

```
Compute Table-Median = Function Median(Arg1 Table-One(ALL))
```

81 ページの『例: 組み込み関数』

関連タスク
37 ページの『組み込み関数の使用 (組み込み関数)』

関連参照
組み込み関数 (*Enterprise COBOL 言語解説書*)

例: 組み込み関数

この例では、Table-Two のクロス・セクションを合計します。

```
Compute Table-Sum = FUNCTION SUM (Table-Two(ALL, 3, ALL))
```

Table-Two が 2x3x2 の配列であるとすると、上記のステートメントによって以下のエレメントが合計されます。

```
Table-Two(1,3,1)
Table-Two(1,3,2)
Table-Two(2,3,1)
Table-Two(2,3,2)
```

この例では、すべての従業員の値を計算します。

```
01  Employee-Table.
    05 Emp-Count      Pic s9(4) usage binary.
    05 Emp-Record     Occurs 1 to 500 times
                        depending on Emp-Count.
       10 Emp-Name    Pic x(20).
       10 Emp-Idme    Pic 9(9).
       10 Emp-Salary  Pic 9(7)v99.
. . .
Procedure Division.
    Compute Max-Salary = Function Max(Emp-Salary(ALL))
    Compute I = Function Ord-Max(Emp-Salary(ALL))
    Compute Avg-Salary = Function Mean(Emp-Salary(ALL))
    Compute Salary-Range = Function Range(Emp-Salary(ALL))
    Compute Total-Payroll = Function Sum(Emp-Salary(ALL))
```

第 5 章 プログラム・アクションの選択と反復

COBOL 制御言語を使用すると、論理テストの結果に基づいてプログラム・アクションを選択すること、プログラムおよびデータの選択された部分を繰り返すこと、および 1 つのグループとして実行すべきステートメントを識別することができます。この制御には、以下のものが含まれます。

- IF ステートメント
- EVALUATE ステートメント
- スイッチおよびフラグ
- PERFORM ステートメント

関連タスク
『プログラム・アクションの選択』
91 ページの『プログラム・アクションの反復』

プログラム・アクションの選択

1 つまたは複数のデータ項目のテストされた値に基づいて、さまざまなプログラム・アクションに備えることができます。

COBOL の IF および EVALUATE ステートメントは、条件式によって 1 つまたは複数のデータ項目をテストします。

関連タスク
『アクションの選択項目のコーディング』
88 ページの『条件式のコーディング』

関連参照
IF ステートメント (*Enterprise COBOL 言語解説書*)
EVALUATE ステートメント (*Enterprise COBOL 言語解説書*)

アクションの選択項目のコーディング

IF . . . ELSE は、2 つの処理アクション間での選択項目をコーディングする場合に使用します。(THEN という語は、COBOL プログラムではオプショナルです。)

```
IF condition-p
  statement-1
ELSE
  statement-2
END-IF
```

処理選択項目の 1 つがアクションを取らない場合は、IF ステートメントに ELSE を指定してもしなくても構いません。ELSE 文節はオプショナルであるため、次のようにコーディングすることができます。

```
IF condition-q
  statement-1
END-IF
```

© Copyright IBM Corp. 1991, 2001

このコーディングは、簡単なプログラミングの場合に適しています。ロジックが複雑になった場合は、おそらく、ELSE 文節を使用する必要があります。たとえば、処理選択項目の 1 つだけに対応するアクションがある、ネストされた IF ステートメントがあるとします。その場合は、次のように、ELSE 文節と CONTINUE ステートメントを使用して IF ステートメントの NULL ブランチをコーディングすることができます。

```
IF condition-q
  statement-1
ELSE
  CONTINUE
END-IF
```

条件が 2 つだけではなく、3 つ以上の可能な条件の間で選択項目をコーディングする場合は、EVALUATE ステートメントを使用します。EVALUATE ステートメントは、IF ステートメントを拡張したものであり、論理エラーやデバッグ問題の共通ソースのコーディングなどの場合に、ネストされた IF ステートメントの使用を回避するために使用することができます。

EVALUATE ステートメントを使用すると、1 つのステートメントの中で条件をいくつでもテストすることができ、それぞれの条件について別々のアクションを指定することができます。構造化プログラミング用語では、これはケース構造と呼ばれます。デシジョン・テーブルと見なすこともできます。

86 ページの『例: THRU 句を使用する EVALUATE』
86 ページの『例: 複数の WHEN ステートメントを使用する EVALUATE』
87 ページの『例: 複数の条件をテストする EVALUATE』

関連タスク
88 ページの『条件式のコーディング』
85 ページの『EVALUATE ステートメントの使用』
『ネストされた IF ステートメントの使用』

ネストされた IF ステートメントの使用

IF ステートメントが、可能な処理分岐の 1 つとして別の IF ステートメントを持つ場合、これらの IF ステートメントはネストされているといいます。論理上は、ネストされた IF ステートメントの深さに制限はありません。ただし、プログラムが 3 つ以上の値について変数をテストしなければならない場合には、EVALUATE を使用する方が適切です。

ネストされた IF ステートメントを多用しないでください。明示的範囲終了符号および適切な字下げが役に立つとはいえ、ロジックをたどるのが困難になる可能性があります。

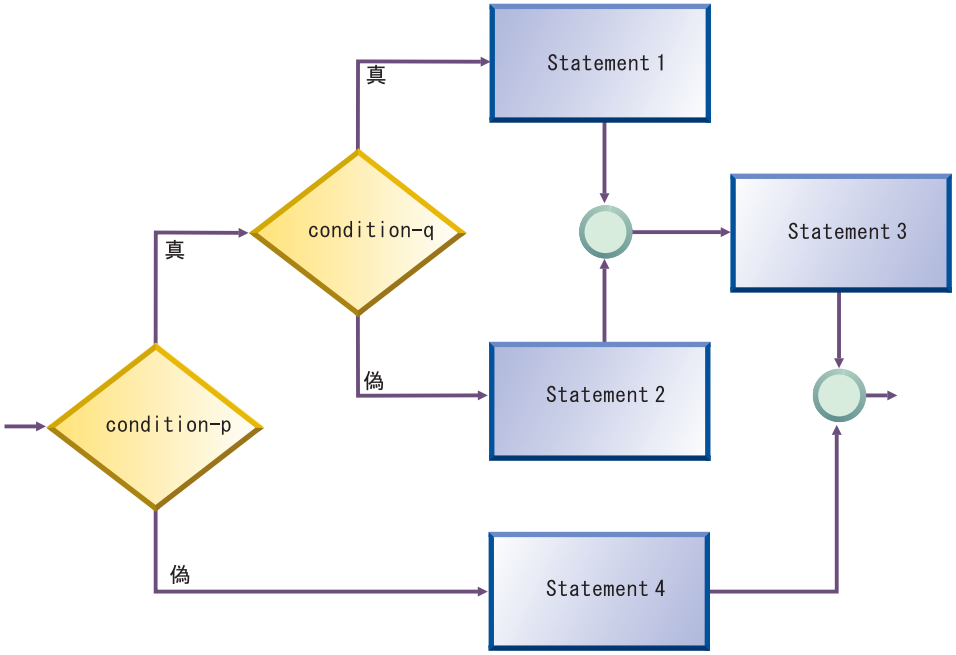
以下に、ネストされた IF ステートメントの疑似コードを示します。

```
IF condition-p
  IF condition-q
    statement-1
  ELSE
    statement-2
  END-IF
```

```
  statement-3
ELSE
  statement-4
END-IF
```

ここでは、1 つの IF が、順次構造とともに、別の IF の 1 つの分岐にネストされています。このような構造では、内側のネストされた IF を閉じる END-IF が非常に重要になります。ピリオドは外側の IF 構造も終了させるため、ピリオドではなく END-IF を使用してください。

次の図は、ネストされた IF ステートメントの論理構造を示しています。



関連タスク

83 ページの『アクションの選択項目のコーディング』

関連参照

明示的範囲終了符号 (*Enterprise COBOL 言語解説書*)

EVALUATE ステートメントの使用

EVALUATE ステートメントを使用すると、いくつかの条件をテストし、それぞれの条件に異なるアクション (*ケース構造* と呼ばれることが多い構成) を設計することができます。テストされる式は、選択サブジェクトと呼ばれます。選択される返答は選択オブジェクトと呼ばれます。同じ構造内に複数のサブジェクトと複数のオブジェクトをコーディングすることができます。

EVALUATE ステートメントを指定すれば、THRU 句および複数の WHEN ステートメントを使用して、複数の条件が同じ処理になるようなケースを処理することができます。

評価されるときは、選択サブジェクトと選択オブジェクトのそれぞれの対が同じクラス (数値、文字、CONDITION TRUE、または FALSE) に属していなければなりません。

EVALUATE ステートメントの実行は、次の場合に終了します。

- 選択された WHEN 句に関連付けられているステートメントが実行された。
- WHEN OTHER 句に関連付けられているステートメントが実行された。
- いずれの WHEN 条件も満たされない。

WHEN 句は、コーディング順にテストされます。したがって、パフォーマンスが最適になるようにこれらの句を順序付ける必要があります。満たされる可能性が最も高い選択オブジェクトを含む WHEN 句をまず最初にコーディングし、その次に 2 番目に可能性の高いものを含む句をコーディングする、といった具合です。例外は WHEN OTHER 句です。これは最後に置かなければなりません。

関連タスク

83 ページの『アクションの選択項目のコーディング』

例: THRU 句を使用する EVALUATE: この例では、THRU 句を使用して、同じ処理アクションに導かれる複数の条件 (値の範囲) をコーディングする方法を示します。この例では、CARPOOL-SIZE が選択サブジェクトで、1、2、および 3 THRU 6 が選択オブジェクトです。

```
EVALUATE CARPOOL-SIZE
  WHEN 1
    MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
  WHEN 2
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  WHEN 3 THRU 6
    MOVE "SMALL GROUP" TO PRINT-CARPOOL STATUS
  WHEN OTHER
    MOVE "BIG GROUP" TO PRINT-CARPOOL STATUS
END-EVALUATE
```

次のネストされた IF ステートメントも同じロジックを表します。

```
IF CARPOOL-SIZE = 1 THEN
  MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
ELSE
  IF CARPOOL-SIZE = 2 THEN
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  ELSE
    IF CARPOOL-SIZE >= 3 and CARPOOL-SIZE <= 6 THEN
      MOVE "SMALL GROUP" TO PRINT-CARPOOL-STATUS
    ELSE
      MOVE "BIG GROUP" TO PRINT-CARPOOL-STATUS
    END-IF
  END-IF
END-IF
```

例: 複数の WHEN ステートメントを使用する EVALUATE: 複数の条件が同じ処理アクションを導くときは、複数の WHEN ステートメントを使用することができます。この方が、THRU 句を使用するよりも柔軟性が高くなります。条件を範囲内の値に評価したり英数字値に評価したりする必要がないからです。

```
EVALUATE MARITAL-CODE
  WHEN "M"
    ADD 2 TO PEOPLE-COUNT
  WHEN "S"
```

```
  WHEN "D"
  WHEN "W"
    ADD 1 TO PEOPLE-COUNT
END-EVALUATE
```

次のネストされた IF ステートメントも同じロジックを表します。

```
IF MARITAL-CODE = "M" THEN
    ADD 2 TO PEOPLE-COUNT
ELSE
  IF MARITAL-CODE = "S" OR
     MARITAL-CODE = "D" OR
     MARITAL-CODE = "W" THEN
       ADD 1 TO PEOPLE-COUNT
  END-IF
END-IF
```

例: 複数の条件をテストする EVALUATE: この例では、WHEN 句の両方の選択サブジェクトが TRUE を満たしていなければなりません。すなわち、TRUE 条件があれば、この句は実行されます。両方のサブジェクトが TRUE に評価されなければ、次の WHEN 句が処理されます。

```
Identification Division.
  Program-ID. MiniEval.
Environment Division.
  Configuration Section.
  Source-Computer. IBM-390.
Data Division.
  Working-Storage Section.
  01   Age               Pic  999.
  01   Sex               Pic  X.
  01   Description       Pic  X(15).
  01   A                 Pic  999.
  01   B                 Pic  9999.
  01   C                 Pic  9999.
  01   D                 Pic  9999.
  01   E                 Pic  99999.
  01   F                 Pic  999999.
Procedure Division.
  PN01.
    Evaluate True Also True
      When Age < 13 Also Sex = "M"
        Move "Young Boy" To Description
      When Age < 13 Also Sex = "F"
        Move "Young Girl" To Description
      When Age > 12 And Age < 20 Also Sex = "M"
        Move "Teenage Boy" To Description
      When Age > 12 And Age < 20 Also Sex = "F"
        Move "Teenage Girl" To Description
      When Age > 19 Also Sex = "M"
        Move "Adult Man" To Description
      When Age > 19 Also Sex = "F"
        Move "Adult Woman" To Description
      When Other
        Move "Invalid Data" To Description
    End-Evaluate
    Evaluate True Also True
      When A + B < 10 Also C = 10
        Move "Case 1" To Description
      When A + B > 50 Also C = ( D + E ) / F
        Move "Case 2" To Description
      When Other
        Move "Case Other" To Description
    End-Evaluate
    Stop Run.
```

条件式のコーディング

IF および EVALUATE ステートメントを使用して、条件式の真理値に従って実行されるプログラム・アクションをコーディングすることができます。 以下の条件のどれでも指定することができます。

- 数値条件
- 英数字条件
- 国別条件
- フィールドのクラス
- 定義したスイッチおよびフラグ
- 符号条件
- UPSI スイッチの状況

国別オペランドを含む条件式はすべて国別条件です。PROGRAM COLLATING SEQUENCE 文節は、国別条件式に影響を及ぼしません。

関連概念
『スイッチおよびフラグ』

関連タスク
89 ページの『スイッチおよびフラグの定義』
90 ページの『スイッチおよびフラグのリセット』
53 ページの『非互換データの検査 (数値のクラス・テスト)』
121 ページの『国別データ項目の比較』

関連参照
条件名値の規則 (*Enterprise COBOL 言語解説書*)
スイッチ状況条件 (*Enterprise COBOL 言語解説書*)
符号条件 (*Enterprise COBOL 言語解説書*)
数値オペランドと英数字オペランドの比較 (*Enterprise COBOL 言語解説書*)
複合条件 (*Enterprise COBOL 言語解説書*)
クラス条件 (*Enterprise COBOL 言語解説書*)

スイッチおよびフラグ

プログラム中のいくつかの決定は、データ項目の値が真か偽か、オンかオフか、はいかいいえかに基づきます。スイッチとして働くレベル 88 項目に意味のある名前 (*条件名*) を付けて定義して、これらの両方向決定を制御してください。

その他のプログラム決定は、データ項目の特定の値または値の範囲に依存します。フィールドに単にオンまたはオフ以外の値を与えるために条件名を使用するときには、そのフィールドはフラグと呼ばれるのが普通です。

フラグおよびスイッチを使用すると、コードの変更が容易になります。条件の値を変更する必要がある場合は、そのレベル 88 条件名の値を変更するだけで済みます。

たとえば、特定の給与範囲についてフィールドをテストするために、プログラムが条件名を使用するとします。別の給与範囲を検査するようにプログラムを変更しな

ければならない場合は、DATA DIVISION 内の条件名の値を変更するだけで済みます。PROCEDURE DIVISION で変更を行う必要はありません。

関連タスク
『スイッチおよびフラグの定義』
90 ページの『スイッチおよびフラグのリセット』

スイッチおよびフラグの定義

DATA DIVISION に、レベル 88 項目を定義して、スイッチまたはフラグとして働く値に意味のある名前 (条件名) を与えてください。

3 つ以上の値をフラグとしてテストするには、複数のレベル 88 項目を使用することによって、フィールドに複数の条件名を割り当ててください。

意味のある条件名が選択されており、かつ割り当てられた値が論理値に関連付けられているならば、プログラムを読むときコードを追跡するのが容易になります。

『例: スイッチ』
『例: フラグ』

例: スイッチ

Transaction-File という名前の入力ファイルについてのファイル終わり (EOF) 条件をテストするには、データ定義を以下のように記述できます。

```
WORKING-STORAGE Section.
01  Switches.
    05  Transaction-EOF-Switch  Pic X  value space.
        88  Transaction-EOF  value "y".
```

レベル 88 記述では、Transaction-EOF-Switch の値が 'y' なら、Transaction-EOF という名前の条件がオンになることが指定されています。PROCEDURE DIVISION で Transaction-EOF を参照するということは、Transaction-EOF-Switch = "y" をテストするのと同じ条件を表します。たとえば、次のステートメントを使用すると、プログラムが Transaction-File の終わりまで読み取り、Transaction-EOF-Switch が 'y' に設定されている場合のみ、報告書が印刷されます。

```
If Transaction-EOF Then
    Perform Print-Report-Summary-Lines
```

例: フラグ

マスター・ファイルを更新するプログラムがあるとします。更新内容は、トランザクション・ファイルから読み取られます。トランザクション・ファイルのレコードには、実行される機能 (追加、変更、または削除) を示すフィールドが入っています。入力ファイルのレコード記述で、レベル 88 項目を使用して機能コード用のフィールドをコーディングします。

```
01  Transaction-Input Record
    05  Transaction-Type            Pic X.
        88  Add-Transaction         Value "A".
        88  Change-Transaction      Value "C".
        88  Delete-Transaction      Value "D".
```

これらの条件名をテストするための PROCEDURE DIVISION 内のコードは、次のように記述します。

```
Evaluate True
  When Add-Transaction
    Perform Add-Master-Record-Paragraph
  When Change-Transaction
    Perform Update-Exisitng-Record-Paragraph
  When Delete-Transaction
    Perform Delete-Master-Record-Paragraph
End-Evaluate
```

スイッチおよびフラグのリセット

プログラムの随所で、スイッチをリセットしたり、フラグをそれらのデータ記述における元の値に戻したりすることが必要になる場合があります。そのためには、SET ステートメントを使用するか、または使用する独自のデータ項目を定義することができます。

SET *condition-name* TO TRUE ステートメントを使用すると、スイッチまたはフラグは、データ記述の中で割り当てられた元の値に設定し直されます。

複数の値を持つレベル 88 項目の場合、SET *condition-name* TO TRUE は、最初の値 (この例では A) を割り当てます。

```
88 Record-is-Active Value "A" "O" "S"
```

SET ステートメントと意味のある条件名を使用すれば、他のプログラマーにもコードが容易に追跡できるようになります。

『例: スイッチをオンに設定する』
91 ページの『例: スイッチをオフに設定する』

例: スイッチをオンに設定する

次の例の SET ステートメントは、Move "y" to Transaction-EOF-Switch と同じことを行います。

```
01  Switches
    05  Transaction-EOF-Switch       Pic X  Value space.
        88  Transaction-EOF                Value "y".
. . .
Procedure Division.
000-Do-Main-Logic.
    Perform 100-Initialize-Paragraph
    Read Update-Transaction-File
      At End Set Transaction-EOF to True
    End-Read
```

次の例では、入力レコードのトランザクション・コードに基づいて、出力レコード内のフィールドに値を割り当てる方法を示します。

```
01  Input-Record.
    05  Transaction-Type             Pic X(9).
    . . .
01  Data-Record-Out.
    05  Data-Record-Type             Pic X.
        88 Record-Is-Active           Value "A".
        88 Record-Is-Suspended        Value "S".
        88 Record-Is-Deleted          Value "D".
    05  Key-Field                    Pic X(5).
. . .

Procedure Division.
    . . .
    Evaluate Transaction-Type of Input-Record
```

```
      When "ACTIVE"
        Set Record-Is-Active to TRUE
      When "SUSPENDED"
        Set Record-Is-Suspended to TRUE
      When "DELETED"
        Set Record-Is-Deleted to TRUE
    End-Evaluate
```

例: スイッチをオフに設定する

以下に示すように、SWITCH-OFF と呼ばれるデータ項目をプログラムのあちこちで使用して、オン / オフ・スイッチをオフに設定することができます。

```
01  Switches
    05  Transaction-EOF-Switch       Pic X  Value space.
        88  Transaction-EOF                Value "y".
01  SWITCH-OFF                       Pic X  Value "n".
. . .
Procedure Division.
    . . .
    Move SWITCH-OFF to Transaction-EOF-Switch
```

このコードは、ファイルの終わりに達していないことを示すためにスイッチをリセットします。

プログラム・アクションの反復

PERFORM ステートメントを使用すると、段落を実行し、その後で次の実行可能なステートメントに暗黙的に制御権を戻すことができます。実際には、PERFORM ステートメントは、プログラムの異なる多くの部分から入ることができる閉じたサブルーチンをコーディングするための手段です。

PERFORM ステートメントを使用すると、設定された回数だけループする (同じコードを繰り返す) か、判断の結果に基づいてループすることができます。

PERFORM ステートメントはインラインまたはライン外にすることができます。

関連タスク

『インラインまたはライン外 PERFORM の選択』
92 ページの『ループのコーディング』
93 ページの『テーブルを操作するループのコーディング』
94 ページの『複数の段落またはセクションの実行』

関連参照

PERFORM ステートメント (*Enterprise COBOL 言語解説書*)

インラインまたはライン外 PERFORM の選択

インライン PERFORM ステートメントの一般規則は、次の点を除き、ライン外 PERFORM ステートメントと同じです。すなわち、インライン PERFORM ステートメント内のステートメントは、ライン外 PERFORM ステートメントで指定されたプロシージャーの範囲内で実行されます。

インラインまたはライン外のいずれの PERFORM ステートメントをコーディングするかを決定するには、以下の点を考慮してください。

- PERFORM ステートメントを複数の場所から使用しますか。

同じコードをプログラム内の複数の場所から使用する場合は、ライン外 PERFORM を使用します。

- どちらのステートメントの配置の方が読みやすいですか。

  PERFORM が複数の画面に渡るため、プログラムのロジック・フローがあまり明確にならない場合は、ライン外 PERFORM を使用します。しかし、PERFORM 段落が短い場合は、インライン PERFORM を使用することによって、コード内でスキップする問題を除くことができます。

- 効率性を優先させますか。

  パフォーマンスが重要である場合は、ライン外 PERFORM への分岐にかかるオーバーヘッドを回避しなければなりません。しかし、ライン外 PERFORM コーディングでもコード最適化を利用できるので、効率性を過度に重要視する必要はありません。

1974 COBOL 標準では、PERFORM ステートメントはライン外であり、このため、別個の段落への明示分岐が必要であり、暗黙の戻りがあります。実行された段落が、プログラムのそれ以降の順次フローの中にある場合は、ロジック・フローの中でもう 1 度実行されます。この追加の実行を回避するためには、段落を通常の順次フローの外側 (たとえば、GOBACK の後) に置くか、または段落のそばに分岐をコーディングしてください。

インライン PERFORM のサブジェクトは、命令ステートメントです。したがって、(インライン PERFORM 内の命令ステートメント以外の) ステートメントは、明示的範囲終了符号を付けてコーディングしなければなりません。各段落では、1 つの論理関数を実行します。

『例: インライン PERFORM ステートメント』

例: インライン PERFORM ステートメント

この例では、インライン PERFORM ステートメントの構造を示します。この例は、必要な範囲終了符号と必要な END-PERFORM ステートメントを持っています。

```
   Perform 100-Initialize-Paragraph
* The following is an inline PERFORM
   Perform Until Transaction-EOF
      Read Update-Transaction-File Into WS-Transaction-Record
         At End
            Set Transaction-EOF To True
         Not At End
            Perform 200-Edit-Update-Transaction
            If No-Errors
               Perform 300-Update-Commuter-Record
            Else
               Perform 400-Print-Transaction-Errors
* End-If is a required scope terminator
            End-If
            Perform 410-Re-Initialize-Fields
* End-Read is a required scope terminator
      End-Read
   End-Perform
```

ループのコーディング

PERFORM . . . TIMES ステートメントは、段落を何回か実行する場合に使用します。

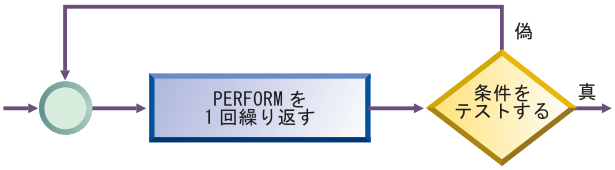
```
PERFORM 010-PROCESS-ONE-MONTH 12 TIMES
INSPECT . . .
```

制御が PERFORM ステートメントに達すると、段落 010-PROCESS-ONE-MONTH のコードが 12 回実行されてから、制御が INSPECT ステートメントに移ります。

PERFORM . . . UNTIL ステートメントは、選択した条件が満たされるまで段落を実行する場合に使用します。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . UNTIL . . .
PERFORM . . . [WITH TEST BEFORE] . . . UNTIL . . .
```

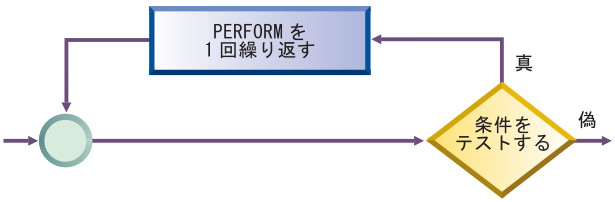
PERFORM . . . WITH TEST AFTER .. . UNTIL は、段落を少なくとも 1 回実行したあと、その後のすべての実行前にテストしたい場合に使用します。このステートメントは、do-until 構造と同等です。



次の例では、暗黙の WITH TEST BEFORE 句によって do-while 構造が提供されます。

```
PERFORM  010-PROCESS-ONE-MONTH
  UNTIL MONTH GREATER THAN 12
  INSPECT . . .
```

制御が PERFORM ステートメントに達すると、条件 (MONTH EQUAL DECEMBER) がテストされます。条件が満たされると、制御が INSPECT ステートメントに移ります。条件が満たされない場合には、010-PROCESS-ONE-MONTH が実行され、条件が再度テストされます。このサイクルは、条件が真になるまで継続されます。(プログラムを読みやすくするために、WITH TEST BEFORE 文節をコーディングすることが必要な場合もあります。)



テーブルを操作するループのコーディング

PERFORM . . . VARYING を使用してテーブルを初期化することができます。この形式の PERFORM ステートメントでは、条件が満たされるまで変数が増分または減分され、テストされます。

そのあと、PERFORM ステートメントを使用して、テーブルを操作するループを制御することができます。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . VARYING . . . UNTIL . . .
PERFORM . . . [WITH TEST BEFORE] . . . VARYING . . . UNTIL .
```

次のコードは、無効なデータがないか、テーブルの初めから終わりまで検査するためのループの例を示しています。

```
PERFORM TEST AFTER VARYING WS-DATA-IX
  FROM 1 BY 1
  UNTIL WS-DATA-IX = 12
  IF WS-DATA (WS-DATA-IX) EQUALS SPACES
    SET SERIOUS-ERROR TO TRUE
    DISPLAY ELEMENT-NUM-MSG5
  END-IF
END-PERFORM
INSPECT . . .
```

上記のコードでは、制御が PERFORM ステートメントに達すると、WS-DATA-IX が 1 に設定され、PERFORM ステートメントが実行されます。次に、条件 (WS-DATA-IX = 12) がテストされます。条件が真である場合には、制御が INSPECT ステートメントに渡ります。偽である場合には、WS-DATA-IX が 1 増加され、PERFORM ステートメントが実行され、条件が再度テストされます。この実行とテストのサイクルは、WS-DATA-IX が 12 になるまで継続されます。

適用の面から見ると、このループは項目 WS-DATA の 12 個のフィールドについての入力検査を制御します。空のフィールドは認められず、このセクションのコードはループし、該当する場合にはエラー・メッセージを発行します。

複数の段落またはセクションの実行

構造化プログラミングでは、実行する段落は通常は 1 つの段落です。しかし、PERFORM . . . THRU ステートメントを使用すると、段落のグループ、1 つのセクション、またはセクションのグループを実行することができます。

PERFORM. . THRU を使用するときには、段落 EXIT ステートメントを使用して一連の段落の終点を明確に示してください。

組み込み関数を使用すると、表の反復処理のタスクがより単純でコーディングしやすいものになります。

関連タスク

80 ページの『組み込み関数を使用したテーブル項目の処理』

第 6 章 ストリングの処理

COBOL には、ストリング・データ項目に関連する以下の操作を実行するための言語構造体が用意されています。

- データ項目の結合と分割
- ヌル終了ストリングの操作 (文字のカウントや移動など)
- 通常の位置 (必要があれば、および長さ) によるサブストリングへの参照
- データ項目の計算および置換 (データ項目内で特定の文字が現れる回数のカウントなど)
- データ項目の変換 (大文字または小文字への変更など)
- データ項目の評価 (データ項目の長さの判別など)

関連タスク
『データ項目の結合 (STRING)』
97 ページの『データ項目の分割 (UNSTRING)』
100 ページの『ヌル終了ストリングの処理』
101 ページの『データ項目のサブストリングの参照』
104 ページの『データ項目の計算および置換 (INSPECT)』
105 ページの『データ項目の変換 (組み込み関数)』
107 ページの『データ項目の評価 (組み込み関数)』
113 ページの『第 7 章 実行時に各国語を使用するためのコーディング』

データ項目の結合 (STRING)

STRING ステートメントは、いくつかのデータ項目の全部または一部を結合して 1 つのデータ項目にするために使用します。1 つの STRING ステートメントを使用することによって、複数の MOVE ステートメントを使用する手間が省けます。

STRING ステートメントは、指定された順序でデータ項目を受け取り項目に転送します。STRING ステートメントでは、以下のものを指定することもできます。

- 1 つの送り出しフィールドを終了させ、別の送り出しフィールドを開始させる区切り文字
- 送り出し文字がすべて処理される前に 1 つの受け取りフィールドがいっぱいになったとき (ON OVERFLOW 条件が発生したとき) に取るべきアクション

任意のリテラルまたは ID オペランドに対して国別項目を指定することができます。ただし、POINTER ID を除きます。ただし、国別項目を指定する場合は、リテラルおよび ID オペランドをすべて (POINTER ID を除く) 国別として指定しなければなりません。

96 ページの『例: STRING ステートメント』

関連タスク
248 ページの『ストリングの結合および分割におけるエラーの処理』

© Copyright IBM Corp. 1991, 2001

関連参照
STRING ステートメント (*Enterprise COBOL 言語解説書*)

例: STRING ステートメント

次の例の STRING ステートメントは、レコード RCD-01 からの情報を出力行として選択し、形式設定します。この出力行は、行番号、顧客の名前と住所、送り状番号、次の請求日、および差し引き不足額によって構成されます。収支は、ドル記号の桁が示すように切り捨てられます。

FILE SECTION では、以下のレコードが定義されます。

```
01  RCD-01.
    05  CUST-INFO.
        10  CUST-NAME       PIC X(15).
        10  CUST-ADDR       PIC X(35).
    05  BILL-INFO.
        10  INV-NO          PIC X(6).
        10  INV-AMT         PIC $$,$$$.99.
        10  AMT-PAID        PIC $$,$$$.99.
        10  DATE-PAID       PIC X(8).
        10  BAL-DUE         PIC $$,$$$.99.
        10  DATE-DUE        PIC X(8).
```

WORKING-STORAGE SECTION では、以下のフィールドが定義されます。

```
77  RPT-LINE               PIC X(120).
77  LINE-POS               PIC S9(3).
77  LINE-NO                PIC 9(5) VALUE 1.
77  DEC-POINT              PIC X VALUE ".".
```

レコード RCD-01 には、以下の情報が含まれています (記号 *b* はブランク・スペースを示します)。

```
J.B.bSMITHbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

PROCEDURE DIVISION では、プログラマーは、RPT-LINE を SPACES に初期化し、LINE-POS (POINTER フィールドとして使用されるデータ項目) を 4 に設定します。(STRING ステートメントの POINTER 句をコーディングすると、明示的なポインター・フィールドを使用して、受け取りフィールドにおけるデータの配置を制御することができます。) そのあと、プログラマーは次の STRING ステートメントをコーディングします。

```
STRING
   LINE-NO SPACE CUST-INFO INV-NO SPACE DATE-DUE SPACE
      DELIMITED BY SIZE
   BAL-DUE
      DELIMITED BY DEC-POINT
   INTO RPT-LINE
   WITH POINTER LINE-POS.
```

STRING プログラムの結果

STRING ステートメントが実行されると、次の表に示されるように、項目は RPT-LINE に移動されます。

項目	位置
LINE-NO	4 - 8
スペース	9
CUST-INFO	10 - 59
INV-NO	60 - 65
スペース	66
DATE-DUE	67 - 74
スペース	75
BAL-DUE の小数点より前の部分	76 - 81

STRING ステートメントの実行後、LINE-POS の値は 82 となり、RPT-LINE は次のようになります。

```
   桁
4      10                                          60      67        76
↓      ↓                                           ↓       ↓         ↓
00001 J.B. SMITH  444 SPRING ST., CHICAGO, ILL  A14275 10/22/76 $2,336
```

データ項目の分割 (UNSTRING)

UNSTRING ステートメントは、1 つの送り出しフィールドを複数の受け取りフィールドに分割するために使用します。1 つの UNSTRING ステートメントを使用することによって、複数の MOVE ステートメントを使用する手間が省けます。

UNSTRING ステートメントでは、以下のものを指定することができます。

- 区切り文字。送り出しフィールド内で区切り文字が検出されると、現行の受け取りフィールドは受け取りを停止し、次の受け取りフィールドが受け取りを開始します。
- 受け取りフィールドに入れられる文字数を保管するフィールド。
- 転送される合計文字数のカウントを保管するフィールド。
- 送り出し項目の終わりに達する前にすべての受け取りフィールドがいっぱいになった場合に取られる、特殊なアクション。

国別項目を送り出しフィールド、受け取りフィールド、および区切り文字として指定できます。ただし、国別項目を指定する場合は、これらのオペランドをすべて国別として指定しなければなりません。

98 ページの『例: UNSTRING ステートメント』

関連概念

115 ページの『Unicode および言語文字のエンコード』

関連タスク

248 ページの『ストリングの結合および分割におけるエラーの処理』

Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド

関連参照
UNSTRING ステートメント (*Enterprise COBOL 言語解説書*)

例: UNSTRING ステートメント

次の例では、入力レコードから情報が選択されます。一部を印刷用に、残りを後続の処理用に編成します。

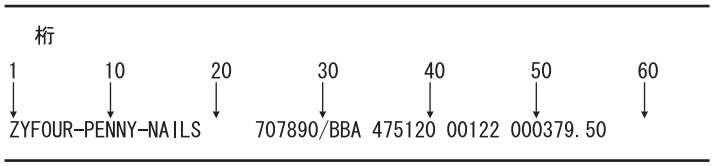
FILE SECTION では、以下のレコードが定義されます。

```
*  Record to be acted on by the UNSTRING statement:
 01  INV-RCD.
     05  CONTROL-CHARS                PIC XX.
     05  ITEM-INDENT                  PIC X(20).
     05  FILLER                       PIC X.
     05  INV-CODE                     PIC X(10).
     05  FILLER                       PIC X.
     05  NO-UNITS                     PIC 9(6).
     05  FILLER                       PIC X.
     05  PRICE-PER-M                  PIC 99999.
     05  FILLER                       PIC X.
     05  RTL-AMT                      PIC 9(6).99.
*
*  UNSTRING receiving field for printed output:
 01  DISPLAY-REC.
     05  INV-NO                       PIC X(6).
     05  FILLER                       PIC X VALUE SPACE.
     05  ITEM-NAME                    PIC X(20).
     05  FILLER                       PIC X VALUE SPACE.
     05  DISPLAY-DOLS                 PIC 9(6).
*
*  UNSTRING receiving field for further processing:
 01  WORK-REC.
     05  M-UNITS                      PIC 9(6).
     05  FIELD-A                      PIC 9(6).
     05  WK-PRICE REDEFINES FIELD-A   PIC 9999V99.
     05  INV-CLASS                    PIC X(3).
*
*  UNSTRING statement control fields
 77  DBY-1                            PIC X.
 77  CTR-1                            PIC S9(3).
 77  CTR-2                            PIC S9(3).
 77  CTR-3                            PIC S9(3).
 77  CTR-4                            PIC S9(3).
 77  DLTR-1                           PIC X.
 77  DLTR-2                           PIC X.
 77  CHAR-CT                          PIC S9(3).
 77  FLDS-FILLED                      PIC S9(3).
```

PROCEDURE DIVISION では、プログラマーは次の UNSTRING ステートメントを書きます。

```
* Move subfields of INV-RCD to the subfields of DISPLAY-REC
* and WORK-REC:
     UNSTRING INV-RCD
       DELIMITED BY ALL SPACES OR "/" OR DBY-1
       INTO ITEM-NAME  COUNT IN CTR-1
          INV-NO  DELIMITER IN DLTR-1  COUNT IN CTR-2
          INV-CLASS
          M-UNITS  COUNT IN CTR-3
          FIELD-A
          DISPLAY-DOLS  DELIMITER IN DLTR-2  COUNT IN CTR-4
       WITH POINTER CHAR-CT
       TALLYING IN FLDS-FILLED
       ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

UNSTRING ステートメントを出す前に、プログラマーは、CHAR-CT(POINTER フィールド) に値 3 を入れて、INV-RCD 内の 2 つの制御文字が処理されることのないようにします。ピリオド (.) が区切り文字として DBY-1 に入れられ、値 0 (ゼロ) が FLDS-FILLED(TALLYING フィールド) に入れられます。そのあと、データが以下のように INV-RCD に読み込まれます。



UNSTRING プログラムの結果

この UNSTRING ステートメントを実行すると、以下のステップで処理が行われます。

1. INV-RCD の桁 3 ～ 18 (FOUR-PENNY-NAILS) が ITEM-NAME に入れられ、区域内で左寄せされ、未使用の 4 つの文字位置にスペースが埋め込まれます。値 16 が CTR-1 に入れられます。
2. ALL SPACES は区切り文字としてコーディングされているため、桁 19 ～ 23 の 5 つの連続した SPACE 文字は区切り文字が 1 回現れただけと見なされます。
3. 桁 24 ～ 29 (707890) が INV-NO に入れられます。区切り文字 / が DLTR-1 に入れられ、値 6 が CTR-2 に入れられます。
4. 桁 31 ～ 33 が INV-CLASS に入れられます。区切り文字は SPACE ですが、区切り文字の受け取り域として定義されたフィールドがないので、桁 34 の SPACE はバイパスされます。
5. 桁 35 ～ 40 (475120) が調べられ、M-UNITS に入れられます。値 6 が CTR-3 に入れられます。区切り文字は SPACE ですが、区切り文字の受け取り域として定義されたフィールドがないので、桁 41 の SPACE はバイパスされます。
6. 桁 42 ～ 46 (00122) が FIELD-A に入れられ、区域内で右寄せされます。高位桁位置には 0 (ゼロ) が埋め込まれます。区切り文字は SPACE ですが、区切り文字の受け取り域として定義されたフィールドがないので、桁 47 の SPACE はバイパスされます。
7. 桁 48 ～ 53 (000379) が DISPLAY-DOLS に入れられます。DBY-1 の中のピリオド (.) 区切り文字が DLTR-2 に入れられ、値 6 が CTR-4 に入れられます。
8. すべての受け取りフィールドが処理され、INV-RCD 内の 2 文字のデータがまだ調べられていないため、ON OVERFLOW 出口が使用され、UNSTRING ステートメントの実行が完了します。

UNSTRING ステートメントの実行後、フィールドには以下の値が入っています。

フィールド	値
DISPLAY-REC	707890 FOUR-PENNY-NAILS 000379
WORK-REC	475120000122BBA
CHAR-CT (POINTER フィールド)	55
FLDS-FILLED (TALLYING フィールド)	6

ヌル終了ストリングの処理

ヌル終了ストリング (たとえば、C プログラムとの間でやりとりされるもの) は、次のようなさまざまなメカニズムによって、構成して操作することができます。

- ヌル終了リテラル定数 (Z". . . ")。
- INSPECT ステートメントを使用して、ヌル終了ストリング内の文字数をカウントする。

```
MOVE 0 TO char-count
INSPECT source-field TALLYING char-count
                     FOR CHARACTERS
                     BEFORE X"00"
```

- UNSTRING ステートメントを使用して、ヌル終了ストリング内の文字をターゲット・フィールドに移動し、文字カウントを得る。

```
WORKING-STORAGE SECTION.
01  source-field          PIC X(1001).
01  char-count     COMP-5 PIC 9(4).
01  target-area.
    02 individual-char OCCURS 1 TO 1000 TIMES DEPENDING ON char-count
                         PIC X.
. . .
PROCEDURE DIVISION.
    . . .
    UNSTRING source-field DELIMITED BY X"00"
                         INTO target-area
                         COUNT IN char-count
      ON OVERFLOW
        DISPLAY "source not null terminated or target too short"
        . . .
    END-UNSTRING
```

- SEARCH ステートメントを使用して、トレーラー・ヌルまたはスペース文字を見つける。検査するストリングを単一文字からなるテーブルとして定義してください。
- ループのフィールドの各文字を検査する (PERFORM)。フィールドの各文字は、source-field (I:1) のような参照修飾子を使用して検査することができます。

『例: ヌル終了ストリング』

関連参照

英数字リテラル (*Enterprise COBOL 言語解説書*)

例: ヌル終了ストリング

以下の例では、ヌル終了ストリングを扱う方法をいくつか示しています。

```
 01  L pic X(20) value z'ab'.
 01  M pic X(20) value z'cd'.
 01  N pic X(20).
 01  N-Length pic 99 value zero.
 01  Y pic X(13) value 'Hello, World!'.
 . . .
* Display null-terminated string
     Inspect N tallying N-length
       for characters before initial x'00'
     Display 'N: ' N(1:N-Length) ' Length: ' N-Length
     . . .
* Move null-terminated string to alphanumeric, strip null
     Unstring N delimited by X'00' into X
     . . .
```

```
* Create null-terminated string
     String Y     delimited by size
            X'00' delimited by size
            into N.
     . . .
* Concatenate two null-terminated strings to produce another
     String L     delimited by x'00'
            M     delimited by x'00'
            X'00' delimited by size
            into N.
```

データ項目のサブストリングの参照

文字ストリング・データ項目 (英数字、DBCS、および国別データ項目を含む) のサブストリングは、参照修飾子を使って参照します。文字ストリング値を戻す組み込み関数は、参照修飾子を含めることができます。

以下の例では、参照修飾子を使用してデータ項目のサブストリングを参照する方法を示します。

```
Move Customer-Record(1:20) to Orig-Customer-Name
```

この例で示すように、データ項目の直後に、次の 2 つの値をコロンで区切って括弧で囲んで指定します。

- サブストリングを開始したい文字の序数位置 (左からの)
- 必要なサブストリングの長さ

長さは任意指定です。長さを省略すると、サブストリングは項目の終わりまでになります。可能であれば、より単純でエラーになりにくいコーディング手法として、長さを省略してください。

2 つの値のいずれかを、変数または算術式としてコーディングすることができます。

算術式が許可されるところであればどこでも数字関数 ID を使用できるので、参照修飾子の中で数字関数 ID を左端文字位置または長さとして使用することができます。

可変長記入項目を含め、テーブル記入項目のサブストリングを参照することもできます。テーブル記入項目のサブストリングを参照するには、参照修飾子の前にサブスクリプト式をコーディングする必要があります。たとえば、PRODUCT-TABLE が適切にコーディングされた文字ストリングのテーブルであると仮定すると、D をテーブル内の 2 番目のストリングの 4 文字目に移動するために、次のステートメントをコーディングできます。

```
MOVE 'D' to PRODUCT-TABLE (2), (4:1)
```

参照修飾子内の両方の数値は少なくとも 1 でなければなりません。また、必要なサブストリングの終わりを超えて参照しないように、それらの合計がデータ項目の全長を 1 より多く超えてはなりません。

左端の文字位置または長さ値が固定小数点の非整数の場合には、整数を作成するために切り捨てが行われます。浮動小数点の非整数の場合には、整数を作成するための丸めが行われます。

以下のオプションを使用すると、範囲外の参照修飾子が検出され、実行時メッセージによって違反が示されます。

- SSRANGE コンパイラー・オプション
- CHECK ランタイム・オプション

国別データ項目を参照変更できます。国別項目の参照修飾子の位置および長さは、国別文字の単位で表されます。

関連概念
『参照修飾子』
115 ページの『Unicode および言語文字のエンコード』

関連タスク
68 ページの『テーブルでの項目の参照』

関連参照
348 ページの『SSRANGE』
参照変更 (*Enterprise COBOL 言語解説書*)
関数定義 (*Enterprise COBOL 言語解説書*)

参照修飾子

システムから現在の時刻を取り出して、その値を拡張形式で表示したいとします。現在の時刻は、ACCEPT ステートメントを使用して取り出すことができます。このステートメントは、次の形式で、時、分、秒、および 100 分の 1 秒を戻します。

```
HHMMSSss
```

しかし、現在の時刻を次の形式で表示したいとします。

```
HH:MM:SS
```

参照修飾子を使用しない場合は、両方の形式についてのデータ項目を定義しなければなりません。さらに、1 つの形式を別の形式に変換するためのコードも書かなければなりません。

参照修飾子を使用する場合は、TIME エレメントを記述するサブフィールドに名前を指定する必要はありません。必要なデータ定義は、システムによって戻される時刻用のデータ定義だけです。たとえば、次のように記述します。

```
01  REFMOD-TIME-ITEM            PIC X(8).
```

次のコードは、時刻値を取り出して、拡張します。

```
     ACCEPT REFMOD-TIME-ITEM FROM TIME.
     DISPLAY "CURRENT TIME IS: "
* Retrieve the portion of the time value that corresponds to
*   the number of hours:
       REFMOD-TIME-ITEM (1:2)
       ":"
* Retrieve the portion of the time value that corresponds to
*   the number of minutes:
       REFMOD-TIME-ITEM (3:2)
       ":"
* Retrieve the portion of the time value that corresponds to
*   the number of seconds:
       REFMOD-TIME-ITEM (5:2)
```

『例: 参照修飾子としての演算式』
『例: 参照修飾子としての組み込み関数』

関連タスク
101 ページの『データ項目のサブストリングの参照』
116 ページの『COBOL での国別データ (Unicode) の使用』

関連参照
参照変更 (*Enterprise COBOL 言語解説書*)

例: 参照修飾子としての演算式

あるフィールドに右寄せされたいくつかの文字が入っている場合に、文字を別のフィールドに移動し、右ではなく左に寄せたいとします。これは、参照修飾子と INSPECT ステートメントを使用して行うことができます。

プログラムに次のデータが入っているとします。

```
01  LEFTY          PIC X(30).
01  RIGHTY         PIC X(30) JUSTIFIED RIGHT.
01  I               PIC 9(9)  USAGE BINARY.
```

プログラムは、先行するスペースの数をカウントし、参照修飾子内の算術式を使用して、右寄せされた文字を別のフィールドに移動し、左寄せします。

```
MOVE SPACES TO LEFTY
MOVE ZERO TO I
INSPECT RIGHTY
   TALLYING I FOR LEADING SPACE.
IF I IS LESS THAN LENGTH OF RIGHTY THEN
   MOVE RIGHTY ( I + 1 : LENGTH OF RIGHTY - I ) TO LEFTY
END-IF
```

MOVE ステートメントは、RIGHTY の文字を、I + 1 で計算された位置から、LENGTH OF RIGHTY - I で計算された長さだけ、フィールド LEFTY に移動します。

例: 参照修飾子としての組み込み関数

次のコード断片が実行されると、Customer-Record のサブストリングが、変数 WS-name に移動されます。サブストリングは、実行時に決定されます。

```
05  WS-name           Pic x(20).
05  Left-posn         Pic 99.
05  I                 Pic 99.
. . .
Move Customer-Record(Function Min(Left-posn I):Function Length(WS-name)) to WS-name
```

整数関数が必要とされる場所で非整数機能を使用したい場合には、INTEGER または INTEGER-PART 関数を使用して、結果を整数に変換することができます。たとえば、次のように記述します。

```
Move Customer-Record(Function Integer(Function Sqrt(I)): ) to WS-name
```

関連参照
INTEGER-PART (*Enterprise COBOL 言語解説書*)
INTEGER (*Enterprise COBOL 言語解説書*)

データ項目の計算および置換 (INSPECT)

INSPECT ステートメントを使用して、以下のことを行うことができます。

- データ項目の選択された部分をある値で埋める。国別データ項目を指定する場合は、国別値も指定する必要があります。
- データ項目の部分を別のデータ項目の対応する部分で置き換える。国別項目またはリテラルをステートメントに指定する場合は、すべての項目を国別として指定する必要があります。
- データ項目内で特定の文字 (0、スペース、アスタリスクなど) が現れる回数をカウントする。国別データ項目の場合、カウントは国別文字の文字数です。

『例: INSPECT ステートメント』

関連概念

115 ページの『Unicode および言語文字のエンコード』

関連参照

INSPECT ステートメント (*Enterprise COBOL 言語解説書*)

例: INSPECT ステートメント

次の例では、INSPECT ステートメントの使用法をいくつか示します。

次の例では、INSPECT ステートメントを使用して、データ項目 DATA-2 内の文字を調べ、置換します。データ項目内で先行 0 が現れる回数が、COUNTR に累算されます。文字 C の最初のインスタンスの後の文字 A のインスタンスはすべて、文字 2 で置き換えられます。

```
77  COUNTR                PIC 9   VALUE ZERO.
01  DATA-2                PIC X(11).
. . .
    INSPECT DATA-2
      TALLYING COUNTR FOR LEADING "0"
      REPLACING FIRST "A" BY "2" AFTER INITIAL "C"
```

DATA-2 (実行前)	COUNTR (実行後)	DATA-2 (実行後)
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

次の例では、INSPECT ステートメントを使用して、データ項目 DATA-3 内の文字を調べ、置換します。データ項目内の引用符 (") の最初のインスタンスより前にある文字はすべて、文字 0 で置き換えられます。

```
77  COUNTR                PIC 9   VALUE ZERO.
01  DATA-3                PIC X(8).
. . .
    INSPECT DATA-3
      REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE
```

DATA-3 (実行前)	COUNTR (実行後)	DATA-3 (実行後)
456”ABEL	0	000“ABEL
ANDES”12	0	00000“12

DATA-3 (実行前)	COUNTR (実行後)	DATA-3 (実行後)
”TWAS BR	0	“TWAS BR

次の例では、AFTER 句と BEFORE 句を指定した INSPECT CONVERTING を使用して、データ項目 DATA-4 内の文字を調べ、置換します。データ項目内で文字 / の最初のインスタンスより後にあり、文字 ? (もしあれば) の最初のインスタンスより前にあるすべての文字が、小文字から大文字に変換されます。

```
01  DATA-4                PIC X(11).
. . .
    INSPECT DATA-4
      CONVERTING
         "abcdefghijklmnopqrstuvwxyz" TO
         "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      AFTER INITIAL "/"
      BEFORE INITIAL"?"
```

DATA-4 (実行前)	DATA-4 (実行後)
a/five/?six	a/FIVE/?six
r/Rexx/RRRr	r/REXX/RRRR
zfour?inspe	zfour?inspe

データ項目の変換 (組み込み関数)

組み込み関数を使用して、文字ストリング・データ項目を以下のものに変換することができます。

- 大文字または小文字
- 逆順
- 数値

INSPECT ステートメントを使用して文字を変換することもできます。

104 ページの『例: INSPECT ステートメント』

NATIONAL-OF および DISPLAY-OF 組み込み関数を使用して、国別 (Unicode) ストリングとの間で変換を行うことができます。

関連概念

115 ページの『Unicode および言語文字のエンコード』

大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)

```
01  Item-1             Pic x(30)  Value "Hello World!".
01  Item-2             Pic x(30).
. . .
    Display Item-1
    Display Function Upper-case(Item-1)
    Display Function Lower-case(Item-1)
    Move Function Upper-case(Item-1) to Item-2
    Display Item-2
```

上記のコードは、次のメッセージをシステムの論理出力装置に表示します。

```
Hello World!
HELLO WORLD!
hello world!
HELLO WORLD!
```

DISPLAY ステートメントは、Item-1 の実際の内容は変更せず、文字の表示方法にのみ影響を与えます。しかし、MOVE ステートメントを使用すると、Item-2 の実際の内容に大文字が移動されます。

逆順への変換 (REVERSE)

次のコードは、Orig-cust-name の中の文字の順序を逆にします。

```
Move Function Reverse(Orig-cust-name) To Orig-cust-name
```

たとえば、開始値が JOHNSON*bbb*であれば、ステートメントの実行後の値は *bbb*NOSNHOJになります (*b* はブランク・スペースを表します)。

国別ストリングの順序を逆にする場合、結果も国別ストリングになります。

数値への変換 (NUMVAL、NUMVAL-C)

NUMVAL および NUMVAL-C 関数は、文字ストリングを数値に変換します。これらの関数を使用すると、自由形式の文字表現数値を含む英数字データ項目を数値形式に変換し、それらを数値として処理することができます。以下にその例を示します。

```
01  R            Pic x(20)  Value "- 1234.5678".
01  S            Pic x(20)  Value "  $12,345.67CR".
01  Total        Usage is Comp-1.
. . .
    Compute Total = Function Numval(R) + Function Numval-C(S)
```

NUMVAL-C は、例で示されているように、引き数に通貨記号またはコンマ (またはその両方) が含まれているときに使用します。文字ストリングの前または後ろに代数符号を入れることができ、その符号が処理されます。引き数は、デフォルト・オプション ARITH(COMPAT) (*互換モード*) でコンパイルするときは 18 桁を超えてはならず、ARITH(EXTEND) (*拡張モード*) でコンパイルするときは 31 桁を超えてはなりません (桁数には編集記号は含みません)。

NUMVAL および NUMVAL-C は、互換モードでは長精度 (64 ビット) 浮動小数点値を戻し、拡張モードでは拡張精度 (128 ビット) 浮動小数点値を戻します。したがって、これらの関数への参照は、数値データ項目への参照を表します。

NUMVAL または NUMVAL-C を使用する場合は、数値データを固定形式で静的に宣言したり、入力データを正確な方法で入力したりする必要はありません。たとえば、次のように入力される数値を定義するとします。

```
01  X            Pic S999V99  leading sign is separate.
. . .
    Accept X from Console
```

アプリケーションのユーザーは、PICTURE 文節で定義されているとおりに正確に数値を入力しなければなりません。たとえば、次のようになります。

```
+001.23
-300.00
```

しかし、NUMVAL 関数を使用する場合は、次のようにコーディングすることができます。

```
01  A            Pic x(10).
01  B            Pic S999V99.
. . .
    Accept A from Console
    Compute B = Function Numval(A)
```

入力は次のようにすることができます。

```
1.23
-300
```

関連概念

46 ページの『数値データの形式』

関連タスク

33 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
118 ページの『国別データの変換』

関連参照

NUMVAL (*Enterprise COBOL 言語解説書*)
NUMVAL-C (*Enterprise COBOL 言語解説書*)
316 ページの『ARITH』

データ項目の評価 (組み込み関数)

データ項目を評価する際には、いくつかの組み込み関数を使用することができます。

- CHAR および ORD は、プログラムで使用される照合シーケンスに関して整数および単一の英数字を評価するためのものです。
- MAX、MIN、ORD-MAX、および ORD-MIN は、一連のデータ項目 (国別データ項目を含む) の中で最大および最小の項目を調べるためのものです。
- LENGTH は、データ項目の長さ (国別データ項目を含む) を調べるためのものです。
- WHEN-COMPILED は、プログラムがコンパイルされた日時を調べるためのものです。

関連概念

115 ページの『Unicode および言語文字のエンコード』

関連タスク

『照合シーケンスに関する単一文字の評価』
108 ページの『最大または最小データ項目の検出』
110 ページの『データ項目の長さの検出』
110 ページの『コンパイルの日付の検出』

照合シーケンスに関する単一文字の評価

特定の文字の照合シーケンスにおける序数位置を調べたい場合は、ORD 関数でその文字を引き数として使用することができます。ORD は、その序数位置を表す整数を

戻します。ある文字の序数位置を調べる便利な方法の 1 つとして、データ項目の 1 文字のサブストリングを ORD への引き数として使用する方法があります。

```
IF Function Ord(Customer-record(1:1)) IS > 194 THEN . . .
```

ある文字の照合シーケンスにおける序数位置がわかっていて、それに対応する文字が知りたい場合には、CHAR でその整数の序数位置を引き数として使用することができます。CHAR は、要求された文字を戻します。

```
INITIALIZE Customer-Name REPLACING ALPHABETIC BY Function Char(65)
```

関連参照
CHAR *(Enterprise COBOL 言語解説書)*
ORD *(Enterprise COBOL 言語解説書)*

最大または最小データ項目の検出

2 つ以上の英数字データ項目のうちのどれが最大値を持つかを知りたい場合には、MAX または ORD-MAX 関数を使用します。問題となっているデータ項目を引き数として与えてください。どの項目に最小値が含まれているのか知りたい場合には、MIN または ORD-MIN を使用します。これらの関数は、照合シーケンスに従って値を評価します。また、MAX、ORD-MAX、MIN、または ORD-MIN を数値に対して使用することができます。その場合は、引き数の代数値が比較されます。

MAX および MIN

MAX および MIN 関数は、指定された変数の 1 つの内容を戻します。

たとえば、次のようなデータ定義があるとします。

```
05  Arg1         Pic x(10)  Value "THOMASSON ".
05  Arg2         Pic x(10)  Value "THOMAS    ".
05  Arg3         Pic x(10)  Value "VALLEJO   ".
```

次のステートメントは、VALLEJO*bbb*を Customer-record の最初の 10 文字位置に割り当てます (ここで *b* はブランク・スペースを表します)。

```
Move Function Max(Arg1 Arg2 Arg3) To Customer-record(1:10)
```

代わりに MIN を使用すると、THOMAS*bbbb* が割り当てられます。

引き数に国別項目を指定する場合は、すべての引き数を国別として指定しなければなりません。

ORD-MAX および ORD-MIN

ORD-MAX および ORD-MIN 関数は、指定された引き数のリストの中で最大値または最小値を持つ引き数の (左からカウントした) 序数位置を表す整数を戻します。

上記の例で ORD-MAX 関数が使用された場合は、無効な場所で数字関数の参照を試みていることになるため、コンパイル時に構文エラー・メッセージを受け取ります。以下に、ORD-MAX 関数の有効な使用例を示します。

```
Compute x = Function Ord-max(Arg1 Arg2 Arg3)
```

このコードは、上記の例と同じ引き数が使用された場合、整数 3 を x に割り当てます。代わりに ORD-MIN を使用した場合には、整数 2 が戻されます。もし Arg1、Arg2、および Arg3 が配列 (テーブル) の連続するエレメントであった場合は、上記の例はより現実的なものになります。

引き数に国別項目を指定する場合は、すべての引き数を国別として指定しなければなりません。

英数字関数で可変長の結果を戻す場合

英数字関数の結果は、関数引き数によって、長さと値が変わる場合があります。

次の例では、R3 に移動されるデータの量および COMPUTE ステートメントの結果は、R1 および R2 の値とサイズによって異なります。

```
01  R1           Pic x(10) value "e".
01  R2           Pic x(05) value "f".
01  R3           Pic x(20) value spaces.
01  L            Pic 99.
. . .
    Move Function Max(R1 R2) to R3
    Compute L = Function Length(Function Max(R1 R2))
```

ここで、R2 は R1 より大きいものと評価されます。したがって、次のようになります。

- ストリング 'f*bbbb*' が R3 に移動されます (ここで *b* はブランク・スペースを表します)。(R3 内の残りの文字位置にはスペースが埋められます。)
- L は値 5 と評価されます。

R1 に 'e' ではなく 'g' が含まれていた場合には、R1 は R2 より大きいものと評価され、次のようになります。

- ストリング 'g*bbbbbbbbb*' が R3 に移動されます。(R3 内の残りの文字位置にはスペースが埋められます。)
- 値 10 が L に割り当てられます。

英数字関数からの可変長出力を処理することになる場合もあります。それに応じてプログラムを設計しなければなりません。たとえば、書き込むレコードによって長さが異なる可能性があるときには、可変長レコード・ファイルの使用を考えることが必要になります。

```
File Section.
FD  Output-File Recording Mode V.
01  Short-Customer-Record   Pic X(50).
01  Long-Customer-Record    Pic X(70).
. . .
Working-Storage Section.
01  R1               Pic x(50).
01  R2               Pic x(70).
. . .
    If R1 > R2
        Write Short-Customer-Record from R1
    Else
        Write Long-Customer-Record from R2
    End-if
```

関連タスク
54 ページの『算術の実行』
80 ページの『組み込み関数を使用したテーブル項目の処理』

関連参照
ORD-MAX (*Enterprise COBOL 言語解説書*)
ORD-MIN (*Enterprise COBOL 言語解説書*)

データ項目の長さの検出

LENGTH 関数をさまざまなコンテキスト (数値データおよびテーブルを含む) で使用して、ストリング項目の長さを判別することができます。

次の COBOL ステートメントは、データ項目を、顧客名を入れるレコードのフィールドに移動する例を示しています。

```
Move Customer-name To Customer-record(1:Function Length(Customer-name))
```

LENGTH 関数は、国別項目の長さを国別文字単位で戻します。

LENGTH OF 特殊レジスターを使用することもできます。この特殊レジスターは国別データの場合でも、長さをバイト単位で戻します。 Function Length(Customer-name) または LENGTH OF Customer-name のどちらをコーディングしても、英数字項目の場合は同じ結果 (Customer-name のバイト単位の長さ) が戻されます。

LENGTH 関数は、算術式が許可される場所でしか使用できません。しかし、LENGTH OF 特殊レジスターはさまざまなコンテキストで使用することができます。たとえば、整数引き数が認められる組み込み関数への引き数として LENGTH OF 特殊レジスターを使用することができます。(組み込み関数を LENGTH OF 特殊レジスターに対するオペランドとして使用することはできません。) LENGTH OF 特殊レジスターは、CALL ステートメントのパラメーターとして使用することもできます。

関連タスク
54 ページの『算術の実行』
80 ページの『組み込み関数を使用したテーブル項目の処理』

関連参照
LENGTH (*Enterprise COBOL 言語解説書*)

コンパイルの日付の検出

プログラムがコンパイルされた日時を知りたい場合は、WHEN-COMPILED 関数を使用することができます。戻される結果には 21 個の文字位置があり、最初の 16 個の文字位置は次の形式になります。

```
YYYYMMDDhhmmsshh
```

これは、コンパイルの 4 桁の年、月、日、および時刻 (時、分、秒、および 100 分の 1 秒) を示します。

WHEN-COMPILED 特殊レジスターは、コンパイルの日時を調べるために使用できるもう 1 つの方法です。形式は次のとおりです。

MM/DD/YYhh.mm.ss

WHEN-COMPILED 特殊レジスターは、2 桁の年のみをサポートし、時刻を秒までしか扱いません。この特殊レジスターは、MOVE ステートメントの送り出しフィールドとしてのみ使用できます。

関連参照
WHEN-COMPILED (*Enterprise COBOL 言語解説書*)

第 7 章 実行時に各国語を使用するためのコーディング

Enterprise COBOL は、実行時に Unicode をサポートします。Unicode は、プレーン・テキストをエンコードするための一貫性のある効率的な方法を提供します。Unicode を使用すると、さまざまな国の言語で動作するソフトウェアを開発できます。

以下の COBOL 機能を使用して、実行時に各国語を使用するためのプログラムをコーディングおよびコンパイルします。

- 国別データ型および国別リテラル
- 2 つの組み込み関数:
  - NATIONAL-OF を使用すると、文字ストリングを UTF-16 (16 ビット UCS transformation format) 表現に戻すことができます。
  - DISPLAY-OF を使用すると、国別ストリングを、選択したコード・ページ (EBCDIC、ASCII、EUC、または UTF-8) に変換できます。
- 2 つのコンパイラー・オプション:
  - CODEPAGE を使用すると、プログラムで英数字および 2 バイト文字セット (DBCS) データに使用するコード・ページを指定できます。
  - NSYMBOL は、リテラル内の N 記号および PICTURE 文節に対して国別処理と DBCS 処理のどちらを使用するかを制御します。

さらに、一定のコーディング状況においては、国別データを扱う際に特別な注意が必要です (ストリングの処理、データのソートおよびマージ、受け渡しなどの場合)。

Enterprise COBOL ステートメント	国別にできるもの	コメント	詳細の参照先
ACCEPT	identifier-1	CONSOLE オプションが直接または間接に指定された場合のみ、EBCDIC から変換されます。	33 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
CALL	identifier-2、identifier-3、identifier-4、identifier-5。literal-2、literal-3		455 ページの『データの受け渡し』
COPY . . . REPLACING	operand-1、operand-2		360 ページの『コンパイラー指示ステートメント』
DISPLAY	identifier-1	CONSOLE オプションが直接または間接に指定された場合のみ、EBCDIC に変換されます。	34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

© Copyright IBM Corp. 1991, 2001

Enterprise COBOL ステートメント	国別にできるもの	コメント	詳細の参照先
INITIALIZE	REPLACING	REPLACING NATIONAL を指定した場合は、identifier-2 および literal-1 はクラス national に属する必要があります。その逆の場合も同様です。	30 ページの『例: 変数の初期化』
INSPECT	identifier-3 (TALLYING ID) 以外の ID およびリテラル	これらのいずれかがクラス national に属する場合は、すべてをクラス national に含める必要があります。	104 ページの『データ項目の計算および置換 (INSPECT)』
INVOKE	identifier-2 または literal-1 としてのメソッド名。BY VALUE 句の identifier-3 または literal-2。		514 ページの『メソッドの呼び出し (INVOKE)』
MERGE	マージ・キー	COLLATING SEQUENCE 句は適用されません。	212 ページの『ソートまたはマージ基準の設定』
MOVE	送り出し側と受け取り側の両方	送り出し側が国別データ項目である場合は、受け取り側も国別データ項目である必要があります。	32 ページの『変数または構造への値の割り当て (MOVE)』
形式 2 (バイナリー) SEARCH	キー・データ項目および検索引き数の両方	検索引き数は、対応キーが国別項目である場合は英数字項目または国別項目のどちらにすることもできます。検索引き数が国別項目である場合は、対応キーは国別データ項目でなければなりません。	79 ページの『二分探索 (SEARCH ALL)』
SORT	ソート・キー	COLLATING SEQUENCE 句は適用されません。	212 ページの『ソートまたはマージ基準の設定』
STRING	identifier-4 (POINTER ID) 以外の任意の ID およびリテラル	これらのいずれかがクラス national に属する場合は、すべてをクラス national に含める必要があります。	95 ページの『データ項目の結合 (STRING)』
UNSTRING	区切り文字 (identifier-2、identifier-3)、データ受け取りフィールド (identifier-4)、区切り文字受け取りフィールド (identifier-5。literal-3、literal-4)	これらのいずれかがクラス national に属する場合は、すべてをクラス national に含める必要があります。	97 ページの『データ項目の分割 (UNSTRING)』

国別データの使用は、以下の組み込み関数に影響を与えます。

組み込み関数	引き数の型を国別にすることが可能か?	関数の型	詳細の参照先
LENGTH	可能	整数	110 ページの『データ項目の長さの検出』
LOWER-CASE、UPPER-CASE	可能	国別	105 ページの『大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)』
MAX, MIN	可能	国別	最大または最小データ項目の検出 (108 ページの『MAX および MIN』)
ORD-MAX、ORD-MIN	可能	整数	最大または最小データ項目の検出 (108 ページの『ORD-MAX および ORD-MIN』)
REVERSE	可能	国別	106 ページの『逆順への変換 (REVERSE)』

関連概念
『Unicode および言語文字のエンコード』

関連タスク
116 ページの『COBOL での国別データ (Unicode) の使用』
118 ページの『国別データの変換』
120 ページの『UTF-8 データの処理』
121 ページの『中国語 GB 18030 データの処理』
121 ページの『国別データ項目の比較』
123 ページの『DBCS データを含む英数字データ項目の処理』
637 ページの『付録 C. 2 バイト文字セット (DBCS) データの変換』

関連参照
318 ページの『CODEPAGE』
335 ページの『NSYMBOL』

Unicode および言語文字のエンコード

Enterprise COBOL では、Unicode に対する基本的な実行時サポートが提供されます。ASCII でサポートされる 256 文字と比較して、Unicode では 65,536 文字の組み合わせを処理できます。

それぞれの文字セット は、あらかじめ定義された文字の集まりですが、コード化表現とは関連付けられていません。*コード化文字セット* (本書ではこれまでコード・ページ とも表記していました) は、文字セットの文字をコード化表現に関連付ける明確な規則の集まりです。それぞれのコード・ページは、文字セットを表すための記号を設定するテーブルのようなものです。それぞれの記号は、固有のビット・パターン、またはコード・ポイント です。それぞれのコード・ページには、固有の *Coded Character Set Identifier (CCSID)* があります。これは 16 ビット数値 (5 桁の 10 進数) です。

Unicode には、Unicode Transformation Format (UTF) と呼ばれるいくつかのエンコード・スキーム (UTF-8、UTF-16、および UTF-32 など) があります。Enterprise COBOL では、国別データ型の表現として、ビッグ・エンディアン形式で UTF-16、CCSID 1200 を使用します。

UTF-8 は、ASCII のインバリアント文字 a ～ z、A ～ Z、0 ～ 9、および一定の特殊文字 (’ @ , . +- = / * ( ) など) を ASCII の場合と同様に表します。UTF-16 は、同じ文字を X’00nn’ (ここで X’nn’ は ASCII での文字表現) と表します。

たとえば、UTF-16 では、ストリング ’ABC’ は NX’004100420043’ となります。UTF-8 では、X’414243’ となります。

1 つまたは複数のエンコード・ユニット を使用して、コード化文字セットで文字を表します。UTF-16 の場合、エンコード・ユニットは 2 バイトのストレージを使用します。任意の EBCDIC コード・ページで定義された文字はいずれも、国別データ表現に変換されたときに 1 つの UTF-16 エンコード・ユニットで表現されます。

関連タスク
113 ページの『第 7 章 実行時に各国語を使用するためのコーディング』

COBOL での国別データ (Unicode) の使用

Enterprise COBOL では、以下の方法で Unicode データを指定できます。

- 国別データ項目
- 国別リテラル
- 国別文字としての形象定数

これらの宣言は、ストレージ必要量に影響を与えます。

国別データ項目

USAGE NATIONAL を使用して Unicode ストリングを保持するようにデータ項目を定義します。1 つまたは複数の PICTURE 記号 N のみから成る PICTURE 文節をコーディングします。

1 つまたは複数の PICTURE 記号 N のみから成るデータ項目に PICTURE 文節を指定するが、USAGE 文節を指定しない場合は、コンパイラー・オプション NSYMBOL(NATIONAL) を使用して、その項目が確実に国別データ項目として扱われるようにする (DBCS 項目として扱うのではなく) ことができます。

国別リテラル

国別リテラルを指定するには、接頭部文字 NX を使用し、オプション NSYMBOL(NATIONAL) を指定してコンパイルします。たとえば、以下のいずれかの表記を使用することができます。

```
N"character-data"
N'character-data'
```

NSYMBOL(DBCS) を使用するときは、リテラルの接頭部文字 N は国別リテラルではなく DBCS リテラルを指定します。

国別リテラルを 16 進値として指定するには、接頭部文字 NX を使用します。以下にその例を示します。

```
NX"hexadecimal-digits"
```

次の各 MOVE ステートメントは、データ項目 X を Unicode 値 AB に設定します。

```
01 X pic NN usage national.
. . .
    Move NX"00410042" to X
    Move N"AB" to X
    Move "AB" to X
```

国別リテラルを必要とするコンテキストで英数字 16 進数リテラルを使用しないでください。そのような使用法は誤解を招きやすくなります。たとえば、次のステートメントの場合です。

```
Move X"C1C2C3C4" to X
```

Unicode AB (ビット・パターン C1C2C3C4 ではなく) を X に移動する結果になります。

形象定数としての国別文字

国別リテラルに形象定数 ALL *literal* を使用することができます。この形象定数は、リテラルを構成するエンコード・ユニットの逐次連結によって生成されるストリングの全体または一部を表します。

形象定数 QUOTE、SPACE、または ZERO を国別文字が要求されるコンテキストで使用するときは、形象定数は国別文字の値を表します。ただし、形象定数 HIGH-VALUE および LOW-VALUE を国別文字が要求されるコンテキスト (MOVE ステートメント、暗黙の MOVE、または国別オペランドを伴う比較条件など) で使用することはできません。

国別リテラルを SPECIAL-NAMES 段落で使用することはできません。

国別データのストレージ

以下のテーブルを使用して英数字 (DISPLAY)、DBCS (DISPLAY-1)、および Unicode (NATIONAL) のエンコード方式を比較し、ストレージの使用法について計画を立ててください。

特性	DISPLAY	DISPLAY-1	NATIONAL
文字エンコード・ユニット	1 バイト	2 バイト	2 バイト
コード・ページ[1]	EBCDIC	EBCDIC	UTF-16
文字ごとのエンコード・ユニット	1	1	1 または 2[2]
文字ごとのバイト数	1 バイト	2 バイト	2 または 4 バイト

1. CODEPAGE コンパイラー・オプションを使用して、英数字データまたは DBCS データに適した EBCDIC コード・ページ用のコード・ページを指定します。ソース・プログラム内の国別リテラルは、実行時に使用できるように UTF-16 に変換されます。
2. 大部分の文字は 1 つのエンコード・ユニットを使用して UTF-16 で表現されます。特に、次の例は文字ごとに単一の UTF-16 エンコード・ユニットを使用して表現されます。
   - COBOL 文字 A ～ Z、a ～ z、0 ～ 9、スペース文字 (SP)、+ -*/= $,;.“()><:’
   - EBCDIC コード・ページから変換されるすべての文字

関連概念
115 ページの『Unicode および言語文字のエンコード』

関連タスク
『国別データの変換』
121 ページの『国別データ項目の比較』
120 ページの『UTF-8 データの処理』

関連参照
318 ページの『CODEPAGE』
335 ページの『NSYMBOL』

国別データの変換

英数字、DBCS、および整数データ項目を国別ストリングに変換するのに、MOVE ステートメントを使用することができます。(他の COBOL ステートメントでもこのような暗黙の変換が行われます。たとえば、英数字データ項目と国別データ項目の比較などがあります。)また、国別 (Unicode) ストリングからの変換、および国別ストリングへの変換を行うには、それぞれ組み込み関数 NATIONAL-OF および DISPLAY-OF を使用します。組み込み関数を使用すると、CODEPAGE コンパイラー・オプションで有効なコード・ページとは異なる変換用のコード・ページを指定することができます。

英数字および整数の国別データへの変換 (MOVE)

MOVE ステートメントを使用して、英数字、DBCS、および整数項目を国別項目に変換することができます。英数字項目を国別項目に移動すると、英数字項目は国別データの受け取り側の表現 (UTF-16 Unicode) に変換されます。英数字編集項目、数値整数、または数字編集項目を国別項目に移動すると、埋め込みや切り捨ての必要がないサイズの英数字データ項目に最初に移動されたものとして処理されます。

変換されたデータの文字数が指定した国別データ項目のサイズより小さい場合は、そのデータにデフォルトの Unicode UTF-16 スペース文字 (NX'0020') が埋め込まれます。変換されたデータの文字数が指定した国別データ項目のサイズより大きい場合は、末尾の文字が切り捨てられます。

英数字データの国別データへの変換 (NATIONAL-OF)

NATIONAL-OF 組み込み関数を使用して、英字、英数字、または DBCS 項目を Unicode (UTF-16) 表現の文字ストリングに変換することができます。ソースが CODEPAGE コンパイラー・オプションで有効なコード・ページとは異なるコード・ページでエンコードされている場合は、ソース・コード・ページを引き数として指定します。

国別データの英数字データへの変換 (DISPLAY-OF)

DISPLAY-OF 組み込み関数を使用して、指定したコード・ページで表現された文字ストリングに国別項目を変換します。コード・ページは引き数として指定するか、または CODEPAGE コンパイラー・オプションで指定します。SBCS 文字と DBCS 文字を組み合わせた EBCDIC コード・ページを指定する場合は、戻されるストリングに

SBCS 文字と DBCS 文字が混在し、シフトインおよびシフトアウト文字によって区切られた DBCS サブストリングが含まれる可能性があります。

デフォルト・コード・ページのオーバーライド

場合によっては、CODEPAGE オプション値として指定された CCSID とは異なる CCSID との間で変換を行う必要が生じます。これを実行するには、項目のコード・ページを明示的に指定した変換関数を使用して項目を変換します。

コード・ページを引き数として DISPLAY-OF に指定し、そのコード・ページが CODEPAGE コンパイラー・オプションで指定したコード・ページと異なる場合は、暗黙の変換 (国別データ項目への割り当て、国別データ項目との比較など) を伴う何らかの操作が生じる DISPLAY-OF 関数を使用しないでください。このような操作は、CODEPAGE コンパイラー・オプションで指定された EBCDIC コード・ページが使用されることを想定しています。

変換例外

国別データと英数字データ間での暗黙的または明示的な変換が失敗して、重大度 3 の言語環境プログラム条件が生成される場合があります。失敗は、以下のいずれかの場合に発生します。

- Unicode 変換サービスがシステムにインストールされていない場合。
- (暗黙的または明示的に) 指定したコード・ページが有効なコード・ページではない場合。
- 暗黙的または明示的に (たとえば、CODEPAGE コンパイラー・オプションを使用して) 指定した CCSID と UTF-16 Unicode CCSID (01200) の組み合わせが、システム上の Unicode 変換サービスに対する有効な変換ペアとしてシステム上に構成されていない場合。

ターゲットの CCSID に対になる文字が存在しない文字については、変換例外は発生しません。このような文字はターゲット・コード・ページの置換文字に変換されます。

『例: 国別データの変換』

関連参照

318 ページの『CODEPAGE』

例: 国別データの変換

次の例は、NATIONAL-OF 組み込み関数と DISPLAY-OF 組み込み関数、および Unicode ストリングとの間で変換を行うための MOVE ステートメントの使用法を示しています。また、同一プログラム内の複数のコード・ページでエンコードされたストリングに対して操作を行うときの明示的変換の必要性も示しています。

```
 CBL CODEPAGE(00037)
* . . .
 01  Data-in-Unicode pic N(100) usage national.
 01  Data-in-Greek pic X(100).
 01  other-data-in-US-English pic X(12) value "PRICE in $ =".
* . . .
     Read Greek-file into Data-in-Greek
     Move function National-of(Data-in-Greek, 00875)
         to Data-in-Unicode
```

```
* . . . process Data-in-Unicode here . . .
     Move function Display-of(Data-in-Unicode, 00875)
         to Data-in-Greek
     Write Greek-record from Data-in-Greek
```

上記のコード例は正常に機能します。Data-in-Greek は CCSID 00875 (Greek) で明示的に表現されたデータとして変換されます。ただし、以下のステートメントを使用すると変換が正しく行われません (項目内のすべての文字がギリシャ語と英語のコード・ページに共通する表現の文字であった場合を除く)。

```
Move Data-in-Greek to Data-in-Unicode
```

Data-in-Greek は、この MOVE ステートメントにより、00037 (米国英語) から UTF-16 への変換に基づいて Unicode に変換されます。この変換が失敗するのは、Data-in-Greek が実際には CCSID 00875 でエンコードされるためです。

CODEPAGE コンパイラー・オプションを正しく CCSID 00875 に設定することができた場合 (つまり、プログラムの残りの部分も EBCDIC データをギリシャ語で処理した場合) は、上記と同じ例を次のようにしてコーディングすることができます。

```
 CBL CODEPAGE(00875)
* . . .
 01  Data-in-Unicode pic N(100) usage national.
 01  Data-in-Greek pic X(100).
     Read Greek-file into Data-in-Greek
* . . . process Data-in-Greek here ...
* . . . or do the following (if need to process data in Unicode)
     Move Data-in-Greek to Data-in-Unicode
* . . . process Data-in-Unicode
     Move function Display-of(Data-in-Unicode) to Data-in-Greek
     Write Greek-record from Data-in-Greek
```

UTF-8 データの処理

UTF-8 データを処理する必要がある場合は、最初にデータを国別データ項目の UTF-16 Unicode に変換します。国別データを処理したあとで、データを出力のために再び UTF-8 に変換します。この変換には、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用します。UTF-8 データにはコード・ページ 01208 を使用します。

EBCDIC データを UTF-8 に変換するには、次の 2 つのステップを実行する必要があります。

1. 関数 NATIONAL-OF を使用して、EBCDIC ストリングを国別ストリング (UTF-16) に変換します。
2. 関数 DISPLAY-OF を使用して、国別ストリング (UTF-16) を UTF-8 に変換します。

次の例は、これらのステップを示しています。ここではギリシャ語の EBCDIC データを UTF-8 に変換します。

```
01 Greek-EBCDIC pic X(10) value "αβγδεζηθ".
01 UnicodeString pic N(10).
01 UTF-8-String pic X(20).
    Move function National-of(Greek-EBCDIC, 00875) to UnicodeString
    Move function Display-of(UnicodeString, 01208) to UTF-8-String
```

中国語 GB 18030 データの処理

Enterprise COBOL では、GB 18030 を明示的にサポートしていません。ただし、GB 18030 文字の処理を以下の方法でサポートします。

- DBCS データを使用して、CCSID 1388 で表現された GB 18030 文字を処理できます。
- Unicode データを使用して、CCSID 01200 (UTF-16 の代用を必要としない GB 18030 文字) で表現された GB 18030 文字を定義および処理できます。
- データを Unicode に変換し、Unicode として処理し、その後データを元のコード・ページ表現に戻すことによって、どのようなコード・ページ (CCSID 1388 を含む) のデータでも処理することができます。

  変換が必要な中国語 GB 18030 データを処理する必要がある場合は、最初に入力データを国別データ項目の UTF-16 Unicode に変換します。国別データを処理したあとで、データを出力のために再び中国語 GB 18030 に変換します。この変換には、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用し、コード・ページ 1388 をそれぞれの 2 番目の引き数として指定します。

  次の例は、このような変換を示しています。

```
 01 Chinese-EBCDIC pic X(16) value "奥林匹克运动会".
 01 Chinese-GB18030-String pic X(16).
 01 UnicodeString pic N(14).
 . . .
     Move function National-of(CHinese-EBCDIC, 1388) to UnicodeString
*  Process data in Unicode
     Move function Display-of(UnicodeString, 1388) to Chinese-GB18030-String
```

GB 18030 文字は、既存の中国語 EBCDIC コード・ページ CCSID 1388 を使用してエンコードされます。このコード・ページは、UTF-16 サロゲート値を必要としない GB 18030 文字を含むように拡張されています。UTF-16 のサロゲート値は、それぞれの文字ごとに 2 つの 2 バイト・エンコード・ユニット (4 バイト) を必要とする文字です。

国別データ項目の比較

国別データ項目は、比較条件で明示的または暗黙的に他の国別データ項目と比較することができます。国別データ項目の条件式を以下のステートメントでコード化することができます。

- EVALUATE
- IF
- INSPECT
- PERFORM
- SEARCH
- STRING
- UNSTRING

国別データ項目は、以下のステートメントの比較条件で明示的または暗黙的に一定の非国別項目と比較することができます。

- EVALUATE

- IF
- PERFORM
- SEARCH

国別オペランドを含む比較はすべて国別比較です。オペランドは有効な比較を行うための規則に従う必要があります。PROGRAM COLLATING SEQUENCE は、国別オペランドを含む比較には影響を与えません。

国別オペランドの比較

長さが等しい 2 つの国別オペランドを比較するときは、対応する文字の対がすべて等しい場合に等しいと判断されます。対応する文字の対に等しくないものがある場合は、等しくない最初の文字の対の 2 進値を比較することによって、より大きい 2 進値を持つオペランドが判別されます。

長さが異なるオペランドを比較するときは、短い方のオペランドの右側にデフォルトの Unicode UTF-16 スペース文字 (NX'0020') が埋め込まれて長いほうのオペランドと同じ長さになったものとして扱われます。

国別オペランドと数値オペランドの比較

国別リテラルまたは国別データ項目を、USAGE DISPLAY の整数リテラルまたは数値データ項目と比較することができます。数値オペランドは基本英数字項目に移動されたかのように扱われ、英数字比較が実行されます。

国別オペランドと英字または英数字オペランドの比較

国別リテラルまたは国別データ項目を、英字または英数字データ項目と比較することができます。英字または英数字オペランドは基本国別項目に移動されたかのように扱われ、国別比較が実行されます。

国別オペランドとグループ・オペランドの比較

国別リテラルまたは国別データ項目をグループと比較することができます。国別オペランドは、グループ・オペランドと同サイズのグループ項目に移動されたかのように扱われ、2 つのグループが比較されます。

関連タスク

116 ページの『COBOL での国別データ (Unicode) の使用』
83 ページの『アクションの選択項目のコーディング』
104 ページの『データ項目の計算および置換 (INSPECT)』
92 ページの『ループのコーディング』
78 ページの『シリアル検索 (SEARCH)』
95 ページの『データ項目の結合 (STRING)』
97 ページの『データ項目の分割 (UNSTRING)』

関連参照

比較条件 (*Enterprise COBOL 言語解説書*)
数値オペランドと英数字オペランドの比較 (*Enterprise COBOL 言語解説書*)

DBCS データを含む英数字データ項目の処理

DBCS 文字を含む英数字データ項目に対してバイト単位の英数字操作 (たとえば、STRING、UNSTRING、または参照変更) を使用すると、結果は予測できません。代わりに以下のステップを実行する必要があります。

1. MOVE ステートメントまたは NATIONAL-OF 関数を使用して、項目を国別データ項目の Unicode に変換します。
2. 国別項目を必要に応じて処理します。
3. DISPLAY-OF 関数を使用して、結果を再度英数字に変換します。

関連タスク

95 ページの『データ項目の結合 (STRING)』
97 ページの『データ項目の分割 (UNSTRING)』
101 ページの『データ項目のサブストリングの参照』
118 ページの『国別データの変換』

第 8 章 ファイルの処理

データの読み取りと書き込みは、すべてのプログラムに不可欠なものです。プログラムは情報を検索し、それを要求どおりに処理した後、結果を示します。

情報のソースおよび結果のターゲットとしては、以下の 1 つまたは 2 つ以上を使用できます。

- 別のプログラム
- 直接アクセス記憶装置
- 磁気テープ
- プリンター
- 端末
- カード読取装置または穿孔装置

外部装置上に存在する情報は、物理レコードまたはブロックとして格納されています。レコードまたはブロックは、入力または出力操作時にシステムによって 1 つの単位として処理される情報の集まりです。

COBOL プログラムは、物理レコードを直接処理しません。論理レコードを処理します。論理レコードは、1 つの完全な物理レコードであることもあるし、1 つの物理レコードの一部であることもあるし、1 つまたは複数の物理レコードの一部または全部を含むこともあります。COBOL プログラムは、論理レコードを、それらが定義されたとおりに正確に処理します。

COBOL では、論理レコードの集合をファイル (プログラムで処理できる情報の列) と言います。

関連概念
『ファイル編成および入出力装置』

関連タスク
127 ページの『ファイル編成およびアクセス・モードの選択』
129 ページの『ファイルの割り振り』
131 ページの『入出力エラーの検査』

ファイル編成および入出力装置

入出力装置によって異なりますが、ファイル編成には、順次、行順次、索引付き、または相対があります。プログラムを設計するときに、使用する装置およびファイル・タイプを決定してください。

以下のファイル編成を選択することができます。

順次ファイル編成
レコードの配置は、ファイル作成時のレコードの入力順によって決まります。それぞれのレコード (最初のレコードは除く) には固有の先行レコード

© Copyright IBM Corp. 1991, 2001

があり、それぞれのレコード (最後のレコードは除く) には固有の後続レコードがあります。いったん確立されると、これらの関係は変わりません。

順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

行順次ファイル編成

行順次ファイルとは、階層ファイル・システム (HFS) に上にある順次ファイルであり、データとしては文字だけが入っています。各レコードは改行文字で終了します。

行順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

索引付きファイル編成

ファイル内のそれぞれのレコードには特殊フィールドが含まれ、そのフィールドの内容がレコード・キーを形成します。このキーの位置は各レコードで同じです。ファイルの論理配置は、ファイルの索引コンポーネントがレコード・キーによって順序付けて確立します。ファイル内のレコードの実際の物理配置は、COBOL プログラムにとっては重要ではありません。

索引付きファイルは、レコード・キーに加えて、代替索引を使用することもできます。これらのキーにより、レコードに関して別の論理順序付けを使用して、ファイルにアクセスできます。

索引付きファイルについて認められるアクセス (レコード伝送) モードは、順次、ランダム、または動的です。索引付きファイルの順次読み取りまたは書き込みの順序は、キー値の順序になります。

相対ファイル編成

ファイルのレコードは、ファイルの始まりからの相対位置によって識別されます。ファイルの最初のレコードの相対レコード番号は 1 で、10 番目のレコードの相対レコード番号は 10 というようになっています。

相対ファイルについて認められるアクセス (レコード伝送) モードは、順次、ランダム、または動的です。相対ファイルの順次読み取りまたは書き込みの順序は、相対レコード番号の順序です。

IBM Enterprise COBOL for z/OS and OS/390 では、レコードの保管および入出力装置からの取り出しに関するオペレーティング・システムへの要求は、QSAM と VSAM の 2 つのアクセス方式、および UNIX ファイル・システムによって処理されます。

データの保管にどの装置タイプを使用するかによって、使用できるファイル編成の選択に影響する可能性があります。ファイル編成オプションで最も柔軟に選択できるのは、直接アクセス記憶装置の場合です。順次専用の装置の場合、編成オプションは制限されますが、有用な他の特性 (テープの可搬性など) があります。

順次専用装置

端末装置、プリンター、カード読取装置、および穿孔装置は、一度に 1 行を処理するので、*ユニット・レコード装置* と呼ばれます。したがって、ユニット・レコード装置との間で読み取りまたは書き込みを行うときは、プログラムで一度にレコードを 1 つずつ順次に処理しなければなりません。

テープでは、レコードは順次に並べられるので、プログラムでそれらを順次に処理しなければなりません。テープ・ファイルを処理するときは、QSAM 物理順次ファイルを使用してください。テープのレコードは固定長でも可変長でも構いません。データ転送の速度は、カードの場合よりも速くなります。

直接アクセス記憶装置

直接アクセス記憶装置には多くのレコードが格納できます。直接アクセス記憶装置の場合は、これらの装置に保管されるファイルのレコード配置によって、プログラムがデータを処理できる方法が決まります。直接アクセス装置を使用すると、以下の複数のファイル編成タイプが使用できるため、プログラムにおける柔軟性が高まります。

- 順次 (VSAM または QSAM)
- 行順次 (UNIX)
- 索引付き (VSAM)
- 相対 (VSAM)

関連タスク

129 ページの『ファイルの割り振り』
133 ページの『第 9 章 QSAM ファイルの処理』
165 ページの『第 10 章 VSAM ファイルの処理』
195 ページの『第 11 章 行順次ファイルの処理』
『ファイル編成およびアクセス・モードの選択』

ファイル編成およびアクセス・モードの選択

アプリケーションで使用するファイル編成およびアクセス・モードを決定する際には、以下の指針を考慮してください。

- アプリケーションがレコード (固定長または可変長レコード) に順次アクセスだけしか行わず、既存のレコード間にレコードを挿入しない場合には、QSAM または VSAM 順次ファイルが最も単純なタイプです。
- 印刷可能文字および特定の制御文字のみを含むレコードに順次にアクセスする UNIX アプリケーションを開発している場合には、行順次ファイルが最も適しています。
- アプリケーションが (固定長または可変長レコードへの) 順次アクセスとランダム・アクセスの両方を必要とする場合には、VSAM 索引付きファイルが最も柔軟性のあるタイプです。
- アプリケーションがランダムにレコードを挿入および削除する場合には、相対ファイルが適しています。

ファイルのかなりの部分がアプリケーションの中で参照または更新される場合には、順次処理の方が、ランダムまたは動的アクセスより速くなります。アプリケーションの実行ごとに処理されるレコードの割合が小さい場合は、ランダム・アクセスまたは動的アクセスを使用します。

次の表は、COBOL ファイルについての可能なファイル編成、アクセス・モード、およびレコード形式を示しています。

ファイル編成	順次 アクセス	ランダム・ アクセス	動的 アクセス	固定長	可変長
QSAM (物理順次)	X			X	X
行順次	X			X[1]	X
VSAM 順次 (ESDS)	X			X	X
VSAM 索引付き (KSDS)	X	X	X	X	X
VSAM 相対 (RRDS)	X	X	X	X	X

1. データ自体は可変長形式ですが、COBOL 固定長レコードに読み込んだり、固定長レコードから書き出すことができます。

関連参照

『入出力のコーディング形式』

196 ページの『許可される制御文字』

入出力のコーディング形式

入出力コーディングの一般形式を、以下に示します。コードの後に、ユーザー指定の情報について説明しています。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT filename ASSIGN TO assignment-name  (1) (2)
    ORGANIZATION IS org ACCESS MODE IS access  (3) (4)
    FILE STATUS IS file-status                 (5)
. . .
DATA DIVISION.
FILE SECTION.
FD  filename
01  recordname                                 (6)
    nn . . . fieldlength & type                (7) (8)
    nn . . . fieldlength & type
. . .
WORKING-STORAGE SECTION
01 file-status  PICTURE 99.
. . .
PROCEDURE DIVISION.
    . . .
    OPEN iomode filename                       (9)
    . . .
    READ filename
    . . .
    WRITE recordname
    . . .
    CLOSE filename
    . . .
    STOP RUN.
```

上記のコードの中のユーザー指定の情報は、次のとおりです。

(1) *filename*

任意の正規 COBOL 名。SELECT ステートメントと FD ステートメント、および READ、OPEN、CLOSE の各ステートメントでは、同じファイル名を使用しなければなりません。さらに、START または DELETE ステートメントを使用する場合にも、ファイル名が必要になります。この名前は、必ずしも、シ

ステムに認識されているデータ・セットの実際の名前でなくても構いません。各ファイルには、独自の SELECT、FD、および入出力ステートメントが必要です。

(2) ***assignment-name***

選択した任意の名前 (COBOL およびシステムの命名規則に従うもの)。名前は、ユーザー定義語である場合には 1 ～ 30 文字の長さにすることができ、リテラルである場合には 1 ～ 160 文字の長さにすることができます。*assignment-name* の *name* 部分は、DD ステートメントで、ALLOCATE コマンド (TSO) で、あるいは環境変数として (たとえば、export コマンドで) (UNIX) 指定することができます。

(3) ***org***

編成は、SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE にすることができます。この文節は、QSAM ファイルについては任意指定です。

(4) ***access***

アクセス・モードは、SEQUENTIAL、RANDOM、または DYNAMIC にすることができます。順次ファイル処理の場合 (行順次ファイルを含む) は、この文節を省略することができます。

(5) ***file-status***

2 文字の COBOL FILE STATUS キー。

(6) ***recordname***

WRITE および REWRITE ステートメントで使用されるレコードの名前。

(7) ***fieldlength***

フィールドの論理長。

(8) ***type***

ファイルのレコード形式。レコード記入項目をレベル 01 記述以上に分ける場合、各エレメントはレコードのフィールドに対して正確にマップしなければなりません。

(9) ***iomode***

INPUT または OUTPUT モード。ファイルから読み取りだけを行う場合は、INPUT をコーディングします。書き込みだけを行う場合は、OUTPUT または EXTEND をコーディングします。読み取りと書き込みの両方を行う場合は、I-O をコーディングします (ただし、LINE SEQUENTIAL 編成の場合を除きます)。

関連タスク

133 ページの『第 9 章 QSAM ファイルの処理』
165 ページの『第 10 章 VSAM ファイルの処理』
195 ページの『第 11 章 行順次ファイルの処理』

ファイルの割り振り

z/OS または UNIX アプリケーション内のどのタイプのファイル (順次、行順次、索引付き、または相対) についても、DD 名または環境変数名のいずれかを使用して外部名を定義することができます。外部名とは、ASSIGN 文節の *assignment-name* 内の名前です。

ファイルが HFS 内にある場合は、DD 定義または環境変数のいずれかを使用して、PATH キーワードでパス名を指定することによって、ファイルを定義することができます。

環境変数名は大文字でなければなりません。その値として使用できる属性は、定義中のファイルの編成によって異なります。

外部名は 2 つの方法のいずれかで定義できるので、COBOL 実行時には、以下のステップを使用してファイルの定義が調べられます。

1. DD 名が明示的に割り振られていれば、それが使用されます。定義は、JCL の DD ステートメント、TSO/E からの ALLOCATE コマンド、またはユーザー開始の動的割り振りから取ることができます。
2. DD 名が明示的に割り振られていないが、同じ名前の環境変数が設定されている場合は、その環境変数の値が使用されます。

   ファイルは、環境変数で指定された属性を使用して動的に割り振られます。最小限、PATH() または DSN() オプションのいずれかを指定しなければなりません。オプションおよび属性はすべて大文字でなければなりません (ただし、PATH オプションの *path-name* サブオプションは例外で、これには大文字小文字の区別があります)。DSN() オプションに一時データ・セット名を指定することはできません。

   以下の場合には、ファイル状況コード 98 が出されます。

   - 環境変数の内容 (ヌルまたはすべてブランクの値を含む) が無効である。
   - ファイルの動的割り振りが失敗する。
   - ファイルの動的割り振り解除が失敗する。

   COBOL 実行時には、各 OPEN ステートメントで環境変数の内容が検査されます。同じ外部名を持つファイルが前の OPEN ステートメントによって動的に割り振られており、その OPEN 以後に環境変数の内容が変更された場合は、実行時に、前の割り振りが動的に割り振り解除され、環境変数に現在設定されているオプションを使用してファイルが再度割り振られます。環境変数の内容が変更されていない場合、実行時には現行の割り振りが使用されます。
3. DD 名も環境変数も定義されていない場合には、次のようになります。
   - 割り振りが QSAM ファイルに対するものであり、CBLQDA ランタイム・オプションが有効であれば、適格ファイルに対して CBLQDA 動的割り振り処理が行われます。この種の「暗黙の」動的割り振りは、実行単位が存続する間持続され、再割り振りすることはできません。
   - それ以外の場合、割り振りは失敗します。

COBOL 実行時には、「暗黙の」CBLQDA 割り振りを除いて、実行単位の終了時にすべての動的割り振りが割り振り解除されます。

関連タスク

429 ページの『環境変数の設定およびアクセス』
150 ページの『QSAM ファイルの定義および割り振り』
146 ページの『CBLQDA による QSAM ファイルの動的作成』
189 ページの『VSAM ファイルの割り振り』

入出力エラーの検査

入力ステートメントまたは出力ステートメントが実行されるたびに、FILE STATUS キーは、操作の成否を示す値で更新されます。FILE STATUS 文節を使用すれば、各入出力ステートメントの実行後に FILE STATUS キーをテストして、0 以外のファイル状況コードが戻された場合に使用するエラー処理手順を設定することができます。

VSAM ファイルの場合は、FILE STATUS 文節に 2 番目の *data-name* を使用すると、追加の VSAM 戻りコード情報を得ることができます。

入出力操作のエラーを処理するもう 1 つの方法は、以下の参照情報で説明しているように、ERROR (EXCEPTION と同義) 宣言をコーディングすることです。

関連タスク

250 ページの『入出力操作でのエラーの処理』
253 ページの『ERROR 宣言のコーディング』
254 ページの『ファイル状況キーの使用』

第 9 章 QSAM ファイルの処理

待機順次アクセス方式 (QSAM) ファイルは、キーなしのファイルであり、レコードは入力順に次々に配置されます。プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で取り出します (READ ステートメントを使って)。各レコードは先行レコードの後に置かれます。

プログラムで QSAM ファイルを処理するには、次のような COBOL 言語ステートメントを使用します。

- ENVIRONMENT DIVISION および DATA DIVISION 内で QSAM ファイルを識別および記述する
- PROCEDURE DIVISION 内でこれらのファイル内のレコードを処理する

レコードの作成後は、その長さまたは位置を変えることはできず、それを削除することはできません。しかし、直接アクセス記憶装置上の QSAM ファイルは (REWRITE を使って) 更新することができます (ただし HFS では不可)。

QSAM ファイルは、テープ、直接アクセス記憶装置 (DASD)、ユニット・レコード装置、および端末装置上に置くことができます。QSAM 処理は、テーブルおよび中間記憶域に最適です。

QSAM を使用して、HFS 内のバイト・ストリーム・ファイルにアクセスすることもできます。これらのファイルは、レコード構造を持たない 2 進数のバイト単位の順次ファイルです。COBOL プログラムでコーディングしたレコード定義と、読み取りおよび書き込みに使う変数の長さによって、転送されるデータの量が決まります。

関連概念
158 ページの『QSAM ファイルのラベル』
z/OS DFSMS: Using Data Sets の『Using access methods』の項

関連タスク
『COBOL での QSAM ファイルおよびレコードの定義』
145 ページの『QSAM ファイルの入出力ステートメントのコーディング』
149 ページの『QSAM ファイルのエラーの処理』
150 ページの『QSAM ファイルに対する作業』
161 ページの『テープ上の QSAM ASCII ファイルの処理』
163 ページの『ASCII ファイルのラベルの処理』

COBOL での QSAM ファイルおよびレコードの定義

FILE-CONTROL 記入項目を使用して、以下のことを行ってください。

- COBOL プログラム内のファイルを QSAM ファイルとして定義します。

© Copyright IBM Corp. 1991, 2001

- それらのファイルを外部ファイル名 (ddnames または環境変数名) と関連付けます。(*外部ファイル名* とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。)

次の例では、COMMUTER-FILE-MST は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

```
FILE-CONTROL.
    SELECT COMMUTER-FILE-MST
    ASSIGN TO S-COMMUTR
    ORGANIZATION IS SEQUENTIAL
    ACCESS MODE IS SEQUENTIAL.
```

ASSIGN 文節 *name* には、ファイルが QSAM ファイルである文書の外部名 (ddname または環境変数名) の前に S- を含めることができます。

ORGANIZATION 文節と ACCESS MODE 文節は、ともに任意指定です。

関連タスク
『レコード形式の指定』
142 ページの『ブロック・サイズの設定』

レコード形式の指定

DATA DIVISION の FD 記入項目で、レコード形式と、レコードがブロック化されるかどうかをコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。

F、V、S、または U のレコード形式を RECORDING MODE 文節でコーディングすることができます。COBOL は、RECORD 文節から、またはファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。レコードがブロック化されるようにしたい場合には、FD 記入項目に BLOCK CONTAINS 文節をコーディングしてください。

次の例は、固定長レコードを持つファイルの場合に FD 記入項目がどのようになるかを示しています。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS F
    BLOCK CONTAINS 0 RECORDS
    RECORD CONTAINS 80 CHARACTERS.
01  COMMUTER-RECORD-MST.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

S の記録モードは、HFS 内のファイルではサポートされません。上記の例はこのようなファイルに適しています。

関連概念
135 ページの『論理レコード』

関連タスク
135 ページの『固定長形式の要求』
136 ページの『可変長形式の要求』
138 ページの『スパン形式の要求』

141 ページの『不定形式の要求』
133 ページの『COBOL での QSAM ファイルおよびレコードの定義』

論理レコード

論理レコード という用語は、COBOL 言語と z/OS QSAM とでは使われ方が若干異なります。形式 V および形式 S ファイルの場合、QSAM 論理レコードではレコードのユーザー・データ部分の前に、COBOL 論理レコードの定義には含まれていない 4 バイトの接頭部が入ります。形式 F および形式 U のファイル、および HFS のバイト・ストリーム・ファイルの場合、QSAM 論理レコードと COBOL 論理レコードの定義は同じです。

本書では、*QSAM 論理レコード* は QSAM 定義を指し、*論理レコード* は COBOL 定義を指しています。

関連参照

136 ページの『形式 F レコードのレイアウト』
137 ページの『形式 V レコードのレイアウト』
140 ページの『形式 S レコードのレイアウト』
141 ページの『形式 U レコードのレイアウト』

固定長形式の要求

固定長レコードは形式 F です。この形式を明示的に要求するには、`RECORDING MODE F` を使用してください。

`RECORDING MODE` 文節は省略することができます。ファイルに関連付けられている最大のレベル 01 レコードの長さが、`BLOCK CONTAINS` 文節でコーディングされているブロック・サイズを超えておらず、かつ次のいずれかを行った場合は、コンパイラーは記録モードを `F` と見なします。

- `RECORD CONTAINS` *integer* 文節 (`RECORD` 文節の形式 1) を使用する。

  この文節を使用すると、ファイルは必ずレコード長 *integer* の固定長形式になります (別の長さを指定した複数のレベル 01 レコード記述がファイルに関連付けられている場合でも)。
- `RECORD CONTAINS` *integer* 文節を省略するが、ファイルに関連付けられているすべてのレベル 01 レコード記述項目を、同じ固定サイズで、`OCCURS DEPENDING ON` 文節を含まないようにコーディングする。この固定サイズがレコード長になります。

非ブロック化形式 F ファイルでは、論理レコードはブロックと同じになります。

ブロック化形式 F ファイルでは、ブロック内の論理レコードの数 (ブロック化因数) は、ファイル内のどのブロックでも一定です。ただし、最後のブロックだけは短くなることがあります。HFS 内のファイルはブロック化されることはありません。

関連概念

『論理レコード』

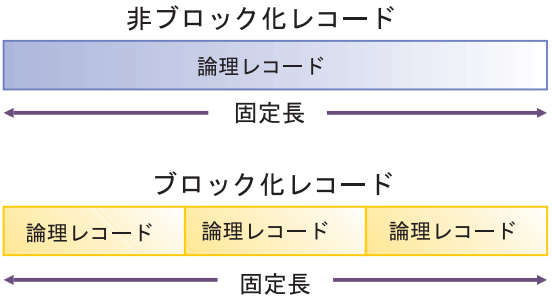
関連タスク

136 ページの『可変長形式の要求』

138 ページの『スパン形式の要求』
141 ページの『不定形式の要求』
134 ページの『レコード形式の指定』

関連参照
『形式 F レコードのレイアウト』

形式 F レコードのレイアウト: 形式 F の QSAM レコードのレイアウトを以下に示します。



関連概念
135 ページの『論理レコード』

関連タスク
135 ページの『固定長形式の要求』
z/OS DFSMS: Using Data Sets の『Fixed-length record formats』の項

関連参照
137 ページの『形式 V レコードのレイアウト』
140 ページの『形式 S レコードのレイアウト』
141 ページの『形式 U レコードのレイアウト』

可変長形式の要求

可変長レコードは形式 V または形式 D です。形式 D のレコードは、ASCII テープ・ファイル上の可変長レコードです。形式 D レコードは、形式 V レコードと同じ方法で処理されます。いずれの場合も、`RECORDING MODE V` を使用してください。

`RECORDING MODE` 文節は省略することができます。ファイルに関連付けられている最大のレベル 01 レコードの長さが、`BLOCK CONTAINS` 文節でコーディングされているブロック・サイズを超えておらず、かつ次のいずれかを行った場合は、コンパイラーは記録モードを `V` と見なします。

- `RECORD IS VARYING` 文節 (`RECORD` 文節の形式 3) を使用する。

  integer-1 および *integer-2* (`RECORD IS VARYING FROM` *integer-1* `TO` *integer-2*) に値を指定すると、最大レコード長は、ファイルに関連するレベル 01 レコード記述項目でコーディングされた長さに関係なく、*integer-2* にコーディングされた値になります。

  integer-1 および *integer-2* を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

- RECORD CONTAINS *integer-1* TO *integer-2* 文節 (RECORD 文節の形式 2) を使用する。*integer-1* および *integer-2* を、ファイルに関連するレベル 01 レコード記述項目の最小長および最大長にしてください。最大レコード長は *integer-2* の値です。
- RECORD 文節を省略するが、サイズが異なるかまたは OCCURS DEPENDING ON 文節を含む複数のレベル 01 レコード (ファイルに関連付けられる) をコーディングする。

  最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

形式 V ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。たとえば、READ ステートメントによって読み込まれる形式 V レコードに MOVE ステートメントを指定すると、移動されるレコードのサイズは、そのレベル 01 レコード記述に相当します。

形式 V ファイルに対する READ ステートメントを指定し、その後にレベル 01 レコードの MOVE を続けて指定すると、実際のレコード長が使用されません。プログラムは、レベル 01 レコード記述で記述されたバイト数を移動しようとします。このバイト数が実際のレコード長を超え、プログラムによってアドレッシング可能な領域外に及ぶと、結果は予測できません。レベル 01 レコード記述によって記述されたバイト数が、読み取られた物理レコードより短い場合は、レベル 01 記述を超えるバイトは切り捨てられます。可変長レコードの実際の長さを調べるには、ファイル定義 (FD) の RECORD 文節の形式 3 で data-name-1 を指定してください。

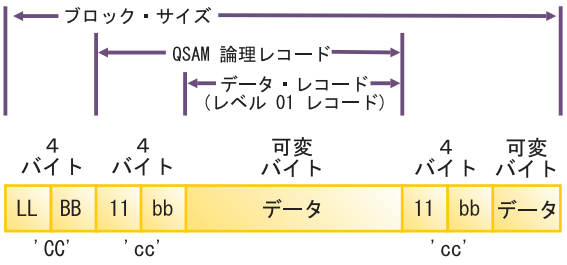
関連タスク

135 ページの『固定長形式の要求』
138 ページの『スパン形式の要求』
141 ページの『不定形式の要求』
134 ページの『レコード形式の指定』

関連参照

『形式 V レコードのレイアウト』
VS COBOL II ランタイムからの移行 (*Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド*)

形式 V レコードのレイアウト: 形式 V の QSAM レコードは、データの前に制御フィールド (以下を参照) を持っています。QSAM 論理レコード長は、プログラム内で定義されたレコード長に 4 バイト (制御フィールド用の) を加算することによって決まりますが、レコードおよびレコード長の記述の中にこれらの 4 バイトを含めてはなりません。



CC 各ブロックの最初の 4 バイトには、制御情報が入ります。

LL は、ブロックの長さ ('CC' フィールドを含む) を指定する 2 バイトを表します。

BB は、システム使用に予約されている 2 バイトを表します。

cc 各論理レコードの最初の 4 バイトには、制御情報が入ります。

11 は、論理レコード長 ('cc' フィールドを含む) を指定する 2 バイトを表します。

bb は、システム使用に予約されている 2 バイトを表します。

ブロック長は次のように決定されます。

- 非ブロック化形式 V レコード: CC + cc + データ部分
- ブロック化形式 V レコード: CC + 各レコードの cc + 各レコードのデータ部分

オペレーティング・システムは、ファイルの作成時に制御バイトを設定します。この制御バイト・フィールドは、プログラムの DATA DIVISION における論理レコードの記述には現れません。COBOL は、制御バイトを収容するのに十分な大きさの入出力バッファーを割り振ります。バッファー内のこれらの制御フィールドをプログラムの中で使用することはできません。可変長レコードがユニット・レコード装置に書き込まれるときには、制御バイトは印刷も穿孔もされません。しかし、それらは、その他の外部ストレージ装置や、ストレージのバッファー域には入れられます。V モード・レコードを入力バッファーから WORKING-STORAGE 域に移動すると、それらは制御バイトなしで移動されます。

HFS 内のファイルはブロック化されることはありません。

関連概念
135 ページの『論理レコード』

関連タスク
136 ページの『可変長形式の要求』

関連参照
136 ページの『形式 F レコードのレイアウト』
140 ページの『形式 S レコードのレイアウト』
141 ページの『形式 U レコードのレイアウト』

スパン形式の要求

スパン・レコードとは、形式 S のレコードで、1 つまたは複数の物理ブロックに含めることができる QSAM 論理レコードのことです。磁気テープまたは直接アクセ

ス装置に割り当てられる QSAM ファイルのスパン・レコードについては、RECORDING MODE S をコーディングすることができます。HFS のファイルについてはスパン・レコードを要求してはなりません。

RECORDING MODE 文節は省略することができます。コンパイラーが記録モードを S と見なすのは、最大レコード長に 4 を加えた値が BLOCK CONTAINS 文節で設定されたブロック・サイズより大きい場合です。

プログラム内の形式 S のファイルの場合、最大レコード長は、形式 V について使用されるのと同じ規則で判別されます。この長さは、RECORD 文節の使用法に基づいています。

形式 S レコードを含むファイルを作成しているときに、レコードがブロック内の残りのスペースよりも大きいと、COBOL では、ブロックがいっぱいになるまでレコードのセグメントが書き込まれます。レコードの残りの部分は、その長さに応じて、次の 1 つまたは複数のブロックに格納されます。COBOL は、32760 バイトまでの長さの QSAM スパン・レコードをサポートします。

形式 S レコードを持つファイルを検索するときには、プログラムは完全なレコードだけを検索することができます。

形式 S ファイルの利点: 形式 S レコードを持つファイルを定義すると、外部ストレージを効率よく利用できる上に、論理レコード長を使ってファイルを編成することができます。

- 直接アクセス装置でトラック容量を効率的に使用するためにブロック長を設定することができます。
- 論理レコード長を装置依存の物理ブロック長に合わせて調整する必要がありません。1 つの論理レコードが 2 つ以上の物理ブロックにわたることができます。
- 異なるタイプの直接アクセス記憶装置間で論理レコードを転送したいときに、柔軟性が増します。

しかし、形式 S ファイルを処理するためには追加のオーバーヘッドが必要になります。

形式 S ファイルと READ INTO: 形式 S ファイルに READ INTO ステートメントを指定すると、そのファイルについて読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。

関連概念

135 ページの『論理レコード』
140 ページの『スパン・ブロック化および非ブロック化ファイル』

関連タスク

135 ページの『固定長形式の要求』
136 ページの『可変長形式の要求』
141 ページの『不定形式の要求』
134 ページの『レコード形式の指定』

関連参照
『形式 S レコードのレイアウト』

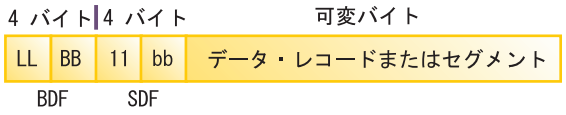
スパン・ブロック化および非ブロック化ファイル: スパン・ブロック化された QSAM ファイルは、それぞれが 1 つまたは複数の論理レコードまたは論理レコードのセグメントを含むブロックから構成されます。論理レコードは固定長であっても可変長であっても構わず、それらのサイズは物理ブロック・サイズと等しくても、より小さくても、より大きくても構いません。論理レコードと物理レコードのサイズの間に必要とされる関係はありません。

スパン非ブロック化ファイルは、それぞれが 1 つの論理レコードまたは論理レコードの 1 つのセグメントを含む物理ブロックから構成されます。論理レコードは固定長であっても可変長であっても構いません。物理ブロックが 1 つの論理レコードを含んでいる場合、ブロック長は論理レコード・サイズによって判別されます。論理レコードをセグメント化しなければならない場合、システムは、常に、できる限り大きな物理ブロックを書き込みます。論理レコード全体が 1 つのトラックに収まらない場合、システムは論理レコードをセグメント化します。

関連概念
135 ページの『論理レコード』

関連タスク
138 ページの『スパン形式の要求』

形式 S レコードのレイアウト: 以下に示すように、スパン・レコードの前には制御フィールドが置かれます。



各ブロックの前には、ブロック記述子フィールドが置かれます。ブロック記述子フィールドは、各物理ブロックの始まりに 1 つだけあります。

ブロック内のレコードの各セグメントの前には、そのセグメントがレコード全体である場合でも、セグメント記述子フィールドが置かれます。セグメント記述子フィールドは、ブロック内の各レコード・セグメントにつき 1 つあります。セグメント記述子フィールドは、そのセグメントが最初、最後、または中間のいずれかであるかも示します。

これらのフィールドは、COBOL プログラムの DATA DIVISION では記述せず、プログラム内で使用することはできません。

関連タスク
138 ページの『スパン形式の要求』

関連参照
136 ページの『形式 F レコードのレイアウト』
137 ページの『形式 V レコードのレイアウト』
141 ページの『形式 U レコードのレイアウト』

不定形式の要求

形式 U レコードは、未定義または未指定の特性を持ちます。形式 U を使用すると、形式 F または形式 V 仕様に適合しないブロックを処理することができます。

コンパイラーが記録モードを U と判別するのは、RECORDING MODE U をコーディングした場合だけです。

レコード長は、RECORD 文節が使用される方法に基づいて、プログラムの中で判別されます。

- RECORD CONTAINS *integer* 文節 (RECORD 文節の形式 1) を使用する場合、レコード長は、ファイルに関連するレベル 01 レコード記述項目の長さに関係なく、*integer* の値になります。
- RECORD IS VARYING 文節 (RECORD 文節の形式 3) を使用する場合、レコード長は、*integer-1* と *integer-2* がコーディングされるかどうかに基づいて判別されます。

  integer-1 および *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*) をコーディングすると、最大レコード長は、ファイルに関連するレベル 01 レコード記述項目の長さに関係なく、*integer-2* の値になります。

  integer-1 および *integer-2* を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。
- RECORD CONTAINS *integer-1* TO *integer-2* 文節 (RECORD 文節の形式 2) を使用し、ファイルに関連するレベル 01 レコード記述項目の最小長を *integer-1* に、最大長を *integer-2* に指定する場合は、最大レコード長は *integer-2* の値になります。
- RECORD 文節を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

形式 U ファイルと READ INTO: 形式 U ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコードのサイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。

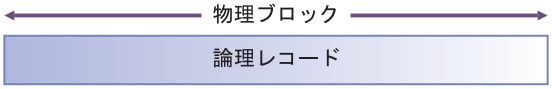
関連タスク

135 ページの『固定長形式の要求』
136 ページの『可変長形式の要求』
138 ページの『スパン形式の要求』
134 ページの『レコード形式の指定』

関連参照

『形式 U レコードのレイアウト』

形式 U レコードのレイアウト: 外部ストレージの各ブロックは 1 つの論理レコードとして扱われます。レコード長やブロック長フィールドはありません。



関連概念
135 ページの『論理レコード』

関連タスク
141 ページの『不定形式の要求』

関連参照
136 ページの『形式 F レコードのレイアウト』
137 ページの『形式 V レコードのレイアウト』
140 ページの『形式 S レコードのレイアウト』

ブロック・サイズの設定

COBOL では、BLOCK CONTAINS 文節を使用して物理レコードのサイズを設定します。この文節を省略すると、コンパイラーはレコードがブロック化されないものと見なします。ディスクまたはテープ上の QSAM ファイルをブロック化すると、処理速度が増し、ストレージ要件を最小限にすることができます。UNIX システム・サービス・ファイル (HFS のファイルを含む)、PDSE メンバー、およびスプール・データ・セットについても、ブロック化することができますが、ブロック化しても、システムがデータを保管する方法に対して何の影響も与えません。

BLOCK CONTAINS 文節でブロック・サイズを明示的に設定する場合には、装置についての最大ブロック・サイズよりも大きくしてはなりません。形式 F ファイルについて設定するブロック・サイズは、レコード長の整数倍でなければなりません。

プログラムでテープ上の QSAM ファイルを使用する場合は、少なくとも 12 ～ 18 バイトの物理ブロック・サイズを使用してください。さもないと、以下のいずれかの実行中にパリティー・チェックが発生すると、ブロックがスキップされます。

- 12 バイトよりも小さいレコード・ブロックの読み取り
- 18 バイトよりも小さいレコード・ブロックの書き込み

通常は、ブロック・サイズが大きい方がパフォーマンスが良くなります。ブロック・サイズがわずか数キロバイトの場合は、特に効率が低下します。最低でも数十キロバイトのブロック・サイズを選択してください。レコードのブロック化を指定してブロック・サイズを省略すると、装置の使用効率とデータ転送速度を考慮した最適なブロック・サイズがシステムによって選択されます。

z/OS にブロック・サイズを決定させる

最高のパフォーマンスを得るためには、COBOL ソース・プログラムのブロック化ファイルについてはブロック・サイズを明示的に設定しないようにお勧めします。新しいブロック化データ・セットの場合は、z/OS にシステム決定のブロック・サイズを提供させるようにする方が簡単です。このフィーチャーを使用するには、以下のガイドラインに従ってください。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- ソース・プログラムに RECORD CONTAINS 0 をコーディングしない。
- JCL DD ステートメントで BLKSIZE 値をコーディングしない。

ブロック・サイズの明示的設定

ブロック・サイズを明示的に設定したい場合は、以下のガイドラインに従うと、プログラムが最も柔軟になります。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- DD 名定義 (JCL DD ステートメント) に BLKSIZE 値をコーディングする。

z/OS における拡張形式データ・セットの場合、DFSMS は物理レコードに 32 バイトのブロック接尾部を追加します。ブロック・サイズを明示的に指定する (JCL または ISPF を介して) 場合には、このブロック接尾部のサイズをブロック・サイズに含めないでください。このブロック接尾部は、プログラムの中で使用することはできません。DFSMS は、ブロック接尾部を読み取るためのスペースを割り振ります。ただし、直接アクセス装置の 1 つのトラックに収まる拡張フォーマット・データ・セットのブロックの数を計算するときは、ブロック・サフィックスのサイズをブロック・サイズに含めることが必要です。

32760 より大きなブロック・サイズを BLOCK CONTAINS で直接指定するか、または BLOCK CONTAINS *n* RECORDS で間接的に指定した場合は、以下の両方の条件が満たされないと、データ・セットの OPEN は失敗し、ファイル状況コード 90 が戻されます。

- OS/390 V2R10.0 DFSMS またはそれ以降のリリースを使用している。
- データ・セットをテープとするように定義している。

既存のブロック化データ・セットの場合は、次のようにすると最も簡単です。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングする。
- DD 名定義に BLKSIZE 値をコーディングしない。

DD 名定義から BLKSIZE を省くと、ブロック・サイズは、システムによってデータ・セット・ラベルから自動的に取得されます。

LBI の利用

大きなブロック・サイズのためのラージ・ブロック・インターフェース (LBI) を提供する OS/390 V2R10.0 DFSMS 以上を使用すると、テープ・データ・セットのパフォーマンスを向上させることができます。この機能が使用可能であれば、COBOL 実行時には、システム決定のブロック・サイズを使用するテープ・ファイルに対して、この機能が自動的に使用されます。LBI は、JCL または BLOCK CONTAINS 文節でブロック・サイズが明示的に定義されるファイルでも、使用されます。LBI を使用すると、ブロック・サイズが 32760 (磁気テープ装置でサポートされる場合) を超えても構いません。

LBI はどんな場合にでも使用されるわけではありません。以下の場合には、32760 を超えるブロック・サイズを使用しようとすると、コンパイル時に診断され、OPEN 障害となります。

- スパン・レコード
- OPEN I-O

32760 を超えるブロック・サイズを使用すると、別のシステムでテープを読み取れなくなる場合があります。32760 より大きなブロック・サイズで作成したテープは、OS/390 V2R10.0 DFSMS またはそれ以降のリリースを使用し、かつ 32760 よ

り大きなブロック・サイズをサポートする磁気テープ装置を持つ MVS システムでしか読み取ることができません。ファイル、装置、またはオペレーティング・システム・レベルに関して、指定したブロック・サイズが大きすぎると、ランタイム・メッセージが出されます。

システム決定のブロック・サイズを 32760 に限定するには、どこにも BLKSIZE を指定せずに、以下のいずれかを 32760 に設定してください。ブロック・サイズの限界値は、以下の優先順位で提供される最初の非ゼロ値です。

1. データ・セットに対する DD ステートメントで BLKSZLIM キーワードを指定する。
2. BLKSZLIM キーワードを使用して、データ・クラスの BLKSZLIM をシステム・プログラマーに設定してもらう。
3. SYS1.PARMLIB の DEVSUP*xx* メンバー内に、キーワード TAPEBLKSZLIM を使用して、システムのブロック・サイズ限界値を設定するようシステム・プログラマーに依頼する。

BLKSIZE または BLKSZLIM 値をどのソースからも利用できないと、システムは BLKSIZE を 32760 に限定します。その後、32760 を超えるブロック・サイズを以下のいずれかの方法で使用可能にすることができます。

- ファイルに対する DD ステートメントで、32760 より大きい BLKSZLIM 値を指定し、COBOL ソースで BLOCK CONTAINS 0 を使用する。
- DD ステートメントまたは COBOL ソースの BLOCK CONTAINS 文節で、32760 より大きな値を BLKSIZE に指定する。

BLKSZLIM は、装置からは独立しています。

ブロック・サイズおよび DCB RECFM サブパラメーター

z/OS のもとでは、DCB RECFM サブパラメーターに S または T オプションをコーディングすることができます。

- 標準ブロック (ファイルの最後のブロックを除き、ファイル内に切り捨てられたブロックまたは埋められていないトラックがないもの) だけを持つ形式 F レコードの場合は、DCB RECFM サブパラメーターで S (標準) オプションを使用してください。S は、テープ上のレコードに関してもサポートされます。これは、レコードが DASD またはテープ上にない場合には無視されます。

  この標準ブロック・オプションを使用すると、特に直接アクセス装置の場合は、入出力のパフォーマンスが向上する可能性があります。
- QSAM ファイルの場合、T (トラック・オーバーフロー) オプションはもう有用ではありません。

関連タスク

133 ページの『COBOL での QSAM ファイルおよびレコードの定義』

z/OS DFSMS: Using Data Sets

関連参照

BLOCK CONTAINS 文節 (*Enterprise COBOL 言語解説書*)

QSAM ファイルの入出力ステートメントのコーディング

QSAM ファイル、または HFS 内のバイト・ストリーム・ファイルを QSAM を使用して処理するには、次の入出力ステートメントをコーディングします。

OPEN ファイルをプログラムで利用できるようにします。

どの QSAM ファイルも、INPUT、OUTPUT、または EXTEND (装置の能力に応じて) としてオープンすることができます。

直接アクセス記憶装置上の QSAM ファイルは、I-O としてオープンすることもできます。HFS ファイルを I-O としてオープンすることはできません。オープンしようとすると、ファイル状況コード 37 が出されます。

READ ファイルからレコードを読み取ります。

順次処理では、プログラムは、レコードをファイルの作成時に入力されたのと同じ順序で 1 つずつ読み取ります。

WRITE ファイル内にレコードを作成します。

プログラムは、ファイルの終わりに新しいレコードを書き込みます。

REWRITE
レコードを更新します。HFS 内のファイルは REWRITE を使用して更新することはできません。

CLOSE ファイルとプログラムの間の接続を解放します。

関連タスク
『QSAM ファイルのオープン』
147 ページの『QSAM ファイルへのレコードの追加』
147 ページの『QSAM ファイルの更新』
147 ページの『QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み』
148 ページの『QSAM ファイルのクローズ』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)
REWRITE ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのオープン

プログラムで READ、WRITE、または REWRITE ステートメントを使用してファイルのレコードを処理するためには、その前に、そのファイルを OPEN ステートメントでオープンしておかなければなりません。

OPEN ステートメントが機能するのは、以下の両方の条件が真である場合です。

- ファイルが使用可能であるか、または動的に割り振られている。
- ファイルの DD 名定義またはデータ・セット・ラベルでコーディングされた*固定ファイル属性* が、COBOL プログラムの SELECT および FD ステートメントでそのファイルにコーディングされた属性と一致する。

ファイル編成属性、コード・セット、最大レコード・サイズ、またはレコード・タイプ (固定長または可変長) が一致していないと、ファイル状況コード 39 が戻され、OPEN ステートメントは失敗します。HFS のファイルのオープン時には、最大レコード・サイズおよびレコード・タイプが一致していなくても、エラーと見なされません。

固定長 QSAM ファイルの場合は、FD に RECORD CONTAINS 0 をコーディングすると、レコード・サイズ属性の矛盾が起こりません。レコード・サイズは DD ステートメントまたはデータ・セット・ラベルから取られ、OPEN ステートメントは正常に終了します。

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

テープ・ファイルを逆順に処理するには、OPEN ステートメントの REVERSED オプションを使用してください。OPEN ステートメントを実行すると、ファイルはその終端に位置付けられます。後続の READ ステートメントでは、データ・レコードが最後のレコードから逆順に読み取られます。REVERSED オプションがサポートされるのは、固定長レコードを持つファイルの場合だけです。

関連タスク
『CBLQDA による QSAM ファイルの動的作成』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)

CBLQDA による QSAM ファイルの動的作成

ファイルが z/OS 上で使用可能であると見なされるのは、DD ステートメント、export コマンド (環境変数)、または TSO ALLOCATE コマンドを使用してオペレーティング・システムに認識されている場合です。(可用性という点では、DD ステートメントの DD 名のミススペルは、DD ステートメントが欠落しているのと同じです。また、無効な値を指定した環境変数は、変数が設定されていないのと同じです。)

ある QSAM ファイルがオペレーティング・システム上で利用可能になっていないときに、COBOL 言語がそのファイルの作成を定義することがあります。ランタイム・オプション CBLQDA を使用しており、以下の状況のいずれか 1 つが存在していれば、そのファイルが暗黙的に作成されます。

- OPTIONAL 句に関係なく、そのファイルが OUTPUT 用にオープンされている。
- OPTIONAL ファイルが EXTEND または I-O としてオープンされている。

  オプショナル・ファイルとは、プログラムが実行されるたびに、必ずしも存在する必要がないファイルです。INPUT、I-O、または EXTEND モードでオープンされたファイルについては、FILE-CONTROL 段落で SELECT OPTIONAL 句を使用すれば、それらをオプショナル・ファイルとして定義することができます。

ファイルは、インストール先で設定されたシステム・デフォルト属性と、プログラムの SELECT および FD ステートメントでコーディングされた属性を用いて割り振られます。

この暗黙の割り振りメカニズムを、環境変数を使用して行うファイルの動的割り振りと混乱しないでください。明示動的割り振りでは、有効な環境変数が設定されていなければなりません。この CBLQDA サポートが使用されるのは、上述されたように QSAM ファイルが利用できず、有効な環境変数が設定されていない場合だけです。

z/OS のもとでは、CBLQDA オプションを使用して作成されるファイルは一時データ・セットであり、プログラムの実行後は存在しません。

関連タスク
145 ページの『QSAM ファイルのオープン』

QSAM ファイルへのレコードの追加

QSAM ファイルに追加を行うためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

I-O としてオープンされたファイルにレコードを追加したい場合には、ファイルをいったんクローズしてから、EXTEND としてオープンしなければなりません。

関連参照
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルの更新

QSAM ファイルを更新できるのは、それが直接アクセス記憶装置に存在する場合のみです。HFS のファイルを更新することはできません。

次のようにして、既存のレコードを同じ長さの別のレコードで置き換えてください。

1. ファイルを I-O としてオープンします。
2. REWRITE を使用してファイル内の既存のレコードを更新します。(REWRITE の前の最後のファイル処理ステートメントは、成功した READ ステートメントでなければなりません。)

関連参照
REWRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み

COBOL には、印刷されるページのサイズを制御するため、およびレコードの垂直位置決めを制御するための言語ステートメントが用意されています。

ページ・サイズの制御

印刷されるページのサイズ (ページの上部マージンと下部マージンの間の行数および脚注域の行数) を制御するには、FD 記入項目の LINAGE 文節を使用します。LINAGE 文節を使用すると、COBOL では ADV コンパイラー・オプションも要求されたかのようにファイルを処理します。

LINAGE 文節を WRITE BEFORE/AFTER ADVANCING *nn* LINES と組み合わせて使用する場合は、注意して値を設定してください。ADVANCING *nn* LINES 文節が使用されると、COBOL はまず、LINAGE-COUNTER と *nn* の合計を計算します。それ以降のアクションは、*nn* のサイズによって異なります。LINAGE-COUNTER が増加されると、END-OF-PAGE 命令ステートメントが実行されます。したがって、LINAGE-COUNTER は、END-OF-PAGE ステートメントの実行時に、現行の脚注域ではなく次の論理ページを指していることがあります。

AT END-OF-PAGE または NOT AT END-OF-PAGE 命令ステートメントが実行されるのは、書き込み操作が正常に完了した場合だけです。書き込み操作が成功しなかった場合には、すべての条件句が省略されて、制御が WRITE ステートメントの終わりに移動します。

レコードの垂直位置決めの制御

印刷ページに書き込むそれぞれのレコードの垂直位置決めを制御するには、WRITE ADVANCING ステートメントを使用します。

. . . BEFORE ADVANCING は、ページが送られる前にレコードを印刷します。
. . . AFTER ADVANCING は、ページが送られた後でレコードを印刷します。

ADVANCING の後に、ページが送られることになる行数を整数 (または *mnemonic-name* を指定した *identifier*) で指定してください。WRITE ステートメントから ADVANCING オプションを省略すると、以下の指定と同じ結果になります。

```
AFTER ADVANCING 1 LINE
```

関連参照

WRITE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのクローズ

CLOSE ステートメントを使用すると、プログラムを QSAM ファイルから切り離すことができます。すでにクローズされているファイルをクローズしようとすると、論理エラーになります。

QSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。ただし、実行単位内の OS/VS COBOL プログラムのいずれかで定義されたファイルを除きます。

- 実行単位が正常に終了すると、実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP(ON) ランタイム・オプションが設定されていれば、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 言語環境プログラムの条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが処理を再開する場合は、実行単位の中で COBOL プログラム (再度呼び出されて再入される可能性のある) のいずれかで定義されている、オープン状態のすべてのファイルがクローズされます。

(条件が処理された後で) プログラムが実行を再開する位置を変更することができます。このためには、言語環境プログラムの CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp のような HLL 言語構造体を使用します。

- COBOL サブプログラムに CANCEL を使用すると、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

このような暗黙の CLOSE 操作が実行されたときは、WORKING-STORAGE SECTION に定義したファイル状況キー・データ項目が設定されますが、EXCEPTION/ERROR および LABEL 宣言は呼び出されません。

マルチスレッド・アプリケーションで QSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようとすると、ファイル状況コードの条件 90 でクローズが失敗します。

関連参照
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

QSAM ファイルのエラーの処理

入出力ステートメントの操作が失敗しても、COBOL がユーザーに代わって訂正処置を取るわけではありません。重大エラーよりは軽度の入出力エラーの場合は、エラーが起こった後でプログラムの実行を継続するかどうかを選択してください。

COBOL は、特定の QSAM 入出力エラーを代行受信して処理するために、以下の方法を提供します。

- ファイル終わり句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 文節
- INVALID KEY 句

FILE STATUS キーまたは宣言をコーディングしなかった場合に、重大な QSAM 処理エラーが起こると、メッセージが出され、言語環境プログラム条件が合図されます。ランタイム・オプション ABTERMENC(ABEND) が指定されていると、この条件の結果として、異常終了が引き起こされます。

FILE STATUS 文節または EXCEPTION/ERROR 宣言を使用する場合は、そのファイルに対する DD ステートメントの DCB で EROPT=ACC をコーディングしてください。そうしなければ、COBOL プログラムは、いくつかのエラー条件が発生した後で処理を継続することができなくなります。

FILE STATUS 文節を使用する場合は、必ずキーを調べ、キー値に基づいて適切なアクションを取るようにしてください。キーを調べなくても、プログラムが実行を継続できる可能性がありますが、予期したものではない結果になることがあります。

関連タスク
250 ページの『入出力操作でのエラーの処理』

QSAM ファイルに対する作業

ここでは、以下のことを説明します。

- 『QSAM ファイルの定義および割り振り』
- 152 ページの『QSAM ファイルの検索』
- 154 ページの『ファイル属性をプログラムと一致させる』
- 156 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

関連参照
157 ページの『QSAM ファイル用のバッファーの割り振り』

QSAM ファイルの定義および割り振り

QSAM ファイルまたは HFS 内の バイト・ストリーム・ファイルは、DD ステートメントまたは環境変数を使用して定義することができます。環境変数を使用する場合、名前は大文字でなければなりません。これらのファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。環境変数を使用して QSAM ファイルを定義する場合は、MVS データ・セットを次のように指定してください。

DSN(*dataset-name*) または DSN(*dataset-name*(*member-name*))。*dataset-name* は完全に修飾されていなければならず、一時データ・セットにすることはできません (& で開始してはなりません)。

オプションとして、DSN の後に、以下の属性を任意の順序で指定することができます。

- 後処理の値 (NEW、OLD、SHR、または MOD)
- TRACKS または CYL
- SPACE(*nnn*,*mmm*)
- VOL(*volume-serial*)
- UNIT(*type*)
- KEEP、DELETE、CATALOG、または UNCATALOG
- STORCLAS(*storage-class*)
- MGMTCLAS(*management-class*)
- DATACLAS(*data-class*)

環境変数または DD 定義を使用して、HFS のファイルを定義することができます。これを行うには、ASSIGN 文節の外部名と一致する名前を使用して、以下のいずれか 1 つを定義してください。

- PATH='*absolute-path-name*' および FILEDATA=BINARY を使用する DD 割り振り
- PATH(*pathname*) の値を持つ環境変数 (*pathname* は / で始まる絶対パス名)

COBOL (OS/390 および VM 版) バージョン 2 リリース 2 より前の COBOL のリリースとの互換性を与えるために、HFS ファイルの DD 割り振りを使用するときに FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。HFS

のテキスト・ファイルを処理する場合は、LINE SEQUENTIAL 編成を使用してください。QSAM を使用して HFS のテキスト・ファイルを処理する場合は、環境変数を使用してファイルを定義することはできません。

QSAM ファイルを定義するときは、指定されたパラメーターを使用して以下の作業を実行してください。

目的	使用する DD パラメーター	使用する EV キーワード
ファイルに名前を付ける。	DSNAME (データ・セット名)	DSN
ファイルに割り振る入出力装置のタイプと数量を選択する。	UNIT	タイプの UNIT の場合のみ
ファイルが常駐するボリュームと、ボリュームの取り付けについて指示を与える。	VOLUME (またはシステムに出力ボリュームを選択させる)	VOL
ファイルに必要なスペースのタイプおよび量を割り振る (直接アクセス記憶装置の場合のみ)。	SPACE	スペースの量 (1 次と 2 次のみ) の場合は、SPACE。スペースのタイプの場合は、TRACKS または CYL。
ファイルに関連付けられるラベルのタイプ (および内容の一部) を指定する。	LABEL	n/a
ジョブ・ステップの完了後にファイルをカタログするか、後続ステップに渡すか、保存するかを指示する。	DISP	NEW、OLD、SHR、MOD に、KEEP、DELETE、CATALOG、または UNCATALOGの指定を追加したもの。
追加したいすべてのデータ制御ブロック情報を完成させる。	DCB サブパラメーター	n/a

QSAM ファイルに関する情報の中には、FILE-CONTROL 記入項目、FD 記入項目、およびその他の COBOL 文節で必ずコーディングしなければならないものがあります。その他の情報は、出力ファイルについての DD ステートメントまたは環境変数でコーディングする必要があります。入力ファイルの場合、システムはファイル・ラベルから情報を取得することができます (標準ラベル・ファイルの場合)。入力ファイルについての DD ステートメントで DCB 情報が指定されると、それはデータ・セット・ラベル上の情報をオーバーライドします。たとえば、新規の直接アクセス装置ファイルに割り振られるスペースの量は、DD ステートメントで SPACE パラメーターを使用して設定することができます。

QSAM ファイルの一部の特性は、COBOL 言語では表現できませんが、ファイルの DD ステートメントで DCB を使用してコーディングすることができます。DCB パラメーターのサブパラメーターで、システムがデータ・セット定義を完成させるのに必要な、以下のものを含む情報をコーディングしてください。

- コンパイル時に、BLOCK CONTAINS 0 RECORDS がコーディングされた場合の、ブロック・サイズ (BLKSIZE=) (推奨方法です)。
- レコードの書き込みまたは読み取りにおいてエラーが発生した場合に実行されるオプション。

- TRACK OVERFLOW または標準ブロック。
- カード読取装置または穿孔装置の操作モード。

DD DUMMY にコーディングされた DCB 属性は、COBOL プログラムの FD 記入項目にコーディングされた属性をオーバーライドしません。

関連タスク

142 ページの『ブロック・サイズの設定』
133 ページの『COBOL での QSAM ファイルおよびレコードの定義』
129 ページの『ファイルの割り振り』

関連参照

『QSAM ファイルを作成するためのパラメーター』
MVS JCL 解説書

QSAM ファイルを作成するためのパラメーター

次に示す DD ステートメントのパラメーターは、しばしば、QSAM ファイルを作成するために使用されます。

```
DSNAME=  [ dataset-name                  ]
DSN=     [ dataset-name(member-name)     ]
         [ &&name                        ]
         [ &&name(member-name)           ]

UNIT=    ( name[,unitcount] )

VOLUME=  ( [PRIVATE] [,RETAIN] [,vol-sequence-num] [,volume-count] ...
VOL=
                          ... [,SER=(volume-serial[,volume-serial]...)] )
                              [,REF=[dsname                          ]
                                    [*.ddname                        ]
                                    [*.stepname.ddname               ]
                                    [*.stepname.procstep.ddname      ]

SPACE=   ([TRK                  ],(primary-quantity[,secondary-quantity][,directory-quantity]))
         [CYL                   ]
         [average-record-length]

LABEL=   ( [data-set-sequece-number,] [ NL  ] [,EXPDT= [ yyddd    ]] )
                                      [ SL  ] [        [ yyyy/ddd ]]
                                      [ SUL ] [,RETPD=xxxx        ]

DISP=    ( [ NEW ] [,DELETE] [,DELETE] )
           [ MOD ] [,KEEP  ] [,KEEP  ]
                   [,PASS  ] [,CATLG ]
                   [,CATLG ]

DCB=     ( subparameter-list )
```

関連タスク

150 ページの『QSAM ファイルの定義および割り振り』

QSAM ファイルの検索

QSAM ファイルは、カタログ化されていてもいなくても、ジョブ制御ステートメントまたは環境変数を使用して検索することができます。

カタログ化されたファイル

ボリュームやスペースなど、すべてのデータ・セット情報は、カタログおよびファイル・ラベルに保管されています。コーディングする必要があるのは、データ・セット名と後処理方法です。DD ステートメントを使用する場合、これは DSNAME パラメーターと DISP パラメーターです。環境変数を使用する場合、これは DSN パラメーターとパラメーター OLD、SHR、または MOD のいずれか 1 つです。

カタログ化されていないファイル

一部の情報はファイル・ラベルに保管されていますが、*dsname* および後処理方法のほかにユニットおよびボリューム情報をコーディングしなければなりません。

JCL を使用しており、そのファイルを現行ジョブ・ステップまたは現行ジョブ内の前のジョブ・ステップで作成した場合は、前の DD ステートメントを参照してデータ・セット情報の大部分を得ることができます。ただし、DSNAME および DISP はコーディングする必要があります。

関連参照

『QSAM ファイルを検索するためのパラメーター』

QSAM ファイルを検索するためのパラメーター

次に示す DD ステートメントのパラメーターは、前に作成したファイルを検索するために使用されます。

```
DSNAME=  [ dataset-name                    ]
DSN=     [ dataset-name(member-name)       ]
         [ *.ddname                        ]
         [ *.stepname.ddname               ]
         [ &&name                          ]
         [ &&name(member-name)             ]

UNIT=    ( name[,unitcount] )

VOLUME=  ( subparameter-list )
VOL=

LABEL=   ( subparameter-list )


DISP=    ( [ OLD    ] [ ,DELETE  ] [ ,DELETE  ] )
           [ SHR    ] [ ,KEEP    ] [ ,KEEP    ]
           [ MOD    ] [ ,PASS    ] [ ,CATLG   ]
                      [ ,CATLG   ] [ ,UNCATLG ]
                      [ ,UNCATLG ]

DCB=     ( subparameter-list )
```

関連タスク

152 ページの『QSAM ファイルの検索』

ファイル属性をプログラムと一致させる

ファイルの DD ステートメントまたはデータ・セット・ラベルでコーディングされた固定ファイル属性と、COBOL プログラムの SELECT および FD ステートメントでそのファイル用にコーディングされた属性が一致していないと、プログラム内の OPEN ステートメントが機能しないことがあります。ファイル編成、レコード形式 (固定長または可変長)、レコード長、またはコード・セットの属性が一致していないと、ファイル状況コードが 39 になり、OPEN ステートメントが失敗します。HFS のファイルについては、例外があります。すなわち、レコード形式およびレコード長が一致していなくても、HFS ではエラーとなりません。

よくあるファイル状況 39 の問題を避けるには、以下にリストされている、既存のファイル、新規のファイル、または COBOL によって動的に作成されるファイルの処理に関する指針に従ってください。

JCL または環境変数内の情報は、データ・セット・ラベル内の情報をオーバーライドするので注意してください。

既存ファイルの処理

プログラムで既存のファイルを処理するときには、COBOL プログラム内のファイルの記述をデータ・セットのファイル属性と矛盾しないようにコーディングします。最大レコード長を定義するには、以下の指針を使用してください。

形式	指定内容
V または S	データ・セットの長さ属性よりも 4 バイト小さくする。
F	データ・セットの長さ属性と同じ。
U	データ・セットの長さ属性と同じ。

可変長 (形式 V) レコードの定義

プログラム内で可変長レコードを定義する最も簡単な方法は、FD 記入項目で RECORD IS VARYING FROM *integer-1* TO *integer-2* を使用し、*integer-2* に適切な値を設定することです。たとえば、データ・セットの長さ属性が 104 (LRECL=104) であると判別している場合を想定してください。最大レコード長がレベル 01 レコード記述ではなく RECORD IS VARYING 文節 (値が設定されている) から判別されることを念頭に置いて、プログラム内で以下のコードを使って形式 V ファイルを定義することができます。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS V
    RECORD IS VARYING FROM 4 TO 100 CHARACTERS.
01  COMMUTER-RECORD-A           PIC X(4).
01  COMMUTER-RECORD-B           PIC X(75).
```

形式 U レコードの定義

前の例の既存のファイルが形式 V ではなく形式 U であった場合を想定してください。104 バイトがすべてユーザー・データである場合、プログラム内で以下のコードを使ってファイルを定義することができます。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS U
```

```
    RECORD IS VARYING FROM 4 TO 104 CHARACTERS.
01  COMMUTER-RECORD-A           PIC X(4).
01  COMMUTER-RECORD-B           PIC X(75).
```

固定長レコードの定義

プログラム内で固定長レコードを定義するには、RECORD CONTAINS *integer* 文節を使用するか、またはすべてのレベル 01 レコード記述を同じ固定サイズにコーディングしてください。いずれのケースでも、データ・セット属性の長さの値と等しい値を使用してください。同じプログラムで実行時にいくつかの異なるファイルを処理することを計画していて、ファイルごとに固定長レコードの長さが異なる場合に、レコード長の矛盾を避けるための推奨される方法は、RECORD CONTAINS 0 をコーディングすることです。

既存のファイルが ASCII データ・セット (DCB=(OPTCD=Q)) である場合には、プログラムのファイルについての FD 記入項目で CODE-SET 文節を使用しなければなりません。

新規ファイルの処理

COBOL プログラムで、プログラムの実行前に使用可能にされる新しいファイルにレコードを書き込む場合は、DD ステートメント、環境変数、または割り振りにおいてコーディングするファイル属性が、プログラムでコーディングした属性と矛盾しないようにしてください。通常、ファイルを事前定義する場合には、最小限のパラメーターをコーディングするだけで済みます。

データ・セットの長さ属性を明示的にコーディングすることが必要であるとき (たとえば ISPF 割り振りパネルを使用する場合、または DD ステートメントがバッチ・ジョブ用のもので、その中でプログラムが RECORD CONTAINS 0 を使用する場合) は、次のようにしてください。

- 形式 V および形式 S ファイルの場合は、プログラム内で定義されているものよりも 4 バイト大きい長さ属性を設定します。
- 形式 F および形式 U ファイルの場合は、プログラム内で定義されているものと同じ長さ属性を設定します。
- ファイルを OUTPUT としてオープンし、それをプリンターに書き込む場合は、ADV コンパイラー・オプションと、プログラム内の COBOL 言語によって、コンパイラーがレコード長に紙送り制御文字のための 1 バイトを追加することがあります。そのような場合には、LRECL をコーディングする際に、追加されるバイトを長さに含めてください。

たとえば、プログラムに、可変長レコードを持つファイルについて以下のコードが含まれているとします。

```
FILE SECTION.
FD  COMMUTER-FILE-MST
    RECORDING MODE IS V
    RECORD CONTAINS 10 TO 50 CHARACTERS.
01  COMMUTER-RECORD-A           PIC X(10).
01  COMMUTER-RECORD-B           PIC X(50).
```

DD ステートメントまたは割り振りにおける LRECL は 54 になります。

COBOL によって動的に作成されるファイルの処理

ファイルが DD ステートメントまたは TSO ALLOCATE コマンドによって使用可能にされていないときに、COBOL プログラムでそのファイルの作成が定義されると、Enterprise COBOL がファイルを動的に割り振ります。ファイルがオープンされるときには、プログラム内でコーディングされたファイル属性が使用されます。このため、ファイル属性の矛盾を心配する必要はありません。

関連タスク

135 ページの『固定長形式の要求』
136 ページの『可変長形式の要求』
141 ページの『不定形式の要求』
146 ページの『CBLQDA による QSAM ファイルの動的作成』

ストライプ拡張形式 QSAM データ・セットの使用

ストライプ拡張形式 QSAM データ・セットは、複数のボリュームにわたり、並列データ・アクセスを認める拡張形式の QSAM データ・セットです。

ストライプ拡張形式 QSAM データ・セットを使用して効果があるのは、次のような特性を持つアプリケーションです。

- アプリケーションが、大量のデータが含まれたファイルを処理する。
- ファイルへの入出力操作にかかる時間が、全体のパフォーマンスに大きな影響を与えている。

QSAM ストライプ・データ・セットを使用することから最大限の効果を得るためには、DFSMS が必要とされる数のバッファーを 16MB 境界より上に割り振ることができなければなりません。

QSAM ストライプ・データ・セットに割り振られるファイルを含むアプリケーションを開発するときには、次のようにしてください。

- バッファーを 16MB 境界より上に割り振ることができないファイルについては、QSAM ストライプ・データ・セットの使用を避けます。
- ファイルについての FILE-CONTROL 段落の記入項目で RESERVE 文節を省略します。RESERVE 文節を省略すると、DFSMS がデータ・セットについての最適なバッファー数を判別できるようになります。
- DATA(31) および RENT コンパイラー・オプションを指定してプログラムをコンパイルし、ロード・モジュールを AMODE(31) にします。
- ファイルが形式 F、形式 V、または形式 U レコードを持つ EXTERNAL ファイルである場合は、ALL31(ON) ランタイム・オプションを指定します。

すべてのストライプ・データ・セットは拡張フォーマット・データ・セットですが、すべての拡張フォーマット・データ・セットがストライプ・データ・セットというわけではありません。

関連タスク

z/OS DFSMS: Using Data Sets の『performance considerations』の項

関連参照

157 ページの『QSAM ファイル用のバッファーの割り振り』

QSAM ファイル用のバッファーの割り振り

DFSMS は、QSAM ファイルの入出力を保管するためのバッファーを、使用されるファイルに応じて、16MB 境界より上または下に自動的に割り振ります。ほとんどの QSAM ファイルのバッファーは、16MB 境界より上に割り振られます。例外は次のとおりです。

- AMODE 24 で実行されるプログラム。
- DATA(24) および RENT オプションを指定してコンパイルされたプログラム。
- NORENT および RMODE(24) オプションを指定してコンパイルされたプログラム。
- NORENT および RMODE(AUTO) オプションを指定してコンパイルされたプログラム。
- EXTERNAL ファイル (ALL31(OFF) ランタイム・オプションが指定されている場合)。ALL31(ON) ランタイム・オプションを指定するためには、実行単位内のすべてのプログラムが 31 ビット・アドレッシング・モードで実行可能でなければなりません。
- TSO 端末に割り振られるファイル。
- 形式 S (スパン) レコードを持つファイル (ファイルが以下のいずれかである場合)。
  - EXTERNAL ファイル (ALL31(ON) オプションが指定されている場合でも)
  - I-O-CONTROL 段落の SAME RECORD AREA 文節で指定されたファイル
  - I-O としてオープンされ、REWRITE ステートメントを使って更新されるブロック化ファイル

関連概念

38 ページの『ストレージとそのアドレス可能度』

関連タスク

156 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

QSAM を使用する HFS ファイルへのアクセス

階層ファイル・システム (HFS) 内のバイト・ストリーム・ファイルを、QSAM を使用して ORGANIZATION SEQUENTIAL ファイルとして処理できます。これを行うためには、次のいずれか 1 つを ASSIGN 文節の *assignment-name* として指定してください。

ddname

キーワード PATH= および FILEDATA=BINARY によってファイルを識別する DD 割り振り。

環境変数名

ファイルの HFS パスの実行時値を持つ環境変数

以下の制約事項に従ってください。

- スパン・レコード形式はサポートされません。
- OPEN I-O および REWRITE はサポートされません。このいずれかの操作を試みると、次のファイル状況条件を受け取ります。
  - OPEN I-O からは 37。

– REWRITE からは 47 (ファイルを I-O として正常にオープンできなかったため)。

使用上の注意

- 次のいずれかについて、ファイル状況 39 (固定ファイル属性の矛盾) が強制されることはありません。
  – レコード長の矛盾
  – レコード・タイプの矛盾 (固定長と可変長)
- READ は、ファイルの最大論理レコード・サイズに等しいバイト数を戻します (ただし、最後のレコードは別で、それより短い可能性があります)。

  たとえば、ファイル定義に、3、5、および 10 バイトの長さの 01 レコード記述が含まれており、この 3 つのレコード、'abc'、'defgh'、および 'ijklmnopqr' をこの順序で書き込むとします。このファイルの最初の READ は 'abcdefghij' を戻し、2 番目の READ は 'klmnopqr ' を戻し、3 番目の READ は AT END 条件となります。

COBOL (OS/390 および VM 版) バージョン 2 リリース 2 より前の IBM COBOL の リリースとの互換性を与えるために、HFS ファイルの DD 割り振りを使用するときに、FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。HFS のテキスト・ファイルを処理する場合は、LINE SEQUENTIAL 編成を使用してください。QSAM を使用して HFS のテキスト・ファイルを処理する場合は、環境変数を使用してファイルを定義することはできません。

関連タスク

129 ページの『ファイルの割り振り』
150 ページの『QSAM ファイルの定義および割り振り』
z/OS DFSMS: Using Data Sets の『Accessing HFS files via BSAM and QSAM』の項

QSAM ファイルのラベル

ラベルを使用して、磁気テープ、直接アクセス・ボリューム、およびデータ・セットを識別することができます。オペレーティング・システムでは、ラベル処理ルーチンを使用して、ラベルを識別および検査し、ボリュームとデータ・セットの位置を突き止めます。

標準と標準外の 2 種類のラベルがあります。IBM Enterprise COBOL for z/OS and OS/390 では、標準外のユーザー・ラベルはサポートされません。さらに、標準ユーザー・ラベルには、関連データ・セットに関するユーザー指定の情報も含まれます。

標準ラベルは、ボリューム・ラベルとデータ・セット・ラベルのグループから構成されます。ボリューム・ラベルは、ボリューム上のデータの前または後に置かれ、そのボリュームを識別および記述します。データ・セット・ラベルは、ボリューム上の各データ・セットの前または後に置かれ、そのデータ・セットを識別および記述します。

- データ・セットの前に置かれるデータ・セット・ラベルは、ヘッダー・ラベル と呼ばれます。

- データ・セットの後ろに置かれるデータ・セット・ラベルは、*トレーラー・ラベル* と呼ばれます。
  トレーラー・ラベルは、データ・セット内のブロックのカウントを含む点を除けば、ヘッダー・ラベルと似ています。
- データ・セット・ラベル・グループには、任意で、標準ユーザー・ラベルを組み込むことができます。
- ボリューム・ラベル・グループには、任意で、標準ユーザー・ラベルを組み込むことができます。

関連タスク
『トレーラー・ラベルおよびヘッダー・ラベルの使用』

関連参照
160 ページの『標準ラベルの形式』

トレーラー・ラベルおよびヘッダー・ラベルの使用

ユーザー・ラベルは、データ・セットまたはボリューム (リール) の始まりまたは終わりに達したときに作成、検査、または更新することができます。ボリューム終わり出口とボリューム開始出口を使用できます。中間トレーラーおよびヘッダーを作成したり、検査することもできます。

データ・セットの各ボリューム上で、最大 8 個のヘッダー・ラベルと 8 個のトレーラー・ラベルを作成、検査、または更新することができます。(QSAM EXTEND は、ファイル開始ラベルが処理されない点を除き、OUTPUT と同じ働きをします。) ラベルは、マルチボリューム・データ・セットの場合、最初のボリュームに置かれます。このボリュームは、トレーラー・ラベルが作成、検査、または更新される予定である場合には、CLOSE として取り付けなければなりません。INPUT または I-O としてオープンされたファイルのトレーラー・ラベルは、AT END 条件に達しているファイルに対して CLOSE ステートメントが実行されるときに処理されます。

誤った位置番号を指定してヘッダーまたはトレーラーをコーディングすると、結果は予測できません。(データ管理は、ラベルを強制的に正しい相対位置にすることがあります。)

標準ラベル処理を使用するときには、データ・セットを記述する DD ステートメントで標準およびユーザー・ラベルのラベル・タイプ (SUL) をコーディングしてください。

ユーザー・ラベル・トラックの入手

直接アクセス・ボリュームに対して SUL の LABEL サブパラメーターを使用すると、データ・セットの作成時に、別個のユーザー・ラベル・トラックが割り振られます。この追加のトラックは初期割り振り時に割り振られ、順次データ・セットの場合はボリュームの終わり (ボリューム切り替え) 時に割り振られます。ユーザー・ラベル・トラック (順次データ・セットの各ボリュームにつき 1 つ) には、ユーザー・ヘッダー・ラベルとユーザー・トレーラー・ラベルの両方が含まれます。LABEL 名がユーザー LABEL 宣言の外側で参照されると、結果は予測できません。

ユーザー・ラベルの処理

USE AFTER LABEL 宣言は、サポートされるファイル上のユーザー・ラベルを処理するためのプロシージャーを提供します。AFTER オプションは、標準ユーザー・ラベルの処理を示します。

ラベルは、ファイルの FD 記入項目で、LABEL RECORDS 文節の *data-names* としてリストしてください。

ファイルのオープンのモード	次のような条件の場合:	結果:
INPUT	USE . . . LABEL 宣言が OPEN オプションまたはファイルにコーディングされている。	ラベルが読み取られ、制御が LABEL 宣言に渡されます。
OUTPUT	USE . . . LABEL 宣言が OPEN オプションまたはファイルにコーディングされている。	ラベル用のバッファー域が提供され、制御が LABEL 宣言に渡されます。
INPUT または I-O	AT END 条件に達しているファイルに対して CLOSE ステートメントが実行される。	トレーラー・ラベルを処理するために制御がLABEL 宣言に渡されます。

GO TO MORE-LABELS ステートメントを使用して、特殊な出口を指定することができます。このステートメントの結果として、ラベル DECLARATIVE SECTION から出た場合は、システムは次のいずれか 1 つを行います。

- 現行の開始または終了ラベルを書き込んだ後、ラベルをさらに作成するために USE セクションの始まりに再入します。最後のラベルを作成した後、システムはセクションの最後のステートメントを実行することによって終了します。
- 追加の開始または終了ラベルを読み取った後、ラベルをさらに検査するために USE セクションの始まりに再入します。ユーザー・ラベルの処理時には、検査する別のユーザー・ラベルがある場合にのみ、システムはそのセクションに再入します。このため、セクションの最後のステートメントを通り抜けるプログラム・パスは必要ありません。

ユーザー・ラベルについて GO TO MORE-LABELS ステートメントが実行されない場合には、その直後のユーザー・ラベルを検査または作成するための、DECLARATIVE SECTION への再入は行われません。

関連概念

158 ページの『QSAM ファイルのラベル』

標準ラベルの形式

標準ラベルは、EBCDIC または ASCII で記録される 80 文字のレコードです。最初の 4 文字は、常に、ラベルを識別するために使用されます。次の図は、テープの場合のこれらの *identifier* を示しています。

ID	説明
VOL1	ボリューム・ラベル
HDR1 または HDR2	データ・セット・ヘッダー・ラベル

ID	説明
EOV1 または EOV2	データ・セット・トレーラー・ラベル (ボリュームの終わり)
EOF1 または EOF2	データ・セット・トレーラー・ラベル (データ・セットの終わり)
UHL1 ～ UHL8	ユーザー・ヘッダー・ラベル
UTL1 ～ UTL8	ユーザー・トレーラー・ラベル

直接アクセス・ボリュームのラベルの形式は、テープ・ボリューム・ラベル・グループのラベル・グループの形式とほとんど同じです。違いは、初期 DASTO ボリューム・ラベルのデータ・セット・ラベルが、データ・セット制御ブロック (DSCB) から構成されるということだけです。DSCB は、ボリューム目録 (VTOC) の中にあり、スペース割り振りなどの制御情報のほかに、テープ・データ・セット・ヘッダーおよびトレーラー情報と等価の情報を含んでいます。

標準ユーザー・ラベル

標準ラベル・グループの中では、ユーザー・ラベルの指定はオプショナルです。

ユーザー・ヘッダー・ラベル (UHL1-8) およびユーザー・トレーラー・ラベル (UTL1-8) に使用される形式は、次のいずれかで記録される長さ 80 文字のラベルから構成されます。

- EBCDIC (DASD 上または IBM 標準ラベル付きテープ上)
- ASCII (ISO/ANSI ラベル付きテープ上)

最初の 3 バイトは、以下のいずれのラベルであるかを識別する文字から構成されます。

- ユーザー・ヘッダー・ラベル (データ・セットの始まりにある) を表す UHL
- ユーザー・トレーラー・ラベル (ボリュームの終わりまたはデータ・セットの終わりにある) を表す UTL

次のバイトには、同じタイプのラベルのセットにおけるこのラベルの相対位置が入ります。1 ～ 8 個のラベルが許可されます。

残りの 76 バイトは、ユーザー指定情報から構成されます。

標準ユーザー・ラベルは、QSAM ストライプ・データ・セットについてはサポートされません。

関連概念

158 ページの『QSAM ファイルのラベル』

テープ上の QSAM ASCII ファイルの処理

プログラムで QSAM ASCII ファイルを処理する場合は、以下のことを行ってください。

1. ASCII アルファベットを要求します。
2. レコード形式を定義します。

3. ddname を (JCL を使用して) 定義します。

さらに、プログラムで ASCII ファイルからの数値データ項目を処理する場合には、別個の符号付き数値データ型 (SIGN IS LEADING SEPARATE) を使用してください。

CODEPAGE コンパイラー・オプションは、ASCII テープをサポートするための ASCII と EBCDIC 間での変換に使用されるコード・ページには影響を与えません。ASCII テープのサポートに使用される CCSID がどのように選択されるか、およびデフォルトの CCSID については、z/OS DFSMS の資料を参照してください。

ASCII アルファベットの要求

SPECIAL-NAMES 段落で、ASCII を表す STANDARD-1 をコーディングします。

```
ALPHABET-NAME IS STANDARD-1
```

ファイルについての FD ステートメントで、次のように指定します。

```
CODE-SET IS ALPHABET-NAME
```

レコード形式の定義

QSAM ASCII テープ・ファイルは、以下のいずれかのレコード形式で処理してください。

- 固定長 (形式 F)
- 不定 (形式 U)
- 可変長 (形式 V)

可変長レコードを使用している場合は、形式 D を明示的にコーディングすることはできません。代わりに、RECORDING MODE V をコーディングしてください。この形式情報は内部で D モードに変換されます。D モード・レコードは、レコードごとに 4 バイトのレコード記述子を持ちます。

DD 名の定義

z/OS のもとでは、ASCII ファイルを処理するには、特別な JCL コーディングが必要です。DCB パラメーターの以下のサブパラメーターを、DD ステートメントにコーディングしてください。

BUFOFF=[*L*|*n*]

L ブロック長 (ブロック接頭語を含む) が入る 4 バイトのブロック接頭語。

n ブロック接頭語の長さ。

- 入力の場合は、0 ～ 99
- 出力の場合は、0 または 4

BLOCK CONTAINS 0 をコーディングした場合には、この値を使用してください。

BLKSIZE=*n*

n ブロックのサイズ (ブロック接頭語の長さを含む)。

LABEL=[AL|AUL|NL]

AL 米国標準規格 (ANS) ラベル

AUL ANS およびユーザー・ラベル

NL ラベルなし

OPTCD=*Q*

Q この値は、ASCII ファイルの場合には必須であり、ファイルが Enterprise COBOL を使用して作成された場合はデフォルトです。

関連タスク

『ASCII ファイルのラベルの処理』

z/OS DFSMS: Using Data Sets の『Converting Character Data』の項

ASCII ファイルのラベルの処理

ASCII ファイルの標準ラベル処理は、EBCDIC ファイルの標準ラベル処理と同じです。システムは、ASCII コードを EBCDIC コードに変換してから処理します。

すべての ANS ユーザー・ラベルは任意指定です。ASCII ファイルは、ユーザー・ヘッダー・ラベル (UHL*n*) およびユーザー・トレーラー・ラベル (UTL*n*) を持つことができます。ファイルの始まりおよび終わりにおけるユーザー・ラベルの数に制限はありません。必要な数だけラベルを作成することができます。すべてのユーザー・ラベルは、長さが 80 バイトでなければなりません。

ユーザー・ラベルを作成または検査するには (ユーザー・ラベル出口)、USE AFTER STANDARD LABEL プロシージャーをコーディングしてください。USE BEFORE STANDARD LABEL プロシージャーを使用することはできません。

テープ上の ASCII ファイルは、以下のようにすることができます。

- ANS ラベル
- ANS およびユーザー・ラベル
- ラベルなし

ASCII テープ上のラベルは、ASCII コードのみでなければなりません。ASCII と EBCDIC の組み合わせが入っているテープは、読み取ることができません。

関連タスク

161 ページの『テープ上の QSAM ASCII ファイルの処理』

第 10 章 VSAM ファイルの処理

仮想記憶アクセス方式 (VSAM) は、直接アクセス記憶装置上のファイルに関するアクセス方式です。VSAM では、以下のことが行えます。

- ファイルをロードする。
- ファイルからレコードを検索する。
- ファイルを更新する。
- ファイル内のレコードの追加、置換、および削除を行う。

VSAM 処理には、QSAM と比較して以下の利点があります。

- 無許可アクセスに対するデータの保護。
- システム間の互換性。
- 装置からの独立性 (ブロック・サイズおよびその他の制御情報に留意する必要がありません)。
- より単純な JCL (システムに必要な情報は統合カタログに入れて提供されます)。
- 索引付きファイル編成または相対ファイル編成を使用する機能。

以下に、VSAM 用語が COBOL 用語や他のよく知られている用語とどのように異なるかをリストしています。

VSAM 用語	COBOL 用語	類似した非 VSAM 用語
データ・セット	ファイル	データ・セット
入力順データ・セット (ESDS)	順次ファイル	QSAM データ・セット
キー順データ・セット (KSDS)	索引付きファイル	ISAM データ・セット
相対レコード・データ・セット (RRDS)	相対ファイル	BDAM データ・セット
制御インターバル・サイズ (CISZ)		ブロック・サイズ
バッファー (BUFNI/BUFND)		BUFNO
アクセス方式制御ブロック (ACB)		データ制御ブロック (DCB)
クラスター (CL)		データ・セット
クラスター定義		データ・セット割り振り
JCL DD ステートメントの AMP パラメーター		JCL DD ステートメントの DCB パラメーター
レコード・サイズ		レコード長

以下の説明の中のファイル という用語は、COBOL ファイルまたは VSAM データ・セットのいずれかを指します。

複雑な要件があるか、または VSAM を頻繁に使用する予定である場合には、使用するオペレーティング・システムの VSAM 資料を調べてください。

関連タスク

168 ページの『VSAM ファイル編成およびレコードの定義』
174 ページの『VSAM ファイルの入出力ステートメントのコーディング』
184 ページの『パスワードによる VSAM ファイルの保護』
183 ページの『VSAM ファイルのエラーの処理』

© Copyright IBM Corp. 1991, 2001

185 ページの『z/OS および UNIX のもとでの VSAM データ・セットの操作』
193 ページの『VSAM パフォーマンスの向上』

関連参照
z/OS DFSMS: Using Data Sets
z/OS DFSMS Macro Instructions for Data Sets
z/OS DFSMS: Access Method Services for Catalogs

VSAM ファイル

VSAM データ・セットの物理編成は、他のアクセス方式で使用されるものとかなり異なります。VSAM データ・セットは、制御インターバルおよび制御域 (CA) で保持されます。それらのサイズは、通常、アクセス方式によって判別され、それらが使用される方法について意識することはありません。

VSAM では、次の 3 種類のファイル編成を使用できます。

VSAM 順次ファイル編成
(VSAM ESDS (入力順データ・セット) 編成とも呼ばれます。) VSAM 順次ファイル編成では、レコードは入力された順番に保管されます。VSAM 入力順データ・セットは、QSAM 順次ファイルと同等です。レコードの順序は固定されます。

VSAM 索引付きファイル編成
(VSAM KSDS (キー順データ・セット) 編成とも呼ばれます。) VSAM 索引付きファイル (KSDS) では、レコードは、組み込み基本キー・フィールドの照合シーケンスの定義に従って配列されます。基本キーは、レコード内の 1 つまたは複数の連続する文字から構成されます。基本キーは、レコードを固有に識別し、それがアクセスされる順序を他のレコードとの関連で判別します。レコードの基本キーは、たとえば、従業員番号や送り状番号にすることができます。

VSAM 相対ファイル編成
(VSAM 固定長または可変長 RRDS (相対レコード・データ・セット) 編成とも呼ばれます。) VSAM 相対レコード・セット (RRDS) には、相対キーによって順序付けされたレコードが入ります。相対キーとは、ファイルの始まりからのレコードの相対的な位置を表す相対レコード番号です。相対レコード番号は、固定長または可変長レコードを識別します。

VSAM 固定長 RRDS では、レコードはストレージ内の一連の固定長スロットに入れられます。各スロットは、相対レコード番号に関連付けられています。たとえば、10 個のスロットが入っている固定長 RRDS では、最初のスロットは相対レコード番号 1 であり、10 番目のスロットは相対レコード番号 10 です。

VSAM 可変長 RRDS では、レコードはそれらの相対レコード番号に従って順序付けられます。レコードの保管および検索は、相対レコード番号の設定に従って行われます。

本書では、*VSAM 相対レコード・データ・セット* (または *RRDS*) という用語は、特に区別する必要がない限り、固定長レコードを持つ相対レコード・データ・セットと可変長レコードを持つ相対レコード・データ・セットの両方を指すために使用されています。

次の表は、各種の VSAM データ・セットをいくつかの特性によって比較しています。

特性	入力順データ・セット (ESDS)	キー順データ・セット (KSDS)	相対レコード・データ・セット (RRDS)
レコードの順序	書き込まれた順序	キー・フィールドによる照合シーケンス	相対レコード番号の順序
アクセス	順次	索引を通してキー順	相対レコード番号順 (キーのように処理される)
代替索引	COBOL ではサポートされませんが、1 つまたは複数の代替索引を持つことができます。	1 つまたは複数の代替索引を持つことができます。	代替索引を持つことはできません。
レコードの相対バイト・アドレス (RBA) と相対レコード番号 (RRN)	RBA を変更することはできません。	RBA を変更できます。	RRN を変更することはできません。
レコード追加用のスペース	データ・セットの終わりにあるスペースが使用されます。	レコードを挿入し、それらの長さを適切に変更するためには、分散フリー・スペースが使用されます。	固定長 RRDS の場合は、データ・セット内の空のスロットが使用されます。 可変長 RRDS の場合は、分散フリー・スペースが使用され、追加されるレコードの長さは適切に変更されます。
レコード削除用のスペース	レコードは削除できませんが、それのスペースを同じ長さのレコードのために再利用することはできます。	削除または短縮されたレコードのスペースは、制御インターバルで自動的に再利用されます。	削除されたレコードのスペースを再利用することができます。
スパン・レコード	スパン・レコードを持つことができます。	スパン・レコードを持つことができます。	スパン・レコードを持つことはできません。
作業ファイルとしての再利用	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	再利用することができます。

関連タスク

168 ページの『VSAM ファイルの順次編成の指定方法』
169 ページの『VSAM ファイルの索引編成の指定方法』
170 ページの『VSAM ファイルの相対編成の指定方法』
186 ページの『VSAM ファイルの定義』

VSAM ファイル編成およびレコードの定義

ENVIRONMENT DIVISION の FILE-CONTROL 記入項目を使用して、COBOL プログラム内のファイルの VSAM ファイル編成およびアクセス・モードを定義してください。

DATA DIVISION の FILE SECTION で、そのファイルのファイル記述 (FD) 記入項目をコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。レコードの論理サイズは、RECORD 文節でコーディングします。

重要: COBOL プログラム内では、アクセス方式サービス・プログラムで定義してからでなければ、VSAM データ・セットを処理できません。

次の表に、VSAM ファイル編成、アクセス・モード、およびレコード形式 (固定長または可変長) を要約しています。

ファイル編成	順次アクセス	ランダム・アクセス	動的アクセス	固定長	可変長
VSAM 順次 (ESDS)	はい	いいえ	いいえ	はい	はい
VSAM 索引付き (KSDS)	はい	はい	はい	はい	はい
VSAM 相対 (RRDS)	はい	はい	はい	はい	はい

関連タスク
『VSAM ファイルの順次編成の指定方法』
169 ページの『VSAM ファイルの索引編成の指定方法』
170 ページの『VSAM ファイルの相対編成の指定方法』
254 ページの『ファイル状況キーの使用』
256 ページの『VSAM 戻りコードの使用 (VSAM ファイルのみ)』
186 ページの『VSAM ファイルの定義』
172 ページの『VSAM ファイルのアクセス・モードの指定』

VSAM ファイルの順次編成の指定方法

COBOL プログラム内では、VSAM ESDS ファイルは、ORGANIZATION IS SEQUENTIAL 文節で識別してください。

順次ファイル内のレコードは、順次にだけしかアクセス (読み取りまたは書き込み) することができません。

レコードをファイルに入れた後は、それを短くしたり、長くしたり、削除したりすることはできません。ただし、長さが変わらなければ、レコードを更新 (REWRITE) することはできます。新しいレコードはファイルの終わりに追加されます。

次の例は、VSAM 順次ファイル (ESDS) の代表的な FILE-CONTROL 記入項目を示しています。

```
SELECT S-FILE
    ASSIGN TO SEQUENTIAL-AS-FILE
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

関連概念
166 ページの『VSAM ファイル』

VSAM ファイルの索引編成の指定方法

COBOL プログラム内では、VSAM KSDS ファイルは、ORGANIZATION IS INDEXED 文節で識別してください。

次の文節を使用して、レコードの基本キーをコーディングしてください。

```
RECORD KEY IS data-name
```

ここで *data-name* は、DATA DIVISION のレコード記述項目で定義したときのキー・フィールドの名前です。索引レコード・キーの照合は、キーのクラスやカテゴリーに関係なく、キーの 2 進値に基づいて行われます。

次の例は、動的にアクセスされる VSAM 索引付きファイル (KSDS) の場合のステートメントを示します。基本キー COMMUTER-NO のほかに、代替キー LOCATION-NO があります。

```
SELECT I-FILE
    ASSIGN TO INDEXED-FILE
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS IFILE-RECORD-KEY
    ALTERNATE RECORD KEY IS IFILE-ALTREC-KEY
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

代替キー

基本キーのほかに、レコードの検索に使用する 1 つまたは複数の代替キーをコーディングすることもできます。代替キーを使用すると、基本キーのシーケンス以外のシーケンスで、索引付きファイルにアクセスしてレコードを読み取ることができます。たとえば、従業員番号ではなく、従業員の部門を介してファイルにアクセスすることができます。代替キーは固有である必要はありません。部門番号がキーであるとすれば、複数のレコードがアクセスされます。これは、代替キーが重複を認めるようにコーディングされた場合に可能です。

COBOL プログラムで代替キーを定義するには、ALTERNATE RECORD KEY 文節を使用します。

```
ALTERNATE RECORD KEY IS data-name
```

ここで *data-name* は、DATA DIVISION のレコード記述項目で定義したときのキー・フィールドの名前です。代替キーの照合は、キーのクラスやカテゴリーに関係なく、キーの 2 進値に基づいて行われます。

代替索引

代替索引を使用するためには、代替索引 (AIX) と呼ばれるデータ・セットを定義する (アクセス方式サービス・プログラムを使って) ことが必要です。AIX には、指定された代替キーのそれぞれの値について 1 つのレコードが入ります。レコードは、代替キー値により順次に配列されます。各レコードには、代替キー値が入っている関連索引付きファイル内のすべてのレコードの対応する基本キーが含まれます。

関連概念
166 ページの『VSAM ファイル』

関連タスク
187 ページの『代替索引の作成』

VSAM ファイルの相対編成の指定方法

COBOL プログラム内では、VSAM RRDS ファイルは、ORGANIZATION IS RELATIVE 文節で識別してください。

各論理レコードを相対レコード番号に関連付けるには、RELATIVE KEY IS 文節を使用します。

次の例は、相対キー ITEM-NO に入れられた値によってランダムにアクセスされる相対レコード・データ・セット (RRDS) を示しています。

```
SELECT R-FILE
    ASSIGN TO RELATIVE-FILE
    ORGANIZATION IS RELATIVE
    ACCESS IS RANDOM
    RELATIVE KEY IS RFILE-RELATIVE-KEY
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

ランダム化ルーチンを使用して、各レコード内のキー値をそのレコードの相対レコード番号と関連付けることができます。レコード・キーを相対レコード番号に変換する技法は数多くありますが、最もよく使用されるランダム化アルゴリズムは除算 / 剰余技法です。この技法では、キーをデータ・セット内のスロットの数で割って、商と剰余を出します。剰余に 1 を加算すると、結果は有効な相対レコード番号になります。

VSAM RRDS では、代替索引はサポートされません。

固定長および可変長 RRDS

固定長レコードの RRDS の場合、各レコードは 1 つのスロットを占有するので、レコードの保管と検索は、そのスロットの相対レコード番号に従って行われます。ファイルをロードするときには、スロットをスキップし、それらを空にしておくこともできます。

可変長レコードの RRDS をロードするときは、相対レコード番号をスキップすることができます。固定長 RRDS とは異なり、可変長 RRDS はスロットを持っていません。代わりに、定義されたフリー・スペースを使用して、より効率的なレコード挿入を行うことができます。

VSAM の可変長 RRDS は、OS/390 および z/OS でサポートされます。

可変長 RRDS のシミュレート

可能なときはいつでも、VSAM 可変長 RRDS を使用してください。ただし、VSAM サポートを使用できない場合に備えて、Enterprise COBOL には、可変長レコードの入った相対レコード・データ・セットを持つための別の方法が用意されています。このサポートは、*COBOL シミュレート可変長 RRDS* と呼ばれ、

SIMVRD|NOSIMVRD ランタイム・オプションによって提供されます。SIMVRD オプションを使用すると、Enterprise COBOL は VSAM KSDS を使って可変長 RRDS をシミュレートします。

COBOL プログラム内で VSAM 可変長 RRDS と COBOL シミュレート可変長 RRDS を識別および記述するために使用するコーディングは類似しています。ただし、SIMVRD ランタイム・オプションの使い方と、VSAM ファイルを RRDS と KSDS のどちらとして定義するかが異なります。

可変長 RRDS を使用するには、RRDS をシミュレートしたいかどうかに応じて、以下のステップを行ってください。

ステップ	VSAM 可変長 RRDS	COBOL シミュレート可変長 RRDS
1	ORGANIZATION IS RELATIVE 文節を使用して、ファイルを COBOL プログラム内で定義します。	同じ
2	COBOL プログラム内で FD ステートメントを使用して、可変長サイズのレコードを記述します。	同じですが、データ・セットにアクセスするすべての COBOL プログラムの FD ステートメントに RECORD IS VARYING もコーディングする必要があります。
3	NOSIMVRD ランタイム・オプションを使用します。	SIMVRD ランタイム・オプションを使用します。
4	アクセス方式サービス・プログラムを介して、VSAM ファイルを RRDS として定義します。	アクセス方式サービス・プログラムを介して、VSAM ファイルを以下のように定義します。 DEFINE CLUSTER INDEXED KEYS(4,0) RECORDSIZE(*avg*,*m*) この場合、 ***avg*** は、COBOL レコードの平均サイズで、厳密に m より小さくなります。 ***m*** は、最大サイズの COBOL レコード + 4 より大か等しくなります。

シミュレート可変長 RRDS の ステップ 2 では、可変長レコード・フォーマットを暗黙指定する他の言語エレメントをコーディングしても COBOL シミュレート可変長 RRDS は得られません。たとえば、以下の文節を単独で使用しても正しいファイル・アクセスは得られません。

- 長さが異なる複数の FD レコード
- レコード定義の OCCURS . . . DEPENDING ON
- RECORD CONTAINS integer-1 TO integer-2 CHARACTERS

シミュレート可変長 RRDS の ステップ 4 でクラスターを定義するときは、以下の制約事項に従ってください。

- 代替索引を定義してはなりません。
- KEYRANGE を指定してはなりません。
- SPANNED を指定してはなりません。

また、レコードを含むファイルを出力のためにオープンするときは、REUSE パラメーターを使用してください。

エラー: シミュレート可変長相対データ・セットと本物の VSAM RRDS データ・セットを使用して作業するときは、COBOL ファイル定義と VSAM データ・セット属性が一致しないと、OPEN ファイル状況 39 が出されます。

関連概念
166 ページの『VSAM ファイル』

関連タスク
186 ページの『VSAM ファイルの定義』

VSAM ファイルのアクセス・モードの指定

VSAM 順次ファイルのレコードは、順次でしかアクセスすることができません。VSAM 索引付きファイルおよび相対ファイルの中のレコードは、順次、ランダム、または動的の 3 つの方法でアクセスすることができます。

順次アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS SEQUENTIAL をコーディングします。その場合、索引付きファイルのレコードは、選択されたキー・フィールド (基本または代替) の順にアクセスされます。相対ファイルのレコードは、相対レコード番号の順にアクセスされます。

ランダム・アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS RANDOM をコーディングします。その場合、索引付きファイルのレコードは、キー・フィールドに入れられた値に従ってアクセスされます。相対ファイルのレコードは、相対キーに入れられた値に従ってアクセスされます。

動的アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS DYNAMIC をコーディングします。動的アクセスは、同じプログラムの中での順次とランダムの混合アクセスです。動的アクセスを使用すると、順次処理とランダム処理の両方を実行する 1 つのプログラムを作成して、あるレコードは順次にアクセスし、別のレコードはキーによってアクセスすることができます。

『例: VSAM ファイルでの動的アクセスの使用』

関連タスク
179 ページの『VSAM ファイルからのレコードの読み取り』

例: VSAM ファイルでの動的アクセスの使用

たとえば、従業員レコードの索引付きファイルがあり、従業員の時間給がレコード・キーを形成しているものとします。プログラムでは、時間給が $10.00 ～ $12.00 の従業員および $20.00 以上の従業員を処理します。

VSAM ファイルの動的アクセスを使用して、プログラムでは次のような処理を行います。

1. 1000 のキーに基づいて最初のレコードをランダムに検索します (ランダム検索 READ を使って)。
2. 給与フィールドが 1200 を超えるまで、順次に読み取ります (READ NEXT を使って)。

3. 今度は 2000 のキーに基づいて、次のレコードをランダム検索します。
4. ファイルの終わりになるまで、順次に読み取ります。

関連タスク
179 ページの『VSAM ファイルからのレコードの読み取り』

VSAM ファイルのレコード長の定義

VSAM レコードは、固定長でも可変長でも構いません。COBOL は、RECORD 文節と、ファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。

VSAM ファイルについては、ブロック化の概念は意味を持たないため、BLOCK CONTAINS 文節は省略して構いません。この文節は構文検査されますが、プログラムの実行方法には影響を及ぼしません。

固定長レコードの定義

レコードを固定長として定義するには、以下のコーディング・オプションの 1 つを使用してください。

RECORD 文節	**文節形式**	**レコード長**	**コメント**
RECORD CONTAINS *integer* をコーディングする。	1	*integer-2*の長さでサイズが固定。	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
RECORD 文節を省略するが、すべてのレベル 01 レコード (ファイルに関連付けられる) を同じサイズとしてコーディングし、OCCURS DEPENDING ON 文節を使用しない。		コーディングされた固定サイズ。	

可変長レコードの定義

レコードを可変長として定義するには、以下のコーディング・オプションの 1 つを使用してください。

RECORD 文節	**文節形式**	**最大レコード長**	**コメント**
RECORD IS VARYING FROM *integer-1* TO *integer-2* をコーディングする。	3	*integer-2* 値	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
RECORD IS VARYING をコーディングする。	3	ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。

RECORD 文節	文節形式	最大レコード長	コメント
RECORD CONTAINS *integer-1* TO *integer-2* をコーディングする。	2	*integer-2* 値	最小レコード長は *integer-1* 値です。
RECORD 文節を省略するが、サイズが異なる OCCURS DEPENDING ON 文節を含む、複数のレベル 01 レコード (ファイルに関連付けられる) をコーディングする。		ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。

形式 V ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。たとえば、READ ステートメントによって読み込まれた形式 V レコードに対して MOVE ステートメントを指定すると、レコードのサイズはそのレベル 01 レコード記述に相当します。

関連タスク
Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド

VSAM ファイルの入出力ステートメントのコーディング

VSAM ファイルを処理するには、以下の COBOL ステートメントを使用してください。

OPEN VSAM データ・セットを処理のために COBOL プログラムに接続します。

WRITE ファイルにレコードを追加するか、またはファイルをロードします。

START READ NEXT ステートメントのためにクラスターにおける現在位置を設定します。

START はレコードを取り出しません。現行レコード・ポインターを設定するだけです。

READ および READ NEXT
ファイルからレコードを取り出します。

REWRITE
レコードを更新します。

DELETE 索引付きファイルおよび相対ファイルからのみ、レコードを論理的に除去します。

CLOSE VSAM データ・セットをプログラムから切り離します。

以下のすべての要素によって、特定の VSAM データ・セットに使用できる入出力ステートメントが決まります。

- アクセス・モード (順次、ランダム、または動的)
- ファイル編成 (ESDS、KSDS、または RRDS)
- OPEN ステートメントのモード (INPUT、OUTPUT、I-O、または EXTEND)

次の表は、順次ファイル (ESDS) に関して可能な組み合わせを示しています。X は、そのステートメントが列の項目に示されているオープン・モードで使用可能であることを示します。

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START				
	READ	X		X	
	REWRITE			X	
	DELETE				
	CLOSE	X	X	X	X

次の表は、索引付き (KSDS) ファイルおよび相対 (RRDS) ファイルに関して使用できる可能な組み合わせを示しています。X は、そのステートメントが列の項目に示されているオープン・モードで使用可能であることを示します。

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	X
ランダム	OPEN	X	X	X	
	WRITE		X	X	
	START				
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
動的	OPEN	X	X	X	
	WRITE		X	X	
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

FILE STATUS 文節でコーディングしたフィールドは、それぞれの入出力ステートメントの後で、操作の成功または失敗を示すために更新されます。

関連概念
『ファイル位置標識』

関連タスク
『ファイルのオープン (ESDS、KSDS、または RRDS)』
179 ページの『VSAM ファイルからのレコードの読み取り』
180 ページの『VSAM ファイル内のレコードの更新』
181 ページの『VSAM ファイルへのレコードの追加』
182 ページの『VSAM ファイル内のレコードの置換』
182 ページの『VSAM ファイルからのレコードの削除』
183 ページの『VSAM ファイルのクローズ』

ファイル位置標識

ファイル位置標識は、順次 COBOL 要求の場合にアクセスされる次のレコードを示します。ファイル位置標識は、プログラマーがプログラム内に設定するものではありません。この標識は、成功した OPEN、START、READ、および READ NEXT ステートメントによって設定されます。その後の READ または READ NEXT 要求は、設定されたファイル位置標識の位置を使用し、それを更新します。

ファイル位置標識は、出力ステートメント WRITE、REWRITE、または DELETE によって使用されたり、影響を受けたりすることはありません。ファイル位置標識は、ランダム処理では意味がありません。

関連タスク
179 ページの『VSAM ファイルからのレコードの読み取り』

ファイルのオープン (ESDS、KSDS、または RRDS)

WRITE、START、READ、REWRITE、または DELETE ステートメントを使用してファイル内のレコードを処理するためには、前もってそのファイルを OPEN ステートメントでオープンしておかなければなりません。 ファイルの可用性および作成は、OPEN 処理、オプショナル・ファイル、およびファイル状況コード 05 と 35 に影響を与えます。

たとえば、オプショナルでなく利用可能な状態にもないファイルの OPEN EXTEND、OPEN I-O、または OPEN INPUT を行うと、ファイル状況が 35 になり、OPEN ステートメントが失敗します。ファイルが OPTIONAL である場合は、OPEN EXTEND、OPEN I-O、または OPEN INPUT を行うと、ファイルが作成され、ファイル状況 05 が戻されます。

OPEN 操作が正しく機能するのは、ファイルの DD ステートメントまたはデータ・セット・ラベルで固定ファイル属性を指定し、しかも COBOL プログラムの SELECT および FD ステートメントでそのファイルについて一貫性のある属性を指定する場合だけです。 以下の項目が一致していないと、ファイル状況コード 39 が戻され、OPEN ステートメントは失敗に終わります。

- ファイル編成の属性 (順次、相対、または索引付き)
- 基本レコード・キー
- 代替レコード・キー
- 最大レコード・サイズ
- レコード・タイプ (固定長または可変長)

COBOL プログラム内で VSAM ファイルについての OPEN ステートメントをコーディングする方法は、ファイルが空である (レコードが入ったことがない) か、ロード済みであるかによって異なります。ファイルのタイプがどちらであっても、プログラムでは、それぞれの OPEN ステートメントの後でファイル状況キーを検査しなければなりません。

関連タスク
『空のファイルのオープン』
179 ページの『ロード済みファイル (レコードが入っているファイル) のオープン』

関連参照
179 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

空のファイルのオープン

レコードが入ったことがないファイル (空のファイル) をオープンするには、ファイルのタイプに応じて以下のステートメントを使用してください。

- ESDS ファイルの場合は、OPEN OUTPUT。
- KSDS および RRDS ファイルの場合は、OPEN OUTPUT または OPEN EXTEND。(どちらのコーディングでも効果は同じです。) ファイルをランダム・アクセスまたは動的アクセス用にコーディングし、それがオプショナル・ファイルであれば、OPEN I-O を使用することができます。

オプショナル・ファイルとは、プログラムが実行されるたびに、必ずしも存在する必要はないファイルです。INPUT、I-O、または OUTPUT モードでオープンされたファイルを、プログラムの FILE-CONTROL セクションで SELECT OPTIONAL 句を使って定義することによって、オプショナルとして定義することができます。

ファイルへのレコードの初期ロードを順次に行う場合: ファイルの初期ロードとは、ファイルに初めてレコードを書き込むことを意味します。これは、レコードがすべて削除されてしまったファイルにレコードを書き込むことと同じではありません。

VSAM ファイルを初期ロードするには、以下のようにしてください。

1. ファイルをオープンします。
2. 順次処理の方が高速であるため、順次処理を使用します (ACCESS IS SEQUENTIAL)。
3. WRITE を使用して、ファイルにレコードを加えます。

OPEN OUTPUT を使って VSAM ファイルをロードすると、プログラムのパフォーマンスが著しく向上します。OPEN I-O または OPEN EXTEND を使用すると、プログラムのパフォーマンスに悪い影響があります。

VSAM 索引付きファイルを順次にロードすると、順次処理ではユーザー定義のフリー・スペースが保持されるため、ロードとその後の処理の両方のパフォーマンスが最適化されます。将来の追加がより効率的になります。

ACCESS IS SEQUENTIAL を使用する場合は、レコードを RECORD KEY の昇順に書き込まなければなりません。

VSAM 相対ファイルを順次にロードすると、レコードは相対レコード番号の昇順にファイルに入れられます。

ファイルの初期ロードをランダムまたは動的に行う場合: ランダム処理または動的処理を使用してファイルをロードすることもできますが、順次処理ほど効率はよくありません。VSAM がランダム処理または動的処理をサポートしていないので、COBOL が、OPEN OUTPUT または OPEN I-O とともに ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用できるようにするために余分な処理を実行しなければなりません。これらのステップでは、ファイルが使用できるよう準備し、それにロード済みファイル (少なくとも一度は使用されたことがある) の状況を与えます。

ファイルが使用できるよう準備するための余分なオーバーヘッドに加えて、ランダム処理ではユーザー定義のフリー・スペースが考慮されません。その結果、将来の追加処理が非効率的になる恐れがあります。順序処理では、ユーザー定義のフリー・スペースが保持されます。

アクセス方式サービス・プログラムを使用した VSAM データ・セットのロード:
アクセス方式サービス・プログラムの REPRO コマンドを使用して、VSAM データ・セットをロードまたは更新することができます。可能なときはいつでも、REPRO を使用してください。

関連タスク
179 ページの『ロード済みファイル (レコードが入っているファイル) のオープン』

関連参照
179 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』
REPRO (*z/OS DFSMS: Access Method Services for Catalogs*)

VSAM ファイルにレコードをロードするために使用されるステートメント

除算	ESDS	KSDS	RRDS
ENVIRONMENT DIVISION	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS PASSWORD ACCESS MODE
DATA DIVISION	FD 記入項目	FD 記入項目	FD 記入項目
PROCEDURE DIVISION	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE

関連タスク
177 ページの『空のファイルのオープン』
180 ページの『VSAM ファイル内のレコードの更新』

ロード済みファイル (レコードが入っているファイル) のオープン

すでにレコードが入っているファイルをオープンするには、OPEN INPUT、OPEN I-O、または OPEN EXTEND を使用してください。

VSAM 入力順または相対レコード・ファイルを EXTEND としてオープンする場合には、追加されたレコードは、ファイル内の最後の既存のレコードの後に置かれます。

VSAM キー順ファイルを EXTEND としてオープンする場合には、追加するそれぞれのレコードが、ファイル内の最高位レコードよりも大きいレコード・キーを持っていなければなりません。

関連タスク
177 ページの『空のファイルのオープン』
185 ページの『z/OS および UNIX のもとでの VSAM データ・セットの操作』

関連参照
『VSAM ファイルにレコードをロードするために使用されるステートメント』
z/OS DFSMS: Access Method Services for Catalogs

VSAM ファイルからのレコードの読み取り

ファイルからレコードを検索 (READ) するには、READ ステートメントを使用します。レコードを読み取るためには、ファイルを INPUT または I-O としてオープンしていなければなりません。プログラムでは、それぞれの READ の後でファイル状況キーを検査しなければなりません。

VSAM 順次ファイルの中のレコードは、それらが書き込まれたシーケンスでしか検索することができません。

VSAM 索引付きファイルおよび相対レコード・ファイルの中のレコードは、次のように検索することができます。

順次 索引付きファイルの場合は、ファイル位置標識のファイル位置標識から、使用中のキー (RECORD KEY または ALTERNATE RECORD KEY) の昇順に従い、相対ファイルの場合は、相対レコード位置の昇順に従って

ランダム
READ 要求の前に、RECORD KEY または ALTERNATE RECORD KEY または RELATIVE KEY をどのように設定するかによって、任意の順序で

動的 順次とランダムの混合で

動的アクセスの場合、順次検索には READ NEXT を使用し、ランダム検索 (キーによる) には READ を使用することによって、特定のレコードの直接読み取りと、いくつかのレコードの順次読み取りを切り替えることができます。

特定のレコードから順次に読み取りを行いたい場合には、READ NEXT ステートメントの前に START ステートメントを使用して、ファイル位置標識を、特定のレコードを指すように設定してください。START の後に READ NEXT をコーディングすると、次のレコードが読み取られ、ファイル位置標識は次のレコードにリセットされます。ファイル位置標識は、START を使用してランダムに移動することができますが、すべての読み取りはそのポイントから順次に行われることになります。

```
START file-name KEY IS EQUAL TO ALTERNATE-RECORD-KEY
```

重複が存在する代替索引に基づいて、VSAM 索引付きファイルに対して直接 READ が実行されると、その代替キーを持つデータ・セット (基本クラスター) の最初のレコードのみが検索されます。同じ代替キーを持つデータ・セット・レコードのそれぞれを検索するためには、一連の READ NEXT ステートメントが必要です。 同じ代替キー値を持つ、読み取るべきレコードがほかにある場合には、02 のファイル状況コードが戻されます。そのキー値を持つ最後のレコードが読み取られると、00 のコードが戻されます。

関連概念
176 ページの『ファイル位置標識』

関連タスク
172 ページの『VSAM ファイルのアクセス・モードの指定』

VSAM ファイル内のレコードの更新

VSAM ファイルを更新するためには、ENVIRONMENT DIVISION および DATA DIVISION のステートメントを使用してレコードを VSAM ファイルにロードし、次の PROCEDURE DIVISION のステートメントを使用して VSAM ファイルを更新します。

アクセス方式	ESDS	KSDS	RRDS
ACCESS IS SEQUENTIAL	OPEN EXTEND WRITE CLOSE *または* OPEN I-O READ REWRITE CLOSE	OPEN EXTEND WRITE CLOSE *または* OPEN I-O READ REWRITE DELETE CLOSE	OPEN EXTEND WRITE CLOSE *または* OPEN I-O READ REWRITE DELETE CLOSE
ACCESS IS RANDOM	適用されない	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE
ACCESS IS DYNAMIC (順次処理)	適用されない	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE
ACCESS IS DYNAMIC (ランダム処理)	適用されない	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE

関連参照

179 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

VSAM ファイルへのレコードの追加

COBOL WRITE ステートメントを使用すると、既存のレコードを置き換えずに、ファイルにレコードを追加することができます。追加されるレコードは、ファイルの定義時に設定された最大レコード・サイズよりも大きいものであってはなりません。プログラムでは、それぞれの WRITE ステートメントの後でファイル状況キーを検査しなければなりません。

レコードを順次に追加する場合

OUTPUT または EXTEND としてオープンされた VSAM ファイルの終わりにレコードを順次に追加するには、ACCESS IS SEQUENTIAL を使用し、WRITE ステートメントをコーディングしてください。

順次ファイルは、必ず順次に書き込まれます。

索引付きファイルの場合は、昇順キー配列で新規のレコードを書かなければなりません。ファイルが EXTEND としてオープンされている場合には、追加されるレコードのレコード・キーは、ファイルがオープンされた時点のファイルの最高位基本レコード・キーよりも大きくなければなりません。

相対ファイルの場合、レコードは正しい順序になっていなければなりません。SELECT 文節に RELATIVE KEY データ項目を組み込むと、書き込まれるレコードの相対レコード番号がそのデータ項目に入れられます。

レコードをランダムまたは動的に追加する場合

レコードを索引付きデータ・セットに書き込むとき、ACCESS IS RANDOM または ACCESS IS DYNAMIC であれば、レコードは任意の順序で書き込むことができます。

VSAM ファイル内のレコードの置換

VSAM ファイル内のレコードを置換するには、I-O としてオープンしたファイルに対する REWRITE を使用してください。 I-O としてオープンされていないファイルに関して REWRITE を使用しようとしても、レコードは再書き込みされず、状況キーが 49 に設定されます。プログラムでは、それぞれの REWRITE ステートメントの後でファイル状況キーを検査しなければなりません。

- 順次ファイルの場合、再書き込みするレコードの長さは元のレコードの長さと同じでなければなりません。
- 索引付きファイルの場合、再書き込みするレコードの長さを変更することができます。
- 可変長相対ファイルの場合、再書き込みするレコードの長さを変更することができます。

レコードをランダムまたは動的に置換する場合には、再書き込みされる側のレコードをプログラムで読み取る必要はありません。その代わりに、更新したいレコードを位置決めするために、以下のことを行ってください。

- 索引付きファイルの場合、レコード・キーを RECORD KEY データ項目に移動してから、REWRITE を発行します。
- 相対ファイルの場合、相対レコード番号を RELATIVE KEY データ項目に移動してから、REWRITE を発行します。

VSAM ファイルからのレコードの削除

索引付きファイルまたは相対ファイルから既存のレコードを削除するには、ファイルを I-O としてオープンし、DELETE ステートメントを使用します。順次ファイルに対して DELETE を使用することはできません。

ACCESS IS SEQUENTIAL を使用する場合、またはファイルにスパン・レコードが含まれている場合には、まず削除すべきレコードをプログラムで読み取らなければなりません。その後、読み取られたレコードを DELETE で除去します。DELETE の前の READ が成功しなかった場合には、削除は行われず、状況キー値が 92 に設定されます。

ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用する場合は、削除すべきレコードを前もってプログラムで読み取る必要はありません。レコードを削除するに

は、削除するレコードのキーを RECORD KEY データ項目に移動してから、DELETE を出します。プログラムでは、それぞれの DELETE ステートメントの後でファイル状況キーを検査しなければなりません。

VSAM ファイルのクローズ

プログラムを VSAM ファイルから切り離すには、CLOSE ステートメントを使用します。すでにクローズされているファイルをクローズしようとすると、論理エラーになります。それぞれの CLOSE ステートメントの後で、ファイル状況キーを検査してください。

VSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。ただし、実行単位内の OS/VS COBOL プログラムのいずれかで定義されたファイルを除きます。

- 実行単位が正常に終了すると、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP(ON) ランタイム・オプションが設定されていれば、その実行単位内の COBOL プログラムのいずれかで定義されているオープン状態のファイルはすべてクローズされます。
- 言語環境プログラム条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

  条件が処理された後でプログラムが実行を再開する位置を変更することができます。この変更を行うには、たとえば、CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp ステートメントのような HLL 言語構造体を使用してください。
- COBOL サブプログラムに CANCEL を出すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

このような暗黙の CLOSE 操作が実行されたときは、WORKING-STORAGE SECTION に定義したファイル状況キー・データ項目が設定されますが、EXCEPTION/ERROR および LABEL 宣言は呼び出されません。

マルチスレッド・アプリケーションで VSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようとすると、ファイル状況コードの条件 90 でクローズが失敗します。

VSAM ファイルのエラーの処理

入力ステートメントまたは出力ステートメントの操作が失敗しても、COBOL がユーザーに代わって訂正処置を取るわけではありません。

VSAM ファイルを扱う際の OPEN および CLOSE エラーはすべて、それがプログラム内の論理エラーであっても外部ストレージ・メディア上の入出力エラーであっても、また、たとえ DECLARATIVE も FILE STATUS 文節もコーディングされていなくても、制御を COBOL プログラムに戻します。

他の入出力ステートメントの操作の失敗については、重大レベルより軽度の入出力エラーの発生後でプログラムに実行を継続させるかどうかを選択します。

COBOL は、特定の VSAM 入出力エラーを代行受信して処理するために、以下の 3 つの方法を提供します。

- ファイル終わり句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 文節 (ファイル状況キーおよび VSAM 戻りコード)
- INVALID KEY 句

プログラム内に定義するそれぞれの VSAM ファイルごとに状況キーを定義しなければなりません。それぞれの入力要求または出力要求 (特に、OPEN および CLOSE) の後で、状況キーの値を検査してください。

FILE STATUS キーまたは宣言をコーディングしなかった場合に、重大な VSAM 処理エラーが起こると、メッセージが出され、言語環境プログラム条件が通知されます。ランタイム・オプション ABTERMENC(ABEND) が指定されていると、この条件の結果として、異常終了が引き起こされます。

関連タスク

250 ページの『入出力操作でのエラーの処理』
256 ページの『VSAM 戻りコードの使用 (VSAM ファイルのみ)』

関連参照

z/OS DFSMS Macro Instructions for Data Sets の『VSAM macro return and reason codes』の項

パスワードによる VSAM ファイルの保護

z/OS システムで推奨するセキュリティー機構は RACF ですが、Enterprise COBOL では、無許可アクセスや無許可更新を防止するため、VSAM ファイルに明示パスワードを使用することもサポートします。

明示パスワードを使用するには、プログラムの SELECT ステートメントで PASSWORD 文節をコーディングしてください。この文節は、ファイルのカタログ記入項目に読み取りまたは更新パスワードが入っている場合にのみ使用してください。

- カタログ記入項目に読み取りパスワードが入っている場合には、パスワード文節が FILE-CONTROL 段落で使用され、DATA DIVISION で記述されていない限り、ファイルは COBOL プログラム内でオープンおよびアクセスできません。ファイルがオープンされるときには、参照された *data-name* に有効なパスワードが入っていなければなりません。
- カタログ記入項目に更新パスワードが入っている場合には、ファイルはオープンおよびアクセスすることはできますが、パスワード文節が FILE-CONTROL 段落で使用され、DATA DIVISION で記述されていない限り、更新できません。

- カタログ記入項目に読み取りパスワードと更新パスワードの両方が入っている場合には、プログラム内でファイルを読み取るときと更新するときの両方に更新パスワードを使用してください。

プログラムでレコードの検索だけを行い、更新を行わない場合には、読み取りパスワードしか必要ありません。プログラムでファイルをロードまたは更新する場合には、カタログされた更新パスワードを指定しなければなりません。

索引付きファイルの場合は、RECORD KEY の PASSWORD データ項目に有効なパスワードが含まれていなければ、ファイルを正常にオープンすることはできません。

VSAM 索引付きファイルをパスワード保護する場合、完全にパスワード保護するためには、すべての代替索引もパスワード保護しなければなりません。それぞれの代替索引はそれ自体のパスワードを持つため、PASSWORD 文節を置く場所が重要になります。PASSWORD 文節は、それが適用されるキー文節の直後になければなりません。

『例: VSAM 索引付きファイルのパスワード保護』

例: VSAM 索引付きファイルのパスワード保護

以下に、パスワード保護を使用する VSAM 索引付きファイルについて使用される COBOL コードの例を示します。

```
. . .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT LIBFILE
      ASSIGN TO PAYMAST
      ORGANIZATION IS INDEXED
      RECORD KEY IS EMPL-NUM
         PASSWORD IS BASE-PASS
      ALTERNATE RECORD KEY IS EMPL-PHONE
         PASSWORD IS PATH1-PASS
. . .
WORKING-STORAGE SECTION.
01 BASE-PASS                  PIC X(8) VALUE "25BSREAD".
01 PATH1-PASS                 PIC X(8) VALUE "25ATREAD".
```

z/OS および UNIX のもとでの VSAM データ・セットの操作

VSAM ファイルを z/OS および UNIX のもとで使用する場合、アクセス方式サービス・プログラム (IDCAMS) コマンド、環境変数、JCL (ジョブ制御言語) のコーディングに関していくつかの特別な考慮事項があります。

VSAM ファイルが*使用可能* となるのは、以下のすべてが真である場合です。

- VSAM ファイルが、アクセス方式サービス・プログラムを使用して定義されている。
- プログラムで使用するために、VSAM ファイルが、それに対応する DD ステートメント、環境変数、または ALLOCATE コマンドを指定して定義されている。
- すでにレコードが入っている。

VSAM ファイルは、定義されていても、レコードが入ったことがなければ、*利用不能* です。

VSAM 順次ファイルに対する OPEN ステートメントの完了時には、必ず戻りコード 0 が戻されます。

ファイルを空にするには、アクセス方式サービス・プログラム REPRO コマンドを使用してください。この方法でレコードを削除すると、ファイルの最も高い相対バイト・アドレス (RBA) が 0 にリセットされます。ファイルは事実上空になり、COBOL にとってはそれがレコードを含んだことがないかのように見えます。

関連タスク

12 ページの『オペレーティング・システムに対してファイルを定義する』
『VSAM ファイルの定義』
187 ページの『代替索引の作成』
189 ページの『VSAM ファイルの割り振り』
191 ページの『RLS による VSAM ファイルの共用』

VSAM ファイルの定義

Enterprise COBOL の VSAM 入力順、キー順、相対レコードの各データ・セットは、アクセス方式サービス・プログラム (IDCAMS) を使用してそれらを定義してからでなければ処理できません。

VSAM クラスター とは、VSAM データ・セットの論理定義であり、1 つまたは 2 つのコンポーネントを持っています。

- VSAM クラスターのデータ・コンポーネントには、データ・レコードが入れられます。
- VSAM キー順クラスターの索引コンポーネントは、索引レコードから構成されます。

VSAM データ・セット (クラスター) を定義するには、アクセス方式サービス・プログラムの DEFINE CLUSTER コマンドを使用してください。このプロセスには、データ転送を行わずに統合カタログ内に項目を作成することが含まれます。

クラスターについては、以下の情報を定義してください。

- 記入項目の名前。
- この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。
- 編成 (順次、索引付き、または相対)。
- データ・セットが占有する装置およびボリューム。
- データ・セットに必要なスペース。
- レコード・サイズおよび制御インターバル・サイズ (CISIZE)。
- 将来のアクセスに必要なパスワード (もしあれば)。

クラスター内のデータ・セットの種類によっては、クラスターごとに以下の情報も定義してください。

- VSAM 索引付きデータ・セット (KSDS) の場合は、レコード内の基本キーの長さおよび位置を指定してください。

- VSAM 固定長相対レコード・データ・セット (RRDS) の場合は、最大の COBOL レコード・サイズに等しいかそれより大きなレコード・サイズを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(n,n)
```

  このようにデータ・セットを定義すると、すべてのレコードは固定スロット・サイズ *n* まで埋め込まれます。RECORD IS VARYING ON *data-name* 形式の RECORD 文節を使用すると、WRITE または REWRITE は、DEPENDING ON *data-name* で指定された長さを、VSAM によって転送されるレコードの長さとして使用します。このデータは、その後、固定スロット・サイズに埋め込まれます。READ ステートメントは必ず、固定スロット・サイズを DEPENDING ON *data-name* 内で戻します。

- VSAM 可変長相対レコード・データ・セット (RRDS) の場合は、予期される平均サイズ COBOL レコードと予期される最大サイズ COBOL レコードを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(avg,m)
```

  予期される平均サイズ COBOL レコードは、予期される最大サイズ COBOL レコードより小さくなければなりません。

- COBOL シミュレート可変長相対レコード・データ・セットの場合は、COBOL レコードの平均サイズと、最大サイズ COBOL レコードに 4 を加えた値に等しいかそれより大きなサイズを指定してください。

```
DEFINE CLUSTER INDEXED
KEYS(4,0)
RECORDSIZE(avg,m)
```

  予期される平均サイズ COBOL レコードは、予期される最大サイズ COBOL レコードより小さくなければなりません。

関連タスク

『代替索引の作成』

189 ページの『VSAM ファイルの割り振り』

170 ページの『VSAM ファイルの相対編成の指定方法』

関連参照

z/OS DFSMS: Access Method Services for Catalogs

代替索引の作成

代替索引は、複数のキーを使用して、データ・セット内のレコードへのアクセスを提供するものです。索引付きデータ・セット (KSDS) の基本索引キーと同じ方法でレコードにアクセスします。

代替索引の使用を計画している場合には、以下のことを知っておかなければなりません。

- 索引が関連付けられるデータ・セット (基本クラスター) のタイプ。
- キーが固有であるか固有でないか。
- 索引をパスワード保護するかどうか。
- 代替索引使用のパフォーマンスの側面。

代替索引は、実際には、VSAM データ・セットのキーへのポインターを含む VSAM データ・セットであるため、代替索引および代替索引パス (代替索引と基本索引の間の関係を確立するエンティティー) を定義しなければなりません。 代替索引を定義した後、代替索引とその基本クラスターの間の関係 (パス) を確立するカタログ記入項目を作成してください。このパスにより、代替キーを介して基本クラスターのレコードにアクセスすることが可能になります。

代替索引を使用するには、以下のステップに従ってください。

1. DEFINE ALTERNATEINDEX コマンドを使用して、代替索引を定義します。このコマンドの中では、以下のことを定義します。
   - 代替索引の名前。
   - 関連する VSAM 索引付きデータ・セットの名前。
   - 代替索引のレコードにおける位置と、代替索引が固有であるかどうか。
   - データ・セットの変更時に代替索引を更新するかどうか。
   - この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。

   COBOL プログラムでは、代替索引は、FILE CONTROL 段落の ALTERNATE RECORD KEY 文節によってのみ識別されます。ALTERNATE RECORD KEY 定義は、カタログ記入項目で作成した定義と一致していなければなりません。カタログしたパスワード項目は、ALTERNATE RECORD KEY 句の後に直接コーディングしなければなりません。
2. DEFINE PATH コマンドを使用して、代替索引を基本クラスター (代替索引を介してアクセスするデータ・セット) に関連付けます。このコマンドの中では、以下のことを定義します。
   - パスの名前。
   - パスが関連付けられている代替索引。
   - 代替索引を含むカタログの名前。

   基本クラスターと代替索引は、同じカタログ内の項目によって記述されます。
3. VSAM 索引付きデータ・セットをロードします。
4. BLDINDEX コマンドを使用して (一般に)、代替索引を作成します。入力ファイルを索引付きデータ・セット (基本クラスター) として識別し、出力ファイルを代替索引またはそのパスとして識別します。この BLDINDEX コマンドは、VSAM 索引付きデータ・セット (つまり、基本クラスター) のすべてのレコードを読み取り、代替索引を作成するのに必要なデータを取り出します。

   あるいは、ランタイム・オプション AIXBLD を使用して、実行時に代替索引を作成することもできます。しかし、これは実行時のパフォーマンスに悪影響を及ぼすことがあります。

189 ページの『例: 代替索引の項目』

関連参照

AIXBLD (COBOL のみ) (*言語環境プログラム プログラミング・リファレンス*)

例: 代替索引の項目

次の例では、COBOL FILE-CONTROL 記入項目と、2 つの代替索引を持つ VSAM 索引付きファイルに対する DD ステートメントまたは環境変数の関係を示しています。

JCL を使用する場合:

```
//MASTERA     DD   DSNAME=clustername,DISP=OLD      (1)
//MASTERA1    DD   DSNAME=path1,DISP=OLD           (2)
//MASTERA2    DD   DSNAME=path2,DISP=OLD           (3)
```

環境変数を使用する場合:

```
export MASTERA=DSN(clustername),OLD                (1)
export MASTERA=DSN(path1),OLD                      (2)
export MASTERA=DSN(path2),OLD                      (3)
. . .
FILE-CONTROL.
  SELECT MASTER-FILE ASSIGN TO MASTERA             (4)
      RECORD KEY IS EM-NAME
      PASSWORD IS PW-BASE                          (5)
      ALTERNATE RECORD KEY IS EM-PHONE             (6)
          PASSWORD IS PW-PATH1
      ALTERNATE RECORD KEY IS EM-CITY              (7)
          PASSWORD IS PW-PATH2.
```

(1) 基本クラスター名は *clustername* です。

(2) 最初の代替索引パスの名前は *path1* です。

(3) 2 番目の代替索引パスの名前は *path2* です。

(4) 基本クラスターの DD 名または環境変数名は、ASSIGN 文節で指定されます。

(5) パスワードはその索引の直後に続きます。

(6) キー EM-PHONE は、最初の代替索引に関連します。

(7) キー EM-CITY は、2 番目の代替索引に関連します。

関連タスク

187 ページの『代替索引の作成』

VSAM ファイルの割り振り

VSAM データ・セットはすべて、アクセス方式サービス・プログラム DEFINE コマンドを介して事前定義し、カタログしなければなりません。VSAM データ・セットに関する情報の大部分は、カタログに入っています。VSAM ファイルを使用するために指定する必要があるのは、DD または環境変数情報だけです。環境変数を使用する場合、名前は大文字でなければなりません。通常、関係がある変数は、入力およびデータ・バッファーだけです。

VSAM ファイル (索引付き、相対、および順次) の割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。環境変数を使用して VSAM ファイルを割り振る場合は、以下のオプションを示された順番で指定する必要があります。他のオプションを指定してはなりません。

- DSN(*dsname*)。ここで、*dsname* は基本クラスターの名前です。
- OLD または SHR

VSAM ファイルに必要な基本 DD ステートメントは、次のとおりです。

```
//ddname   DD   DSN=dsname,DISP=SHR,AMP=AMORG
```

対応する export コマンドは、次のとおりです。

```
export evname="DSN(dsname),SHR"
```

いずれの場合も、*dsname* は、アクセス方式サービス・プログラムの DEFINE CLUSTER または DEFINE PATH コマンドで使用した名前と同じ名前でなければなりません。データ・セットはすでにカタログされているため、DISP は OLD または SHR でなければなりません。JCL を使用する際に MOD を指定すると、そのデータ・セットは OLD として扱われます。

AMP は、データ・セットについてプログラムによって提供される情報を補足するために使用される VSAM JCL パラメーターです。AMP が有効になるのは、プログラムが VSAM ファイルをオープンしたときです。AMP パラメーターを介して設定されたパラメーターは、カタログ内の情報またはプログラムによって提供される情報に優先します。

AMP パラメーターは、次の場合を除いて、必要ありません。

- ダミー VSAM データ・セットを使用する場合。次はその例です。

  ```
  //ddname  DD   DUMMY,AMP=AMORG
  ```

- 追加の索引またはデータ・バッファーを要求する場合。次はその例です。

  ```
  //ddname   DD  DSN=VSAM.dsname,DISP=SHR,
  //             AMP=('BUFNI=4,BUFND=8')
  ```

環境変数を使用して VSAM データ・セットを割り振る場合は、AMP を指定することはできません。

VSAM 基本クラスターの場合は、COBOL プログラムの SELECT ステートメントの ASSIGN 文節で指定した名前と同じシステム名 (DD 名または環境変数名) を指定してください。

COBOL プログラムで代替索引を使用する場合は、基本クラスターのシステム名 (DD ステートメントまたは環境変数) だけでなく、各代替索引パスごとのシステム名も指定する必要があります。プログラム内で代替索引パスのシステム名を明示的に宣言するための言語メカニズムはありません。このため、それぞれの代替索引パスのシステム名 (DD 名または環境変数名) を以下のガイドラインに従って作成する必要があります。

- 基本クラスター名に整数を連結します。
- プログラム内のファイルに定義された最初の代替レコード (SELECT ステートメントの ALTERNATE RECORD KEY 文節) と関連付けられたパスに 1 を使用します。
- そのファイルのその後の代替レコード定義と関連付けられたパスごとに 1 ずつ増やします。

たとえば、基本クラスターのシステム名が ABCD である場合、プログラム内のファイルに定義された最初の代替索引パスは ABCD1、2 番目の代替索引パスのシステム名は ABCD2 となり、以後同様に続きます。

基本クラスター・システム名とシーケンス番号を合わせた長さが 8 文字を超える場合は、システム名の基本クラスター部分の右側が切り捨てられ、連結結果が 8 文字になるよう切り詰められます。たとえば、基本クラスターのシステム名が ABCDEFGH である場合、最初の代替索引パスのシステム名は ABCDEFG1、10 番目の代替索引パスのシステム名は ABCDEF10 となり、以後同様に続きます。

関連タスク
129 ページの『ファイルの割り振り』

関連参照
MVS JCL 解説書

RLS による VSAM ファイルの共用

VSAM JCL パラメーター RLS を使用することによって、VSAM に関するレコード・レベル共用の使用法を指定することができます。整合性を保つ読み取りプロトコルが必要なときは RLS=CR を使用し、読み取り保全性プロトコルが必要でないときは RLS=NRI を使用してください。RLS パラメーターを指定することは、COBOL プログラム実行時に RLS モードを要求する唯一の方法です。

環境変数を使用して VSAM データ・セットを割り振る場合は、RLS を指定することはできません。

RLS モードでの VSAM ファイルに関する更新問題の回避

VSAM データ・セットが RLS モードで I-O (更新) 用にオープンされると、RLS=CR と RLS=NRI のどちらが指定されているかに関係なく、最初の READ によって、レコードが入っている制御インターバルの排他ロックが起こります。排他ロックは、WRITE または REWRITE が出されるか、または別のレコードについて別の READ が出されると解除されます。

RLS=CR を指定すると、レコードはロックされ、別のレコードに対する別の READ が要求されるまで、そのレコードへの更新を妨げることができます。読み取り中のレコードに関するロックが有効な間、他のユーザーは同じレコードに対する READ を要求することはできますが、読み取りロックが解除されるまで、そのレコードを更新することはできません。RLS=NRI を指定した場合は、入力に関する READ が出されると、ロックは有効でなくなり、別のユーザーがそのレコードを更新する可能性があります。

RLS=CR のロック規則により、アプリケーションがレコード・ロックが使用可能になるのを待つ可能性があり、このために入力の READ の速度が遅くなることがあります。

RLS=CR の機能を使用するためには、アプリケーションのロジックを修正することが必要な場合があります。アプリケーションが複数更新者環境で正しく機能することを確認するまでは、回復不能範囲を更新するバッチ・ジョブには、RLS JCL パラメーターを使用しないでください。

VSAM データ・セットを RLS モードで INPUT または I-O 処理用にオープンするときには、OPEN または START を出すのは READ の*直前* が良いでしょう。OPEN または START と実際の READ の間に時間差があると、OPEN または START の実行後に、アプリケーションが位置を定めたレコードの前の位置に他のユーザーがレコー

ドを追加する可能性が高くなります。COBOL 実行時には、OPEN が要求された時点で VSAM データ・セットの開始位置が明示的に指し示されますが、READ が遅れると、他のユーザーによって追加されたレコードによって、VSAM データ・セットの実際の開始位置が変わる可能性があります。

RLS を使用する際の制約事項

RLS モードには以下の制約事項が適用されます。

- VSAM ファイルを RLS モードでオープンするときには、VSAM クラスター属性 KEYRANGE および IMBED はサポートされません。
- RLS モードでは、VSAM クラスター属性 REPLICATE は推奨されません。これは、システム規模のバッファー・プールと、ストレージ階層内の潜在的に大きな CF キャッシュ構造によって、利点が打ち消されるためです。
- VSAM ファイルを RLS モードでオープンするときには、VSAM が空のパスのオープンを認めないため、AIXBLD ランタイム・オプションはサポートされません。代替索引データ・セットを作成するために AIXBLD ランタイム・オプションが必要である場合には、VSAM データ・セットを非 RLS モードでオープンしなければなりません。
- RLS モードでは、一時データ・セットは認められません。
- RLS モードでオープンされた VSAM ファイルについては、SIMVRD ランタイム・オプションはサポートされません。

RLS モードでの VSAM ファイルのエラーの処理

アプリケーションが RLS モードの VSAM データ・セットにアクセスする場合には、*それぞれの*要求の後でファイル状況および VSAM フィードバック・コードを検査するようにしてください。

入出力処理時にアプリケーションが「SMSVSAM サーバー利用不能」に直面した場合には、VSAM ファイルの再オープンを試行する前にそれを明示的にクローズしてください。VSAM は、「SMSVSAM サーバー利用不能」タイプの障害を表す戻りコード 16 を生成し、フィードバック・コードはありません。COBOL プログラムでは、VSAM 戻りコード 16 に関して 2 番目のファイル状況域の最初の 2 バイトを検査することができます。

COBOL 実行時には、OPEN 処理中にエラーが発生すると、IGZ0205W メッセージが生成され、ファイルは自動的にクローズされます。

その他のすべての RLS モード・エラーは、VSAM 戻りコード 4、8、または 12 を戻します。

VSAM パフォーマンスの向上

多くの場合、COBOL および VSAM のパフォーマンスの調整を担当するのはシステム・プログラマーです。アプリケーション・プログラマーとしては、次の表にリストされている VSAM の局面を制御することができます。

VSAM の局面	行うことができること	理由の説明とコメント
アクセス方式サービス・プログラムの呼び出し	IDCAMS を使用した、代替索引の事前作成	
バッファリング	順次アクセスの場合は、もっと多くのデータ・バッファーを要求し、ランダム・アクセスの場合は、もっと多くの索引バッファーを要求します。ACCESS IS DYNAMIC の場合は、BUFND と BUFNI の両方を指定します。 アプリケーションが対話式に実行される場合以外は、追加のバッファーのコーディングを避けます。さらに、入出力遅延が原因であることが考えられる応答時間問題が生じた場合にのみ、バッファーをコーディングするようにします。	デフォルトは、1 つの索引バッファー (BUFNI) と 2 つのデータ・バッファー (BUFND) です。
アクセス方式サービス・プログラムを使ってのレコードのロード	以下の場合は、アクセス方式サービス・プログラムの REPRO コマンドを使用します。 • ターゲット索引付きデータ・セットにすでにレコードが入っている。 • 入力順次データ・セットに、更新されるかまたは索引付きデータ・セットに挿入されるレコードが入っている。 COBOL プログラムを使用してファイルをロードする場合は、OPEN OUTPUT および ACCESS SEQUENTIAL を使用します。	これらの条件のもとでは、REPRO コマンドは、COBOL プログラムと同じ速さまたはそれよりも速く、索引付きデータ・セットを更新することができます。
ファイル・アクセス・モード	最高のパフォーマンスを得るためには、順次にレコードにアクセスします。	動的アクセスは、順次アクセスよりは効率が低く、ランダム・アクセスよりは効率的です。ランダム・アクセスでは、VSAM がそれぞれの要求について索引にアクセスしなければならないため、EXCP が増加します。
キーの設計	レコード内のキーを、高位部分が比較的一定になり、低位部分が頻繁に変わるように設計します。	この方法を使用すると、キーが最適に圧縮されます。

VSAM の局面	行うことができること	理由の説明とコメント
複数の代替索引	複数の代替索引の使用を避けます。	更新は基本パスを介して適用されなければならず、複数の代替パスを介して反映されることになるため、複数の代替索引を使用すると、パフォーマンスが低下する可能性があります。
相対ファイル編成	VSAM 可変長相対データ・セットではなく、VSAM 固定長相対データ・セットを使用します。	VSAM 固定長相対データ・セットは、COBOL シミュレート相対データ・セットに類似したパフォーマンス特性を持つ VSAM 可変長相対データ・セットと比較して、スペース的には効率がよくありませんが、実行時の効率はよくなります。
制御インターバル・サイズ (CISZ)	システム・プログラマーに、VSAM データ・セットのデータ・アクセスおよび将来の成長性に関する情報を提供してください。この情報から、システム・プログラマーは最適な制御インターバル・サイズ (CISZ) および FREESPACE サイズ (FSPC) を判別することができます。 制御域 (CA) の分割を最小限にするために、CISZ および FSPC に適切な値を選択します。クラスターに対して LISTCAT ALL コマンドを出して、現在の CA 分割の数を診断してから、すべての CA 分割を定期的に省略するためにクラスターを (EXPORT または IMPORT または REPRO を使用して) 圧縮することができます。	VSAM は、直接アクセス記憶装置 (DASD) の使用アルゴリズムに最も適するように CISZ を計算しますが、これがアプリケーションにとっては効率的でないことがあります。 4K の平均 CISZ はほとんどのアプリケーションに適しています。CISZ をより小さくすることは、挿入を犠牲にして (つまり、CISZ 分割がより多くなり、結果としてデータ・セット内のスペースがより多くなる)、ランダム処理の検索をより速くすることを意味します。CISZ をより大きくすると、各 READ についてチャネル間で転送されるデータがより多くなります。これは、大きな OS BLKSIZE と同様に、順次処理の場合に効率がよりよくなります。 多くの制御域 (CA) 分割は、VSAM のパフォーマンスにとっては不利です。FREESPACE 値は、ファイルの使用法に応じて、CA 分割に影響を与える場合があります。

関連タスク

172 ページの『VSAM ファイルのアクセス・モードの指定』

z/OS DFSMS: Using Data Sets の『Deciding how big a virtual resource pool to provide』の項

z/OS DFSMS: Using Data Sets の『Selecting the optimal percentage of free space』の項

関連参照

z/OS DFSMS: Access Method Services for Catalogs

第 11 章 行順次ファイルの処理

行順次ファイルは、階層ファイル・システム (HFS) に常駐し、印刷可能文字と特定の制御文字だけをデータとして含むファイルです。各レコードは、EBCDIC 改行文字 (X'15') で終了します (この文字はレコードの長さには含まれません)。これらは順次ファイルなので、各レコードは入力順序に従って次々に入れられます。プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で検索します (READ ステートメントを使って)。新しいレコードは、前のレコードの後に入れられます。

プログラムで行順次ファイルを処理するには、次のような COBOL 言語ステートメントを使用します。

- ENVIRONMENT DIVISION および DATA DIVISION 内でファイルを識別および記述する
- PROCEDURE DIVISION 内でファイルのレコードを処理する

レコードの作成後は、その長さまたは位置を変えることはできず、それを削除することはできません。

関連概念
UNIX システム・サービス ユーザーズ・ガイド

関連タスク
『COBOL での行順次ファイルおよびレコードの定義』
196 ページの『行順次ファイルの構造の記述』
198 ページの『行順次ファイル用の入出力ステートメントのコーディング』
201 ページの『行順次ファイルのエラーの処理』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
196 ページの『許可される制御文字』

COBOL での行順次ファイルおよびレコードの定義

ENVIRONMENT DIVISION の FILE-CONTROL 記入項目を使用して、以下のことを行ってください。

- COBOL プログラム内のファイルを行順次ファイルとして定義する。
- それらのファイルを外部ファイル名 (*ddnames* または環境変数名) と関連付ける。外部ファイル名とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。

次の例では、COMMUTER-FILE は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

© Copyright IBM Corp. 1991, 2001

```
FILE-CONTROL.
    SELECT COMMUTER-FILE
    ASSIGN TO COMMUTR
    ORGANIZATION IS LINE SEQUENTIAL
    ACCESS MODE IS SEQUENTIAL
    FILE STATUS IS ECODE.
```

外部名の前に、ASSIGN *assignment-name* 文節に編成フィールド (S- または AS-) を含めてはなりません。ACCESS 句と FILE STATUS 句はオプショナルです。

関連タスク
『行順次ファイルの構造の記述』
198 ページの『行順次ファイル用の入出力ステートメントのコーディング』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
『許可される制御文字』

許可される制御文字

印刷可能文字以外で行順次ファイルに入れることができるのは、次の表に示す制御文字だけです。16 進値は、EBCDIC で示されています。

16 進値	制御文字
X'05'	水平タブ
X'0B'	垂直タブ
X'0C'	書式送り
X'0D'	復帰
X'0E'	DBCS シフトアウト
X'0F'	DBCS シフトイン
X'15'	改行
X'16'	バックスペース
X'2F'	アラーム

改行文字は、レコード区切り文字として扱われます。他の制御文字はデータとして扱われ、レコードの一部分です。

関連タスク
195 ページの『COBOL での行順次ファイルおよびレコードの定義』

行順次ファイルの構造の記述

DATA DIVISION の FILE SECTION で、そのファイルのファイル記述 (FD) 記入項目をコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。

レコードの論理サイズは、RECORD 文節でコーディングします。行順次ファイルはストリーム・ファイルです。行順次ファイルの本質は文字中心であるので、物理レコードは可変長です。

次の例は、行順次ファイルの FD 記入項目を示しています。

固定長レコードの場合:

```
FILE SECTION.
FD  COMMUTER-FILE
    RECORD CONTAINS 80 CHARACTERS.
01  COMMUTER-RECORD.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

可変長レコードの場合:

```
FILE SECTION.
FD  COMMUTER-FILE
    RECORD VARYING FROM 16 TO 80 CHARACTERS.
01  COMMUTER-RECORD.
    05  COMMUTER-NUMBER        PIC  X(16).
    05  COMMUTER-DESCRIPTION   PIC  X(64).
```

同一の固定サイズをコーディングし、どのレベル 01 レコード記述項目の OCCURS DEPENDING ON 文節もファイルと関連付けられていない場合には、その固定サイズが論理レコード長になります。しかし、レコードの終わりにあるブランクはファイルに書き込まれないので、物理レコードは可変長になります。

関連タスク

195 ページの『COBOL での行順次ファイルおよびレコードの定義』
198 ページの『行順次ファイル用の入出力ステートメントのコーディング』
『行順次ファイルの定義および割り振り』

関連参照

データ部—ファイル記述記入項目 (*Enterprise COBOL 言語解説書*)

行順次ファイルの定義および割り振り

HFS の行順次ファイルは、DD ステートメントまたは環境変数を使用して定義することができます。これらのファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

行順次ファイルを定義するには、ASSIGN 文節の外部名と一致する名前を使用して、以下のいずれか 1 つを定義してください。

- DD 割り振り:
  - PATH='*absolute-path-name*' を指定する DD ステートメント
  - PATH('*absolute-path-name*') を指定する TSO 割り振り

  以下のオプションを指定することもできます。
  - PATHOPTS
  - PATHMODE
  - PATHDISP
- PATH(*absolute-path-name*) の値を持つ環境変数。これ以外の値を指定することはできません。

  たとえば、COBOL プログラムに HFS ファイル /u/myfiles/comtfile を、COMMUTR の *assignment-name* を持つ COBOL ファイルとして使用させるには、次のコマンドを使用してください。

  ```
  export COMMUTR="PATH(/u/myfiles/commuterfile)"
  ```

関連タスク
129 ページの『ファイルの割り振り』
195 ページの『COBOL での行順次ファイルおよびレコードの定義』

関連参照
MVS JCL 解説書

行順次ファイル用の入出力ステートメントのコーディング

行順次ファイルを処理するには、次の入出力ステートメントをコーディングします。

OPEN ファイルをプログラムに使用可能にします。

行順次ファイルは、INPUT、OUTPUT、または EXTEND としてオープンすることができます。行順次ファイルを I-O としてオープンすることはできません。

READ ファイルからレコードを読み取ります。

順次処理では、プログラムは、レコードをファイルの作成時に入力されたのと同じ順序で 1 つずつ読み取ります。

WRITE ファイルにレコードを作成します。

プログラムは、ファイルの終わりに新しいレコードを書き込みます。

CLOSE ファイルとプログラムの間の接続を解除します。

関連タスク
195 ページの『COBOL での行順次ファイルおよびレコードの定義』
196 ページの『行順次ファイルの構造の記述』
『行順次ファイルのオープン』
199 ページの『行順次ファイルからのレコードの読み取り』
199 ページの『行順次ファイルへのレコードの追加』
200 ページの『行順次ファイルのクローズ』
201 ページの『行順次ファイルのエラーの処理』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
READ ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのオープン

プログラムが READ または WRITE ステートメントを使用してファイルのレコードを処理するためには、その前に、プログラムは OPEN ステートメントでファイルをオープンしなければなりません。

OPEN ステートメントは、ファイルが使用可能であるか、または動的に割り振られていれば、機能します。

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

関連タスク
『行順次ファイルからのレコードの読み取り』
『行順次ファイルへのレコードの追加』
200 ページの『行順次ファイルのクローズ』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルからのレコードの読み取り

行順次ファイルから読み取りを行うには、ファイルをオープンして、READ ステートメントを使用します。

順次処理では、プログラムは、ファイルの作成時に入力された同じ順序でレコードを次々に読み取ります。

ファイル・レコード内の文字は、以下のいずれかの条件が起こるまで、一度に 1 文字ずつレコード域に読み込まれます。

- レコード区切り文字 (EBCDIC 改行文字) が検出される。

  区切り文字は廃棄され、レコード域の残りの部分はスペースで埋められます。(レコード域はファイル・レコードより長くなります。)
- レコード域全体が文字で埋められる。

  まだ読み取られていない次の文字がレコード区切り文字であると、その文字は廃棄されます。次の READ は、次のレコードの先頭文字から読み取りを行います。(レコード域はファイル・レコードと同じ長さになります。)

  これ以外の場合、まだ読み取られていない次の文字が、次の READ によって読み取られる先頭文字になります。(レコード域はファイル・レコードより短くなります。)

関連タスク
198 ページの『行順次ファイルのオープン』
『行順次ファイルへのレコードの追加』
200 ページの『行順次ファイルのクローズ』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルへのレコードの追加

行順次ファイルに追加するためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

レコード域の終わりにあるブランクは除去され、レコード区切り文字が追加されます。レコード域内の文字 (先頭文字から、追加されたレコード区切り文字まで) は、1 つのレコードとしてファイルに書き込まれます。

行順次ファイルに書き込まれるレコードは、USAGE DISPLAY および DISPLAY-1 項目だけを含んでいなければなりません。外部 10 進データ項目は符号なしでなければなりません。もし符号付きの場合は、SEPARATE CHARACTER 句で宣言しなければなりません。

関連タスク
198 ページの『行順次ファイルのオープン』
199 ページの『行順次ファイルからのレコードの読み取り』
『行順次ファイルのクローズ』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
OPEN ステートメント (*Enterprise COBOL 言語解説書*)
WRITE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのクローズ

プログラムを行順次ファイルから切り離すには、CLOSE ステートメントを使用します。すでにクローズされているファイルをクローズしようとすると、論理エラーになります。

行順次ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。

- 実行単位が正常に終了したとき。
- 実行単位が異常終了したときは、TRAP(ON) ランタイム・オプションが設定されている場合。
- 言語環境プログラム条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

  (条件が処理された後で) プログラムが再開する位置を変更することができます。これは、言語環境プログラムの CEEMRCR 呼び出し可能サービスで再開カーソルを移動するか、または C longjmp 呼び出しのような HLL 言語構造体を使用して行います。

これらの暗黙の CLOSE 操作が実行されると、ファイル状況コードが設定されますが、EXCEPTION/ERROR 宣言は呼び出されません。

関連タスク
198 ページの『行順次ファイルのオープン』
199 ページの『行順次ファイルからのレコードの読み取り』
199 ページの『行順次ファイルへのレコードの追加』
197 ページの『行順次ファイルの定義および割り振り』

関連参照
CLOSE ステートメント (*Enterprise COBOL 言語解説書*)

行順次ファイルのエラーの処理

入出力ステートメントの操作が失敗しても、COBOL がユーザーに代わって訂正処置を取るわけではありません。入出力エラーが発生した後でプログラムの実行を継続するかどうかを選択してください。

COBOL は、特定の行順次入出力エラーを代行受信して処理するために、以下の技法を提供します。

- End of file phrase (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 文節

これらの技法を使用しなかった場合に、入出力処理でエラーが起こると、言語環境プログラム条件が発生します。

FILE STATUS 文節を使用する場合は、必ずキーを調べ、キー値に基づいて適切なアクションを取るようにしてください。キーを調べなくても、プログラムが実行を継続できる可能性がありますが、予期したものではない結果になることがあります。

関連タスク

198 ページの『行順次ファイル用の入出力ステートメントのコーディング』
250 ページの『入出力操作でのエラーの処理』

第 12 章 ファイルのソートおよびマージ

SORT または MERGE ステートメントを使用すると、レコードを特定のシーケンスで並べることができます。

SORT ステートメント
(ファイルまたは内部プロシージャーから) 順序付けられていない入力を受け入れ、要求されたシーケンスで出力を (ファイルまたは内部プロシージャーに) 作成します。ソートの前に、レコードを追加、削除、または変更することができます。

MERGE ステートメント
2 つ以上の順序付けられたファイルからのレコードを比較し、それらを順序正しく結合します。マージの前に、レコードを追加、削除、または変更することができます。

同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。プログラムにいくつのソート操作およびマージ操作を含めても構いません。また、同じ操作を何度も実行しても構いませんし、異なる操作を実行しても構いません。ただし、1 つの操作が終了してからでなければ、別の操作を開始することはできません。

Enterprise COBOL では、ソートおよびマージ用の IBM ライセンス・プログラムは DFSORT または同等のプログラムでなければなりません。DFSORT に言及している場所では、任意の同等のソートまたはマージ製品を使用することができます。

SORT または MERGE ステートメントを含む COBOL プログラムは、16MB 境界より上または下のいずれにでも常駐させることができます。

ソートまたはマージの一般的な手順は、次のとおりです。

1. ソートまたはマージに使用するソート・ファイルまたはマージ・ファイルを記述する。
2. ソートまたはマージする入力を記述する。レコードをソート前に処理したい場合には、入力プロシージャーをコーディングしてください。
3. ソートまたはマージからの出力を記述する。レコードをソートまたはマージした後に処理したい場合には、出力プロシージャーをコーディングしてください。
4. ソートまたはマージを要求する。
5. ソートまたはマージ操作が成功したかどうかを判別する。

制約事項:

- UNIX のもとでは、SORT または MERGE ステートメントを含む COBOL プログラムを実行することはできません。この制約には、BPXBATCH も含まれます。
- THREAD オプションを指定してコンパイルしたプログラムで SORT ステートメントまたは MERGE ステートメントを使用することはできません。これには、オブジェクト指向構文を使用するプログラムとマルチスレッド・アプリケーションも含まれます。これらはいずれも THREAD オプションを必要とします。

© Copyright IBM Corp. 1991, 2001

関連概念
『ソートおよびマージ・プロセス』

関連タスク
『ソートまたはマージ・ファイルの記述』
205 ページの『ソートまたはマージへの入力の記述』
208 ページの『ソートまたはマージからの出力の記述』
211 ページの『ソートまたはマージの要求』
215 ページの『ソートまたはマージの成否の判断』
217 ページの『FASTSRT を使ってのソートのパフォーマンスの向上』
220 ページの『ソート動作の制御』
DFSORT アプリケーション・プログラミングの手引き

関連参照
224 ページの『CICS SORT アプリケーションの制約事項』
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 言語解説書*)

ソートおよびマージ・プロセス

ファイルのソート時、そのすべてのレコードは、各レコード内の 1 つまたは複数のフィールドの内容 (キー) に従って順序付けされます。複数のキーがある場合は、レコードはまず最初の (基本) キーの内容に従ってソートされ、次に 2 番目のキーの内容に従ってソートされる、というようになります。レコードは、各キーの昇順または降順にソートすることができます。

ファイルをソートするには、COBOL の `SORT` ステートメントを使用します。

複数のファイルのマージ時には (これらのファイルはソート済みでなければなりません)、レコードは、各レコード内の 1 つまたは複数のキーの内容に従って結合され、順序付けされます。ソートの場合と同様、レコードはまず最初の (基本) キーの内容に従って順序付けされ、次に 2 番目のキーの内容に従って順序付けされる、というようになります。レコードは、各キーの昇順または降順に順序付けすることができます。

`MERGE . . . USING` を使用して、順序付けられた 1 つのファイルとして結合したい複数のファイルの名前を指定します。マージ操作では、入力ファイルのレコード内のキーを比較し、順序付けられたレコードを 1 つずつ、出力プロシージャーの `RETURN` ステートメントに、または `GIVING` 句で指定されたファイルに渡します。

関連参照
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 言語解説書*)

ソートまたはマージ・ファイルの記述

ソートまたはマージに使用するソート・ファイルは、次のように記述してください。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つまたは複数の SELECT ステートメントをコーディングし、ソート・ファイルの名前を指定します。たとえば、次のように記述します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Sort-Work-1 ASSIGN TO SortFile.
```

   Sort-Work-1 は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. そのソート・ファイルを、DATA DIVISION の FILE SECTION の SD 記入項目で記述します。

   それぞれの SD 記入項目がレコード記述を含んでいなければなりません。たとえば、次のように記述します。

```
DATA DIVISION.
FILE SECTION.
SD  Sort-Work-1
    RECORD CONTAINS 100 CHARACTERS.
01  SORT-WORK-1-AREA.
    05  SORT-KEY-1   PIC  X(10).
    05  SORT-KEY-2   PIC  X(10).
    05  FILLER       PIC  X(80).
```

ソートまたはマージについての SELECT ステートメントと SD 記入項目は、WORKING-STORAGE からのデータ項目をソートまたはマージするだけの場合でも必要です。

SD 記入項目で記述するファイルは、ソートまたはマージ操作に使用される作業ファイルです。このファイルに入力または出力操作を実行することはできません。このファイルの DD 名定義を与える必要はありません。

関連参照

16 ページの『FILE SECTION の記入項目』

ソートまたはマージへの入力の記述

ソートまたはマージ用の入力ファイルは、以下の手順に従って記述してください。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つまたは複数の SELECT ステートメントをコーディングし、入力ファイルの名前を指定します。たとえば、次のように記述します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Input-File ASSIGN TO InFile.
```

   Input-File は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. その入力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。たとえば、次のように記述します。

```
DATA DIVISION.
FILE SECTION.
FD  Input-File
    LABEL RECORDS ARE STANDARD
```

```
       BLOCK CONTAINS 0 CHARACTERS
       RECORDING MODE IS F
       RECORD CONTAINS 100 CHARACTERS.
   01  Input-Record   PIC X(100).
```

関連タスク

207 ページの『入力プロシージャーのコーディング』
211 ページの『ソートまたはマージの要求』

関連参照

16 ページの『FILE SECTION の記入項目』

例: SORT 用のソート・ファイルおよびの入力ファイルの記述

次の例は、ソート・ファイルおよび入力ファイルを記述するのに必要な ENVIRONMENT DIVISION および DATA DIVISION の記入項目を示しています。

```
 ID Division.
 Program-ID. SmplSort.
 Environment Division.
 Input-Output Section.
 File-Control.
*
* Assign name for a working file is
* treated as documentation.
*
     Select Sort-Work-1 Assign To SortFile.
     Select Sort-Work-2 Assign To SortFile.
     Select Input-File  Assign To InFile.
 . . .
 Data Division.
 File Section.
 SD  Sort-Work-1
     Record Contains 100 Characters.
 01  Sort-Work-1-Area.
     05  Sort-Key-1    Pic  X(10).
     05  Sort-Key-2    Pic  X(10).
     05  Filler        Pic  X(80).
 SD  Sort-Work-2
     Record Contains 30 Characters.
 01  Sort-Work-2-Area.
     05  Sort-Key      Pic  X(5).
     05  Filler        Pic  X(25).
 FD  Input-File
     Label Records Are Standard
     Block Contains 0 Characters
     Recording Mode is F
     Record Contains 100 Characters.
 01  Input-Record      Pic  X(100).
 . . .
 Working-Storage Section.
 01  EOS-Sw            Pic  X.
 01  Filler.
     05  Table-Entry Occurs 100 Times
             Indexed By X1    Pic X(30).
     . . .
```

関連タスク

211 ページの『ソートまたはマージの要求』

入力プロシージャーのコーディング

入力ファイルのレコードを、それらがソート・プログラムに解放される前に処理したい場合には、SORT ステートメントの INPUT PROCEDURE 句を使用してください。入力プロシージャーを使用して、以下のことを行うことができます。

- データ項目を WORKING-STORAGE からソート・ファイルに解放する。
- プログラム内の別な場所ですでに読み取られているレコードを解放する。
- 入力レコードからレコードを読み取り、それらを選択または処理し、それらをソート・ファイルに解放する

それぞれの入力プロシージャーは、段落またはセクションのいずれかで構成されなければなりません。たとえば、WORKING-STORAGE の表からのレコードをソート・ファイル SORT-WORK-2 に解放するには、次のようにコーディングすることができます。

```
    SORT SORT-WORK-2
      ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE 600-SORT3-INPUT-PROC
    . . .
600-SORT3-INPUT-PROC SECTION.
    PERFORM WITH TEST AFTER
      VARYING X1 FROM 1 BY 1 UNTIL X1 = 100
      RELEASE SORT-WORK-2-AREA FROM TABLE-ENTRY (X1)
    END-PERFORM.
```

レコードをソート・プログラムに転送するためには、すべての入力プロシージャーに少なくとも 1 つの RELEASE または RELEASE FROM ステートメントが含まれていなければなりません。たとえば、X から A を解放するには、次のように入力することができます。

```
MOVE X TO A.
RELEASE A.
```

あるいは、次のようにコーディングできます。

```
RELEASE A FROM X.
```

次の表では、RELEASE ステートメントと RELEASE FROM ステートメントを比較しています。

RELEASE	**RELEASE FROM**

MOVE EXT-RECORD
  TO SORT-EXT-RECORD
PERFORM RELEASE-SORT-RECORD
. . .
RELEASE-SORT-RECORD.
  RELEASE SORT-RECORD
	PERFORM RELEASE-SORT-RECORD
	. . .
	RELEASE-SORT-RECORD.
	  RELEASE SORT-RECORD
	  FROM SORT-EXT-RECORD

関連参照

210 ページの『入出力プロシージャーに関する制約事項』

RELEASE ステートメント (*Enterprise COBOL 言語解説書*)

ソートまたはマージからの出力の記述

ソートまたはマージからの出力がファイルである場合は、以下の手順に従ってファイルを記述しなければなりません。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に SELECT ステートメントをコーディングし、出力ファイルの名前を指定します。たとえば、次のように記述します。

   ```
   ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
       SELECT Output-File ASSIGN TO OutFile.
   ```

   Output-File は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. その出力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。たとえば、次のように記述します。

   ```
   DATA DIVISION.
   FILE SECTION.
   FD  Output-File
       LABEL RECORDS ARE STANDARD
       BLOCK CONTAINS 0 CHARACTERS
       RECORDING MODE IS F
       RECORD CONTAINS 100 CHARACTERS.
   01  Output-Record   PIC X(100).
   ```

関連タスク

『出力プロシージャーのコーディング』

211 ページの『ソートまたはマージの要求』

関連参照

16 ページの『FILE SECTION の記入項目』

出力プロシージャーのコーディング

ソート済みのレコードをソート作業ファイルから別のファイルに書きこむ前に、それらの選択、編集、またはそれ以外の変更を行いたい場合には、SORT ステートメントの OUTPUT PROCEDURE 句を使用してください。

それぞれの出力プロシージャーは、セクションまたは段落のいずれかで構成されなければなりません。また、出力プロシージャーには、以下の両方を含めなければなりません。

- 少なくとも 1 つの RETURN または RETURN INTO ステートメント。
- レコードの処理に必要なステートメント。レコードは、RETURN ステートメントによって一度に 1 つずつ使用可能になります。

RETURN ステートメントによって、ソート済みの各レコードが出力プロシージャーから使用可能になります (ソート・ファイルに対する RETURN ステートメントは、入力ファイルに対する READ ステートメントに似ています)。

RETURN ステートメントとともに AT END および END-RETURN 句を使用することができます。AT END 句の命令ステートメントは、ソート・ファイルからすべてのレコー

ドが戻された後で実行されます。END-RETURN 明示的範囲終了符号は、RETURN ステートメントの有効範囲を区切る役割をします。

RETURN ではなく RETURN INTO ステートメントを使用すると、レコードは WORKING-STORAGE または出力域に戻されます。

DFSORT を使用する際のコーディングの考慮事項

RETURN ステートメントが、COBOL プログラムが実行を終了する前に AT END 条件を検出しないと、SORT ステートメントが異常終了し、DFSORT メッセージ IEC025A が出されることがあります。この状況を回避するためには、次のようにしてください。

1. AT END 句を指定した RETURN ステートメントをコーディングする。
2. AT END 条件が検出されるまで、RETURN ステートメントが実行されるようにする。

AT END 条件が起こるのは、最後のレコードがソート作業ファイルからプログラムに戻され、後続の RETURN ステートメントが実行された後です。

『例: DFSORT を使用する際の出力プロシージャーのコーディング』

関連参照
210 ページの『入出力プロシージャーに関する制約事項』
RETURN ステートメント (*Enterprise COBOL 言語解説書*)

例: DFSORT を使用する際の出力プロシージャーのコーディング

次の例は、プログラムが実行を終了する前に RETURN ステートメントが AT END 条件に出合うようにしたコーディング手法を示しています。AT END 句をコーディングした RETURN ステートメントは、AT END 条件が起こるまで実行されます。

```
IDENTIFICATION DIVISION.
DATA DIVISION.
FILE SECTION.
SD  OUR-FILE.
01  OUR-SORT-REC.
    03  SORT-KEY               PIC X(10).
    03  FILLER                 PIC X(70).
. . .
WORKING-STORAGE SECTION.
01  WS-SORT-REC                PIC X(80).
01  END-OF-SORT-FILE-INDICATOR  PIC X VALUE 'N'.
    88  NO-MORE-SORT-RECORDS          VALUE 'Y'.
. . .
PROCEDURE DIVISION.
A-CONTROL SECTION.
    SORT OUR-FILE ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE IS B-INPUT
      OUTPUT PROCEDURE IS C-OUTPUT.
    . . .
B-INPUT SECTION.
    MOVE . . .. . .. TO WS-SORT-REC.
    RELEASE OUR-SORT-REC FROM WS-SORT-REC.
    . . .
C-OUTPUT SECTION.
    DISPLAY 'STARTING READS OF SORTED RECORDS: '.
    RETURN OUR-FILE
      AT END
        SET NO-MORE-SORT-RECORDS TO TRUE.
```

```
PERFORM WITH TEST BEFORE UNTIL NO-MORE-SORT-RECORDS
  IF SORT-RETURN = 0 THEN
    DISPLAY 'OUR-SORT-REC = ' OUR-SORT-REC
    RETURN OUR-FILE
      AT END
        SET NO-MORE-SORT-RECORDS TO TRUE
  END-IF
END-PERFORM.
```

入出力プロシージャーに関する制約事項

SORT によって呼び出されるそれぞれの入出力プロシージャー、および MERGE によって呼び出されるそれぞれの出力プロシージャーには、以下の制約事項が適用されます。

- プロシージャーに SORT または MERGE ステートメントを含めてはなりません。
- プロシージャーに STOP RUN、EXIT PROGRAM、または GOBACK ステートメントを含めてはなりません。
- プロシージャーの中で ALTER、GO TO、および PERFORM ステートメントを使用することによって、入力または出力プロシージャーの外側にあるプロシージャー名を参照することができます。しかし、GO TO または PERFORM ステートメントの後で、その入力または出力プロシージャーに制御権を戻さなければなりません。
- PROCEDURE DIVISION のほかの部分に、入力または出力プロシージャーの内部への制御権の移動を記述してはなりません (ただし、宣言セクションからの制御権の戻りは例外です)。
- 入力または出力プロシージャーの中から、標準リンケージ規約に従うプログラムを呼び出すことができます。ただし、呼び出されるプログラムから、SORT または MERGE ステートメントを出してはなりません。
- SORT または MERGE 操作時には、SD データ項目が使用されます。出力プロシージャーの中で、最初の RETURN が実行される前に、このデータ項目を使用してはなりません。最初の RETURN ステートメントの前に、データをこのレコード域に移動すると、戻される最初のレコードが上書きされます。
- 言語環境プログラムの条件処理では、入力または出力プロシージャーの中でユーザー作成条件処理ルーチンを設定することは許可されません。

関連タスク

207 ページの『入力プロシージャーのコーディング』
208 ページの『出力プロシージャーのコーディング』
リンク・エディットおよび実行の計画 (*言語環境プログラム プログラミング・ガイド*)

ソートおよびマージ・データ・セットの定義

z/OS のもとで DFSORT を使用するには、実行時 JCL に DD ステートメントをコーディングして、必要なデータ・セットを記述しなければなりません。

ソートまたはマージ作業域

少なくとも 3 つのデータ・セットを定義します。SORTWK01、SORTWK02、SORTWK03、...、SORTWK*nn* (*nn* は 99 以下) などです。これらのデータ・セットは HFS にあってはなりません。

SYSOUT データ・セット名を変更しない限り、ソート診断メッセージ用にこのデータ・セットを定義します。(名前は、SORT-CONTROL データ・セット内の OPTION 制御ステートメントの MSGDDN キーワードを使用して、または SORT-MESSAGE 特殊レジスターを使用して変更してください。)

SORTCKPT
ソートまたはマージでチェックポイントを設定する場合に、このデータ・セットを定義します。

入出力 必要に応じて、入出力データ・セットを定義します。

SORTLIB (DFSORT ライブラリー)
ソート・モジュールを含むライブラリー (たとえば、SYS1.SORTLIB) を定義します。

関連タスク
220 ページの『ソート動作の制御』
223 ページの『DFSORT によるチェックポイント・リスタートの使用』

可変長レコードのソート

ソートへの入力ファイルに可変長レコードが入っていても、ソート作業ファイルは可変長として定義しないと可変長になりません。

コンパイラーがソート作業ファイルを可変長と見なすのは、そのファイルの SD 記入項目で以下のいずれかをコーディングした場合です。

- RECORD IS VARYING 文節
- サイズが異なるレコードを定義する 2 つ以上のレコード記述 (あるいは、OCCURS DEPENDING ON 文節を含むレコード)

SD 記入項目では RECORDING MODE 文節は認められないため、ソート作業ファイルについて RECORDING MODE V をコーディングすることはできません。

可変長ファイルに対するソートのパフォーマンスを向上させるには、SMS= 制御カードまたは SORT-MODE-SIZE 特殊レジスターで、入力ファイルに最も頻繁に出現するレコード長 (モーダル長) を指定します。

関連タスク
221 ページの『制御ステートメントによる DFSORT デフォルトの変更』
220 ページの『ソート動作の制御』

ソートまたはマージの要求

入力ファイル (MERGE の場合は複数のファイル) からレコードを直接読み取るには、SORT ... USING または MERGE ... USING と、SELECT ステートメントで宣言した入力ファイル (複数の場合もある) の名前を使用してください。コンパイラーは、入力ファイルをオープンし、レコードを読み取り、レコードをソートまたはマージ・プログラムに解放し、そして入力ファイルをクローズするための入力プロシージャーを生成します。SORT または MERGE ステートメントが実行を開始するとき、入力ファイルがオープンされていてはなりません。

ソース済みまたはマージ済みレコードをそれ以上の処理を行わずに、ソート・プログラムまたはマージ・プログラムから別のファイルに直接転送するには、SORT ... GIVING またはMERGE ... GIVING と、SELECT ステートメントで宣言した出力ファイルの名前を使用してください。コンパイラーは、出力ファイルをオープンし、レコードを戻し、レコードを書き込み、そしてファイルをクローズするための出力プロシージャーを生成します。SORT または MERGE ステートメントが実行を開始するとき、出力ファイルがオープンしていてはなりません。たとえば、次のように記述します。

```
SORT Sort-Work-1
    ON ASCENDING KEY Sort-Key-1
    USING Input-File
    GIVING Output-File.
```

SORT または MERGE ステートメントの USING または GIVING ファイルは、HFS に常駐する順次ファイルにすることができます。

206 ページの『例: SORT 用のソート・ファイルおよびの入力ファイルの記述』

レコードがソートされる前にソート・レコードに入力プロシージャーを実行したい場合には、SORT ... INPUT PROCEDURE を使用してください。ソートされたレコードに出力プロシージャーを実行したい場合には、SORT ... OUTPUT PROCEDURE を使用してください。たとえば、次のように記述します。

```
SORT Sort-Work-1
    ON ASCENDING KEY Sort-Key-1
    INPUT PROCEDURE EditInputRecords
    OUTPUT PROCEDURE FormatData.
```

213 ページの『例: 入出力プロシージャーを使用したソート』

制約事項: MERGE ステートメントで入力プロシージャーを使用することはできません。マージ操作への入力ソースは、すでにソート済みのファイルの集合でなければなりません。ただし、マージされたレコードに出力プロシージャーを実行したい場合には、MERGE ... OUTPUT PROCEDURE を使用してください。たとえば、次のように記述します。

```
MERGE Merge-Work
    ON ASCENDING KEY Merge-Key
    USING Input-File-1 Input-File-2 Input-File-3
    OUTPUT PROCEDURE ProcessOutput.
```

Merge-Work は、DATA DIVISION の FILE SECTION の SD ステートメントで定義しなければならず、入力ファイルは、FILE SECTION の FD ステートメントで定義しなければなりません。

ソートまたはマージ基準の設定

ソートまたはマージ基準を設定するには、以下のステップに従ってください。

1. ソートまたはマージするファイルのレコード記述の中で、その操作を実行する対象となるキー (複数の場合もある) を定義します。

   キーの最大数は規定されていませんが、キーはレコード記述の最初の 4092 バイト内になければなりません。

キーの全長が 4092 バイトを超えてはなりません。ただし、EQUALS キーワードが DFSORT OPTION 制御ステートメントにコーディングされている場合は別で、その場合はキーの全長が 4088 バイトを超えてはなりません。

制約事項: キーは可変位置にすることはできません。

2. SORT または MERGE ステートメントに、順序付けに使用するキー・フィールドを指定します。キーは昇順または降順としてコーディングできます。複数のキーをコーディングする場合、一部を昇順にし、残りを降順にすることができます。

   左端のキーが基本キーです。次のキーが 2 次キー、というようになります。

国別データ項目をソートおよびマージのキーとして指定することができます。2 進照合シーケンスはこのようなキーに適用されます。ソートまたはマージのキーとして国別データ項目を指定する場合は、SORT または MERGE ステートメントの COLLATING SEQUENCE 句はいずれもそのキーに適用されません。

同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。プログラムはいくつのソート操作およびマージ操作でも実行することができます。ただし、1 つの操作が終了してからでなければ、次の操作を開始することはできません。

関連概念

631 ページの『付録 B. 複合 OCCURS DEPENDING ON』 (可変位置項目)

関連タスク

210 ページの『ソートおよびマージ・データ・セットの定義』

関連参照

SORT 制御ステートメント (*DFSORT アプリケーション・プログラミングの手引き*)
SORT ステートメント (*Enterprise COBOL 言語解説書*)
MERGE ステートメント (*Enterprise COBOL 言語解説書*)

例: 入出力プロシージャーを使用したソート

以下は、SORT ステートメントにおける入出力プロシージャーの使用例です。この例では、基本キー SORT-GRID-LOCATION と 2 次キー SORT-SHIFT を SORT ステートメントで使用する前に、それらのキーをどのように DATA DIVISION に定義できるかも示します。

```
DATA DIVISION.
. . .
SD  SORT-FILE
    RECORD CONTAINS 115 CHARACTERS
    DATA RECORD SORT-RECORD.
01  SORT-RECORD.
    05  SORT-KEY.
        10  SORT-SHIFT              PIC X(1).
        10  SORT-GRID-LOCATION      PIC X(2).
        10  SORT-REPORT             PIC X(3).
    05  SORT-EXT-RECORD.
        10  SORT-EXT-EMPLOYEE-NUM   PIC X(6).
        10  SORT-EXT-NAME           PIC X(30).
        10  FILLER                  PIC X(73).
. . .
WORKING-STORAGE SECTION.
01  TAB1.
    05 TAB-ENTRY OCCURS 10 TIMES
```

```
           INDEXED BY TAB-INDX.
        10  WS-SHIFT               PIC X(1).
        10  WS-GRID-LOCATION       PIC X(2).
        10  WS-REPORT              PIC X(3).
        10  WS-EXT-EMPLOYEE-NUM    PIC X(6).
        10  WS-EXT-NAME            PIC X(30).
        10  FILLER                 PIC X(73).
. . .
PROCEDURE DIVISION.
    . . .
    SORT SORT-FILE
        ON ASCENDING KEY SORT-GRID-LOCATION SORT-SHIFT
        INPUT PROCEDURE 600-SORT3-INPUT
        OUTPUT PROCEDURE 700-SORT3-OUTPUT.
    . . .
600-SORT3-INPUT.
    PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
        RELEASE SORT-RECORD FROM TAB-ENTRY(TAB-INDX)
    END-PERFORM.
. . .
700-SORT3-OUTPUT.
    PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
        RETURN SORT-FILE INTO TAB-ENTRY(TAB-INDX)
            AT END DISPLAY 'Out Of Records In SORT File'
        END-RETURN
    END-PERFORM.
```

関連タスク

211 ページの『ソートまたはマージの要求』

代替照合シーケンスの選択

レコードのソートまたはマージは、EBCDIC または ASCII 照合シーケンスか、あるいは別の照合シーケンスで行うことができます。OBJECT-COMPUTER 段落で PROGRAM COLLATING SEQUENCE 文節をコーディングしない限り、デフォルトの照合シーケンスは EBCDIC です。PROGRAM COLLATING SEQUENCE 文節で指定されたシーケンスをオーバーライドするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句を使用してください。プログラムの SORT または MERGE ステートメントごとに異なる照合シーケンスを使用することができます。

ASCII ファイルをソートまたはマージするときには、ASCII 照合シーケンスを要求しなければなりません。このためには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句をコーディングし、SPECIAL-NAMES 段落に STANDARD-1 として *alphabet-name* を定義します。

関連タスク

10 ページの『照合シーケンスの指定』

関連参照

SORT ステートメント (*Enterprise COBOL 言語解説書*)

ウィンドウ化日付フィールドに基づいたソート

使用している DFSORT のバージョンが Y2PAST オプションをサポートする場合は、ウィンドウ化日付フィールドをソート・キーとして指定することができます。このオプションがサポートされていれば、DFSORT は、ウィンドウ化日付シーケンスに基づいてソートまたはマージすることができます。

ウィンドウ化日付フィールドに基づいてソートするためには、DATE FORMAT 文節を使用してウィンドウ化日付フィールドを定義し、そのフィールドをソート・キーとして使用してください。DFSORT は、コンパイル単位で使用された同じ世紀ウィンドウを使用します。YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを指定してください。

DFSORT は年末尾型ウィンドウ化日付フィールドをサポートしますが、コンパイラー自体は、MERGE または SORT 以外のステートメントの年末尾型ウィンドウ化日付フィールドについては、自動ウィンドウ操作を行いません。

関連タスク
579 ページの『日付によるソートおよびマージ』

関連参照
358 ページの『YEARWINDOW』
DATE FORMAT 文節 (*Enterprise COBOL 言語解説書*)
OPTION 制御ステートメント (Y2PAST オプション) (*DFSORT アプリケーション・プログラミングの手引き*)

同じキーを持つレコードのオリジナル・シーケンスの保持

以下のいずれかの方法で、同じ照合レコードの順序を入力から出力まで保持することができます。

- EQUALS オプションをデフォルトとして指定して DFSORT をインストールします。
- 実行時に、IGZSRTCD データ・セットの OPTION カードにEQUALS キーワードを与えます。
- SORT ステートメントで WITH DUPLICATES IN ORDER 句を使用します。これにより、IGZSRTCD データ・セットの OPTION カードに EQUALS キーワードが追加されます。

  OPTION カードで NOEQUALS キーワードを使用せず、かつ DUPLICATES 句を使用するようにしてください。さもないと、実行単位が終了してしまいます。

関連参照
OPTION 制御ステートメント (*DFSORT アプリケーション・プログラミングの手引き*)

ソートまたはマージの成否の判断

DFSORT プログラムは、ソートまたはマージの終了後に、以下のいずれかの完了コードを戻します。

0 ソートまたはマージの正常終了

16 ソートまたはマージの不成功な完了

完了コードは、SORT-RETURN 特殊レジスターに保管されます。このレジスターの内容は、それぞれの SORT または MERGE ステートメントが実行されるたびに変更されます。

それぞれの SORT または MERGE ステートメントの後で、正常終了かどうかをテストしなければなりません。たとえば、次のように記述します。

```
    SORT SORT-WORK-2
        ON ASCENDING KEY SORT-KEY
        INPUT PROCEDURE IS 600-SORT3-INPUT-PROC
        OUTPUT PROCEDURE IS 700-SORT3-OUTPUT-PROC.
    IF SORT-RETURN NOT=0
        DISPLAY "SORT ENDED ABNORMALLY. SORT-RETURN = " SORT-RETURN.
    . . .
600-SORT3-INPUT-PROC SECTION.
    . . .
700-SORT3-OUTPUT-PROC SECTION.
    . . .
```

プログラムの中で SORT-RETURN をまったく参照しないと、COBOL ランタイムで戻りコードがテストされます。戻りコードが 16 であると、COBOL はランタイム診断メッセージを出します。

SORT または MERGE ステートメントの 1 つまたは 2 つ以上 (しかし、必ずしも全部ではない) について SORT-RETURN をテストすると、COBOL ランタイムで戻りコードが検査されません。

デフォルトでは、DFSORT 診断メッセージは SYSOUT データ・セットに送られます。このデフォルトを変更したい場合は、DFSORT の OPTION 制御カードの MSGDDN パラメーターを使用するか、SORT-MESSAGE 特殊レジスターを使用してください。

関連タスク

219 ページの『NOFASTSRT によるソート・エラーの検査』
220 ページの『ソート動作の制御』

関連参照

DFSORT メッセージおよび戻りコード (*DFSORT アプリケーション・プログラミングの手引き*)

ソートまたはマージ操作の途中停止

ソートまたはマージ操作を停止するには、SORT-RETURN 特殊レジスターを使用します。次のいずれかの方法で、レジスターに整数 16 を移動してください。

- 入力または出力プロシージャーの中で MOVE を使用する。

  ソートまたはマージ処理は、次の RELEASE または RETURN ステートメントが実行された直後に停止されます。

- USING または GIVING ファイルの処理中に入る宣言セクションの中でこのレジスターをリセットする。

  ソートまたはマージ処理は、次の暗黙の RELEASE または RETURN (USING または GIVING ファイルに対して、あるレコードの読み取りまたは書き込みの後で起こる) が実行された直後に停止されます。

制御は、その後、SORT または MERGE ステートメントの次のステートメントに戻ります。

FASTSRT を使ってのソートのパフォーマンスの向上

FASTSRT コンパイラー・オプションを使用すると、ほとんどのソート操作のパフォーマンスが向上します。FASTSRT を使用した場合は、DFSORT プロダクトが (Enterprise COBOL の代わりに) 、以下のステートメントで指定された入出力ファイルに入出力操作を実行します。

```
SORT . . . USING
SORT . . . GIVING
```

コンパイラーは通知メッセージを出し、FASTSRT がパフォーマンスを向上させることができるステートメントを指摘します。

使用上の注意

- FASTSRT を使用する場合は、DFSORT オプション SORTIN または SORTOUT は使用できません。FASTSRT コンパイラー・オプションは、USING または GIVING ファイルとして使用している行順次ファイルには適用されません。
- FASTSRT を使用する場合は、ファイル状況を指定しても、ソート操作中はファイル状況は無視されます。

関連参照

327 ページの『FASTSRT』
『JCL に関する FASTSRT の要件』
『ソート入出力ファイルに関する FASTSRT の要件』

JCL に関する FASTSRT の要件

実行時 JCL では、ソート作業ファイル (SORTWK*nn*) を、テープ・データ・セットではなく、直接アクセス装置に割り当てなければなりません。

入出力ファイルの場合、JCL の DD ステートメントの DCB パラメーターは FD 記述と一致していなければなりません。

ソート入出力ファイルに関する FASTSRT の要件

FASTSRT を指定しても、FASTSRT の要件が満たされていないと、コンパイラーはメッセージを出し、代わりに COBOL ランタイムが入出力を実行します。その場合、プログラムはパフォーマンスの向上を望めなくなります。

FASTSRT を使用するためには、ソートへの入力ファイルおよびソートからの出力ファイルを、次のように記述し、処理しなければなりません。

- USING 句では 1 つの入力ファイルのみ指定することができます。GIVING 句では 1 つの出力ファイルのみ指定することができます。
- 入力ファイルに対して入力プロシージャーを使用したり、出力ファイルに対して出力プロシージャーを使用したりしてはなりません。

  入力または出力プロシージャーを使用する代わりに、次の DFSORT 制御ステートメントを使用できます。

  – INREC
  – OUTREC
  – INCLUDE

- OMIT
- STOPAFT
- SKIPREC
- SUM

  多くの DFSORT の機能では、入力または出力プロシージャーでよく使用されるのと同様の操作を実行します。代わりに適切な DFSORT 制御ステートメントをコーディングし、それらを IGZSRTCD データ・セットまたは SORTCNTL データ・セットに入れてください。

- 出力 FD 記入項目に LINAGE 文節をコーディングしないでください。
- ソートで使用する FD に適用するために、INPUT 宣言 (入力ファイル用)、OUTPUT 宣言 (出力ファイル用)、またはファイル特定宣言 (入力ファイルまたは出力ファイル用) をコーディングしてはなりません。
- 可変長の相対ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 行順次ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 入力ファイルまたは出力ファイルのいずれかについて、SD と FD 記入項目のレコード記述では、両方とも同じ形式 (固定長または可変長) を定義しなければなりません。また、SD と FD の最大レコード・サイズでは、同じレコード長を定義しなければなりません。

出力ファイルに RELATIVE KEY 文節をコーディングしても、これはソートでは設定されないことに注意してください。

パフォーマンスのヒント: 入出力レコードをブロック化すると、ソート処理のパフォーマンスが大幅に向上する可能性があります。

QSAM の要件

- QSAM ファイルのレコード形式は、固定長、可変長、またはスパンでなければなりません。
- QSAM 入力ファイルは空であっても構いません。
- 入力と出力の両方に同じ QSAM ファイルを使用する場合は、2 つの異なる DD ステートメントを使用してこのファイルを記述しなければなりません。たとえば、FILE-CONTROL SECTION では、次のようにコーディングすることができます。

  ```
  SELECT FILE-IN ASSIGN INPUTF.
  SELECT FILE-OUT ASSIGN OUTPUTF.
  ```

  DATA DIVISION で、FILE-IN と FILE-OUT の両方についての FD 記入項目を持つことになります。この場合、FILE-IN と FILE-OUT は、それらの名前を除いて同じです。

  PROCEDURE DIVISION では、SORT ステートメントを次のように記述します。

  ```
  SORT file-name
    ASCENDING KEY data-name-1
    USING FILE-IN GIVING FILE-OUT
  ```

  JCL では、次のようにコーディングします。

```
//INPUTF DD DSN=INOUT,DISP=SHR
//OUTPUTF DD DSN=INOUT,DISP=SHR
```

ここで、データ・セット INOUT はカタログされています。

これに反して、USING 句と GIVING 句に同じファイル名をコーディングするか、または入力ファイルと出力ファイルに同じ DD 名を割り当てた場合は、そのファイルは入力と出力のいずれかについて FASTSRT に受け入れられますが、両方については受け入れられません。ファイルが入力について FASTSRT に不適格となるような条件が特になければ、ファイルは入力について FASTSRT に受け入れられますが、出力については受け入れられません。ファイルが入力について FASTSRT に不適格である場合でも、出力について FASTSRT に適格である可能性があります。

FASTSRT に対して適格である QSAM ファイルは、SORT ステートメントの実行中に COBOL プログラムからアクセスすることができます。たとえば、ファイルが入力で FASTSRT に使用されている場合には、それを出力プロシージャーでアクセスすることができ、ファイルが出力で FASTSRT に使用されている場合には、それを入力プロシージャーでアクセスすることができます。

VSAM の要件

- VSAM 入力ファイルは空であってはなりません。
- VSAM ファイルをパスワード保護することはできません。
- USING 句と GIVING 句の両方で同じ VSAM ファイルを指定することはできません。
- FASTSRT に対して適格である VSAM ファイルは、SORT ステートメントの処理が完了するまで、COBOL プログラムからアクセスすることはできません。たとえば、ファイルが入力で FASTSRT に適格である場合には、それを出力プロシージャーでアクセスしてはなりません。逆も同様です。(そのような処理を行うと、OPEN は失敗します。)

関連タスク
DFSORT アプリケーション・プログラミングの手引き

NOFASTSRT によるソート・エラーの検査

NOFASTSRT オプションを使用してコンパイルすると、ソート・プロセスは、SORT ステートメントの USING または GIVING 句で参照されるファイルに関して、オープン、クローズ、または入出力操作時にエラーの有無を検査しません。このため、SORT ステートメントが正常に完了したかどうかを調べることが必要になる場合があります。

必要なコードは、次の表に示されているように、USING および GIVING 句で参照されるファイルに FILE STATUS 文節または ERROR 宣言がコーディングされているかどうかによって異なります。

FILE STATUS 文節の有無	ERROR 宣言の有無	必要なコード
なし	なし	特別なコーディングは不要です。ソート・プロセス時に障害が起こると、プログラムは異常終了します。

FILE STATUS 文節の有無	ERROR 宣言の有無	必要なコード
あり	なし	SORT ステートメントの後で SORT-RETURN 特殊レジスターをテストし、ファイル状況キーをテストします。(ファイル状況コードは設定されますが、COBOL はこれを検査できないため、ファイル状況検査を完了したい場合にはお勧めしません。)
どちらでも可	あり	ERROR 宣言で、SORT-RETURN 特殊レジスターを 16 に設定して、ソート・プロセスを停止し、それが失敗したことを示します。SORT ステートメントの後で、SORT-RETURN 特殊レジスターをテストします。

関連タスク

215 ページの『ソートまたはマージの成否の判断』
254 ページの『ファイル状況キーの使用』
253 ページの『ERROR 宣言のコーディング』
216 ページの『ソートまたはマージ操作の途中停止』

ソート動作の制御

以下の手段を使用して、ソート動作のさまざまな局面を制御することができます。

- ソート前に特殊レジスターに値を挿入する
- コンパイラー・オプションを使用する
- 制御ステートメントのキーワードを使用する

ソートの後で特殊レジスターの内容を調べることによって、ソート動作を検査することもできます。

次の表は、特殊レジスターまたはコンパイラー・オプションの影響を受ける可能性のあるソート動作の局面、および等価のソート制御ステートメント・キーワードをリストしたものです。

設定またはテスト対象	特殊レジスターまたはコンパイラー・オプションを使用する場合	制御ステートメント (該当する場合は、およびキーワード) を使用する場合
予約される主記憶域の量	SORT-CORE-SIZE 特殊レジスター	OPTION (キーワード RESINV)
使用される主記憶域の量	SORT-CORE-SIZE 特殊レジスター	OPTION (キーワード MAINSIZE または MAINSIZE=MAX)
可変長レコードを持つファイルのレコードのモーダル長	SORT-MODE-SIZE 特殊レジスター	SMS=*nnnnn*
ソート制御ステートメント・データ・セットの名前 (デフォルト IGZSRTCD)	SORT-CONTROL 特殊レジスター	なし
ソート・メッセージ・ファイルの名前 (デフォルト SYSOUT)	SORT-MESSAGE 特殊レジスター	OPTION (キーワード MSGDDN)

設定またはテスト対象	特殊レジスターまたはコンパイラー・オプションを使用する場合	制御ステートメント (該当する場合は、およびキーワード) を使用する場合
ソート・レコードの数	SORT-FILE-SIZE 特殊レジスター	OPTION (キーワード FILSZ)
ソート完了コード	SORT-RETURN 特殊レジスター	なし
日付フィールドに基づいてソートまたはマージするための世紀ウィンドウ	YEARWINDOW コンパイラー・オプション	OPTION (キーワード Y2PAST)
ソート・キーまたはマージ・キーとして使用されるウィンドウ化日付フィールドの形式	(PICTURE、USAGE、および DATE FORMAT 文節から得られる)	SORT (キーワード FORMAT=Y2*x*)

ソート特殊レジスター

SORT-CONTROL は、ソート・制御ステートメント・ファイルの DD 名が入れられる 8 文字の COBOL特殊レジスターです。IGZSRTCD というデフォルトの DD 名を使用したくない場合は、ソート制御ステートメントが含まれているデータ・セットの DD 名を SORT-CONTROL に割り当ててください。

SORT-CORE-SIZE、SORT-FILE-SIZE、SORT-MESSAGE、および SORT-MODE-SIZE 特殊レジスターは、それらにデフォルト以外の値を割り当てた場合には、SORT インターフェースで使用されます。しかし、実行時には、ソート制御ステートメント・データ・セット内の制御ステートメントのパラメーターは、特殊レジスター内の対応する設定をオーバーライドし、その影響に対するメッセージが出されます。

SORT-RETURN 特殊レジスターを使用すると、ソートまたはマージが正常に実行されたかどうかを判断したり、ソートまたはマージ操作を途中で停止することができます。

プログラム内に設定したソート特殊レジスターごとに、コンパイラー警告メッセージ (W レベル) が出されます。

関連タスク

215 ページの『ソートまたはマージの成否の判断』
216 ページの『ソートまたはマージ操作の途中停止』
『制御ステートメントによる DFSORT デフォルトの変更』
223 ページの『ソート・ファイル用のスペースの割り振り』
DFSORT プログラム制御ステートメントの使用 (*DFSORT アプリケーション・プログラミングの手引き*)

関連参照

222 ページの『IGZSRTCD データ・セットのデフォルトの特性』

制御ステートメントによる DFSORT デフォルトの変更

ソート・パフォーマンスを向上させるために DFSORT のシステム・デフォルトを変更したい場合には、実行時データ・セット IGZSRTCD に含めた制御ステートメントを介して DFSORT に情報を渡してください。

IGZSRTCD データ・セットに含めることができる制御ステートメント (リストされた順序で)は、次のとおりです。

1. SMS=*nnnnn*。*nnnnn* は、最も出現頻度の高いレコード・サイズの長さ (バイト単位) です。(SD ファイルが可変長である場合にのみ使用します。)
2. OPTION (キーワード SORTIN または SORTOUT を除く)。
3. その他の DFSORT 制御ステートメント (SORT、MERGE、RECORD、END を除く)。

制御ステートメントは、2 桁目から 71 桁目の間にコーディングしてください。行をコンマで終わらせ、次の行を新しいキーワードで開始すれば、制御ステートメント・レコードを継続することができます。レコードではラベルまたはコメントは認められず、レコード自体を DFSORT コメント・ステートメントにすることはできません。

関連タスク
220 ページの『ソート動作の制御』
DFSORT プログラム制御ステートメントの使用 (*DFSORT アプリケーション・プログラミングの手引き*)

関連参照
『IGZSRTCD データ・セットのデフォルトの特性』

IGZSRTCD データ・セットのデフォルトの特性

- LRECL=80.
- BLKSIZE=400.
- DD 名は IGZSRTCD です。(SORT-CONTROL 特殊レジスターにコーディングすることによって、別の DD 名を使用することができます。)

IGZSRTCD データ・セットは任意指定です。SORT-CONTROL データ・セットの DD 名を定義したときに、メッセージ IGZ0027W を受け取った場合は、OPEN 障害が起こっており、問題を調べる必要があります。

関連タスク
220 ページの『ソート動作の制御』

ソートまたはマージ操作のためのストレージの割り振り

DFSORT のインストール時に設定される特定のパラメーターによって、DFSORT が使用するストレージの量が決まります。一般的に、DFSORT が使用できるストレージが多いほど、ソートまたはマージ操作の実行速度が速くなります。

しかし、DFSORT インストールで、領域内のすべてのフリー・スペースを COBOL 操作用に割り振ってはなりません。プログラムの実行時には、以下のために使用できるストレージがなければなりません。

- 入力または出力プロシージャーから動的に呼び出される COBOL プログラム
- 言語環境プログラムのランタイム・ライブラリー・モジュール
- 入力または出力プロシージャーで使用するために領域にロードされる可能性があるデータ管理モジュール
- これらのモジュールによって獲得されるストレージ

ソートまたはマージのある特定の実行のために、インストール時に設定された値をオーバーライドすることができます。これを行うには、ソート制御ステートメント・データ・セット内の OPTION 制御ステートメントに MAINSIZE および RESINV キーワードをコーディングするか、または SORT-CORE-SIZE 特殊レジスターを使用してください。

領域内のすべてのフリー・スペースが COBOL プログラム内のソート操作に使用されるほど、ストレージ割り振りをオーバーライドすることがないように注意してください。

関連タスク
220 ページの『ソート動作の制御』
DFSORT インストールおよびカスタマイズ

関連参照
OPTION 制御ステートメント (*DFSORT アプリケーション・プログラミングの手引き*)

ソート・ファイル用のスペースの割り振り

NOFASTSRT または入力プロシージャーを使用する場合は、ソートするファイルのサイズを DFSORT は認識しません。このため、大きいファイルをソートするときにスペース不足状態に陥ったり、小さいファイルをソートするときにリソースの過剰割り振りが発生する可能性があります。このような状況が発生した場合は、SORT-FILE-SIZE 特殊レジスターを使用して、ソートに必要なリソースの量 (たとえば、ワークスペースまたは *ハイパースペース*) を DFSORT が判別できるようにします。SORT-FILE-SIZE を、妥当な入力レコード見積数に設定します。この値は、FILSZ=E*n* の値として DFSORT に渡されます。

関連タスク
220 ページの『ソート動作の制御』
207 ページの『入力プロシージャーのコーディング』
DFSORT アプリケーション・プログラミングの手引き

DFSORT によるチェックポイント・リスタートの使用

z/OS のもとで DFSORT の実行中に取られたチェックポイントは、DFSORT によって取られたものでない限り、再始動には使用できません。SORT または MERGE ステートメントの実行中に COBOL プログラムによって取られたチェックポイントは無効です。すなわち、再始動は検出されますが、取り消されます。

ソートまたはマージ操作中にチェックポイントを取るには、以下のステップを使用してください。

1. SORTCKPT についての DD ステートメントを JCL に追加します。
2. I-O-CONTROL 段落で RERUN 文節をコーディングします。

   RERUN ON *assignment-name*
3. ソート制御ステートメント・データ・セット (デフォルトの DD 名 IGZSRTCD) 内の OPTION 制御ステートメントで CKPT (または CHKPT) キーワードをコーディングします。

関連概念
553 ページの『第 30 章 割り込みとチェックポイント・リスタート』

関連タスク
221 ページの『制御ステートメントによる DFSORT デフォルトの変更』
553 ページの『チェックポイントの設定』

CICS のもとでのソート

CICS のもとでサポートされる IBM ソート・プロダクトはありません。しかし、SORT ステートメント (および CICS のもとで稼動する独自のソート・プログラム) を使用すれば、少量のデータをソートすることができます。

SORT ステートメントには、入力プロシージャーと出力プロシージャーの両方を指定しなければなりません。入力プロシージャーでは、RELEASE ステートメントを使用して、ソートが実行される前にレコードを COBOL プログラムからソート・プログラムに転送します。出力プロシージャーでは、RETURN ステートメントを使用して、ソートが実行された後でレコードをソート・プログラムから COBOL プログラムに転送します。

関連タスク
207 ページの『入力プロシージャーのコーディング』
208 ページの『出力プロシージャーのコーディング』

関連参照
『CICS SORT アプリケーションの制約事項』

CICS SORT アプリケーションの制約事項

CICS のもとで実行され、SORT ステートメントを使用する COBOL アプリケーションには、以下の制約事項が適用されます。

- USING または GIVING 句を含む SORT ステートメントはサポートされません。
- ソート制御データ・セットはサポートされません。SORT-CONTROL 特殊レジスターのデータは無視されます。
- 入力または出力プロシージャーで以下の CICS コマンドを使用すると、予測できない結果が生じる可能性があります。
  - CICS LINK
  - CICS XCTL
  - CICS RETURN
  - CICS HANDLE
  - CICS IGNORE
  - CICS PUSH
  - CICS POP
- 上記以外の CICS コマンドは、NOHANDLE または RESP オプションを使用する場合には、使用できます。NOHANDLE または RESP を使用しないと、予測できない結果が生じる恐れがあります。

第 13 章 XML 文書の処理

XML PARSE ステートメントを使用して、COBOL プログラムから XML 文書を処理することができます。XML PARSE ステートメントは、COBOL ランタイムの一部である高速 XML パーサーとの間の COBOL 言語インターフェースです。XML 文書を処理するには、XML パーサーとの間で制御を受け渡しする必要があります。このような制御の受け渡しを実行するには、XML PARSE ステートメントを使用します。このステートメントでは、XML パーサーから制御を受け取り、パーサー・イベントを処理する処理プロシージャーを指定します。処理プロシージャーで特殊レジスターを使用して、パーサーと情報を交換します。

XML 文書を処理するには、以下の COBOL 機能を使用してください。

- XML PARSE ステートメントは、XML 構文解析を開始して、文書および処理プロシージャーを識別します。
- 処理プロシージャーは構文解析を制御します。すなわち、XML イベントおよび関連文書フラグメントを受け取って処理し、オプションで例外を処理します。
- 以下の特殊レジスターは、情報の受け渡しを行います。
  - XML-CODE は、XML 構文解析の状況を判別します。
  - XML-EVENT は、各 XML イベントの名前を受け取ります。
  - XML-TEXT は、英数字文書から XML 文書フラグメントを受け取ります。
  - XML-NTEXT は、国別文書から XML 文書フラグメントを受け取ります。

関連概念
『COBOL でのXML パーサー』

関連タスク
227 ページの『XML 文書へのアクセス』
227 ページの『XML 文書の構文解析』
228 ページの『XML イベントの処理』
241 ページの『XML 文書のエラー処理』
240 ページの『XML 文書のエンコードについての理解』

関連参照
643 ページの『付録 D. XML 参照資料』
XML 仕様 (www.w3c.org/xml)

COBOL でのXML パーサー

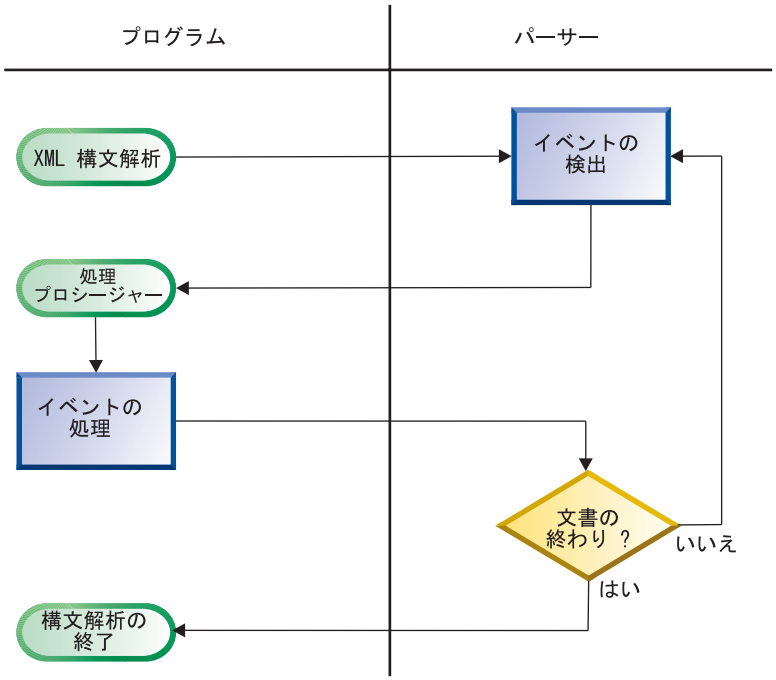
Enterprise COBOL ではイベント・ベースのインターフェースが提供されるため、これを使用して XML 文書を構文解析し、さらに COBOL データ構造に変換することができます。XML パーサーが文書内の (XML イベントに関連付けられた) フラグメントを検出し、作成した処理プロシージャーによってこれらのフラグメントに対する操作が実行されます。それぞれの XML イベントを処理するために独自のプロシージャーをコーディングします。この操作の間中、制御がパーサーとプロシージャーの間を行き来します。

© Copyright IBM Corp. 1991, 2001

パーサーとの受け渡しを開始するには、XML PARSE ステートメントを使用します。このステートメントに処理プロシージャーを指定します。この XML PARSE ステートメントを実行すると、構文解析が開始されてパーサーでの処理プロシージャーが確立されます。プロシージャーが実行されるたびに XML パーサーは XML 文書の解析を継続して次のイベントを報告し、検出したフラグメント (新規エレメントの開始など) をプロシージャーに戻します。また、XML PARSE ステートメントに、構文解析の終了時に制御を渡したい 2 つの命令ステートメントを指定することもできます。1 つは正常終了の場合、もう 1 つは例外条件が存在する場合のためのステートメントです。

パーサーとプログラム間の基本的な制御の受け渡しを示した高度な概略図を以下に示します。

XML 構文解析フローの概要



通常、構文解析は XML 文書全体が構文解析されるまで継続されます。

XML パーサーが XML 文書を構文解析する際は、XML 文書のさまざまな側面が XML 仕様の定義に従った整形式になっているかどうか検査します。文書が整形式であるのは、XML 構文規則に準拠し、その他のいくつかの規則 (終了タグの適切な使用、属性名が固有であることなど) に従っている場合です。

関連タスク

227 ページの『XML 文書へのアクセス』
227 ページの『XML 文書の構文解析』
234 ページの『XML を処理するためのプロシージャーの作成』
241 ページの『XML 文書のエラー処理』
240 ページの『XML 文書のエンコードについての理解』

関連参照
XML 仕様 (http://www.w3c.org/xml)
652 ページの『XML 適合性』

XML 文書へのアクセス

XML PARSE ステートメントを使用して XML 文書を構文解析する前に、その文書をプログラムで使用できるようにしておく必要があります。文書を取得する方法として最も適当なのは、MQSeries メッセージ、CICS 一時キューまたは 連絡域 (commarea)、あるいは IMS メッセージ処理キューから取得する方法です。

構文解析する XML 文書がファイル内に保管されている場合は、以下に示す通常の COBOL 機能を使用して文書をプログラムのデータ項目に入れてください。

- FILE-CONTROL 記入項目でプログラムに対してファイルを定義します。
- OPEN ステートメントでファイルをオープンします。
- READ ステートメントで、ファイルからすべてのレコードを読み取り、プログラムの WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION に定義された英数字データ項目または国別データ項目に入れます。
- (オプション) STRING ステートメントで、個別レコードすべてを 1 つの連続ストリームに結合し、無関係なブランクを除去し、可変長レコードを処理します。

関連タスク
408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
425 ページの『第 22 章 IMS のもとでの COBOL プログラムの実行』

XML 文書の構文解析

XML 文書を構文解析するには、次の例に示すように XML PARSE ステートメントを使用します。

```
XML PARSE XMLDOCUMENT
    PROCESSING PROCEDURE XMLEVENT-HANDLER
  ON EXCEPTION
     DISPLAY 'XML document error ' XML-ERROR
     STOP RUN
  NOT ON EXCEPTION
     DISPLAY 'XML document was successfully parsed.'
END-XML
```

XML PARSE ステートメントで、まず XML 文書文字ストリームを含むデータ項目 (この例では XMLDOCUMENT) を識別します。DATA DIVISION で、英数字データ項目または国別データ項目として ID を宣言することができます。英数字データ項目の場合は、サポートされているいずれかの単一バイト文字セット (EBCDIC または ASCII) でその内容をエンコードする必要があります。国別データ項目の場合は、その内容を Unicode UTF-16 CCSID 1200 でエンコードする必要があります。エンコード宣言が含まれていない英数字の XML 文書は、CODEPAGE コンパイラー・オプションで指定したコード・ページで構文解析されます。

次に、文書から検出した XML イベントを処理するプロシージャーの名前 (この例では XMLEVENT-HANDLER) を指定します。

さらに、構文解析の終了時に制御を受け取る以下の命令ステートメントのいずれかまたは両方を指定できます。

- ON EXCEPTION は、未処理の例外が発生したときに制御を受け取ります。
- NOT ON EXCEPTION は、それ以外の場合に制御を受け取ります。

XML PARSE ステートメントを終了するには、END-XML を使用します。この範囲終了符号を使用して、XML PARSE ステートメントを条件ステートメント内または別の XML PARSE ステートメント内にネストします。

XML パーサーと処理プロシージャー間での制御の受け渡しは、以下のいずれかの状況が発生するまで継続されます。

- XML 文書全体の構文解析が完了し、それが END-OF-DOCUMENT イベントによって示された場合。
- パーサーが文書内にエラーを検出し、EXCEPTION イベントをシグナル通知した場合。処理プロシージャーは、パーサーに戻るまでは特殊レジスター XML-CODE をゼロにリセットしません。
- パーサーに戻る前に、特殊レジスター XML-CODE を -1 に設定して、構文解析プロセスを故意に終了した場合。

関連タスク
240 ページの『XML 文書のエンコードについての理解』

関連参照
XML PARSE ステートメント (*Enterprise COBOL 言語解説書*)
制御フロー (*Enterprise COBOL 言語解説書*)

XML イベントの処理

XML-EVENT 特殊レジスターを使用すると、パーサーから処理プロシージャーに渡されるイベントを決定することができます。XML-EVENT には、'START-OF-ELEMENT' などのイベント名が入ります。パーサーは、XML PARSE ステートメントの XML ID のタイプに基づいて、特殊レジスター XML-TEXT または XML-NTEXT に入っているイベントの内容を渡します。

これらのイベントは、基本的に、このサンプル XML 文書に出現する順番で示されています。「サンプル XML テキスト」の部分に示されたテキストは、このサンプルからとったものです。実際のテキストは、区切り文字 <<>> の間に示されています。

```
<?xml version="1.0" encoding="ibm-1140" standalone="yes" ?>
<!--This document is just an example-->
<sandwich>
  <bread type="baker&apos;s best" />
  <?spread please use real mayonnaise ?>
  <meat>Ham &amp; turkey</meat>
  <filling>Cheese, lettuce, tomato, etc.</filling>
  <![CDATA[We should add a <relish> element in future!]]>
</sandwich>junk
```

START-OF-DOCUMENT

説明 構文解析する文書の先頭に一度だけ出現します。XML テキストは文書全体であり、これには、LF (改行) や NL (改行) など行制御文字がすべて含まれています。

サンプル XML テキスト
このサンプル・テキストの長さは 336 文字です。

VERSION-INFORMATION

説明 バージョン情報についてのオプションの XML 宣言内に出現します。XML テキストには、バージョン値が入ります。*XML 宣言* は、使用する XML のバージョンおよび文書のエンコード方式を指定する XML テキストです。

サンプル XML テキスト
<<1.0>>

ENCODING-DECLARATION

説明 オプションのエンコード宣言についての XML 宣言内に出現します。XML テキストには、エンコード値が入ります。

サンプル XML テキスト
<<ibm-1140>>

STANDALONE-DECLARATION

説明 オプションのスタンドアロン宣言についての XML 宣言内に出現します。XML テキストには、スタンドアロン値が入ります。

サンプル XML テキスト
<<yes>>

DOCUMENT-TYPE-DECLARATION

説明 パーサーが文書タイプ宣言 (DTD) を検出したときに出現します。文書タイプ宣言は、文字シーケンス <!DOCTYPE で始まり > 文字で終わりますが、この間には、内容を記述するかなり複雑な文法規則が入ります (詳細については、*XML 仕様* を参照してください)。このイベントの場合、XML テキストには宣言全体 (開始と終了の文字シーケンスを含む) が入ります。これは、XML テキストに区切り文字が含まれる唯一のイベントです。

サンプル XML テキスト
サンプルには、文書タイプ宣言はありません。

COMMENT

説明 XML 文書についてのコメントがある場合に出現します。XML テキストには、コメントの開始と終了を示すコメント区切り <!-- と --> に囲まれてデータが入ります。

サンプル XML テキスト

```
<<This document is just an example>>
```

START-OF-ELEMENT

説明 要素開始タグまたは空要素タグごとに一度出現します。XML テキストは要素名に設定されます。

サンプル XML テキスト

START-OF-ELEMENT イベントとして、次の順序で出現します。

1. <<sandwich>>
2. <<bread>>
3. <<meat>>
4. <<filling>>

ATTRIBUTE-NAME

説明 有効な名前を認識後、要素開始タグまたは空要素タグの属性ごとに出現します。XML テキストには属性名が入ります。

サンプル XML テキスト

```
<<type>>
```

ATTRIBUTE-CHARACTERS

説明 属性値のフラグメントごとに出現します。XML テキストにはフラグメントが入ります。通常、属性値は単一ストリングのみで構成され、複数の行に分割されている場合も同様です。ただし、場合によっては、属性値が複数のイベントで構成されることがあります。

サンプル XML テキスト

ATTRIBUTE-CHARACTERS イベントとして、次の順序で出現します。

1. <<baker>>
2. <<s best>>

サンプルの「type」属性の値は、ストリング 'baker'、単一文字 '''、およびストリング 's best' という 3 つのフラグメントで構成されます。単一文字のフラグメント ''' は、ATTRIBUTE-CHARACTER イベントとして個別に渡されます。

ATTRIBUTE-CHARACTER

説明 定義済みエンティティー参照 '&'、'''、'>'、'<'、および '"' 用の属性値に出現します。定義済みエンティティーの詳細については、*XML 仕様* を参照してください。

サンプル XML テキスト

```
<<'>>
```

ATTRIBUTE-NATIONAL-CHARACTER

説明 '&#dd..;' または '&#hh..;' (d は 10 進数字を表し、h は 16 進数字を表す) の形式で表された数字参照 (Unicode コード・ポイント、つまり、「スカラー値」) 用の属性値に出現します。国別文字のスカラー値が 65,535 (NX'FFFF') より大きい場合、XML-NTEXT には 2 つのエンコード・ユニット (サロゲート・ペア) が入り、長さは 4 バイトになります。このエンコード・ユニットのペアは、2 つで 1 つの文字を表現します。このペアを分離すると無効な文字が作成されますので、分離しないでください。

サンプル XML テキスト

サンプルには、数字参照はありません。

END-OF-ELEMENT

説明 パーサーが、要素終了タグまたは空要素タグの終了不等号括弧を認識するたびに出現します。

サンプル XML テキスト

```
<<bread>>
```

PROCESSING-INSTRUCTION-TARGET

説明 処理命令 (PI) 開始の文字シーケンス '<?' の後に名前をパーサーが認識したときに出現します。PI を使用すると、アプリケーション用の特別な命令を XML 文書に定義することができます。

サンプル XML テキスト

```
<<spread>>
```

PROCESSING-INSTRUCTION-DATA

説明 PI ターゲットの直後に始まり、PI 終了の文字シーケンス '?>' の直前で終わるデータについて出現します。XML テキストには、PI データが入ります。これには末尾の空白文字が含まれますが、先頭の空白文字は含まれません。

サンプル XML テキスト

```
<<please use real mayonnaise >>
```

CONTENT-CHARACTERS

説明 このイベントは、XML 文書の「中身」を表します。つまり、開始タグと終了タグの間の文字データです。XML テキストにはこのデータが含まれており、通常、これは単一のストリングで構成されます。複数の行に分割されている場合も同様です。要素の内容に参照や別の要素が含まれている場合、内容全体が複数のイベントで構成されることがあります。パーサーでは、CONTENT-CHARACTERS イベントを使用して、CDATA セクションのテキストをプログラムに渡す場合もあります。

サンプル XML テキスト

CONTENT-CHARACTERS イベントとして、次の順序で出現します。

1. <<Ham >>
2. << turkey>>
3. <<Cheese, lettuce, tomato, etc.>>
4. <<We should add a <relish> element in future!>>

このサンプルでは、「中身」となる要素の内容は、ストリング 'Ham '、文字 '&'、およびストリング ' turkey' で構成されます。単一文字のフラグメント '&' は、CONTENT-CHARACTER イベントとして個別に渡されます。この 2 つのストリングのフラグメントには、先頭と末尾の空白があります。

CONTENT-CHARACTER

説明 定義済みエンティティー参照 '&'、'''、'>'、'<'、および '"' に対応する要素値に出現します。定義済みエンティティーの詳細については、*XML 仕様* を参照してください。

サンプル XML テキスト

<<&>>

CONTENT-NATIONAL-CHARACTER

説明 '&#dd..;' または '&#hh..;' (d は 10 進数字を表し、h は 16 進数字を表す) の形式で表された数字参照 (Unicode コード・ポイント、つまり、「スカラー値」) に対応する要素値に出現します。国別文字のスカラー値が 65,535 (NX'FFFF') より大きい場合、XML-NTEXT には 2 つのエンコード・ユニット (サロゲート・ペア) が入り、長さは 4 バイトになります。このエンコード・ユニットのペアは、2 つで 1 つの文字を表現します。このペアを分離すると、無効な文字が作成されるので、ペアを分離しないでください。

サンプル XML テキスト

サンプルには、数字参照はありません。

END-OF-ELEMENT

説明 パーサーが、要素終了タグまたは空要素タグの終了不等号括弧を認識するたびに出現します。XML テキストには要素名が入ります。

サンプル XML テキスト

END-OF-ELEMENT イベントとして、次の順序で出現します。

1. <<bread>>
2. <<meat>>
3. <<filling>>
4. <<sandwich>>

START-OF-CDATA-SECTION

説明 CDATA セクションを開始するときに出現します。CDATA セクションは、ストリング '<![CDATA['で開始され、ストリング ']]>' で終了します。CDATA セクションは、XML マークアップとして認識される文字を含むテキストのブロックを「エスケープ」するために使用します。XML テキストには、必ず開始の文字シーケンス '<![CDATA[' が含まれています。パーサーでは、これらの区切り文字で囲まれた CDATA セクションの内容を単一の CONTENT-CHARACTERS イベントとして渡します。

サンプル XML テキスト

```
<<<![CDATA[>>
```

END-OF-CDATA-SECTION

説明 パーサーが CDATA セクションの終了を認識したときに出現します。

サンプル XML テキスト

```
<<]]>>>
```

UNKNOWN-REFERENCE-IN-ATTRIBUTE

説明 5 つの定義済みエンティティー参照 (前述の ATTRIBUTE-CHARACTER に記載) 以外のエンティティー参照用の属性値に出現します。

サンプル XML テキスト

サンプルには、不明なエンティティー参照はありません。

UNKNOWN-REFERENCE-IN-CONTENT

説明 定義済みエンティティー参照 (前述の CONTENT-CHARACTER に記載) 以外のエンティティー参照用の要素内容に出現します。

サンプル XML テキスト

サンプルには、未定義のエンティティー参照はありません。

END-OF-DOCUMENT

説明 文書の構文解析が完了したときに出現します。

サンプル XML テキスト

END-OF-DOCUMENT イベント用の XML テキストは空です。

EXCEPTION

説明 XML 文書の処理中にエラーが検出されたときに出現します。構文解析が開始される前にシグナル通知されるエンコード矛盾例外については、XML-TEXT は長さ 0 (ゼロ) になるか、文書のエンコード宣言値のみが入ります。

サンプル XML テキスト

例外 (<sandwich> 要素の終了タグに続く余分な「ジャンク」) が検出された地点までの構文解析済みの部分。

関連参照

XML-EVENT (*Enterprise COBOL 言語解説書*)
4.6 定義済みエンティティー (*XML 仕様* — URL: www.w3.org/TR/REC-xml#sec-predefined-ent)
2.8 Prolog および文書タイプ宣言 (*XML 仕様* — URL: www.w3.org/TR/REC-xml#sec-prolog-dtd)

XML を処理するためのプロシージャーの作成

処理プロシージャーには、XML イベントを処理するためのステートメントをコーディングします。

パーサーは、イベントを検出すると、次の表に示す特殊レジスター内の処理プロシージャーに情報を渡します。これらのレジスターは、データ構造の取り込みと、処理の制御に使用します。

XML 特殊レジスターの定義		
レジスター	**内容**	**暗黙的な定義および使用法**
XML-EVENT[1]	XML イベントの名前	PICTURE X(30) USAGE DISPLAY VALUE SPACE
XML-CODE	各 XML イベント用の例外コードまたはゼロ	PICTURE S9(9) USAGE BINARY VALUE ZERO
XML-TEXT[1]	XML PARSE ID として英数字データ項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)	可変長英数字データ項目、16MB のサイズ制限
XML-NTEXT[1]	XML PARSE ID として国別データ項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)	可変長国別データ項目、16MB のサイズ制限
1. この特殊レジスターを受け取りデータ項目として使用することはできません。		

これらの特殊レジスターがネストされたプログラムで使用された場合は、最外部のプログラムで GLOBAL として暗黙的に定義されます。

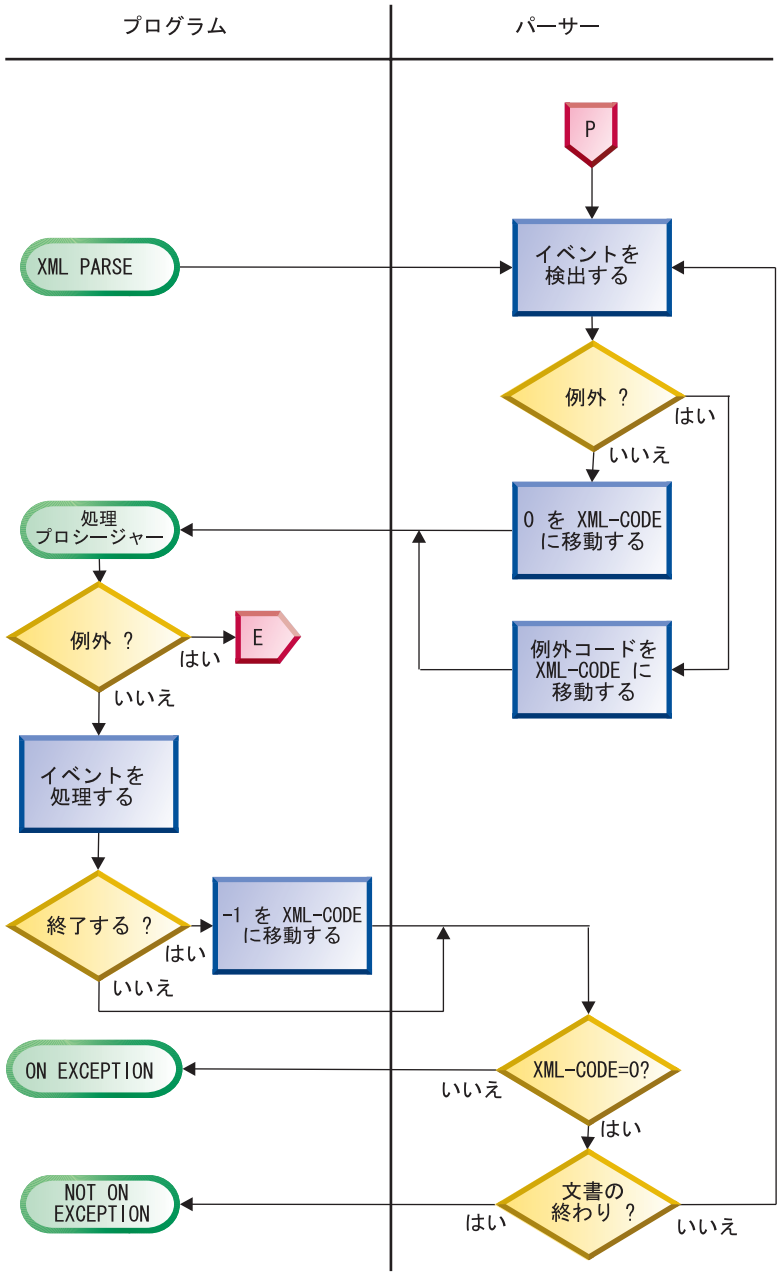
XML-CODE の内容を理解する

パーサーから XML PARSE ステートメントに制御が戻ったとき、XML-CODE には、パーサーまたは処理プロシージャーによって設定された最新の値が入っています。

すべてのイベント (EXCEPTION イベントを除く) において、XML-CODE 特殊レジスターの値はゼロです。EXCEPTION 以外のイベントで XML パーサーに制御を戻す前に XML-CODE 特殊レジスターを -1 に設定すると、戻された XML-CODE 値 -1 で指定されるユーザー開始例外で処理が停止します。イベントから戻る前に XML-CODE をその他の非ゼロ値に変更した場合、結果は保証されません。

EXCEPTION イベントの場合、特殊レジスター XML-CODE は例外コードに設定されます。次の図では、パーサーと処理プロシージャーの間の制御フロー、およびこの 2 つの間で情報を渡すために XML-CODE がどのように使用されるかを示します。この章に示す複数の図表は、 P などの別ページへの接続記号によって連結されています。具体的には、次の図の E は図「XML 例外の制御フロー」(243 ページ) に連結されており、 P は「XML CCSID 例外制御フロー」(246 ページ) から連結しています。

XML-CODE の使用法を示す、XML パーサーとプログラムの間の制御フロー



XML-TEXT および XML-NTEXT を使用する

特殊レジスター XML-TEXT と XML-NTEXT は、相互に排他的です。指定する XML ID のタイプによって、どの特殊レジスターが設定されるかが決まります、ただし、このことは、ATTRIBUTE-NATIONAL-CHARACTER イベントと CONTENT-NATIONAL-CHARACTER イベントには適用されません。これらのイベントの場合、XML PARSE ID に指定するデータ項目に関係なく、XML-NTEXT が設定されます。

パーサーによって XML-TEXT が設定されると、XML-NTEXT は未定義になります (長さ 0)。パーサーによって XML-NTEXT が設定されると、XML-TEXT は未定義になります (長さ 0)。

XML-NTEXT に含まれる国別文字の数を決定するには、LENGTH 関数を使用します。LENGTH OF XML-NTEXT には、文字数ではなく、XML-NTEXT で使用されるバイト数が入ります。

XML テキストを COBOL データ項目に変換する

XML データは固定長ではなく、固定形式でもないので、XML データを COBOL データ項目に移動するときは、特別な技法を使用する必要があります。

英数字項目の場合、XML データを COBOL 項目の左端 (デフォルト) に配置するか、右端に配置するかを決める必要があります。右端に配置する場合は、COBOL 項目の宣言に JUSTIFIED RIGHT 文節を指定します。

数字の XML 値、特に、'$1,234.00' や '$1234' といった「修飾」された通貨値には特別な配慮が必要です。この 2 つの値は XML では同じものを意味しますが、COBOL 送信フィールドとしてはまったく別の宣言になります。このような場合は、以下のいずれかの技法を使用してください。

- 適度に規則性のあるフォーマットの場合は、MOVE を使用して、適切な数字編集項目として再定義された英数字項目に移動します。次に、数字編集項目から移動して編集解除することにより、数字 (操作) 項目への最終的な移動を行います (規則性のあるフォーマットとは、たとえば、小数点以下の桁数が同じで、999 より大きい値には常にコンマ区切り文字が付くフォーマットです)。
- 単純性と高度な柔軟性を実現するために、英数字の XML データについては、以下の技法を使用してください。
  - 単純な数字を表現している XML データから単純な数値を取り出してデコードするには、関数 NUMVAL を使用します。
  - 通貨数量を表現している XML データから数値を取り出してデコードするには、関数 NUMVAL-C を使用します。

  ただし、これらの関数を使用するとパフォーマンスが下がります。

処理プロシージャーに関する制約

処理プロシージャーでは、XML PARSE ステートメントを直接実行することはできません。ただし、INVOKE ステートメントまたは CALL ステートメントを使用して、処理プロシージャーからメソッドまたは最外部のプログラムに制御が渡されると、ターゲットとなるメソッドまたはプログラムでは、同一または別の XML PARSE ステートメントを実行することができます。複数のスレッドで実行されているプログラムから、同一または別の XML ステートメントを同時に実行することもできます。

処理プロシージャーを終了する

コンパイラーは、処理プロシージャーの最後のステートメントの後に、戻り機構を挿入します。処理プロシージャーに STOP RUN ステートメントをコーディングすると、実行単位を終了することができます。ただし、GOBACK ステートメントや EXIT PROGRAM ステートメントの場合、パーサーに制御を戻しません。これらのステートメントを使用すると、処理プロシージャーで重大なエラーが発生する原因となります。

『例: XML の構文解析』

関連タスク

116 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

643 ページの『続行を許可する XML 例外』
648 ページの『続行を許可しない XML 例外』
XML-CODE (*Enterprise COBOL 言語解説書*)
XML-EVENT (*Enterprise COBOL 言語解説書*)
XML-NTEXT (*Enterprise COBOL 言語解説書*)
XML-TEXT (*Enterprise COBOL 言語解説書*)

例: XML の構文解析

以下の例では、XML PARSE ステートメントおよび処理プロシージャーの基本的な構成を示します。この XML 文書は、構文解析のフローを確認できるように、ソースに示されています。プログラムの出力を以下に示します。プログラムの出力と文書を比較して、パーサーと処理プロシージャーとの間の相互作用を確認し、イベントと文書フラグメントを突き合わせてください。

```
 Process flag(i,i)
 Identification division.
   Program-id. xmlsampl.

 Data division.
  Working-storage section.
*****************************************************************
* XML document, encoded as initial values of data-items.        *
*****************************************************************
   1 xml-document.
    2 pic x(39) value '<?xml version="1.0" encoding="ibm-1140"'.
    2 pic x(19) value ' standalone="yes"?>'.
    2 pic x(39) value '<!--This document is just an example-->'.
    2 pic x(10) value '<sandwich>'.
    2 pic x(35) value '  <bread type="baker&apos;s best"/>'.
    2 pic x(41) value '  <?spread please use real mayonnaise  ?>'.
    2 pic x(31) value '  <meat>Ham &amp; turkey</meat>'.
    2 pic x(40) value '  <filling>Cheese, lettuce, tomato, etc.'.
    2 pic x(10) value '</filling>'.
    2 pic x(35) value '  <![CDATA[We should add a <relish>'.
    2 pic x(22) value ' element in future!]]>'.
    2 pic x(31) value '  <listprice>$4.99 </listprice>'.
    2 pic x(27) value '  <discount>0.10</discount>'.
    2 pic x(11) value '</sandwich>'.
   1 xml-document-length computational pic 999.

*****************************************************************
* Sample data definitions for processing numeric XML content.   *
*****************************************************************
   1 current-element pic x(30).
```

```
   1 xfr-ed pic x(9) justified.
   1 xfr-ed-1 redefines xfr-ed pic 999999.99.
   1 list-price computational pic 9v99 value 0.
   1 discount computational pic 9v99 value 0.
   1 display-price pic $$9.99.

 Procedure division.
  mainline section.

     XML PARSE xml-document PROCESSING PROCEDURE xml-handler
       ON EXCEPTION
         display 'XML document error ' XML-CODE
       NOT ON EXCEPTION
         display 'XML document successfully parsed'
     END-XML

*****************************************************************
*    Process the transformed content and calculate promo price.  *
*****************************************************************
     display ' '
     display '-----++++++++++ Using information from XML '
         '++++++++++-----'
     display ' '
     move list-price to display-price
     display '  Sandwich list price: ' display-price
     compute display-price = list-price * (1 - discount)
     display '  Promotional price:   ' display-price
     display '  Get one today!'

     goback.

  xml-handler section.
     evaluate XML-EVENT
* ==> Order XML events most frequent first
       when 'START-OF-ELEMENT'
         display 'Start element tag: <' XML-TEXT '>'
         move XML-TEXT to current-element
       when 'CONTENT-CHARACTERS'
         display 'Content characters: <' XML-TEXT '>'
* ==> Transform XML content to operational COBOL data item...
         evaluate current-element
           when 'listprice'
* ==> Using function NUMVAL-C...
             compute list-price = function numval-c(XML-TEXT)
           when 'discount'
* ==> Using de-editing of a numeric edited item...
             move XML-TEXT to xfr-ed
             move xfr-ed-1 to discount
         end-evaluate
       when 'END-OF-ELEMENT'
         display 'End element tag: <' XML-TEXT '>'
         move spaces to current-element
       when 'START-OF-DOCUMENT'
         compute xml-document-length = function length(XML-TEXT)
         display 'Start of document: length=' xml-document-length
             ' characters.'
       when 'END-OF-DOCUMENT'
         display 'End of document.'
       when 'VERSION-INFORMATION'
         display 'Version: <' XML-TEXT '>'
       when 'ENCODING-DECLARATION'
         display 'Encoding: <' XML-TEXT '>'
       when 'STANDALONE-DECLARATION'
         display 'Standalone: <' XML-TEXT '>'
       when 'ATTRIBUTE-NAME'
         display 'Attribute name: <' XML-TEXT '>'
       when 'ATTRIBUTE-CHARACTERS'
```

```
          display 'Attribute value characters: <' XML-TEXT '>'
        when 'ATTRIBUTE-CHARACTER'
          display 'Attribute value character: <' XML-TEXT '>'
        when 'START-OF-CDATA-SECTION'
          display 'Start of CData: <' XML-TEXT '>'
        when 'END-OF-CDATA-SECTION'
          display 'End of CData: <' XML-TEXT '>'
        when 'CONTENT-CHARACTER'
          display 'Content character: <' XML-TEXT '>'
        when 'PROCESSING-INSTRUCTION-TARGET'
          display 'PI target: <' XML-TEXT '>'
        when 'PROCESSING-INSTRUCTION-DATA'
          display 'PI data: <' XML-TEXT '>'
        when 'COMMENT'
          display 'Comment: <' XML-TEXT '>'
        when 'EXCEPTION'
          compute xml-document-length = function length (XML-TEXT)
          display 'Exception ' XML-CODE ' at offset '
              xml-document-length '.'
        when other
          display 'Unexpected XML event: ' XML-EVENT '.'
      end-evaluate
      .
 End program xmlsampl.
```

構文解析例からの出力: 以下の出力では、構文解析の各イベントが、どの文書フラグメントから発生しているかを確認することができます。

```
Start of document: length=390 characters.
Version: <1.0>
Encoding: <ibm-1140>
Standalone: <yes>
Comment: <This document is just an example>
Start element tag: <sandwich>
Content characters: <  >
Start element tag: <bread>
Attribute name: <type>
Attribute value characters: <baker>
Attribute value character: <'>
Attribute value characters: <s best>
End element tag: <bread>
Content characters: <  >
PI target: <spread>
PI data: <please use real mayonnaise  >
Content characters: <  >
Start element tag: <meat>
Content characters: <Ham >
Content character: <&>
Content characters: < turkey>
End element tag: <meat>
Content characters: <  >
Start element tag: <filling>
Content characters: <Cheese, lettuce, tomato, etc.>
End element tag: <filling>
Content characters: <  >
Start of CData: <<![CDATA[>
Content characters: <We should add a <relish> element in future!>
End of CData: <]]>>
Content characters: <  >
Start element tag: <listprice>
Content characters: <$4.99 >
End element tag: <listprice>
Content characters: <  >
Start element tag: <discount>
Content characters: <0.10>
End element tag: <discount>
End element tag: <sandwich>
```

```
End of document.
XML document successfully parsed

-----++++++++++ Using information from XML **********-----

  Sandwich list price:  $4.99
  Promotional price:    $4.49
  Get one today!
```

XML 文書のエンコードについての理解

XML PARSE ステートメントは、以下のいずれかのデータ項目を含む XML 文書のみをサポートします。

- Unicode UTF-16 を使用してエンコードされた国別データ項目
- サポートされている単一バイト EBCDIC または ASCII 文字集合のいずれかを使用してエンコードされた英数字データ項目

XML 文書にエンコード宣言が含まれている場合は、その宣言が、XML PARSE ステートメントで指定されるエンコード情報および文書の基本エンコードと矛盾しないようにする必要があります。 パーサーは、最大 3 つの情報ソースを以下の順序で使用することにより、文書のエンコード方式を判別します。

1. 文書の最初の文字群
2. XML PARSE ステートメントで指定されるエンコード情報
3. ステップ 2 が成功した場合、文書のエンコード宣言

したがって、サポートされるコード・ページのいずれかが指定されたエンコード宣言を含む XML 宣言で始まっている XML 文書の場合、エンコード宣言がその文書の基本エンコード方式や XML PARSE ステートメントにより指定されたエンコード情報と矛盾しない限り、パーサーはそのエンコード宣言に従います。

XML 宣言のない XML 文書の場合、あるいは XML 宣言でエンコード宣言が省略されている場合、その文書の基本エンコードと矛盾しない限り、パーサーは、XML PARSE ステートメントにより指定されたエンコード情報を使用して、文書を処理します。

これらのソース間に矛盾があることを検出すると、パーサーは XML 例外イベントをシグナル通知します。

コード・ページの指定

XML 文書の多くは XML 宣言で始まりますが、この XML 宣言で文書のエンコード情報を指定することができます。エンコード宣言を含む XML 宣言の例を以下に示します。

```
<?xml version="1.0" encoding="ibm-1140" ?>
```

エンコード宣言は、以下のいずれかの方法で指定することができます。

- 先行ゼロ付きまたは先行ゼロなしの CCSID 番号に、次のいずれかの接頭部を付けて指定します (大文字小文字混合形式)。
  - IBM-
  - IBM_

– CP
– CP-
– CP_
– CCSID-
– CCSID_

- 次のいずれかのサポートされる別名を使用します (大文字小文字混合形式)。

コード・ページ	サポートされる別名
037	EBCDIC-CP-US、EBCDIC-CP-CA、EBCDIC-CP-WT、EBCDIC-CP-NL
500	EBCDIC-CP-BE、EBCDIC-CP-CH
813	iso-8859-7、iso_8859-7
819	iso-8859-1、iso_8859-1
920	iso-8859-9、iso_8859-9
1200	UTF-16

その他のコード・ページを使用した文書の構文解析

明示的にサポートされている単一バイト・コード・ページ以外のコード・ページでエンコードされた XML 文書を構文解析するには、NATIONAL-OF 関数を使用して、該当するコード・ページを国別データ項目で Unicode UTF-16 に変換します。特殊レジスター XML-NTEXT で処理プロシージャーに渡されるそれぞれの文書テキスト部分は、必要に応じて DISPLAY-OF 関数を使用して元のコード・ページに戻すことができます。

関連タスク

118 ページの『国別データの変換』

関連参照

XML 文書のコード化文字セット (*Enterprise COBOL 言語解説書*)

XML 文書のエラー処理

次の機能を使用すると、XML 文書のエラー処理が容易になります。

- ユーザー定義の処理プロシージャー
- XML PARSE ステートメントの ON EXCEPTION 句
- 特殊レジスター XML-CODE

XML パーサーは、XML 文書でエラーを検出すると、XML 例外イベントを生成します。パーサーでは、次の情報に基づいて処理プロシージャーに制御を渡すことにより、この例外イベントを戻します。

- 特殊レジスター XML-EVENT に 'EXCEPTION' が含まれている。
- 特殊レジスター XML-CODE に数値例外コードが含まれている。
- XML-TEXT に、例外が検出された地点までの文書テキストが含まれている。

エラー・コードの値が、

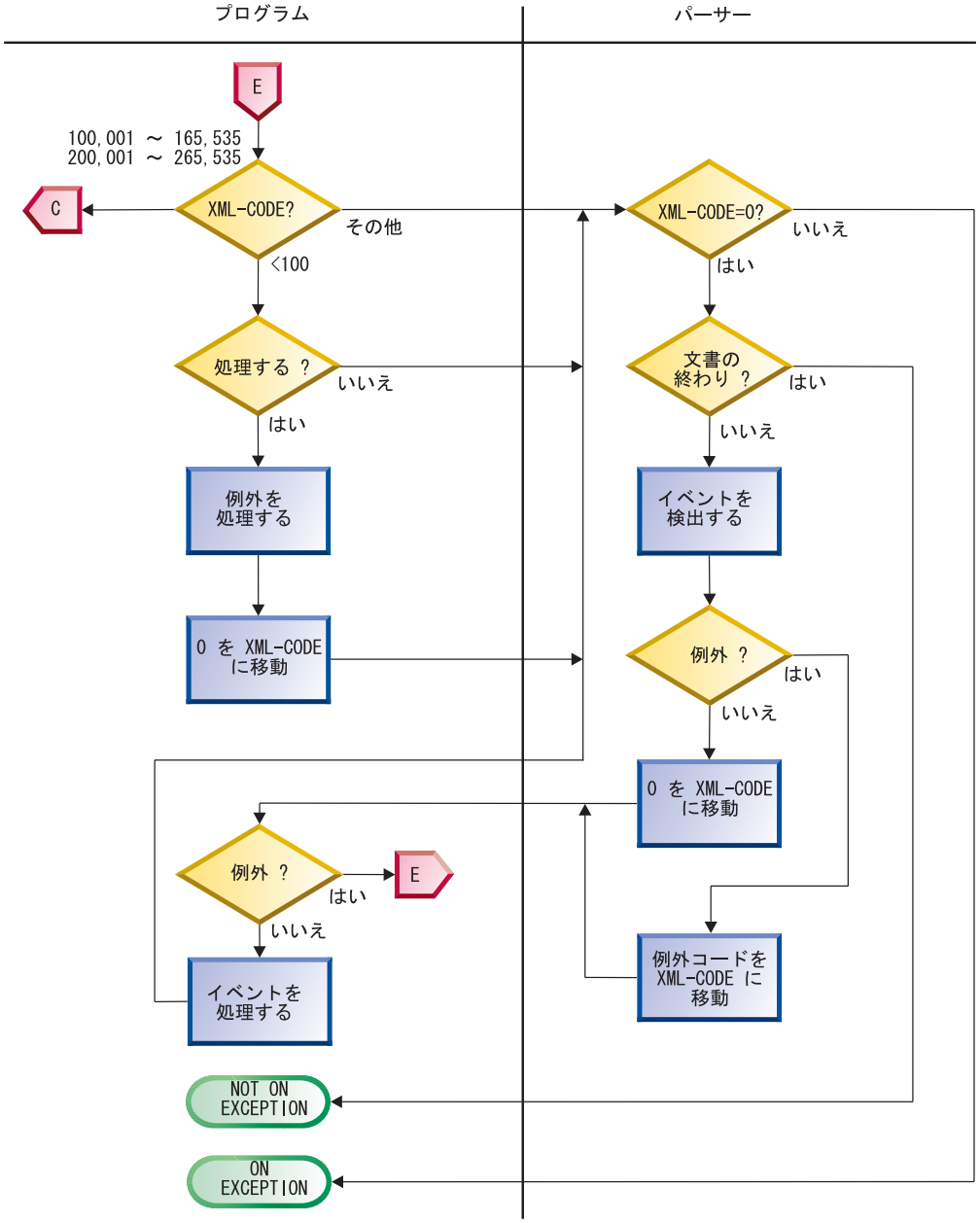
- 1 ～ 99
- 100,001 ～ 165,535
- 200,001 ～ 265,535

のいずれかの範囲にある場合は、処理プロシージャーで例外を処理し、構文解析を続けることができます。エラー・コードがその他の非ゼロ値である場合は、構文解析を続けることができません。エンコード方式の矛盾の例外 (50 ～ 99 および 300 ～ 499) は、文書の構文解析が開始される前にシグナル通知されます。このような例外の場合、XML-TEXT は長さゼロになるか、文書のエンコード宣言値のみが入ります。

1 ～ 49 の範囲の例外は XML 仕様に基づく致命的エラーであり、ユーザーが例外を処理しても、パーサーでは通常の構文解析を続けることはできません。ただし、パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、その他のエラーのスキャンを続けます。このような例外の場合、パーサーでは、END-OF-DOCUMENT イベント以外については、追加の標準イベントをシグナル通知しません。

次の図では、パーサーと処理プロシージャーの間の制御フローをわかりやすく示しています。この図は、特定の例外の処理方法、および XML-CODE を使用して例外を識別する方法を示してます。この章に示す複数の図表は、 P などの別ページへの接続記号によって連結されています。具体的には、次の図の C は図「XML CCSID 例外の制御フロー」(246 ページ) に連結されています。この図では、 E と E は、別ページへの接続記号およびページ内接続記号の両方として使用されています。

XML 例外の制御フロー



未処理の例外

例外を処理する必要がない場合は、XML-CODE の値を変更しないでパーサーに制御を戻します。この場合、パーサーは、ON EXCEPTION ステートメントで指定されたステートメントに制御権を移動します。ON EXCEPTION 句がコーディングされていない場合、制御は XML PARSE ステートメントの終わりに移動します。

例外の処理

処理プロシージャーで例外イベントを処理するには、次の手順に従います。

1. XML-CODE の内容を処置の手引きとして使用します。

2. XML-CODE を、例外が処理されたことを示すゼロに設定します。
3. パーサーに制御を戻します。これにより、例外条件がなくなります。

構文解析の終了時点までに未処理の例外がなかった場合は、NOT ON EXCEPTION 句に指定したステートメントに制御が渡されます (構文解析の正常終了)。NOT ON EXCEPTION 句をコーディングしなかった場合、制御は XML PARSE ステートメントの終わりに渡されます。特殊レジスター XML-CODE はゼロに設定されます。

この方法で例外を処理できるのは、XML-CODE で渡される例外コードが次のいずれかの範囲にある場合だけです。

- 1 ～ 99
- 100,001 ～ 165,535
- 200,001 ～ 265,535

これらの範囲にない場合、パーサーは追加のイベントをシグナル通知せず、ON EXCEPTION 句に指定されているステートメントに制御を渡します。この場合、処理プロシージャーが XML-CODE をゼロに設定してからパーサーに制御を戻したとしても、XML-CODE は元の例外番号に設定されます。

XML-CODE の値を元の例外コード以外の非ゼロ値に設定してパーサーに制御を戻した場合、結果は保証されません。

構文解析の終了

処理プロシージャーで XML-CODE を -1 に設定すると、標準の XML イベント (つまり、EXCEPTION 以外のイベント) からパーサーに戻る前に、構文解析を意図的に終了することができます。この技法は、目的に応じて文書を十分に確認できた場合、あるいは文書に何らかの不規則性があるためにそれ以上処理を続けても意味がないことがわかった場合に使用することができます。

この場合、例外条件が存在しても、パーサーは追加のイベントをシグナル通知しません。したがって、ON EXCEPTION 句が指定されている場合は、この句に制御が戻ります。このとき、XML-CODE が -1 (構文解析が意図的に終了されたことを示す) であるか調べることができます。ON EXCEPTION 句が指定されていない場合、制御は XML PARSE ステートメントの終わりに戻ります。

XML-CODE を変更せずにパーサーに戻ることにより、例外 XML イベント後に構文解析を終了することもできます。この場合、結果は意図的に終了した場合と似ていますが、XML-CODE が例外番号に設定された状態でパーサーが XML PARSE ステートメントに戻る点が異なります。

CCSID 矛盾例外

XML-CODE の例外コードが 100,001 ～ 165,535 または 200,001 ～ 265,535 の範囲にある例外イベントの場合は、特別なケースが適用されます。 これらの例外コード範囲は、文書の CCSID (文書の先頭、およびエンコード宣言を調べることで判別される) が XML PARSE ステートメントの CCSID と矛盾することを示します。

この場合は、XML-CODE の値から 100,000 または 200,000 (EBCDIC CCSID であるか ASCII CCSID であるかによって決まる) を減算することにより、文書の CCSID を判別することができます。たとえば、XML-CODE が 101,140 に設定されている場合、文書の CCSIDは 1140 です。

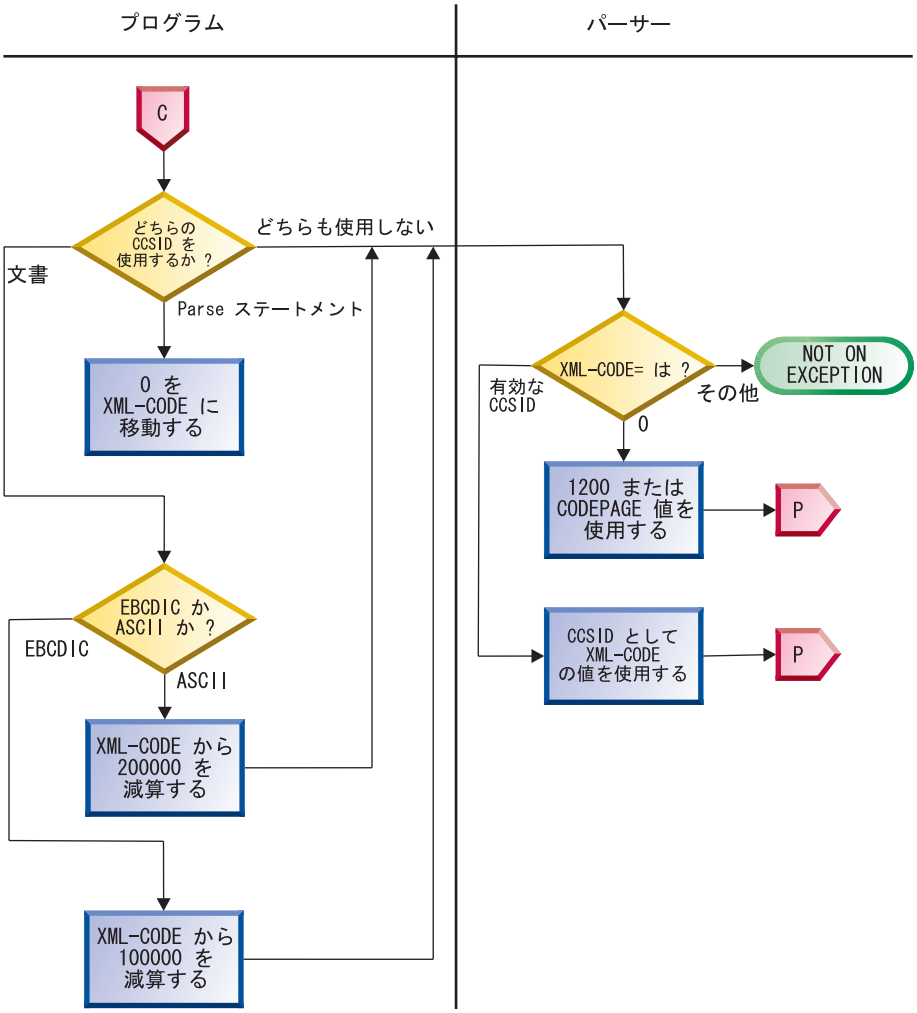
XML PARSE ステートメントの CCSID は、XML PARSE ID のタイプによって決まります。ID が国別データ項目の場合、CCSID は Unicode を示す 1200 です。XML PARSE ID が英数字の場合、CCSID は CODEPAGE コンパイラー・オプションの値によって指定される値です。

パーサーは、CCSID 矛盾例外イベント用の処理プロシージャーから戻ると、次のいずれかの処置をとります。

- XML-CODE をゼロに設定した場合、パーサーでは、XML PARSE ステートメントの CCSID (国別データ項目の場合は 1200、それ以外の場合は CODEPAGE コンパイラー・オプションの値) を使用します。
- XML-CODE を文書の CCSID (すなわち、元の XML-CODE 値から 100,000 または 200,000 (該当する方) を減算した値) に設定した場合、パーサーでは文書の CCSID を使用します。処理プロシージャーからの戻り時に XML-CODE が非ゼロ値に設定されていたとき、パーサーが処理を続けるのは、このケースに該当する場合だけです。
- それ以外の場合、パーサーは文書の処理を停止し、例外条件とともに制御を XML PARSE ステートメントに戻します。XML-CODE は、当初に例外イベントへ渡された例外コードに設定されます。

次の図は、これらの処置を示しています。この章に示す複数の図表は、 などの別ページへの接続記号によって連結されています。具体的には、次の図の は図「XML-CODE の使用法を示す、XML パーサーとプログラムの間の制御フロー」 (235 ページ) に連結されており、 は「XML 例外の制御フロー」 (243 ページ) から連結しています。

XML CCSID 例外のフロー制御



関連参照

643 ページの『続行を許可する XML 例外』
648 ページの『続行を許可しない XML 例外』
XML-CODE (*Enterprise COBOL 言語解説書*)

第 14 章 エラーの処理

コーディングまたはシステムの問題が起こることを予期して、それらの問題を処理するためのコードをプログラムに入れておきます。このようなコードは、備え付けの救難信号装置または救命艇のようなものです。このコードによって、ユーザーは問題があることを知ることになりますが、出力データやファイルが破壊されてはなりません。

エラー処理コードでは、状態の処理、メッセージの発行、プログラムの停止などのアクションを取ることができます。どのようなイベントであっても、警告メッセージをコーディングするのはよいことです。

データ入力エラーまたはご使用のシステムで定義されたエラーに対して、独自のエラー検出ルーチンを作成することができます。

COBOL には、エラー状態を予想して訂正するのに役に立つ特殊なエレメントがいくつか含まれています。

- ユーザー要求ダンプ
- STRING および UNSTRING 操作の ON OVERFLOW
- 算術演算の ON SIZE ERROR
- 入力または出力エラーを処理する技法
- CALL ステートメントの ON EXCEPTION または ON OVERFLOW
- エラー処理用のユーザー作成ルーチン

関連タスク

248 ページの『ストリングの結合および分割におけるエラーの処理』
249 ページの『算術演算でのエラーの処理』
250 ページの『入出力操作でのエラーの処理』
259 ページの『プログラム呼び出し時のエラーの処理』
259 ページの『エラー処理用のルーチンの作成』

ダンプの要求

言語環境プログラム・サービス CEE3DMP を呼び出せば、ランタイム環境の定様式ダンプを入手することができます。システム・ダンプを入手するには、ゼロのクリーンアップ値を指定して言語環境プログラム・サービス CEE3ABD を呼び出せば、クリーンアップなしの異常終了を要求することができます。

定様式ダンプの作成

プログラム内の事前に指定した任意のポイントで、言語環境プログラム・ランタイム環境およびメンバー言語ライブラリーのダンプを取ることができます。単に、言語環境プログラム呼び出し可能サービス CEE3DMP への呼び出しをコーディングするだけです。たとえば、次のようになります。

© Copyright IBM Corp. 1991, 2001

```
77  Title-1          Pic x(80)   Display.
77  Options          Pic x(255)  Display.
01  Feedback-code    Pic x(12)   Display.
. . .
    Call "CEE3DMP" Using Title-1, Options, Feedback-code
```

言語環境プログラムによって作成される定様式ダンプに記号変数を組み込むためには、TEST コンパイラー・オプションの SYM サブオプションを使用してコンパイルし、CEE3DMP の VARIABLES サブパラメーターを使用しなければなりません。

さらに、ランタイム・オプションを介して、選択したエラー状態に合ったダンプを作成するように要求することができます。

システム・ダンプの作成

プログラム内の事前に指定した任意のポイントでシステム・ダンプを取ることができます。単に、ゼロのクリーンアップ値を指定した言語環境プログラム・サービス CEE3ABD を呼び出して、クリーンアップなしの異常終了を要求するだけです。

この呼び出し可能サービスは実行単位を即刻終了させ、異常終了が出されると、システム・ダンプが要求されます。

関連参照
言語環境プログラム デバッグのガイド
CEE3DMP — ダンプの生成 (*言語環境プログラム プログラミング・リファレンス*)

ストリングの結合および分割におけるエラーの処理

ストリングの結合や分割を行うときに、STRING または UNSTRING によって使用されるポインターが受け取りフィールドの範囲を超えることがあります。オーバーフロー条件が存在する可能性はありますが、COBOL ではオーバーフローの発生を許可しません。この場合、STRING 操作や UNSTRING 操作は完了せず、受け取りフィールドは未変更のままとなり、制御は次の順次ステートメントに移動します。STRING ステートメントや UNSTRING ステートメントには ON OVERFLOW 文節が定義されておらず、未完了の操作については通知されません。

次のステートメントを考えてください。

```
String Item-1 space Item-2 delimited by Item-3
    into Item-4
    with pointer String-ptr
    on overflow
        Display "A string overflow occurred"
End-String
```

以下に、ステートメントの実行前と実行後のデータ値を示しています。

データ項目	PICTURE	実行前の値	実行後の値
Item-1	X(5)	AAAAA	AAAAA
Item-2	X(5)	EEEAA	EEEAA
Item-3	X(2)	EA	EA
Item-4	X(8)	*bbbbbbbb*[1]	*bbbbbbbb*[1]

データ項目	PICTURE	実行前の値	実行後の値
String-ptr	9(2)	0	0
1. 記号 *b* はブランク・スペースを表します。			

String-ptr の値は 0 で、受け取りフィールドには達しないため、オーバーフロー条件が発生し、STRING 操作は完了しません (String-ptr が 9 より大きい場合も同じ結果になります)。ON OVERFLOW が指定されていなかった場合は、Item-4 の内容が未変更のままであることについて通知されません。

算術演算でのエラーの処理

算術演算の結果が、それらを入れる固定小数点フィールドより大きかったり、0 除算が試みられたりすることがあります。いずれの場合も、ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの後の ON SIZE ERROR 文節でその状況を処理することができます。

固定小数点オーバーフローおよび 10 進数オーバーフローに対して ON SIZE ERROR を正しく機能させるためには、TRAP(ON) ランタイム・オプションを指定しなければなりません。

以下の場合は、ON SIZE ERROR 文節の命令ステートメントが実行され、結果フィールドは変更されません。

- 固定小数点オーバーフロー
- 0 による除算
- 0 の 0 乗
- 0 の負数乗
- 負数の分数乗

浮動小数点指数オーバーフローは、浮動小数点算術計算の値を z900 の浮動小数点オペランド形式で表すことができない場合に起こります。このタイプのオーバーフローは SIZE ERROR を起こしません。代わりに、異常終了が起こります。異常終了を代行受信し、独自のエラー・リカバリー論理を提供するために、ユーザー作成条件処理ルーチンをコーディングすることができます。

『例: 0 による除算の検査』

例: 0 による除算の検査

通知メッセージを出すように、ON SIZE ERROR 命令ステートメントをコーディングしてください。たとえば、次のようになります。

```
DIVIDE-TOTAL-COST.
    DIVIDE TOTAL-COST BY NUMBER-PURCHASED
        GIVING ANSWER
        ON SIZE ERROR
          DISPLAY "ERROR IN DIVIDE-TOTAL-COST PARAGRAPH"
          DISPLAY "SPENT " TOTAL-COST, " FOR " NUMBER-PURCHASED
          PERFORM FINISH
    END-DIVIDE
    . . .
   FINISH.
    STOP RUN.
```

この例では、0 による除算を行うと、プログラムは問題を識別するメッセージを書き出し、プログラム実行を停止します。

入出力操作でのエラーの処理

入力または出力操作が失敗しても、COBOL が自動的に訂正処置をとることはありません。重大エラーではない入出力エラーが発生した後でプログラムの実行を継続するかどうかを選択してください。

特定の入力または出力エラーの代行受信および処理には、以下のいずれかの技法を使用することができます。

- ファイル終わり条件 (AT END)
- ERROR 宣言
- ファイル状況キー
- ファイル・システム戻りコード
- INVALID KEY 句
- READ または WRITE ステートメント上の命令ステートメント句
- ERROR 宣言
- FILE STATUS 文節

VSAM ファイルについては、FILE STATUS 文節を指定した場合は、VSAM 戻りコードをテストして、エラー処理論理に対してプログラムを指示することもできます。

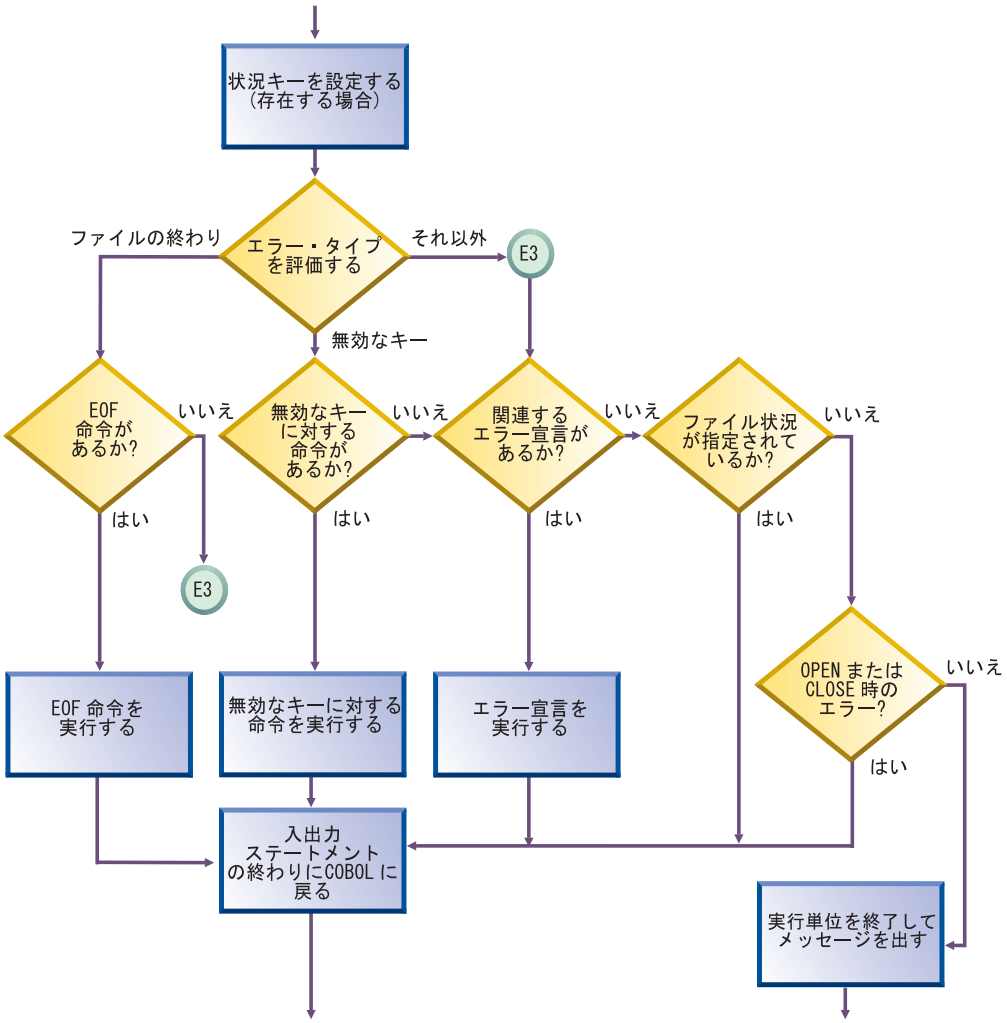
(エラー処理コードを設計に組み込むことによって) プログラムの継続を選択した場合には、適切なエラー・リカバリー手順もコーディングする必要があります。たとえば、ファイル状況キーの値を検査する手順をコーディングすることができます。

入力または出力エラーを前述の方法で処理しなかったときは、重大度 3 の言語環境プログラム条件がシグナル通知され、この条件が処理されない場合には、実行単位が終了します。

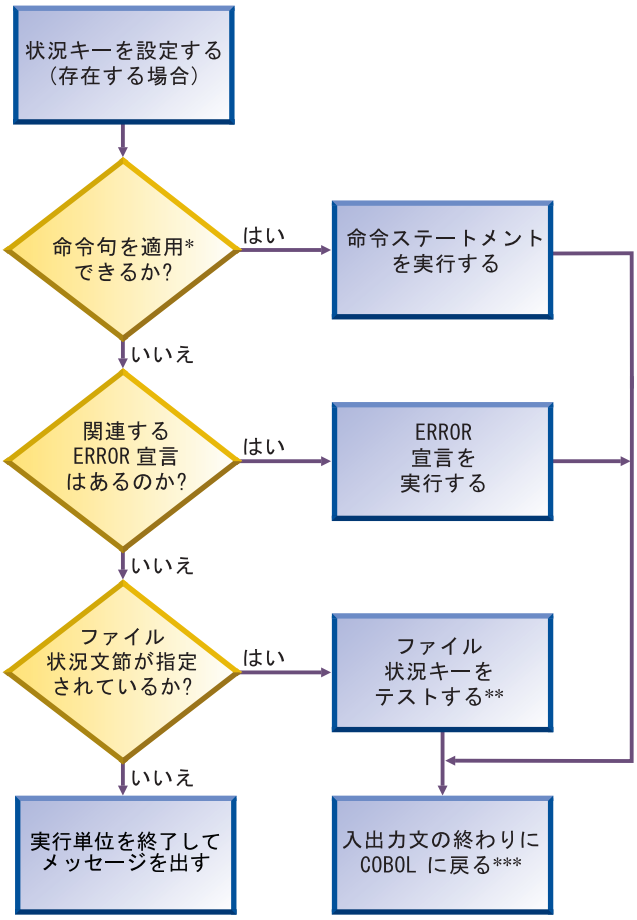
次の図は、示されたエラーの後のロジック・フローを示しています。

- VSAM 入力または出力エラー
- QSAM および行順次の入力または出力エラー

次の図は、VSAM 入力または出力エラーの後のロジック・フローを示しています。



次の図は、QSAM または行順次ファイルの出力エラー後のロジック・フローを示しています。このエラーは、REEL/UNIT 文節が指定された READ ステートメント、WRITE ステートメント、または CLOSE ステートメントから生じた可能性があります (QSAMのみ)。



*QSAM の場合の可能な句は、AT END、AT END-OF-PAGE、および INVALID KEY です。行順次の場合は、AT END です。

**ファイル状況キーをテストするコードを書く必要があります。

***COBOL プログラムの実行は、エラーを引き起こした入出力ステートメントの後で継続されます。

関連タスク

『ファイル終わり条件 (AT END) の使用』
253 ページの『ERROR 宣言のコーディング』
254 ページの『ファイル状況キーの使用』
149 ページの『QSAM ファイルのエラーの処理』
256 ページの『VSAM 戻りコードの使用 (VSAM ファイルのみ)』
258 ページの『INVALID KEY 句のコーディング』

ファイル終わり条件 (AT END) の使用

READ ステートメントの AT END 句をコーディングすると、エラーまたは正常な条件をプログラムの設計に従って処理することができます。AT END 句をコーディングすると、ファイルの終わりにその句が実行されます。AT END 句をコーディングしないと、対応する ERROR 宣言が実行されます。

多くの設計では、ファイルの終わりまでの順次読み取りが意図的に行われており、AT END 条件が予期されます。たとえば、マスター・ファイルを更新するために、トランザクションが入っているファイルを処理していると想定します。

```
PERFORM UNTIL TRANSACTION-EOF = "TRUE"
  READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD
    AT END
      DISPLAY "END OF TRANSACTION UPDATE FILE REACHED"
      MOVE "TRUE" TO TRANSACTION-EOF
  END READ
  . . .
END-PERFORM
```

コーディングした NOT AT END 句が実行されるのは、READ ステートメントが正常に完了した場合だけです。ファイル終わり以外の何らかの条件のために READ 操作が失敗すると、AT END 句も NOT AT END 句も実行されません。その代わりに、関連する宣言型プロシージャーを実行した後で、READ ステートメントの終わりに制御が渡されます。

AT END 句または EXCEPTION 宣言型プロシージャーのいずれもコーディングせずに、ファイルの状況キー文節をコーディングすることもできます。その場合は、ファイル終わり条件を検出した入力ステートメントまたは出力ステートメントの後の次の順次命令に、制御が渡されます。おそらく、適切なアクションを取るためのコードをその場所に記述することになります。

関連参照
AT END 句 (*Enterprise COBOL 言語解説書*)

ERROR 宣言のコーディング

COBOL プログラムの中に、入力または出力エラーが発生した場合に制御が与えられる ERROR 宣言型プロシージャーを 1 つ以上コーディングすることができます。以下のものを定義することができます。

- プログラム全体用の単一の共通プロシージャー
- ファイル・オープン・モード (INPUT、OUTPUT、I-O、または EXTEND) ごとに 1 つのプロシージャー
- それぞれのファイルごとの個々のプロシージャー

このようなプロシージャーをそれぞれ PROCEDURE DIVISION の宣言セクションに入れます。

プロシージャーでは、訂正処置の試行、操作の再試行、実行の継続または終了などを選択することができます。エラーについてさらに詳しい分析を行う場合は、ERROR 宣言型プロシージャーとファイル状況キーを組み合わせて使用することができます。

ブロック化ファイルの処理を継続する場合、エラーが発生したレコードより後ろにあるブロック内の残りのレコードが失われることがあります。

エラーが発生した後、システムが制御をプログラムに戻すようにしたい場合は、ERROR 宣言型プロシージャーを作成します。ERROR 宣言型プロシージャーを書き込まないと、エラー発生後、ジョブが取り消されるか、異常終了します。

マルチスレッド化: マルチスレッドのアプリケーションで入出力宣言をコーディングするときは、デッドロックが発生しないように注意する必要があります。入出力操作の結果として制御が入出力宣言に移動した場合、その宣言内のステートメントが実行されている間は、そのファイルと関連付けられた自動逐次化ロックが保持されます。したがって、宣言内に入出力操作をコーディングすると、ロジックは以下のサンプルに示すデッドロックに陥ることがあります。

```
Declaratives.
D1 section.
Use after standard error procedure on F1
. . .
    Read F2.
. . .
D2 section.
Use after standard error procedure on F2
. . .
    Read F1.
. . .
End declaratives.
. . .
    Rewrite R1.
    Rewrite R2.
```

このプログラムを 2 つのスレッドで実行している場合、次の一連のイベントが発生するおそれがあります。

1. スレッド 1: 再書き込み R1 が F1 に対するロックを獲得し、入出力エラーが発生する。
2. スレッド 1: F1 に対するロックを保持したまま、宣言 D1 に入る。
3. スレッド 2: 再書き込み R2 が F2 に対するロックを獲得し、入出力エラーが発生する。
4. スレッド 2: 宣言 D2 に入る。
5. スレッド 1: 宣言 D1 から F2 を読み取り、スレッド 2 によって保持されている F2 のロックに対し待機する。
6. スレッド 2: 宣言 D2 から F1 を読み取り、スレッド 1 によって保持されている F1 ロックに対し待機する。
7. デッドロック。

関連参照
EXCEPTION/ERROR 宣言 (*Enterprise COBOL 言語解説書*)

ファイル状況キーの使用

入力ステートメントまたは出力ステートメントがそれぞれファイルに実行された後、システムは、ファイル状況キーの 2 桁の値を更新します。一般的に、最初の桁の 0 は正常操作を示し、両方の桁に 0 が入っていると、報告すべき異常なことが何もないことを示します。ファイル状況キーは、FILE-CONTROL 段落の FILE STATUS 文節および DATA DIVISION のデータ定義を使用して設定します。

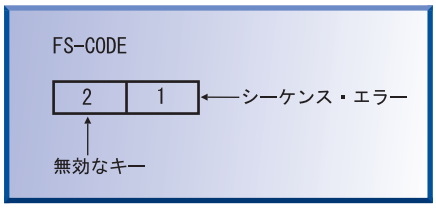
```
FILE STATUS IS data-name-1
```

変数 *data-name-1* は、2 文字の COBOL ファイル状況キーを指定します。これは WORKING-STORAGE SECTION に定義しなければなりません。この *data-name* は可変位置名にすることはできません。

プログラムは COBOL ファイル状況キーを検査して、エラーが発生しているかどうかを発見し、エラーがある場合には、どのようなタイプのエラーであるかを識別することができます。たとえば、FILE STATUS が

```
FILE STATUS IS FS-CODE
```

とコーディングされると、次のような状況に関する情報を保持するために、FS-CODE が COBOLによって使用されます。



各ファイルごとに、次の規則に従ってください。

- ファイルごとに異なるファイル状況キーを定義します。

  そうすると、アプリケーション論理エラーやディスク・エラーのような、ファイル入力または出力例外の原因を判別することができます。
- それぞれの入力または出力要求の後で、ファイル状況キーを検査します。

  0 以外の値が入っている場合、プログラムはエラー・メッセージを出すか、またはその値に基づいて処置を取ることができます。

  状況キー・コードをリセットする必要はありません。状況キー・コードは各入出力が試みられた後で設定されます。

VSAM ファイルの場合は、FILE STATUS 文節に 2 番目の ID をコーディングすることで、ファイル状況キーだけでなく、VSAM 入力または出力要求に関する詳細情報を取得することができます。

状況キーは単独でも、INVALID KEY オプションと一緒でも使用でき、EXCEPTION または ERROR 宣言を補足するためにも使用できます。このように状況キーを使用すると、それぞれの入力または出力操作の結果に関する正確な情報が得られます。

『例: ファイル状況キー』

例: ファイル状況キー

以下の COBOL コードは、ファイルのオープン後に状況キーに簡単な検査を実行します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIMCHK.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTERFILE ASSIGN TO AS-MASTERA
    FILE STATUS IS MASTER-CHECK-KEY
    . . .
DATA DIVISION.
. . .
WORKING-STORAGE SECTION.
01  MASTER-CHECK-KEY       PIC X(2).
. . .
```

```
PROCEDURE DIVISION.
    . . .
    OPEN INPUT MASTERFILE
    IF MASTER-CHECK-KEY NOT = "00"
        DISPLAY "Nonzero file status returned from OPEN " MASTER-CHECK-KEY
    . . .
```

VSAM 戻りコードの使用 (VSAM ファイルのみ)

2 文字の FILE STATUS コードは一般的すぎるため、要求の後処理を正確に示すことができないこともあります。2 番目の状況域をコーディングすることによって、VSAM 入力または出力要求に関する詳細情報を入手することができます。

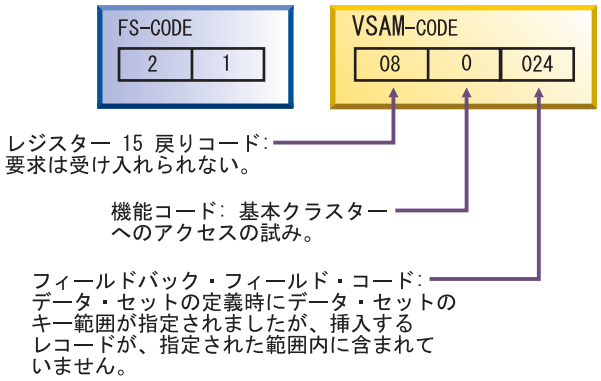
```
FILE STATUS IS data-name-1 data-name-8
```

変数 *data-name-1* は、2 文字の COBOL ファイル状況キーを指定します。変数 *data-name-8* は、COBOL ファイル状況キーが 0 でないときに VSAM 戻りコードが入れられる 6 バイトのデータ項目を指定します。

2 番目の状況域 *data-name-8* は、以下の VSAM-CODE のように、WORKING-STORAGE SECTION に定義することができます。

```
01 RETURN-STATUS.
    05 FS-CODE                  PIC X(2).
    05 VSAM-CODE.
       10 VSAM-R15-RETURN       PIC S9(4) Usage Binary.
       10 VSAM-FUNCTION         PIC S9(4) Usage Binary.
       10 VSAM-FEEDBACK         PIC S9(4) Usage Binary.
```

Enterprise COBOL は、*data-name-8* を使用して、VSAM から提供される情報を渡します。次の例では、FS-CODE が *data-name-1* に相当し、VSAM-CODE が *data-name-8* に相当します。



『例: VSAM 状況コードの検査』

例: VSAM 状況コードの検査

この COBOL コードは、以下の処理を行います。

- 索引付きファイルを読み取る (5 番目のレコードから)
- それぞれの入力または出力要求の後で、ファイル状況キーを検査する
- ファイル状況キーが 0 でないとき VSAM コードを表示する

さらに、以下に、処理中のファイルに 6 つのレコードが入っていることを想定した場合の、このプログラムからの出力を図示しています。

```
IDENTIFICATION DIVISION
PROGRAM-ID. EXAMPLE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT VSAMFILE ASSIGN TO VSAMFILE
    ORGANIZATION IS INDEXED
    ACCESS DYNAMIC
    RECORD KEY IS VSAMFILE-KEY
    FILE STATUS IS FS-CODE VSAM-CODE.
DATA DIVISION.
FILE SECTION.
FD  VSAMFILE
    RECORD  30.
01  VSAMFILE-REC.
    10 VSAMFILE-KEY             PIC X(6).
    10 FILLER                   PIC X(24).
WORKING-STORAGE SECTION.
01  RETURN-STATUS.
    05 FS-CODE                  PIC XX.
    05 VSAM-CODE.
       10 VSAM-RETURN-CODE      PIC S9(2) Usage Binary.
       10 VSAM-COMPONENT-CODE   PIC S9(1) Usage Binary.
       10 VSAM-REASON-CODE      PIC S9(3) Usage Binary.
PROCEDURE DIVISION.
    OPEN  INPUT VSAMFILE.
    DISPLAY "OPEN INPUT VSAMFILE FS-CODE: " FS-CODE.

    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
       STOP RUN
    END-IF.

    MOVE "000005" TO VSAMFILE-KEY.
    START VSAMFILE KEY IS EQUAL TO VSAMFILE-KEY.
    DISPLAY "START VSAMFILE KEY="  VSAMFILE-KEY
            " FS-CODE: "  FS-CODE.
    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
    END-IF.

    IF FS-CODE = "00"
       PERFORM READ-NEXT UNTIL FS-CODE NOT = "00"
    END-IF.

    CLOSE VSAMFILE.
    STOP RUN.

READ-NEXT.
    READ VSAMFILE NEXT.
    DISPLAY "READ NEXT VSAMFILE FS-CODE: " FS-CODE.
    IF FS-CODE NOT = "00"
       PERFORM VSAM-CODE-DISPLAY
    END-IF.
    DISPLAY VSAMFILE-REC.

VSAM-CODE-DISPLAY.
    DISPLAY "VSAM-CODE ==>"
            " RETURN: "  VSAM-RETURN-CODE,
            " COMPONENT: "  VSAM-COMPONENT-CODE,
            " REASON: "  VSAM-REASON-CODE.
```

以下に、VSAM コード情報を検査するプログラム例からの出力例を示しています。

```
OPEN INPUT VSAMFILE FS-CODE: 00
START VSAMFILE KEY=000005 FS-CODE: 00
READ NEXT VSAMFILE FS-CODE: 00
000005 THIS IS RECORD NUMBER 5
READ NEXT VSAMFILE FS-CODE: 00
000006 THIS IS RECORD NUMBER 6
READ NEXT VSAMFILE FS-CODE: 10
VSAM-CODE ==> RETURN: 08 COMPONENT: 2 REASON: 004
```

INVALID KEY 句のコーディング

INVALID KEY 句は、VSAM 索引付きファイルおよび相対ファイルに対する READ、START、WRITE、REWRITE、および DELETE 要求に含めることができます。INVALID KEY 句に制御が与えられるのは、誤った索引キーのために入力エラーまたは出力エラーが発生した場合です。

状況キーを評価し、特定の INVALID KEY 条件を判別するには、INVALID KEY と一緒に FILE STATUS 文節を使用してください。

さらに、QSAM ファイルに対する WRITE 要求に INVALID KEY を含めることもできますが、この INVALID KEY 句の意味は限定されています。この句が使用されるのは、いっぱいになっているディスクに書き込もうとした場合に限られます。

INVALID KEY および ERROR 宣言

INVALID KEY 句は、次の点で ERROR 宣言とは異なります。

- INVALID KEY 句は、限定された種類のエラーに対してのみ働きますが、ERROR 宣言はあらゆる形式をカバーします。
- INVALID KEY 句は入出力動詞に直接コーディングされますが、ERROR 宣言は別個にコーディングされます。
- INVALID KEY 句は 1 つの入出力操作に固有ですが、ERROR 宣言はもっと汎用的なものです。

INVALID KEY 条件を引き起こすステートメントで INVALID KEY をコーディングすると、制御は INVALID KEY 命令ステートメントに転送されます。このため、ERROR 宣言をコーディングしても、それらの宣言は実行されません。

NOT INVALID KEY

コーディングした NOT INVALID KEY 句が実行されるのは、そのステートメントが正常に完了した場合だけです。INVALID KEY 以外の条件のために操作が失敗すると、INVALID KEY 句も NOT INVALID KEY 句も実行されません。その代わりに、関連する ERROR 宣言を実行した後で、制御がステートメントの終わりに渡されます。

『例: FILE STATUS および INVALID KEY』

例: FILE STATUS および INVALID KEY

たとえば、マスター顧客レコードが入っているファイルがあり、トランザクション更新ファイルの情報でそれらのレコードを更新しなければならないとします。プログラムは、各トランザクション・レコードを読み取り、マスター・ファイルの中の対応するレコードを見つけ、必要な更新を行います。どちらのファイルのレコードにもそれぞれ顧客番号用のフィールドがあり、マスター・ファイルの中の各レコードには固有の顧客番号があります。

顧客レコードのマスター・ファイル用の FILE-CONTROL 記入項目には、索引編成を定義するステートメント、ランダム・アクセス、基本レコード・キーとして MASTER-CUSTOMER-NUMBER、およびファイル状況キーとして CUSTOMER-FILE-STATUS が含まれています。次の例では、FILE STATUS と INVALID KEY を一緒に使用して、入力ステートメントまたは出力ステートメントが失敗した理由をより厳密に判別する方法を示します。

```
.
.  (read the update transaction record)
.
MOVE "TRUE" TO TRANSACTION-MATCH
MOVE UPDATE-CUSTOMER-NUMBER TO MASTER-CUSTOMER-NUMBER
READ MASTER-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD
  INVALID KEY
    DISPLAY "MASTER CUSTOMER RECORD NOT FOUND"
    DISPLAY "FILE STATUS CODE IS: " CUSTOMER-FILE-STATUS
    MOVE "FALSE" TO TRANSACTION-MATCH
END-READ
```

プログラム呼び出し時のエラーの処理

プログラムが、別個にコンパイルされたプログラムを動的に呼び出すとき、呼び出されるプログラムがシステムから使用できないことがあります。たとえば、システムがストレージ不足だったり、ロード・モジュールを見つけることができない場合です。CALL ステートメントに ON EXCEPTION 句も ON OVERFLOW 句もない場合、アプリケーションは異常終了します。

一連のステートメントを実行してユーザー定義のエラー処理を行う場合は、ON EXCEPTION 句を使用します。たとえば、次のようになります。

```
MOVE "REPORTA" TO REPORT-PROG
CALL REPORT-PROG
  ON EXCEPTION
    DISPLAY "Program REPORTA not available, using REPORTB.'
    MOVE "REPORTB" TO REPORT-PROG
    CALL REPORT-PROG
    END-CALL
END-CALL
```

プログラム REPORTA が使用できない場合、制御は ON EXCEPTION 句で継続されます。

ON EXCEPTION 句は、呼び出されるプログラムの可用性についてのみ適用されます。呼び出されるプログラムの実行中にエラーが発生した場合、ON EXCEPTION 句は実行されません。

関連タスク

Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド

エラー処理用のルーチンの作成

ON EXCEPTION 句、ON SIZE ERROR 句、またはその他の言語構成要素を使用すると、プログラムの実行中に発生する可能性のあるエラー条件の大部分を処理することができます。しかし、マシン・チェックなどの特別な条件が発生すると、通常、アプリケーションには制御が戻りません。そのため、異常終了します。

しかし、Enterprise COBOL および言語環境プログラムでは、そのような条件が発生したときにユーザー作成プログラムが制御を獲得できる方法を提供しています。言語環境プログラム条件処理を使用して、独自のエラー処理プログラムを COBOL で書くことができます。これらのプログラムでは、報告、分析、修正を行うことができ、ユーザー・プログラムが実行を再開できるようにします。

言語環境プログラムがユーザー作成のエラー・プログラムに制御を渡すようにさせるには、まずその入り口点を言語環境プログラムに対して示し、登録する必要があります。PROCEDURE-POINTER データ項目を使用して、プロシージャー入り口点の入り口アドレスを言語環境プログラム・サービスに渡すことができます。

関連タスク

452 ページの『プロシージャー・ポインターと関数ポインターの使用』

第 2 部 プログラムのコンパイルおよびデバッグ

第 15 章 z/OS のもとでのコンパイル 263
JCL を使用したコンパイル 264
カタログ式プロシージャーの使用 264
コンパイル・プロシージャー (IGYWC) . . . 265
コンパイルおよびリンク・エディット用プロシージャー (IGYWCL) 267
コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG) 268
コンパイル、ロード、実行用のプロシージャー (IGYWCG) 269
コンパイル、プリリンク、リンク・エディット用のプロシージャー (IGYWCPL) 270
コンパイル、プリリンク、リンク・エディット、実行用のプロシージャー (IGYWCPLG) . 271
プリリンクおよびリンク・エディット用のプロシージャー (IGYWPL) 272
コンパイル、プリリンク、ロード、実行用のプロシージャー (IGYWCPG) 273
プログラムをコンパイルするための JCL の作成 274
例: コンパイル用のユーザー作成 JCL . . . 275
TSO のもとでのコンパイル 276
例: TSO のもとでコンパイルするための ALLOCATE および CALL 277
例: TSO のもとでコンパイルするための CLIST 277
アセンブラー・プログラムからコンパイラーを開始する 278
コンパイラー入出力の定義 279
z/OS のもとでコンパイラーによって使用されるデータ・セット 280
論理レコード長とブロック・サイズ 281
ソース・コード・データ・セットの定義 (SYSIN) 282
ソース・ライブラリーの指定 (SYSLIB) 283
出力データ・セットの定義 (SYSPRINT) . . . 283
コンパイラー・メッセージの端末への送信 (SYSTERM) 283
オブジェクト・コードの作成 (SYSLIN または SYSPUNCH) 284
関連データ・ファイルの作成 (SYSADATA) . . 284
出力 Java データ・セット (SYSJAVA) の定義 285
デバッグ・データ・セットの定義 (SYSDEBUG) 285
z/OS のもとでのコンパイラー・オプションの指定 285
PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定 286
例: JCL を使用したコンパイラー・オプションの指定 287
例: TSO のもとでのコンパイラー・オプションの指定 287
z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力 287
複数プログラムのコンパイル (バッチ・コンパイル) 289
例: バッチ・コンパイル 290
バッチ・コンパイルでのコンパイラー・オプションの指定 291
例: バッチ・コンパイルでのオプションの優先順位 292
例: バッチ・コンパイルでの LANGUAGE オプション 292
ソース・プログラムのエラーの訂正 293
コンパイル・エラー・メッセージのリストの生成 294
コンパイラー検出エラーに関するメッセージおよびリスト 294
コンパイル・エラー・メッセージの形式 . . . 295
コンパイル・エラー・メッセージの重大度コード 296

第 16 章 UNIX のもとでのコンパイル 299
UNIX のもとでの環境変数の設定 299
UNIX のもとでのコンパイラー・オプションの指定 300
cob2 コマンドを使用したコンパイルおよびリンク 301
入出力の定義 302
DLL の作成 302
例: UNIX のもとでコンパイルするための cob2 の使い方 303
cob2 303
cob2 入出力ファイル 305
スクリプトを使用したコンパイル 306

第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行 307
UNIX のもとでのオブジェクト指向アプリケーションの準備 308
UNIX 環境外でのオブジェクト指向プログラムのコンパイル 309
オブジェクト指向プログラムの実行 309

第 18 章 コンパイラー・オプション 311
COBOL 85 標準に準拠するオプション設定値 . . . 313
矛盾するコンパイラー・オプション 313
ADATA 315
ADV 315
ARITH 316
AWO 316
BUFSIZE 317
CICS 317
CODEPAGE 318
COMPILE 319
CURRENCY 320
DATA 321
DATEPROC 322
DBCS 323
DECK 323
DIAGTRUNC 324

© Copyright IBM Corp. 1991, 2001

DLL 324
DUMP 325
DYNAM 326
EXIT 327
EXPORTALL 327
FASTSRT 327
FLAG 328
FLAGSTD 329
INTDATE 330
LANGUAGE 331
LIB 332
LINECOUNT 332
LIST 333
MAP 333
NAME 334
NSYMBOL 335
NUMBER 336
NUMPROC 336
OBJECT 338
OFFSET 338
OPTIMIZE 339
未使用データ項目 339
OUTDD 340
PGMNAME 340
PGMNAME(COMPAT) 341
PGMNAME(LONGUPPER) 341
PGMNAME(LONGMIXED) 342
QUOTE/APOST 343
RENT 343
RMODE 344
SEQUENCE 345
SIZE 346
SOURCE 346
SPACE 347
SQL 347
SSRANGE 348
TERMINAL 349
TEST 349
THREAD 352
TRUNC 353
TRUNC の例 1 355
TRUNC の例 2 355
VBREF 356
WORD 357
XREF 357
YEARWINDOW 358
ZWB 359
コンパイラー指示ステートメント 360

第 19 章 デバッグ 365
ソース言語によるデバッグ 366
プログラム・ロジックのトレース 366
入出力エラーの検出および処理 367
データの妥当性検査 367
初期化されていないデータの検出 368
プロシージャーに関する情報の生成 368
デバッグ行 368
デバッグ・ステートメント 368
例: USE FOR DEBUGGING 369
コンパイラー・オプションを使用したデバッグ . . 370
コーディング・エラーの検出 371
構文検査のみ 372
条件付きコンパイル 372
行シーケンス問題の検出 372
有効範囲の検査 372
診断するエラーのレベルの選択 373
例: 組み込みメッセージ 374
プログラム・エンティティー定義および参照の検出 375
データ項目のリスト 376
リストの入手 377
例: 短縮リスト 379
例: SOURCE および NUMBER 出力 381
例: MAP 出力 382
例: 組み込みマップ要約 383
MAP 出力で使用される用語 384
LIST および MAP 出力で使用される記号 . . . 385
例: ネストされたプログラム・マップ 386
LIST 出力の読み取り 386
例: プログラム初期化コード 387
シグニチャー情報バイト: コンパイラー・オプション 389
シグニチャー情報バイト: DATA DIVISION 391
シグニチャー情報バイト: ENVIRONMENT DIVISION 392
シグニチャー情報バイト: PROCEDURE DIVISION 動詞 392
シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目 394
例: ソース・コードから生成されるアセンブラー・コード 395
例: TGT メモリー・マップ 396
例: WORKING-STORAGE の位置とサイズ 397
例: DSA メモリー・マップ 397
例: XREF 出力 - データ名相互参照 399
例: XREF 出力 - プログラム名相互参照 . . . 400
例: 組み込み相互参照 400
例: OFFSET コンパイラー出力 401
例: VBREF コンパイラー出力 402
デバッガーの使用準備 402

第 15 章 z/OS のもとでのコンパイル

Enterprise COBOL プログラムは、z/OS のもとでは、ジョブ制御言語 (JCL)、TSO コマンド、CLIST、または ISPF パネルを使用してコンパイルすることができます。

- JCL を使ったコンパイルの場合、IBM は一連のカタログ式プロシージャーを用意しています。これを使用すれば、書かなければならない JCL コーディングの量を減らすことができます。カタログ式プロシージャーが要件に合わない場合は、独自の JCL を書くことができます。JCL を使用して、単一のプログラムをコンパイルすること、または複数のプログラムをバッチ・ジョブの中でコンパイルすることができます。
- TSO のもとでコンパイルするときは、TSO コマンド、CLIST、または ISPF パネルを使用することができます。

また、Enterprise COBOL コンパイラーを呼び出すツールまたはインターフェースを開発している場合には、Enterprise COBOL コンパイラーをアセンブラー・プログラムから開始することもできます。

JCL を使用するか、または TSO のもとでコンパイルするときは、コンパイル・ステップ中でコンパイルに必要なデータ・セットを定義し、プログラムおよび求める出力に必要なコンパイラー・オプションを指定する必要があります。

コンパイラーは、COBOL プログラムを、コンピューターが処理できる言語 (オブジェクト・コード) に変換します。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムのデバッグおよび調整に役立つ補足情報を提供します。コンパイルを制御するには、コンパイラー指示ステートメントおよびコンパイラー・オプションを使用してください。

プログラムをコンパイルした後、コンパイルの結果を検討して、コンパイラー検出エラーがあれば、それを訂正する必要があります。

関連タスク

264 ページの『JCL を使用したコンパイル』
276 ページの『TSO のもとでのコンパイル』
278 ページの『アセンブラー・プログラムからコンパイラーを開始する』
279 ページの『コンパイラー入出力の定義』
285 ページの『z/OS のもとでのコンパイラー・オプションの指定』
289 ページの『複数プログラムのコンパイル (バッチ・コンパイル)』
293 ページの『ソース・プログラムのエラーの訂正』

関連参照

360 ページの『コンパイラー指示ステートメント』
280 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』
287 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

© Copyright IBM Corp. 1991, 2001

JCL を使用したコンパイル

コンパイルの場合は、JCL に以下の情報を含める必要があります。

- ジョブ記述
- コンパイラーを実行するためのステートメント
- 必要なデータ・セットの定義 (HFS ファイルを使用する場合はディレクトリー・パスを含む)

z/OS のもとでプログラムをコンパイルする最も簡単な方法は、カタログ式プロシージャーを使用する JCL をコーディングすることです。*カタログ式プロシージャー*とは、プロシージャー・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに配置された一連のジョブ制御ステートメントです。

次の JCL は、カタログ式プロシージャーの一般形式を示します。

```
//jobname  JOB  parameters
//stepname EXEC [PROC=]procname[,{PARM=|PARM.stepname=}'options']
//SYSIN    DD  data set parameters
. . .         (source program to be compiled)
/*
//
```

カタログ式プロシージャーを使用してオブジェクト指向のプログラムをコンパイルするときには、追加の考慮事項があります。

473 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク

『カタログ式プロシージャーの使用』
274 ページの『プログラムをコンパイルするための JCL の作成』
285 ページの『z/OS のもとでのコンパイラー・オプションの指定』
291 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』
470 ページの『DLL を作成するためのプログラムのコンパイル』

関連参照

280 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

カタログ式プロシージャーの使用

カタログ式プロシージャーは、JCL の EXEC ステートメントに指定してください。

次の JCL は、Enterprise COBOL プログラムをコンパイルし、必要なデータ・セットを定義するために、IBM 提供のカタログ式プロシージャー IGYWC を呼び出します。

```
//JOB1      JOB1
//STEPA     EXEC PROC=IGYWC ←――――――(カタログ式プロシージャーの名前)
//COBOL.SYSIN DD *←――――――――――――(データ・セットを記述する適切なパラメーター)
000100 IDENTIFICATION DIVISION←――――(ソース・コード)
.
.
.
/*                                   (オプションの区切りステートメント)
```

これらのプロシージャーは、z/OS の一部であるすべてのジョブ・スケジューラーで使用することができます。スケジューラーは、不要なパラメーターを検出すると、それらを無視するか、または代替パラメーターで置き換えます。

コンパイラー・オプションがプロシージャーで明示的に指定されない場合、インストール時に設定されたデフォルトのオプションが適用されます。これらのデフォルト・オプションは、必要なオプションを含めた EXEC を使用してオーバーライドすることができます。

対応する DD ステートメントをオーバーライドすれば、階層ファイル・システム内のデータ・セットを指定することができます。ただし、指定するコンパイラー・ユーティリティー・ファイル (SYSUT*x*) およびコピー・ライブラリー (SYSLIB) は MVS データ・セットでなければなりません。

カタログ式プロシージャーの呼び出し、EXEC ステートメントのオーバーライドと追加、および DD ステートメントのオーバーライドと追加の詳細は、言語環境プログラム情報の中で記載されています。

関連タスク
言語環境プログラム プログラミング・ガイド

関連参照
『コンパイル・プロシージャー (IGYWC)』
267 ページの『コンパイルおよびリンク・エディット用プロシージャー (IGYWCL)』
268 ページの『コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)』
269 ページの『コンパイル、ロード、実行用のプロシージャー (IGYWCG)』
270 ページの『コンパイル、プリリンク、リンク・エディット用のプロシージャー (IGYWCPL)』
271 ページの『コンパイル、プリリンク、リンク・エディット、実行用のプロシージャー (IGYWCPLG)』
272 ページの『プリリンクおよびリンク・エディット用のプロシージャー (IGYWPL)』
273 ページの『コンパイル、プリリンク、ロード、実行用のプロシージャー (IGYWCPG)』
z/OS DFSMS: Program Management

コンパイル・プロシージャー (IGYWC)

IGYWC プロシージャーは、プログラムをコンパイルするための単一ステップのプロシージャーです。オブジェクト・モジュールを作成します。コンパイラーを呼び出す他のどのカタログ式プロシージャーにおいても、コンパイル・ステップは類似しています。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (または適切なパラメーター)
```

IGYWC カタログ式プロシージャーは、以下のステートメントから構成されます。

```
//IGYWC  PROC  LNGPRFX='IGY.V3R1M0',SYSLBLK=3200
//*
//*  COMPILE A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL   EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))        (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。

『例: HFS を使用するコンパイル用の JCL』

例: HFS を使用するコンパイル用の JCL: 次のジョブは、IGYWC というプロシージャーを使用して、階層ファイル・システム (HFS) に置かれている COBOL プログラム demo.cbl をコンパイルします。このジョブは、生成されたコンパイラー・リスト demo.lst、オブジェクト・ファイル demo.o、および SYSADATA ファイル demo.adt を HFS に書き込みます。

```
//HFSDEMO JOB ,
// TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,REGION=50M,
//  NOTIFY=&SYSUID,USER=&SYSUID
//COMPILE EXEC IGYWC,
// PARM.COBOL='LIST,MAP,RENT,FLAG(I,I),XREF,ADATA'
//SYSPRINT DD PATH='/u/userid/cobol/demo.lst',    (1)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),              (2)
// PATHMODE=SIRWXU,                               (3)
// FILEDATA=TEXT                                  (4)
//SYSLIN DD PATH='/u/userid/cobol/demo.o',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSADATA DD PATH='/u/userid/cobol/demo.adt',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSIN DD PATH='/u/userid/cobol/demo.cbl',
// PATHOPTS=ORDONLY,
// FILEDATA=TEXT,
// RECFM=F
```

(1) PATH には、HFS ファイルのパス名を指定します。

(2) PATHOPTS は、ファイルのアクセス (読み取り、読み取り / 書き込みなど) を示し、ファイルの状況 (付加、作成、切り捨てなど) を設定します。

(3) PATHMODE は、ファイルの作成時に設定される許可、すなわちファイル・アクセス属性を示します。

(4) FILEDATA は、データをテキストまたはバイナリーのいずれとして扱うかを指定します。

この例に示すコンパイル用 DD ステートメントでは、オーバーライドとして、HFS (PATH='*hfs-directory-path*') と MVS データ・セット (DSN=*traditional-data-set-name*) の組み合わせを使用することができます。ただし、コンパイラー・ユーティリティー・ファイル (DD ステートメント SYSUT*x*) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS データ・セットでなければなりません。

関連参照

UNIX システム・サービス コマンド解説書

MVS JCL 解説書

280 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

コンパイルおよびリンク・エディット用プロシージャー (IGYWCL)

IGYWCL プロシージャーは、プログラムのコンパイルとリンク・エディットを行う 2 ステップのプロシージャーです。COBOL ジョブ・ステップは、リンケージ・エディターまたはバインダーへの入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (または適切なパラメーター)
```

IGYWCL カタログ式プロシージャーは、以下のステートメントから構成されます。

```
//IGYWCL PROC  LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLOCK SIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))             (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                (3)
//             DISP=SHR
```

```
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(3) SYSLIB は、インストール先によって異なります。

コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)

IGYWCLG プロシージャーは、プログラムのコンパイル、リンク・エディット、および実行を行う 3 ステップのプロシージャーです。

COBOL ジョブ・ステップは、リンケージ・エディターまたはバインダーへの入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。COBOL プログラムでデータ・セットを参照する場合は、これらのデータ・セットを定義する DD ステートメントも指定しなければなりません。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (または適切なパラメーター)
```

以下に、IGYWCLG カタログ式プロシージャーを構成するステートメントを示しています。

```
//IGYWCLG PROC LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                  (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                    (3)
//             DISP=SHR
```

```
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(3) SYSLIB は、インストール先によって異なります。

コンパイル、ロード、実行用のプロシージャー (IGYWCG)

IGYWCG プロシージャーは、プログラムのコンパイル、ロード、および実行を行う 2 ステップのプロシージャーです。COBOL ジョブ・ステップは、ローダーへの入力となるオブジェクト・モジュールを作成します。COBOL プログラムでデータ・セットを参照する場合は、これらのデータ・セットを定義する DD ステートメントも指定しなければなりません。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (または適切なパラメーター)
```

以下に、IGYWCG カタログ式プロシージャーを構成するステートメントを示しています。

```
//IGYWCG PROC  LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE'
//*
//*  COMPILE, LOAD AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                   (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,                 (2)
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                 (3)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//GO     EXEC PGM=LOADER,COND=(8,LT,COBOL),REGION=2048K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                   (4)
//             DISP=SHR
//SYSLOUT  DD  SYSOUT=*
```

```
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSLIN は、HFS 上に置くことができます。
(3) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(4) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、リンク・エディット用のプロシージャー (IGYWCPL)

IGYWCPL プロシージャーは、プログラムのコンパイル、プリリンク、およびリンク・エディットを行う 3 ステップのプロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//SYSIN DD  *       (または適切なパラメーター)
```

以下に、IGYWCPL カタログ式プロシージャーを構成するステートメントを示しています。

```
//IGYWCPL PROC LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',PLANG=EDCPMSGE,
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE, PRELINK AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLOCK SIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))              (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
```

```
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                (3)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(3) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、リンク・エディット、実行用のプロシージャー (IGYWCPLG)

IGYWCPLG プロシージャーは、プログラムのコンパイル、プリリンク、リンク・エディット、および実行を行う 4 ステップのプロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//SYSIN DD  *       (または適切なパラメーター)
```

以下に、IGYWCPLG カタログ式プロシージャーを構成するステートメントを示しています。

```
//IGYWCPLG PROC LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             PLANG=EDCPMSGE,
//             LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, PRELINK, LINK EDIT, AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                    (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))                  (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

```
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                    (3)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(3) SYSLIB は、インストール先によって異なります。

プリリンクおよびリンク・エディット用のプロシージャー (IGYWPL)

IGYWPL プロシージャーは、プログラムのプリリンクおよびリンク・エディットを行う 2 ステップのプロシージャーです。

IGYWPL カタログ式プロシージャーは、以下のステートメントから構成されます。

```
//IGYWPL PROC  PLANG=EDCPMSGE,SYSLBLK=3200,
//             LIBPRFX='CEE',
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  PRELINK AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   PLANG     EDCPMSGE         PRELINK MESSAGES MEMBER NAME
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //PLKED.SYSIN DD . . .
//*
//PLKED   EXEC PGM=EDCPRLK,PARM='',
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,                (1)
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
```

```
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=&SYSLBLK)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//LKED   EXEC PGM=HEWL,COND=(4,LT,PLKED),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,               (2)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=SYSDA,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
//SYSIN    DD  DUMMY
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSLIB は、インストール先によって異なります。

コンパイル、プリリンク、ロード、実行用のプロシージャー (IGYWCPG)

IGYWCPG プロシージャーは、プログラムのコンパイル、プリリンク、ロード、および実行を行う 4 ステップのプロシージャーです。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD  *       (または適切なパラメーター)
```

IGYWCPG カタログ式プロシージャーは、以下のステートメントで構成されます。

```
//IGYWCPG PROC LNGPRFX='IGY.V3R1M0',SYSLBLK=3200,
//             PLANG=EDCPMSGE,
//             LIBPRFX='CEE'
//*
//*  COMPILE, PRELINK, LOAD, AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V3R1M0       PREFIX FOR LANGUAGE DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   PLANG     EDCPMSGE         PRELINKER MESSAGES MODULE
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL   EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,                 (1)
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))               (2)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//PLKED   EXEC PGM=EDCPRLK,PARM='',COND=(8,LT,COBOL),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
```

```
//             DISP=SHR
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(&PLANG),
//             DISP=SHR
//SYSLIB   DD  DUMMY
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS), (3)
//             SPACE=(32000,(100,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//GO       EXEC PGM=LOADER,COND=(8,LT,COBOL),REGION=2048K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,                  (4)
//             DISP=SHR
//SYSLOUT  DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&PLKSET,DISP=(OLD,DELETE)
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

(1) STEPLIB は、インストール先によって異なります。
(2) SYSUT5 は、LIB オプションが使用される場合のみ必要になります。
(3) SYSMOD は、HFS 上に置くことができます。
(4) SYSLIB は、インストール先によって異なります。

プログラムをコンパイルするための JCL の作成

カタログ式プロシージャーでは、より複雑なプログラムに必要となる z/OS のプログラミングの柔軟性が得られない場合は、独自のジョブ制御ステートメントを作成してください。次の JCL は、プログラムのコンパイルに使用される一般形式を示します。

```
//jobname  JOB  acctno,name,MSGCLASS=1                (1)
//stepname EXEC PGM=IGYCRCTL,PARM=(options)           (2)
//STEPLIB  DD   DSNAME=IGY.V3R1M0.SIGYCOMP,DISP=SHR   (3)
//SYSUT1   DD   UNIT=SYSDA,SPACE=(subparms)           (4)
//SYSUT2   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT3   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT4   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT5   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT6   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT7   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSPRINT DD   SYSOUT=A                              (5)
//SYSLIN   DD   DSNAME=MYPROG,UNIT=SYSDA,             (6)
//              DISP=(MOD,PASS),SPACE=(subparms)
//SYSIN    DD   DSNAME=dsname,UNIT=device,            (7)
//              VOLUME=(subparms),DISP=SHR
```

(1) JOB ステートメントは、ジョブの始まりを示します。

(2) EXEC ステートメントは、Enterprise COBOL コンパイラー (IGYCRCTL) を呼び出すことを指定します。

(3) この DD ステートメントは、Enterprise COBOL コンパイラーが常駐するデータ・セットを定義します。

(4) SYSUT DD ステートメントは、コンパイラーがソース・プログラムを処理するために使用するユーティリティー・データ・セットを定義します。SYSUT ファイルはすべて、直接アクセス記憶装置上に置かなければなりません。

(5) SYSPRINT DD ステートメントは、LIST や MAP などのオプションからの出力を受け取るデータ・セットを定義します。SYSOUT=A は、システム出力装置を宛先とするデータ・セットの標準指定です。

(6) SYSLIN DD ステートメントは、OBJECT オプション (オブジェクト・モジュール) からの出力を受け取るデータ・セットを定義します。

(7) SYSIN DD ステートメントは、ジョブ・ステップ (ソース・コード) への入力として使用するデータ・セットを定義します。

以下のデータ・セットに対するコンパイル DD ステートメントには、HFS (PATH='*hfs-directory-path*') と MVS データ・セット (DSN=*traditional-data-set-name*) を混在させることができます。

- ソース・ファイル
- オブジェクト・ファイル
- リスト
- ADATA ファイル
- デバッグ・ファイル
- 実行可能モジュール

ただし、コンパイラー・ユーティリティー・ファイル (DD ステートメント SYSUT*x*) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS データ・セットでなければなりません。

『例: コンパイル用のユーザー作成 JCL』
473 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連参照
MVS JCL 解説書

例: コンパイル用のユーザー作成 JCL

次の例は、基本 JCL を適応させることができる可能性をいくつか示します。

```
//JOB1     JOB                                          (1)
//STEP1    EXEC PGM=IGYCRCTL,PARM='OBJECT'              (2)
//STEPLIB  DD   DSNAME=IGY.V3R1M0.SIGYCOMP,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=MYPROG,UNIT=SYSDA,
//              DISP=(MOD,PASS),SPACE=(TRK,(3,3))
//SYSIN    DD   *                                       (3)
000100 IDENTIFICATION DIVISION.
  . . .
/*                                                      (4)
```

(1) JOB1 は、ジョブの名前です。

(2) STEP1 は、ジョブ内の単一ジョブ・ステップの名前です。この EXEC ステー

トメントは、生成されたオブジェクト・コードを、後でリンク・ステップへの入力として使用するディスクまたはテープに入れることも指定しています。

(3) アスタリスクは、入力ストリームの中で入力データ・セットが続くことを示します。

(4) 区切りステートメント /* は、入力ストリームの中で、データとその後の制御ステートメントとを区切ります。

TSO のもとでのコンパイル

TSO のもとでは、TSO コマンド、コマンド・リスト (CLIST)、REXX EXEC、または ISPF を使用すると、従来の MVS データ・セットを使用するプログラムをコンパイルすることができます。HFS ファイルを使用するプログラムの場合は、TSO コマンドまたは REXX EXEC を使用してコンパイルすることができます。いずれの方式を使用する場合も、データ・セットを割り振り、コンパイルを要求しなければなりません。

1. ALLOCATE コマンドを使用してデータ・セットを割り振ります。

   どのコンパイルでも、作業データ・セット (SYSUT*n*) と、SYSIN および SYSPRINT データ・セットを割り振らなければなりません。特定のコンパイラー・オプションを指定する場合は、以下のデータ・セットを割り振ってください。

   - OBJECT コンパイラー・オプションを指定した場合は、SYSLIN データ・セットを割り振って、オブジェクト・モジュールを作成します。
   - TERMINAL オプションを指定した場合は、SYSTERM データ・セットを割り振って、端末でコンパイラー・メッセージを入手します。
   - LIB オプションを指定した場合に、Enterprise COBOL プログラムで COPY ステートメントや REPLACE ステートメントを使用したときは、SYSLIB データ・セットを割り振ります。これは、HFS パスではなく、従来の MVS データ・セットでなければなりません。
   - プログラムで LIB オプションを指定した場合は、SYSUT5 を割り振ります。このユーティリティー・データ・セットは、HFS パスではなく、従来の MVS データ・セットでなければなりません。

   データ・セットはどんな順序で割り振っても構いません。ただし、コンパイルを開始する前に、必要なすべてのデータ・セットを割り振っておく必要があります。

2. CALL コマンドを使用してコンパイルを要求します。

```
READY
CALL 'IGY.V3R1M0.SIGYCOMP(IGYCRCTL)'
```

ALLOCATE コマンドおよび CALL コマンドは、TSO コマンド行に指定することができます。また、HFS ファイルを使用していない場合は、これらのコマンドを CLIST に指定することもできます。

SYSUT*x* ユーティリティー・データ・セットおよび SYSLIB ライブラリーを除いて、どのコンパイラー・データ・セットにも HFS ファイルを割り振ることができます。ALLOCATE ステートメントの形式は、次のとおりです。

```
Allocate File(SYSIN) Path('/u/myu/myap/std/prog2.cbl')
Pathopts(ORDONLY) Filedata(TEXT)
```

『例: TSO のもとでコンパイルするための ALLOCATE および CALL』
『例: TSO のもとでコンパイルするための CLIST』

例: TSO のもとでコンパイルするための ALLOCATE および CALL

次の例は、TSO のもとでコンパイルするときの ALLOCATE および CALL コマンドの指定方法を示します。

```
READY
ALLOCATE FILE(SYSUT1) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT2) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT3) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT4) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT5) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT6) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSUT7) CYLINDERS SPACE(1 1)
READY
ALLOCATE FILE(SYSPRINT) SYSOUT
READY
ALLOCATE FILE(SYSTERM) DATASET(*)
READY
ALLOCATE FILE(SYSLIN) DATASET(PROG2.OBJ) NEW TRACKS SPACE(3,3)
READY
ALLOCATE FILE(SYSIN) DATASET(PROG2.COBOL) SHR
READY
CALL 'IGY.V3R1M0.SIGYCOMP(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E),TERMINAL'
  .
   (COBOL listings and messages)
  .
READY
FREE FILE(SYSUT1,SYSUT2,SYSUT3,SYSUT4,SYSUT5,SYSUT6,SYSUT7,SYSPRINT,SYSTERM,SYSIN,SYSLIN)
READY
```

例: TSO のもとでコンパイルするための CLIST

以下に、TSO のもとでコンパイルするためのサンプル CLIST を示します。この例では、FREE コマンドは必要ありません。しかし、上手なプログラミングを行うには、ファイルを割り振る前に、それらを解放しておく必要があります。

```
PROC 1 MEM
CONTROL LIST
FREE (SYSUT1)
FREE (SYSUT2)
FREE (SYSUT3)
FREE (SYSUT4)
FREE (SYSUT5)
FREE (SYSUT6)
FREE (SYSUT7)
FREE (SYSPRINT)
FREE (SYSIN)
FREE (SYSLIN)
ALLOC F(SYSPRINT) SYSOUT
ALLOC F(SYSIN) DA(COBOL.SOURCE(&MEM))  SHR REUSE
ALLOC F(SYSLIN) DA(COBOL.OBJECT(&MEM)) OLD REUSE
```

```
ALLOC F(SYSUT1) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT2) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT3) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT4) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT5) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT6) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
ALLOC F(SYSUT7) NEW SPACE(5,5) TRACKS UNIT(SYSDA)
CALL 'IGY.V3R1M0.SIGYCOMP(IGYCRCTL)'
```

アセンブラー・プログラムからコンパイラーを開始する

アセンブラー・プログラムから Enterprise COBOL コンパイラーを開始するには、動的起動によって、ATTACH マクロ命令または LINK マクロ命令を使用します。動的起動を使用する場合、次の情報を Enterprise COBOL に渡す必要があります。

- コンパイルに指定するオプション
- Enterprise COBOL コンパイラーが処理時に使用するデータ・セットの DD 名

形式

```
symbol   {LINK|ATTACH} EP=IGYCRCTL,
               PARAM=(optionlist[,ddnamelist]),VL=1
```

EP Enterprise COBOL コンパイラーのシンボル名を指定します。制御プログラム (ライブラリー・ディレクトリー記入項目からの) が、プログラムが実行を開始すべき入り口点を決定します。

PARAM アセンブラー・プログラムから Enterprise COBOL コンパイラーに渡されるアドレス・パラメーターをサブリストとして指定します。アドレス・パラメーター・リストの最初のフルワードには、COBOL*optionlist* のアドレスが入っています。2 番目のフルワードには、*ddnamelist* のアドレスが入っています。

3 番目と 4 番目のフルワードには、NULL パラメーターまたは 0 が入っています。

optionlist

コンパイル用に指定された COBOL オプションが入っている可変長リストのアドレスを指定します。リストを提示しない場合でも、このアドレスは指定しなければなりません。

optionlist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。オプションが指定されない場合、このカウントは 0 でなければなりません。*optionlist* は、各フィールドが次のフィールドとコンマで区切られて、自由形式です。リストにはブランクまたは 0 を含むことはできません。コンパイラーが認識するのは、そのリストの最初の 100 文字だけです。

ddnamelist

COBOL コンパイラー処理中に使用されるデータ・セットの代替の DD 名を含んでいる可変長リストのアドレスを指定します。標準 DD 名を使用する場合は、*ddnamelist* を省略することができます。

ddnamelist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。8 バイト未満の名前は、左寄せにしてブランクで埋め込む必要がありま

す。代替 DD 名がリストから省略されている場合、標準名が想定されます。リスト内で名前が省略されている場合、8 バイトの記入項目には 2 進数 0 が入っていなければなりません。リストを短縮して、終わりから名前を省略することができます。

指定された SYSUT*n* データ・セットはすべて直接アクセス記憶装置上になければならず、物理順次編成をもっていなければなりません。これらのデータ・セットが HFS に常駐していてはなりません。

次の表に、*ddnamelist* 内の 8 バイトの記入項目のシーケンスを示します。

DD 名 8 バイト記入項目	置き換えられる名前
1	SYSLIN
2	適用されない
3	適用されない
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4
12	SYSTERM
13	SYSUT5
14	SYSUT6
15	SYSUT7
16	SYSADATA
17	SYSJAVA
18	SYSDEBUG

VL アドレス・パラメーター・リストの最後のフルワードの符号ビットを 1 に設定するように指定します。

Enterprise COBOL コンパイラーは、処理を完了すると、レジスター 15 に戻りコードを書き込みます。

コンパイラー入出力の定義

コンパイラーが作業を行うために使用する数種類のデータ・セットを定義しなければなりません。コンパイラーは、入力データ・セットおよびライブラリーを使用して、さまざまなタイプの出力 (オブジェクト・コード、リスト、およびメッセージを含む) を作成します。また、コンパイラーは、コンパイル時にユーティリティー・データ・セットも使用します。

関連タスク

282 ページの『ソース・コード・データ・セットの定義 (SYSIN)』
283 ページの『ソース・ライブラリーの指定 (SYSLIB)』
283 ページの『出力データ・セットの定義 (SYSPRINT)』
283 ページの『コンパイラー・メッセージの端末への送信 (SYSTERM)』
284 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』
284 ページの『関連データ・ファイルの作成 (SYSADATA)』
285 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』

関連参照
『z/OS のもとでコンパイラーによって使用されるデータ・セット』

z/OS のもとでコンパイラーによって使用されるデータ・セット

次の表は、コンパイラーが使用する各データ・セットの機能、装置要件、および許可される装置クラスをリストしたものです。

タイプ	ddname	機能	必須かどうか?	装置要件	許可される装置クラス	HFS に常駐
入力	SYSIN[1]	ソース・プログラムを読み取る	はい	カード読取装置 (中間記憶装置)	任意	はい
	SYSLIB またはその他のコピー・ライブラリー[1]	ユーザー・ソース・ライブラリー (PDS) を読み取る	プログラムに COPY または BASIS ステートメントが含まれている場合 (LIB は必須)	直接アクセス	SYSDA	いいえ
ユーティリティー	SYSUT1、SYSUT2、SYSUT3、SYSUT4、SYSUT6[2]	コンパイル時にコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSDA	いいえ
	SYSUT5[2]	コンパイル時にコンパイラーによって使用される作業データ・セット	プログラムに COPY、REPLACE、または BASIS ステートメントが含まれている場合 (LIB は必須)	直接アクセス	SYSDA	いいえ
	SYSUT7[2]	リストを作成するためにコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSDA	いいえ

タイプ	ddname	機能	必須かどうか?	装置要件	許可される装置クラス	HFS に常駐
出力	SYSPRINT[1]	ストレージ・マップ、リスト、およびメッセージを書き込む	はい	プリンター (中間記憶装置)	SYSSQ、SYSDA、標準出力クラス A	はい
	SYSTERM	進行メッセージおよび診断メッセージを書き込む	TERM が有効な場合	出力装置 (TSO 端末)		はい
	SYSPUNCH	オブジェクト・コードを作成する	DECK が有効な場合	カード穿孔装置 (直接アクセス)	SYSSQ、SYSDA	はい
	SYSLIN	コンパイラーからの出力およびリンケージ・エディターへの入力として、オブジェクト・モジュール・データ・セットを作成する	OBJECT が有効な場合	直接アクセス	SYSSQ、SYSDA	はい
	SYSADATA	関連データ・ファイル・レコードを書き込む	ADATA または EVENTS が有効な場合	出力装置		はい
	SYSJAVA	クラス定義用の生成済み Java ソース・ファイルを作成する	クラス定義をコンパイルする場合	(HFS ファイルでなければならない)		はい
	SYSUDUMP、SYSABEND、または SYSMDUMP	ダンプを書き込む	DUMP が有効な場合 (まれにしか使用されない)	直接アクセス	SYSDA	はい
	SYSDEBUG	オブジェクト・モジュールとは別のデータ・セットに記号デバッグ情報テーブルを書き込む	TEST(. . .,SYM,SEPARATE) が有効な場合	直接アクセス	SYSDA	はい

1. EXIT オプションを使用して、これらのデータ・セットからユーザー出口を提供することができます。
2. これらのデータ・セットは単一ボリュームでなければなりません。

関連参照
『論理レコード長とブロック・サイズ』
327 ページの『EXIT』

論理レコード長とブロック・サイズ

作業データ・セット (SYSUT*n*) および HFS ファイル以外のコンパイラー・データ・セットの場合は、DCB パラメーターの BLKSIZE サブパラメーターを使用してブロック・サイズを設定することができます。この値は、データ・セットが常駐する装置に許されているものでなければなりません。設定する値は、データ・セットが固定長か可変長かによって異なります。

固定長レコード (RECFM=F または RECFM=FB) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL × *n* に等しくなります (*n* はブロック化因数)。次の表は、固定長データ・セットに定義されている値を示しています。一般には、これらの値を変更してはなりませんが、以下の値は変更可能です。

- SYSDEBUG: リストされた範囲内で、任意の LRECL を指定することができます。推奨値は 1024 です。
- SYSPRINT、SYSDEBUG: BLKSIZE=0 を指定でき、その結果として、システム決定のブロック・サイズになります。

データ・セット	RECFM	LRECL (バイト)	BLKSIZE[1]
SYSIN	F または FB	80	80 x *n*
SYSLIN	F または FB	80	80 x *n*
SYSPUNCH	F または FB	80	80 x *n*
SYSLIB または他のコピー・ライブラリー	F または FB	80	80 x *n*
SYSPRINT[2]	F または FB	133	133 x *n*
SYSTERM	F または FB	80	80 x *n*
SYSDEBUG[2]	F または FB	80 ～ 1024	LRECL x *n*
1. *n* = ブロック化因数 2. BLKSIZE=0 を指定すると、システムがブロック・サイズを決定します。			

可変長レコード (RECFM=V) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL + 4 に等しくなります。

データ・セット	RECFM	LRECL (バイト)	BLKSIZE (バイト) 最小許容値
SYSADATA	VB	1020	1024

ソース・コード・データ・セットの定義 (SYSIN)

次に示すように、SYSIN DD ステートメントを使用して、ソース・コードを入れるデータ・セットを定義してください。

```
//SYSIN    DD  DSNAME=dsname,UNIT=SYSSQ,
//             VOLUME=(subparms),DISP=SHR
```

ソース・コードまたは BASIS ステートメントを入力ストリームに直接入れることができます。その場合は、次の SYSIN DD ステートメントを使用してください。

```
//SYSIN    DD *
```

DD * の表記法を使用する場合、ソース・コードまたは BASIS ステートメントをこのステートメントの後に続けなければなりません。コンパイルの後に別のジョブ・ステップが続いている場合は、そのステップの EXEC ステートメントが /* ステートメントまたは最後のソース・ステートメントの後に続きます。

ソース・ライブラリーの指定 (SYSLIB)

プログラムに COPY または BASIS ステートメントが含まれている場合は、SYSLIB DD ステートメントを追加してください。これらの DD ステートメントは、(ソース・コード内の) COPY ステートメントまたは入力ストリーム内の BASIS ステートメントによって要求されるデータが入っているライブラリー (区分データ・セット) を定義します。

```
//SYSLIB   DD  DSNAME=copylibname,DISP=SHR
```

複数のコピー・ライブラリーまたは基本ライブラリーがある場合には、複数の DD ステートメントを連結します。

```
//SYSLIB   DD  DSNAME=PROJECT.USERLIB,DISP=SHR
//         DD  DSNAME=SYSTEM.COPYX,DISP=SHR
```

ライブラリーは直接アクセス記憶装置に置かれます。TSO のもとで JCL を使用してコンパイルする場合、ライブラリーが階層ファイル・システムに入っていてはなりません。

NOLIB オプションが有効な場合は、SYSLIB DD ステートメントは必要ありません。

出力データ・セットの定義 (SYSPRINT)

次に示すように、SYSPRINT を使用すればリストを出力することができます。

```
//SYSPRINT DD   SYSOUT=A
```

出力は、SYSOUT データ・セット、プリンター、直接アクセス記憶装置、または磁気テープ装置に送ることができます。このリストには、PARM パラメーターのデフォルト・オプションまたは要求されたオプションの結果 (すなわち、診断メッセージ、オブジェクト・コード・リスト) が入れられます。

定義するデータ・セットについては、以下を指定してください。

- 順次データ・セットの名前、PDS または PDSE メンバーの名前、あるいは HFS パス。
- データ・セットの LRECL として 133。
- データ・セットの RECFM として F または B。
- ブロック・サイズ (DCB パラメーターの BLKSIZE サブパラメーターを使用して指定するか、システム決定のデフォルトのブロック・サイズをシステムに設定させます)。

コンパイラー・メッセージの端末への送信 (SYSTERM)

TSO のもとでコンパイルしている場合は、次に示すように、SYSTERM データ・セットを定義して、コンパイラー・メッセージを端末に送ることができます。

```
ALLOC F(SYSTERM) DA(*)
```

SYSTERM は、さまざまな方法で定義することができます (SYSOUT データ・セット、ディスク上のデータ・セット、HFS 内のファイル、別の印刷クラスなど)。

オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)

OBJECT コンパイラー・オプションを使用すると、オブジェクト・コードを、従来の MVS データ・セットまたは HFS ファイルとしてディスクに保管したり、テープに保管することができます。コンパイラーは、SYSLIN DD ステートメントまたは SYSPUNCH DD ステートメントで定義されたファイルを使用して、オブジェクト・コードを保管します。

```
//SYSLIN   DD  DSNAME=dsname,UNIT=SYSDA,
//             SPACE=(subparms),DISP=(MOD,PASS)
```

SYSLIN DD ステートメントの DISP パラメーターを使用して、オブジェクト・コード・データ・セットが以下のうちどのように処理されるかを示すことができます。

- リンケージ・エディターまたはバインダーに渡される
- カタログ化される
- 保存される
- カタログ化された既存のライブラリーに追加される

上記の例では、データが作成され、別のジョブ・ステップであるリンケージ・エディターまたはバインダーのジョブ・ステップに渡されます。

インストール先で、DECK オプションおよび SYSPUNCH DD ステートメントを使用することができます。

```
//SYSPUNCH DD  SYSOUT=B
```

B は、穿孔データ・セットの標準出力クラスです。

NOOBJECT オプションが有効な場合、SYSLIN DD ステートメントは必要ではありません。NODECK オプションが有効な場合は、SYSPUNCH DD ステートメントは必要ではありません。

関連参照
338 ページの『OBJECT』
323 ページの『DECK』

関連データ・ファイルの作成 (SYSADATA)

ADATA コンパイラー・オプションまたは EVENTS コンパイラー・オプションを使用する場合は、SYSADATA データ・セットを定義してください。

```
//SYSADATA DD  DSNAME=dsname,UNIT=SYSDA
```

SYSADATA DD ステートメントで定義されるファイルは、コンパイル中に収集されたプログラムに関する情報が入っている特定のレコード・タイプを含む順次ファイルです。このファイルは、従来の MVS データ・セットまたは HFS ファイルにすることができます。

関連参照
315 ページの『ADATA』

出力 Java データ・セット (SYSJAVA) の定義

オブジェクト指向プログラムをコンパイルする場合は、SYSJAVA DD ステートメントを追加します。生成された Java ソース・ファイルは、SYSJAVA DD 名に書き込まれます。

```
//SYSJAVA   DD PATH='/u/userid/java/Classname.java',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

ファイルは、HFS ファイルでなければなりません。

関連タスク

307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

デバッグ・データ・セットの定義 (SYSDEBUG)

JCL または TSO からコンパイルするときに、TEST(. . .,SYM,SEPARATE) コンパイラー・オプションを指定すると、シンボリック・デバッグ情報テーブルは、SYSDEBUG DD ステートメントで指定されるデータ・セットに書き込まれます。以下に例を示します。

```
//SYSDEBUG DD  DSNAME=dsname,UNIT=SYSDA
```

定義するデータ・セットについては、以下を指定してください。

- 順次データ・セットの名前、PDS または PDSE メンバーの名前、あるいは HFS パス。
- データ・セットの LRECL として 80 以上、1024 以下の数。SYSDEBUG のデフォルトの LRECL は 1024 です。
- データ・セットの RECFM として F または FB。
- ブロック・サイズ (DCB パラメーターの BLKSIZE サブパラメーターを使用して指定するか、システム決定のデフォルトのブロック・サイズをシステムに設定させます)。

関連参照

349 ページの『TEST』

z/OS のもとでのコンパイラー・オプションの指定

コンパイラーは、デフォルトのコンパイラー・オプションを使用してインストールされ、セットアップされます。コンパイラーのインストール時に、設置場所のシステム・プログラマーは、たとえば、パフォーマンスの向上やある程度の標準の維持を確実に行うために、コンパイラー・オプションの設定を修正することができます。インストール先で固定として設定したコンパイラー・オプションはどれもオーバーライドすることはできません。固定されていないオプションについては、以下のいずれかの方法でコンパイラー・オプションを指定して、デフォルト設定をオーバーライドすることができます。

- COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
- コンパイラーの開始時に組み込む (JCL の EXEC ステートメントの PARM パラメーター、または TSO のコマンド行のいずれか)。

コンパイラーは、以下の優先順位で高位から下位へ、オプションを認識します。

1. 設置場所で固定されているインストール先デフォルト
2. バッチ内の最初のプログラムで有効にされた BUFSIZE、LIB、OUTDD、SIZE、および SQL コンパイラー・オプションの値
3. IDENTIFICATION DIVISION の前に置かれた PROCESS (または CBL) ステートメント
4. コンパイラー呼び出しで指定されたオプション (JCL PARM パラメーターまたは TSO CALL コマンド)
5. 固定されていないインストール先デフォルト

この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定されたときにどのオプションが有効かも決定します。

大部分のオプションは対になっているため、いずれか一方を選択します。たとえば、相互参照リストのオプションの対は XREF|NOXREF です。相互参照リストを希望する場合は、XREF を指定してください。希望しない場合は、NOXREF を指定してください。

サブパラメーターがあるオプションもあります。たとえば、リストで 1 ページ当たり 44 行が必要な場合には、LINECOUNT(44) と指定します。

287 ページの『例: JCL を使用したコンパイラー・オプションの指定』
287 ページの『例: TSO のもとでのコンパイラー・オプションの指定』

関連タスク

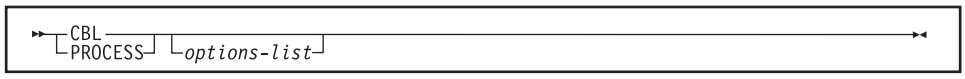
『PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定』
291 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』

関連参照

313 ページの『矛盾するコンパイラー・オプション』
311 ページの『第 18 章 コンパイラー・オプション』

PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定

COBOL ソースのプログラムの PROCESS ステートメントにコンパイラー・オプションをコーディングすることができます。



PROCESS ステートメントは、IDENTIFICATION DIVISION ヘッダーの前、かつコメント行またはコンパイラー指示ステートメントの前に入れなければなりません。

PROCESS は 1 ～ 66 桁目の間で開始することができます。シーケンス・フィールドとしては 1 ～ 6 桁目を使用できます。シーケンス・フィールドを付けて使用する

ときは、PROCESS は 8 ～ 66 桁目の間で開始できます。シーケンス・フィールドが使用される場合には、そこに 6 文字が含まれていなければならず、最初の文字は数字でなければなりません。

CBL を PROCESS の同義語として使用することができます。CBL は 1 ～ 70 桁目の間で開始することができます。シーケンス・フィールドを付けて使用する場合には、CBL は 8 ～ 70 桁目の間で開始することができます。

PROCESS と *options-list* の最初のオプションとは、1 つ以上のブランクで区切らなければなりません。オプションはコンマまたはブランクで区切ります。個々のオプションとそのサブオプションの間にスペースを入れてはなりません。

2 つ以上の PROCESS ステートメントを使用することができます。複数の PROCESS ステートメントを使用する場合は、別の種類のステートメントを間に入れずに、PROCESS ステートメントを続けて指定する必要があります。複数の PROCESS ステートメントにわたってオプションを継続することはできません。

プログラミングの編成で、COBOL コンパイラーのデフォルトのオプション・モジュールと一緒に PROCESS ステートメントを使用することを禁止することができます。編成で許可されていない PROCESS ステートメントが COBOL プログラムで見つかった場合、COBOL コンパイラーはエラー診断を生成します。

例: JCL を使用したコンパイラー・オプションの指定

次の例は、z/OS のもとで JCL を使用してコンパイラー・オプションを指定する方法を示します。

```
. . .
//STEP1     EXEC PGM=IGYCRCTL,
//               PARM='LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

例: TSO のもとでのコンパイラー・オプションの指定

次の例は、OS/390 のもとでコンパイラー・オプションを指定する方法を示します。

```
. . .

READY

CALL 'SYS1.LINKLIB(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力

コンパイラーがソース・プログラムの処理を完了すると、選択されたコンパイラー・オプションによっては、次のうちの 1 つまたは複数が作成されていることになります。

コンパイラー・オプション	コンパイラー出力	出力のタイプ
ADATA または EVENTS	コンパイル中のプログラムに関する情報	関連データ・ファイル
DLL	DLL サポートで使用可能なオブジェクト・モジュール	オブジェクト
DUMP	コンパイルが異常終了した場合は、システム・ダンプ (SYSUDUMP、SYSABEND、または SYSMDUMP DD ステートメントが必要)。まれにしか使用されません。	リスト

コンパイラー・オプション	コンパイラー出力	出力のタイプ
EXPORTALL	DLL のエクスポートされた記号	オブジェクト
FLAG	コンパイラーがプログラムで検出したエラーのリスト	リスト
LIST	マシン言語およびアセンブラー言語でのオブジェクト・コードのリスト	リスト
MAP	プログラムのデータ項目のマップ	リスト
NUMBER	ユーザー提供の行番号がリストに示される	リスト
COMPILE を指定した OBJECT または DECK	オブジェクト・コード	オブジェクト
OFFSET	オブジェクト・コードの相対アドレスのマップ	リスト
OPTIMIZE	OBJECT が有効な場合は、最適化オブジェクト・コード	オブジェクト
RENT	OBJECT が有効な場合は、再入可能オブジェクト・コード	オブジェクト
SOURCE	ソース・プログラムのリスト	リスト
SQL	DB2 バインド処理用の SQL ステートメントおよびホスト変数情報	データベース要求モジュール
SSRANGE	テーブル内の参照を検査するための余分のコード	オブジェクト内
TERMINAL	端末に送信される進行メッセージと診断メッセージ	端末
TEST(*hook location suboption*)	デバッグ・ツールのコンパイルされたフック	オブジェクト内の余分のコード
TEST(SYM,NOSEP)	デバッグ・ツールおよび定様式ダンプのための情報テーブル	オブジェクト
TEST(SYM,SEP)	デバッグ・ツールおよび定様式ダンプのための情報テーブル	デバッグ情報サイド・ファイル
VBREF	ソース・プログラム内の動詞の相互参照リスト	リスト
XREF	プロシージャー、プログラム、およびデータの名前に関するソート済み相互参照リスト	リスト

コンパイルからのリスト出力は、SYSPRINT で定義されたデータ・セットに入れられ、オブジェクト出力は SYSLIN または SYSPUNCH に入れられます。進行メッセージと診断メッセージは、SYSPRINT データ・セットに入れるだけでなく、SYSTERM データ・セットに送ることができます。関連データ・ファイルのレコードは、SYSADATA で定義されたデータ・セットに入れられます。データベース要求モジュール (DBRM) は、DBRMLIB 内に定義されるデータ・セットです。デバッグ情報サイド・ファイルは、SYSDEBUG 内に定義されるデータ・セットです。

コンパイル時に作成されたリストを保管してください。デバッグまたは調整が必要になったときに、作業のテスト段階でこのリストを使用することができます。

コンパイル後、次に行うステップは、コンパイラーがプログラムで見つけたエラーを修正することです。

エラーが 1 つも見つからなかった場合には、プロセスの次のステップである、プログラムのリンク・エディットまたはバインドに進むことができます (コンパイラー・オプションを使用してオブジェクト・コードの生成を抑止した場合、オブジェクト・コードを入手するには再コンパイルする必要があります)。

関連参照

294 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』
311 ページの『第 18 章 コンパイラー・オプション』

関連タスク

言語環境プログラムのもとでのリンク・エディットおよび実行の準備 (*言語環境プログラム プログラミング・ガイド*)

複数プログラムのコンパイル (バッチ・コンパイル)

コンパイラーを 1 回呼び出すだけで、一連の別個の COBOL プログラムをコンパイルすることができます。このコンパイルから作成されたオブジェクト・プログラムをリンクして、1 つのロード・モジュールまたは複数の別個のロード・モジュールにすることができます (これは NAME コンパイラー・オプションで制御されます)。

バッチ・ジョブの一部として複数のプログラムをコンパイルするときは、以下を行う必要があります。

- 1 つのロード・モジュールを作成するのか、複数のロード・モジュールを作成するのかを決定する。
- シーケンスの中の各プログラムを終了させる。
- コンパイラー・オプションを指定する (バッチ・ジョブ内のプログラムで指定されたコンパイラー・オプションの影響を認識する)。

別個のロード・モジュールを作成するには、それぞれのモジュール・セットの前に NAME オプションを付けなければなりません。コンパイラーが NAME コンパイラー・オプションを検出すると、次の NAME コンパイラー・オプションが見つかるまで、そのシーケンス内の最初のプログラムおよび後続のすべてのプログラムが 1 つのロード・モジュールにリンク・エディットされます。このようにして、NAME オプションを使用してコンパイルされる連続するプログラムは、それぞれ 1 つのロード・モジュールに入れられます。

シーケンス内の各プログラムは、END PROGRAM マーカーで終了させる必要があります (ただし、バッチの最後のプログラムは例外であり、END PROGRAM マーカーは省略可能です)。あるいは、シーケンスの中の各プログラムの前に CBL または PROCESS ステートメントを付けることができます。

プログラム (プログラム・シーケンスの最後のプログラムを除く) で END PROGRAM マーカーを省略すると、そのシーケンス内の次のプログラムが直前のプログラムにネストされます。以下のいずれかの状況では、エラーとなる可能性があります。

- この結果ネストされるプログラムの中に PROCESS ステートメントがある。
- CBL ステートメントがシーケンス番号域 (桁 1 ～ 6) に完全にコーディングされていない。

  CBL ステートメントがシーケンス番号域 (桁 1 ～ 6) に完全にコーディングされている場合は、それはソース・ステートメント行のラベルと見なされるため、その CBLステートメントに関するエラー・メッセージは出されません。

290 ページの『例: バッチ・コンパイル』

関連タスク
291 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』

関連参照
334 ページの『NAME』

例: バッチ・コンパイル

次の例は、IGYWCL カタログ式プロシージャーの 1 回の呼び出しを示しています。このプロシージャーは、3 つのプログラム（PROG1、PROG2、および PROG3）をバッチ・コンパイルし、2 つのロード・モジュールを作成します。

- PROG1 と PROG2 が一緒にリンク・エディットされ、PROG2 という名前の 1 つのロード・モジュールが形成されます。このロード・モジュールの入り口点は、デフォルトによって、ロード・モジュールの最初のプログラムである PROG1 が使用されます。
- PROG3 は単独でリンク・エディットされ、PROG3 という名前のロード・モジュールが作成されます。この場合は、ロード・モジュール内の唯一のプログラムであるので、入り口点も PROG3 です。

```
//jobname  JOB  acctno,name,MSGLEVEL=1
//stepname EXEC IGYWCL
//COBOL.SYSIN DD *
010100 IDENTIFICATION DIVISION.
010200  PROGRAM-ID PROG1.
        . . .
019000  END PROGRAM PROG1.
020100 IDENTIFICATION DIVISION.
020200  PROGRAM-ID PROG2.
        . . .
029000  END PROGRAM PROG2.
 CBL NAME
030100 IDENTIFICATION DIVISION.
030200  PROGRAM-ID PROG3.
        . . .
039000  END PROGRAM PROG3.
/*
//LKED.SYSLMOD DD DSN=&&GOSET                    (1)
/*
//P2       EXEC PGM=PROG2
//STEPLIB  DD   DSN=&&GOSET,DISP=(SHR,PASS)      (2)
. . .                                            (3)
/*
//P3       EXEC PGM=PROG3
//STEPLIB  DD   DSN=&&GOSET,DISP=(SHR,PASS)      (2)
. . .                                            (3)
/*
//
```

(1) LKED ステップ SYSLMOD のデータ・セット名が、一時名 &&GOSET に変更されます。メンバー名は指定しません。

(2) 一時データ・セット &&GOSET は、コンパイル済みプログラムを実行するために、ステップ P2 および P3 の STEPLIB として使用されます。言語環境プログラムのライブラリーが共用ストレージに常駐していない場合は、さらに、ライブラリー・データ・セットを STEPLIB の DD ステートメントとして追加する必要があります。

(3) PROG1 および PROG2 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

(4) PROG3 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

関連参照
IBM 提供のカタログ式プロシージャー (*言語環境プログラム プログラミング・ガイド*)

バッチ・コンパイルでのコンパイラー・オプションの指定

バッチ・シーケンスの各プログラムのコンパイラー・オプションは、以下のいずれかの通常の方法で指定できます。

- CBL または PROCESS ステートメントをプログラムの前に置く
- コンパイラーの呼び出し

CBL または PROCESS ステートメントが現行プログラムに指定されていると、コンパイラーは、最初のプログラムの前に有効なオプションと一緒に CBL または PROCESS ステートメントを解決します。現行プログラムに CBL または PROCESS ステートメントが含まれていない場合は、コンパイラーは、前のプログラムで有効であったオプションの設定を使用します。

特定のコンパイラー・オプションは、シーケンスの中の各プログラムのコンパイラー・オプション設定の優先順位に影響を及ぼすことに注意してください。

1. 設置場所で固定されているインストール先デフォルト。
2. バッチ内の最初のプログラムで有効にされた BUFSIZE、LIB、OUTDD、SIZE、および SQL コンパイラー・オプションの値。
3. 現行プログラムに対する CBL または PROCESS ステートメント上のオプション (ある場合)。
4. コンパイラー呼び出しで指定されたオプション (JCL PARM または TSO CALL)。
5. 固定されていないインストール先デフォルト。

バッチ・シーケンス内のいずれかのプログラムで BUF、LIB、OUTDD、SIZE、または SQL オプションが必要な場合は、バッチ・シーケンス内の最初のプログラムでそのオプションを有効にする必要があります (BASIS、COPY、または REPLACE ステートメントを処理するとき、コンパイラーは、バッチ処理のすべてのプログラムを 1 つの入力ファイルとして処理します)。

バッチで LIB オプションを指定する場合は、バッチ・コンパイル時に NUMBER および SEQUENCE オプションを変更することはできません。コンパイラーは、LIB オプションのもとで NUMBER および SEQUENCE を処理するとき、バッチ内のすべてのプログラムを 1 つの入力ファイルとして扱います。このため、入力ファイル全体のシーケンス番号が昇順になっていなければなりません。

コンパイラーが CBL または PROCESS ステートメントの LANGUAGE オプションをエラーと診断すると、言語選択は、コンパイラーが最初の CBL または PROCESS ステートメントを検出する前に有効であった状態に戻します。バッチ・コンパイル時に有効な言語は、その環境での CBL または PROCESS ステートメントの処理規則に従います。

『例: バッチ・コンパイルでのオプションの優先順位』
『例: バッチ・コンパイルでの LANGUAGE オプション』

例: バッチ・コンパイルでのオプションの優先順位

以下のリストは、バッチ・コンパイルのコンパイラー・オプション階層を示しています。

```
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0   Date 09/30/2001 Time 10:34:14 Page 1

INVOCATION PARAMETERS:
NOTERM

PROCESS(CBL) statements:
        CBL  CMPR2,FLAG(I,I)

Options in effect:     All options are installation defaults unless otherwise noted:
    NOADATA
     ADV
      QUOTE
    NOAWO
      BUFSIZE(4096)
      CMPR2            Process option PROGRAM 1
      .
      .
      FLAG(I,I)        Process option PROGRAM 1
      .
      .
    NOTERM             INVOCATION option
      .
      .
      End of compilation for program 1
      .
      .
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0   Date 09/30/2001 Time 10:34:14 Page 4
PROCESS(CBL) statements:
       CBL APOST
Options in effect:
    NOADATA
      ADV
    APOST              Process option in effect for PROGRAM 2
    NOAWO
      BUFSIZE(4096)
    NOCMPR2            Returns to installation option for PROGRAM 2, and subsequent prog
      .
      .
      FLAG(I)          Returns to installation option
      .
      .
    NOTERM             INVOCATION option remains in effect
      .
      .
End of compilation for program 2
```

例: バッチ・コンパイルでの LANGUAGE オプション

次の例は、バッチ環境での LANGUAGE コンパイラー・オプションの動作を示しています。デフォルトのインストール・オプションは ENGLISH (省略形は EN) で、呼び出しオプションは XX (存在しない言語) です。

```
            Source                        Language in
                                            effect
-------------------------------------------------------------------------
CBL   LANG(JP),FLAG(I,I),APOST,SIZE(MAX)   |EN|  Installation default -- EN
      IDENTIFICATION DIVISION.             |JP|  Invocation           -- XX
      PROGRAM-ID.  COMPILE1.               |:|
            .  .  .                        |:|
      END PROGRAM  COMPILE1.               |:|
CBL    LANGUAGE(YY)                        |EN|  CBL resets language
CBL    SIZE(2048K),LANGUAGE(JP),LANG(!!)   |:|  to EN. LANGUAGE(YY)
      IDENTIFICATION DIVISION.             |JP|  is ignored because it
      PROGRAM-ID.  COMPILE2.               |:|  is superseded by (JP).
            .  .  .                        |:|  (!!) is not alpha-
      END PROGRAM  COMPILE2.               |:|  numeric and is
      IDENTIFICATION DIVISION.             |:|  discarded.
      PROGRAM-ID.  COMPILE3.               |:|
            .  .  .                        |:|
      END PROGRAM  COMPILE3.               |:|
CBL   LANGUAGE(JP),LANGUAGE(YY)            |EN|  CBL resets language
            .  .  .                        |:|  to EN. LANGUAGE(YY)
            .  .  .                        |:|  supersedes (JP) but
            .  .  .                        |:|  is nonexistent.
```

プログラム COMPILE1 の場合、コンパイラーが呼び出しオプションを走査するときには、デフォルト言語 ENGLISH (EN) が有効です。XX は存在しない言語 ID なので、診断メッセージは英大 / 小文字混合で出されます。デフォルトの EN は、コンパイラーが CBL ステートメントを走査するときにも有効なままです。CBL ステートメントの認識されないオプション APOST は、英大 / 小文字混合で診断されます。これは、CBL ステートメントが処理を完了しておらず、EN が最後の有効な言語オプションであったからです。コンパイラーが CBL オプションを処理した後は、有効となる言語は日本語 (JP) です。

プログラム COMPILE2 では、最初のプログラムが使用される前に有効な言語は英語なので、コンパイラーは、英大 / 小文字混合の CBL ステートメント・エラーを診断します。LANGUAGE オプションが 2 つ以上指定された場合は、最後に指定された有効な言語だけが使用されます。この例では、最後の有効な言語は日本語（JP）なので、CBL オプションの処理完了時に有効となる言語は日本語です。CBL または PROCESS ステートメントのオプションを日本語で診断したい場合は、COMPILE1 より前に有効な言語が日本語でなければなりません。

プログラム COMPILE3 には CBL ステートメントがなく、したがって、前のコンパイルからの有効な言語である日本語 (JP) が継承されます。

コンパイラーは、COMPILE3 をコンパイルした後は、CBL ステートメントのために、有効な言語を英語 (EN) にリセットします。CBL ステートメントの言語オプションは、最後に指定された 2 文字の英数字言語 ID である YY を解決します。YY は存在しない言語であるため、有効な言語は英語のままです。

ソース・プログラムのエラーの訂正

ソース・コード・エラーに関するメッセージは、そのエラーが発生した場所 (LINEID) を示し、メッセージのテキストは何が問題なのかを知らせます。この情報を使用して、ソース・プログラムを訂正し、再コンパイルすることができます。

エラーを訂正するとしても、すべてのエラーを修正する必要はありません。警告レベルまたは通知レベルのメッセージは、プログラムの中に残っていても支障はなく、それらのエラーを除去するために、再コーディングやコンパイルを行う必要はありません。ただし、重大レベルやエラー・レベルのエラーは、プログラム障害の可能性が大きいため、訂正しなければなりません。

回復不能レベルのエラーは、きわめて重大なエラーです。低い方の 4 つのレベルのエラーとは対照的に、U レベルのエラーは、ソース・プログラムの間違いの結果として生じたものではない場合があります。コンパイラー自体あるいはオペレーティング・システム中の欠陥から生じる可能性があります。この場合は、コンパイラーが強制的に早期終了させられ、完全なオブジェクト・コードとリストを作成しないため、問題を解決するしかありません。多くの S レベルの構文エラーを持つプログラムに関するメッセージを受け取った場合には、これらのエラーを訂正し、プログラムを再度コンパイルしてください。また、コンパイル・ジョブに変更を加えることによって、ジョブ・セットアップ・ケース (データ・セット定義の欠落やコンパイラー処理用のストレージの不足など) を解決することが可能です。コンパイル・ジョブ・セットアップが正しく、S レベルの構文エラーを訂正した場合は、IBM に連絡して他の U レベル・エラーの調査を要求してください。

ソース・プログラムのエラーを訂正した後で、プログラムを再コンパイルしてください。この 2 回目のコンパイルが成功した場合は、リンク・エディット・ステップに進んでください。コンパイラーによってまだ問題が検出される場合は、通知メッセージだけが戻されるようになるまで、上記の手順を繰り返さなければなりません。

関連タスク
『コンパイル・エラー・メッセージのリストの生成』

関連参照
『コンパイラー検出エラーに関するメッセージおよびリスト』

コンパイル・エラー・メッセージのリストの生成

コンパイラー診断メッセージとその説明の完全なリストを生成することができます。これを行うには、次のように、PROGRAM-ID 段落で指定された ERRMSG のプログラム名を指定してプログラムをコンパイルします。

```
Identification Division.
Program-ID. ErrMsg.
```

プログラムの残りの部分は省略することができます。

コンパイラー検出エラーに関するメッセージおよびリスト

コンパイラーはソース・プログラムを処理するときに、COBOL 言語を調べてエラーがないかどうかを検査します。エラーが見つかるたびに、コンパイラーはメッセージを出します。これらのメッセージは、コンパイラー・リスト内では (FLAG オプションに従って) 照合されます。

リスト内の各メッセージは、以下の情報を与えます。

- エラーの性質
- エラーを検出したコンパイラー・フェーズ

- エラーの重大度レベル

可能な限り、メッセージはエラーを訂正するための具体的な指示を与えます。

コンパイラー・オプション、CBL および PROCESS ステートメント、あるいは BASIS、COPY、および REPLACE ステートメントの処理中に検出されたエラーに関するメッセージは、リストの上部近くに表示されます。

プログラムの中で検出されたコンパイル・エラーに関するメッセージは、プログラムごとにリストの終わり近くに表示されます。

コンパイル中に検出されたすべてのエラーの要約は、リストの下部に表示されます。

関連タスク
293 ページの『ソース・プログラムのエラーの訂正』
294 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照
『コンパイル・エラー・メッセージの形式』
296 ページの『コンパイル・エラー・メッセージの重大度コード』
328 ページの『FLAG』

コンパイル・エラー・メッセージの形式

コンパイラーが出すメッセージはそれぞれ、次のような形式になっています。

nnnnnn IGY*ppxxxx-l message-text*

nnnnnn
コンパイラーが処理している最後の行のソース・ステートメントの番号。ソース・ステートメント番号は、プログラムのソース印刷出力にリストされます。コンパイル時に NUMBER オプションを指定すると、それらは元のソース・プログラム番号になります。NONUMBER を指定すると、番号はコンパイラーによって生成された番号になります。

IGY このメッセージが COBOL コンパイラーから出されたものであることを識別する接頭部。

pp コンパイラーのどのフェーズまたはサブフェーズでエラーが発見されたかを識別する 2 文字。アプリケーション・プログラマーはこの情報を無視することができます。コンパイラー・エラーの疑いがあると診断した場合は、IBM にサポートを依頼してください。

xxxx エラー・メッセージを識別する 4 桁の数字。

l エラーの重大度レベルを示す文字 (I、W、E、S、または U)。

message-text
メッセージ・テキスト。エラー・メッセージの場合は、エラーの原因となった条件の簡単な説明。

ヒント: FLAG オプションを使用してメッセージを抑止した場合は、プログラム内にさらにエラーがある可能性があることを認識していてください。

関連参照
『コンパイル・エラー・メッセージの重大度コード』
328 ページの『FLAG』

コンパイル・エラー・メッセージの重大度コード

コンパイラーが検出できるエラーの重大度は、次の 5 つのカテゴリーに分けられます。

メッセージのレベル	目的
通知 (I) (戻りコード=0)	通知にすぎません。アクションを取る必要はありません。プログラムは正しく実行されます。
警告 (W) (戻りコード=4)	エラーの可能性があることを示します。プログラムはおそらく、書かれたとおりに正しく実行されます。
エラー (E) (戻りコード=8)	明確にエラーである条件があることを意味します。コンパイラーはエラーの訂正を試みましたが、プログラムの実行結果は予期したものではない可能性があります。エラーを訂正しなければなりません。
重大 (S) (戻りコード=12)	重大なエラーを示す条件があることを意味します。コンパイラーはエラーを訂正できませんでした。プログラムは正しく実行されず、また、実行を試みてはなりません。オブジェクト・コードは作成されない可能性があります。
回復不能 (U) (戻りコード=16)	コンパイルが終了するほどの重大なエラー条件があることを示します。

以下の例では、メッセージが出される原因となったステートメントの部分を引用符で囲んであります。

行 ID	メッセージ・コード	メッセージ・テキスト
2	IGYDS0009-E	“PROGRAM” を、領域 “A” から書き始めてはいけません。領域 “B” で検出されたものとして処理されました。
2	IGYDS1089-S	“PROGRAM” は無効でした。走査は次の領域 “A” 項目、レベル番号、または次の文節の先頭から再開されました。
2	IGYDS0017-E	“ID” は領域 “A” から書き始めなければなりません。領域 “A” で検出されたものとして処理されました。
2	IGYDS1003-E	“PROGRAM-ID” 段落が見つかりませんでした。プログラム名 “CBLPGM01” が想定されました。
2	IGYSC1082-E	ピリオドを指定する必要がありました。“ID” の前にピリオドが想定されました。
2	IGYDS1102-E	“DIVISION” が指定されているはずでしたが、“ALONGPRO” が見つかりました。“ALONGPRO” の前に、“DIVISION” が想定されました。
2	IGYDS1082-E	ピリオドを指定する必要がありました。“ALONGPRO” の前にピリオドが想定されました。
2	IGYDS1089-S	“ALONGPRO” は無効でした。走査は次の領域 “A” 項目、レベル番号、または次の文節の先頭から再開されました。

行 ID	メッセージ・コード	メッセージ・テキスト
2	IGYDS1003-E	“PROGRAM-ID” 段落が見つかりませんでした。プログラム名 “CBLPGM02” が想定されました。
3	IGYPS0017-E	“PROCEDURE” は領域 “A” から書き始めなければなりません。領域 “A” で検出されたものとして処理されました。
34	IGYSC0137-E	プログラム名 “ALONGPRO” は、オープンされているプログラムのいずれの名前にも一致しませんでした。プログラム “CBLPGM02” を終了させるために、“END PROGRAM” マーカーが想定されました。
34	IGYSC0136-E	ソース・ファイルの終りに達したか、または含まれるプログラムの “END PROGRAM” マーカーが見つかりましたが、プログラム “CBLPGM01” の “END PROGRAM” マーカーが見つかりませんでした。

第 16 章 UNIX のもとでのコンパイル

Enterprise COBOL プログラムを UNIX のもとでコンパイルするには、cob2 コマンドを使用します。 UNIX のもとでは、z/OS のもとでコンパイルする任意の COBOL ソース・プログラムをコンパイルすることができます。UNIX のもとで COBOL コンパイラーによって生成されたオブジェクト・コードは、z/OS のもとで実行することができます。

コンパイル・ステップの一部として、コンパイルに必要なファイルを定義し、プログラムおよび求める出力に必要なコンパイラー・オプションを指定する必要があります。

コンパイラーの主な仕事は、COBOL プログラムを、コンピューターで処理できる言語 (オブジェクト・コード) に変換することです。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムをデバッグして調整するのに役立つ補足情報を提供します。コンパイルを制御するには、コンパイラー指示ステートメントおよびコンパイラー・オプションを使用してください。

関連タスク
『UNIX のもとでの環境変数の設定』
300 ページの『UNIX のもとでのコンパイラー・オプションの指定』
301 ページの『cob2 コマンドを使用したコンパイルおよびリンク』
306 ページの『スクリプトを使用したコンパイル』

関連参照
280 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』
287 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

UNIX のもとでの環境変数の設定

環境変数 とは、文字ストリングに関連付けられる名前です。環境変数を使用して、プログラム (コンパイラーを含む) が必要とする値を設定します。コンパイラーに必要な環境変数は、export シェル・コマンドを使用して設定します。たとえば、SYSLIB 変数を設定するには、シェルまたはスクリプト・ファイルから export コマンドを出してください。

```
export SYSLIB=/u/mystuff/copybooks
```

環境変数に割り当てる値には、他の環境変数または変数自体を含めることができます。これらの変数の値が適用されるのは、export コマンドを出すシェルからコンパイルする場合だけです。環境変数を設定しなかった場合は、デフォルト値が適用されるか、またはその変数は定義されません。環境変数名は大文字でなければなりません。

コンパイラーが使用するために設定できる環境変数は、次のとおりです。

© Copyright IBM Corp. 1991, 2001

COBOPT

コンパイラー・オプションをブランクまたはコンマで区切って指定します。サブオプションはコンマで区切ってください。 変数値の始まりまたは終わりにあるブランクは無視されます。オプションのリストに、ブランクまたは UNIX シェルにとって意味のある文字が含まれている場合には、リストを引用符で囲んでください。たとえば、次のようになります。

```
export COBOPT="TRUNC(OPT) XREF"
```

SYSLIB

COPY ステートメントで明示的なライブラリー名を指定しなかったときは、COBOL コピーブックの検索の際に使用するディレクトリーへのパスを指定します。複数のパスはコロンで区切ってください。パスは、export コマンドの最初のパスから最後のパスへと順番に評価されます。同じ名前の複数のファイルを持つ変数を設定した場合は、そのファイルの最初に見つかったコピーが使用されます。

COPY ステートメントに明示的なライブラリー名が指定されていない場合、コンパイラーは、以下の順序でコピーブックを検索します。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

library-name

COPY ステートメントで明示的なライブラリー名を指定する場合の、コピー元のディレクトリー・パスを指定します。この環境変数名は、プログラム内の *library-name* と同じです。それぞれのライブラリーごとに環境変数を設定しなければなりません。さもないと、エラーになります。環境変数名 *library-name* は大文字でなければなりません。

text-name

テキストのコピー元のファイルの名前を指定します。この環境変数名は、プログラム内の *text-name* と同じです。環境変数名 *text-name* は大文字でなければなりません。

関連タスク

『UNIX のもとでのコンパイラー・オプションの指定』
301 ページの『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照

360 ページの『コンパイラー指示ステートメント』
311 ページの『第 18 章 コンパイラー・オプション』
COPY ステートメント (*Enterprise COBOL 言語解説書*)

UNIX のもとでのコンパイラー・オプションの指定

コンパイラーは、デフォルトのコンパイラー・オプションを使用してインストールされ、セットアップされます。コンパイラーのインストール時に、設置場所のシステム・プログラマーは、たとえば、パフォーマンスの向上やある程度の標準の維持を確実に行うために、コンパイラー・オプションの設定を修正することができます。インストール先で固定として設定したコンパイラー・オプションはどれもオー

バーライドすることはできません。固定されていないオプションについては、以下のいずれかの方法でコンパイラー・オプションを指定して、デフォルト設定をオーバーライドすることができます。

1. COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
2. cob2 コマンドの -q オプションを指定する。
3. COBOPT 環境変数を設定する。

コンパイラーは、上記の優先順位でオプションを認識します。

この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定された場合に有効となるオプションも決定します。cob2 コマンドを使用してコンパイルするときは、コンパイラー・オプションは、以下の優先順位 (最高位から最下位へ) の順序で認識されます。

1. オーバーライド不能として固定されているインストール先デフォルト。
2. バッチ・コンパイルの最初のプログラムで有効にされた BUFSIZE、LIB、SQL、OUTDD、および SIZE オプションの値。
3. COBOL ソース・プログラム内の PROCESS または CBL ステートメントで指定された値。
4. cob2 コマンドの -q オプション・ストリングで指定された値。
5. COBOPT 環境変数で指定された値。
6. 固定されていないインストール先デフォルト。

関連タスク
286 ページの『PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定』
299 ページの『UNIX のもとでの環境変数の設定』
『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照
313 ページの『矛盾するコンパイラー・オプション』
311 ページの『第 18 章 コンパイラー・オプション』

cob2 コマンドを使用したコンパイルおよびリンク

UNIX シェルから COBOL プログラムのコンパイルおよびリンクを行うには、cob2 コマンドを使用します。オプションおよび入力ファイル名は (オプションと名前をスペースで区切って) どのような順序で指定しても構いません。指定したオプションは、コマンド行のすべてのファイルに適用されます。

複数のファイルをコンパイルする (バッチ・コンパイル) には、複数のソース・ファイル名を指定してください。

UNIX の場合、COBOL プログラムをコンパイルするには、RENT オプションが必要になります。cob2 コマンドは、COBOL コンパイラー・オプション RENT および TERM を自動的に組み込みます。

cob2 コマンドは、標準の MVS 探索順序で見つかった COBOL コンパイラーを呼び出します。COBOL コンパイラーが LNKLST に設定されていない場合や、複数の

レベルの IBM COBOL コンパイラーがシステムにインストールされている場合は、使用するコンパイラーの PDS を STEPLIB 環境変数で指定することができます。たとえば、次のようになります。

```
export STEPLIB=IGY.V3R1M0.SIGYCOMP
```

cob2 コマンドは、リンク・ステップには UNIX シェル・コマンド c89 を暗黙的に使用します。c89 は、リンカー (DFSMS プログラム管理バインダー) へのシェル・インターフェースです。

入出力の定義

コンパイラー入出力のデフォルトの位置は、現行ディレクトリーです。

.cbl という拡張子を持つファイルだけがコンパイラーに渡されます。cob2 は、その他のファイルをすべてリンカーに渡します。

リンカーによって、最初のメインプログラムから実行が開始されます。

COBOL ソース・プログラム *file*.cbl のコンパイルから要求したリスト出力は、*file*.lst に書き込まれます。リンカーから要求したリスト出力は、stdout に書き込まれます。

DLL の作成

DLL を作成するには、cob2 オプション -bdll を指定しなければなりません。DLL を作成する場合は、COBOL コンパイラー・オプション DLL、EXPORTALL、および RENT が必要であり、-bdll cob2 オプションを指定すると、これらのオプションが自動的に組み込まれるようになります。たとえば、次のようになります。

```
cob2 -o mydll -bdll mysub.cbl
```

cob2 -bdll を指定すると、リンク・ステップによって DLL 定義サイド・ファイルが作成されます。このファイルには、DLL によってエクスポートされたそれぞれの名前に対応する IMPORT 制御ステートメントが入っています。DLL 定義サイド・ファイルの名前は、出力ファイル名に基づきます。出力名に拡張子が付いている場合は、その拡張子を “x” で置き換えて、サイド・ファイル名が作られます。たとえば、出力ファイル名が foo.dll であれば、サイド・ファイル名は foo.x になります。

後で、DLL を呼び出すモジュールを作成するときに DLL 定義サイド・ファイルを使用する場合は、リンク用に指定する他のオブジェクト・ファイル (*file*.o) と一緒にサイド・ファイルを指定します。たとえば、次のようになります。

```
cob2 -o myappl -qdll myappl.cbl mydll.x
```

303 ページの『例: UNIX のもとでコンパイルするための cob2 の使い方』

関連タスク

470 ページの『DLL を作成するためのプログラムのコンパイル』
言語環境プログラム プログラミング・ガイド
UNIX システム・サービス ユーザーズ・ガイド

関連参照

303 ページの『cob2』

305 ページの『cob2 入出力ファイル』
280 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』
287 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』
UNIX システム・サービス コマンド解説書

例: UNIX のもとでコンパイルするための cob2 の使い方

次の例では、cob2 の使用法を示します。

- 1 つのファイル (ここでは、alpha.cbl という名前のファイル) をコンパイルするには、次のように入力します。

  ```
  cob2 -c alpha.cbl
  ```

  コンパイルされたファイルには、alpha.o という名前が付けられます。
- 2 つのファイル (ここでは、alpha.cbl と beta.cbl) をコンパイルするには、次のように入力します。

  ```
  cob2 -c alpha.cbl beta.cbl
  ```

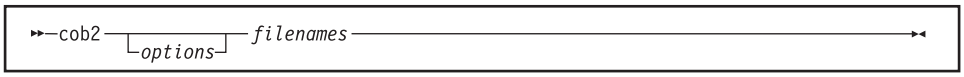
  コンパイルされたファイルには、alpha.o および beta.o という名前が付けられます。
- 2 つのファイルをリンクするには、-c オプションを指定せずにそれらのファイルをコンパイルします。たとえば、alpha.cbl および beta.cbl をコンパイルしてリンクし、gamma を生成するには、次のように入力します。

  ```
  cob2 alpha.cbl beta.cbl -o gamma
  ```

  このコマンドは、alpha.o と beta.o を作成し、その後に alpha.o、beta.o、および COBOL ライブラリーをリンクします。リンク・ステップが成功すると、gamma という名前の実行可能プログラムが作成されます。
- LIST および NODATA オプションを使用して alpha.cbl をコンパイルするには、次のように入力します。

  ```
  cob2 -qlist,noadata alpha.cbl
  ```

cob2



cob2 は、大文字で表してはなりません。

cob2 のオプションは、次のとおりです。

-b*xxx*

ストリング *xxx* をパラメーターとしてリンカーに渡します。*xxx* は、コンマで区切られた、*name=value* 形式のリンカー・オプションのリストです。名前と値の両方とも省略せずにつづらなければなりません。大文字小文字の区別はありません。-b と *xxx* の間にスペースを入れてはなりません。

オプションの値を指定しなかった場合は、デフォルト値として YES が使用されます。ただし、以下のオプションは例外で、これらは指定されたデフォルト値を持っています。

- LIST=NOIMPORT
- ALIASES=ALL
- COMPAT=CURRENT
- DYNAM=DLL

特殊値となるのは、*xxx* が dll (実行可能モジュールが DLL であることを指定する) の場合です。この値はリンカーに渡されません。

-c プログラムをコンパイルしますが、リンクしません。

-comprc_ok=*n*

コンパイラーからの戻りコードに基づいて cob2 動作を制御します。戻りコードが *n* 以下であれば、cob2 は継続してリンク・ステップに進むか、またはコンパイルのみの場合には、ゼロの戻りコードで終了します。コンパイラーから戻された戻りコードが *n* より大きい場合は、cob2 は同じ戻りコードで終了します。リンク・ステップの cob2 によって c89 コマンドが暗黙的に呼び出された場合は、c89 コマンドからの終了値が、cob2 コマンドからの戻りコードとして使用されます。

デフォルトは -comprc_ok=4 です。

-e *xxx*

モジュールの入り口点として使用するプログラムの名前を指定します。-e を指定しなかった場合、デフォルト入り口点は、cob2 コマンド呼び出しでファイル名として指定された最初のプログラム (*file*.cbl) またはオブジェクト・ファイル (*file*.o) になります。

-g デバッグに備えてプログラムを準備します。サブオプションなしで TEST オプションを指定するのと等価です。

-I*xxx*

library-name が指定されていないコピーブックの検索に使用するディレクトリーへのパス *xxx* を追加します。

複数のパスを指定するには、複数の -I オプションを使用するか、単一の -I オプション値の中に複数のパス名をコロンで区切って指定してください。

COPY ステートメントに明示的なライブラリー名が指定されていない場合、コンパイラーは、以下の順序でコピーブックを検索します。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

COPY ステートメントを使用する場合には、LIB コンパイラー・オプションを必ず有効にしなければなりません。

-L *xxx*

-l オペランドで指定されたアーカイブ・ライブラリーを探索するために使用するディレクトリー・パスを指定します。

-l *xxx*

リンカーのためのアーカイブ・ライブラリーの名前を指定します。cob2 コマンドは、lib*xxx*.a という名前を、-L オプションで指定されたディレクトリーで探索し、そのあと通常の探索順序で探索します (これは、大文字の I (アイ) ではなく、小文字の l (エル) です)。

-o *xxx*

オブジェクト・モジュール *xxx* を指定します。-o オプションが使用されない場合、オブジェクト・モジュールの名前は a.out になります。

-q*xxx*

xxx をコンパイラーに渡します (*xxx* は、ブランクまたはコンマで区切られたコンパイラー・オプションのリストです)。

括弧がオプションまたはサブオプションの一部である場合、またはブランクを使用してオプションを区切る場合は、*xxx* を引用符で囲んでください。-q と *xxx* の間にスペースを入れないでください。

-v コンパイル・ステップおよびリンク・ステップで cob2 によって出される生成済みコマンド (渡されるオプションを含む) を表示し、それらを実行します。以下に、出力例を示します。

```
cob2 -v -o mini -qssrange mini.cbl
compiler: ATTCRCTL PARM=RENT,TERM,SSRANGE /u/userid/cobol/mini.cbl
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0  in progress ...
End of compilation 1,  program mini,  no statements flagged.
linker: /bin/c89 -o mini -e // mini.o
```

-# コンパイル・ステップとリンク・ステップを表示しますが、それらを実行しません。

関連タスク

301 ページの『cob2 コマンドを使用したコンパイルおよびリンク』
299 ページの『UNIX のもとでの環境変数の設定』

cob2 入出力ファイル

cob2 コマンドを使用するときは、入力ファイル名として以下のファイルを指定することができます。

ファイル名	説明	コメント
file.cbl	コンパイルおよびリンクされる COBOL ソース・ファイル	cob2 オプション -c を指定した場合は、リンクされません。
file.a	アーカイブ・ファイル	リンク・エディット・フェーズで使用できるように、ar コマンドによって作成されます。
file.o	リンク・エディットされるオブジェクト・ファイル	COBOL コンパイラー、C/C++ コンパイラー、またはアセンブラーによって作成することができます。
file.x	DLL 定義サイド・ファイル	ダイナミック・リンク・ライブラリー (DLL) を参照するアプリケーションのリンク・エディット・フェーズで使用されます。

cob2 コマンドを使用すると、以下のファイルが現行ディレクトリーに作成されます。

ファイル名	説明	コメント
ファイル	実行可能モジュールまたは DLL	cob2 オプション -o *file* を指定した場合に、リンカーによって作成されます。
a.out	実行可能モジュールまたは DLL	cob2 オプション -o を指定しなかった場合に、リンカーによって作成されます。
file.adt	入力 COBOL ソース・プログラム *file*.cbl に対応する関連データ (ADATA) ファイル	コンパイラー・オプション ADATA を指定した場合に、コンパイラーによって作成されます。
file.dbg	入力 COBOL ソース・プログラム *file*.cbl に対応する、デバッグ・ツール用の記号情報テーブル	コンパイラー・オプション TEST(*hook*,SYM,SEPARATE) を指定した場合に、コンパイラーによって作成されます。
file.lst	入力 COBOL ソース・プログラム *file*.cbl に対応するリスト・ファイル	コンパイラーによって作成されます。
file.o	入力 COBOL ソース・プログラム *file*.cbl に対応するオブジェクト・ファイル	コンパイラーによって作成されます。
file.x	DLL 定義サイド・ファイル	*file*.dll という名前の DLL を作成する場合に、cob2 リンク・フェーズで作成されます。
class.java	Java クラス定義 (ソース)	クラス定義をコンパイルしたときに作成されます。

関連タスク

301 ページの『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照

315 ページの『ADATA』
349 ページの『TEST』
UNIX システム・サービス コマンド解説書

スクリプトを使用したコンパイル

シェル・スクリプトを使って cob2 タスクを自動化するためには、無効な構文をシェルが cob2 に渡さないようにするため、以下の構文を使用してください。

- コンパイラー・サブオプションを指定するには、左括弧と右括弧ではなく、それぞれ、等号とコロンを使用してください。たとえば、-qOPT(FULL),XREF ではなく、-qOPT=FULL:,XREF と指定します。
- コンパイラー・オプションのサブオプションを区切るのにアポストロフィが必要な場合は、アポストロフィではなく下線を使用してください。
- オプション・ストリングの中でブランクを使用しないでください。

第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行

INVOKE ステートメントやクラス定義などのオブジェクト指向 (OO) 構文を含む COBOL プログラムや、Java のサービスを使用する COBOL プログラムをコンパイルするには、RENT、DLL、THREAD、および DBCS の各オプションを使用する必要があります (RENT コンパイラー・オプションと DBCS コンパイラー・オプションはデフォルトです)。各クラス定義は、それぞれ個別にコンパイルしてください。

オブジェクト指向または Java 対応 COBOL プログラムを実行するための準備として、次のことを行います。

1. COPY ステートメントを使用してプログラムに JNI.cpy ファイルを組み込む場合は、cobol ディレクトリーの include サブディレクトリー (通常は /usr/lpp/cobol/include) を、コンパイル時に使用されるコピーブックの検索順序に指定します。このためには、cob2 コマンドの -I オプションを使用するか、SYSLIB 環境変数を設定します。
2. オブジェクト・ファイルを次の 2 つのサイド・デックとリンクさせて、実行可能モジュールを作成します。
   - libjvm.x (IBM Developer Kit for OS/390, Java 2 Technology Edition で提供)
   - igzcjava.x (COBOL で提供)

クラス定義が含まれている COBOL ソース・ファイルをコンパイルすると、次の 2 つの出力ファイルが生成されます。

- プログラム用のオブジェクト・ファイル。
- COBOL クラス定義に対応する Java クラス定義が含まれている Java ソース・プログラム。この生成済みの Java クラス定義は、決して編集しないでください。COBOL プログラムを変更する場合には、更新された COBOL プログラムを再コンパイルして、オブジェクト・ファイルと Java クラス定義の両方を再生成しなければなりません。

実行に向けてクラス定義を準備するために、以下の手順に従って、この 2 つのファイルを処理する必要があります。

1. オブジェクト・ファイルを次の 2 つの DLL サイド・デックとリンクさせて、実行可能 DLL モジュールを作成します。
   - libjvm.x (IBM Developer Kit for OS/390, Java 2 Technology Edition で提供)
   - igzcjava.x (COBOL で提供)

   結果として生成される DLL モジュールの名前には、lib*Classname*.so (*Classname* はこのクラスの外部クラス名) を使用する必要があります。
2. 生成された Java ソースを Java コンパイラーによりコンパイルして、.class ファイルを生成します。

igzcjava.x DLL サイド・デックは、HFS の cobol ディレクトリーの lib サブディレクトリーに置かれます。通常、完全パス名は次のようになります。

© Copyright IBM Corp. 1991, 2001

/usr/lpp/cobol/lib/igzcjava.x

この DLL サイド・デックは、言語環境プログラムの一部である SCEELIB PDS のメンバー IGZCJAVA としても使用可能です。

UNIX のもとでのオブジェクト指向アプリケーションの準備

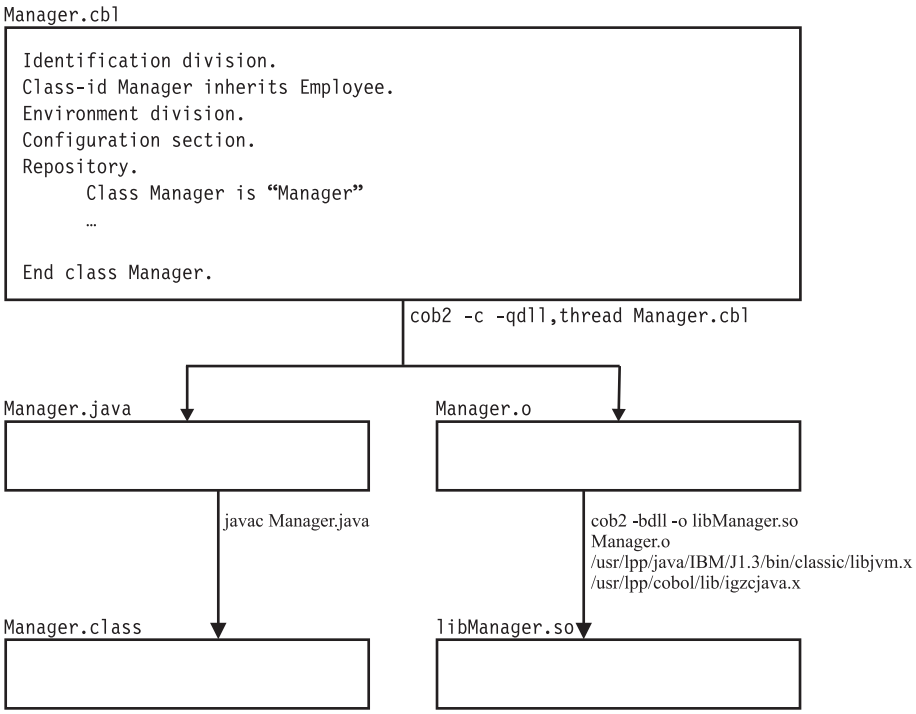
UNIX シェルで開発する場合、COBOL アプリケーションのコンパイルとリンクには cob2 コマンドを使用し、Java クラス定義のコンパイルには javac コマンドを使用します。

各 COBOL クラス定義をリンクして、HFS 内の独立した DLL モジュールを作成します。

Classname に対応するクラス定義が格納されている COBOL ソース・ファイル *Classname*.cbl の場合は、次のコマンドを使用します。

コマンド	入力	出力
cob2 -c -qdll,thread *Classname*.cbl	*Classname*.cbl	*Classname*.o *Classname*.java
cob2 -bdll -o lib*Classname*.so *Classname*.o /usr/lpp/java/IBM/J1.3/bin/classic/libjvm.x /usr/lpp/cobol/lib/igzcjava.x	*Classname*.o	lib*Classname*.so
javac *Classname*.java	*Classname*.java	*Classname*.class

次の図は、クラス定義のコンパイルに関係するコマンドとファイルを示しています。



DLL モジュールの名前には、ここに示すように、lib*Classname*.so (*Classname* はこのクラスの外部クラス名) というファイル名を使用する必要があります。このクラスがパッケージの一部であり、外部クラス名にピリオド (.) が使用されている場合は、DLL モジュール名ではピリオドを下線に変更する必要があります。たとえば、Account クラスが com.acme パッケージの一部である場合、このクラスの外部クラス名 (クラスの REPOSITORY 段落記入項目に定義されている) は com.acme.Account であり、このクラスの DLL モジュール名は libcom_acme_Account.so でなければなりません。

cob2 コマンドと javac コマンドが正常に実行されると、プログラムの実行可能コンポーネントである、実行可能 DLL モジュール lib*Classname*.so と class ファイル *Classname*.class が生成されます。

これらのコマンドによって生成されるファイルは、すべて現行作業ディレクトリーに置かれます。

UNIX 環境外でのオブジェクト指向プログラムのコンパイル

TSO またはバッチ JCL を使用してオブジェクト指向プログラムをコンパイルした場合、通常、生成されたオブジェクト・ファイルは SYSLIN DD 名または SYSPUNCH DD 名で識別されるデータ・セットに書き込まれます。生成された Java ソース・ファイルは、次の例に示すように、SYSJAVA DD 名に書き込まれます。

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU,
// FILEDATA=TEXT
```

オブジェクト指向プログラムの実行

オブジェクト指向 COBOL アプリケーションは、UNIX アプリケーションとして実行する必要があります。COBOL クラスのための DLL が置かれているディレクトリーを LIBPATH 環境変数で指定します。さらに、それらの COBOL クラスと関連付けられた Java .class ファイルが置かれているディレクトリーを CLASSPATH 環境変数で指定します。

COBOL と Java の混合アプリケーションのメインプログラムが COBOL プログラムである場合は、通常どおりに、コマンド・プロンプトでプログラム名を指定してアプリケーションを実行します。混合アプリケーションが Java クラスの main メソッドで始動される場合は、java コマンドを使用し、main メソッドが保管されているクラスの名前を指定することにより、アプリケーションを実行します。

関連タスク

299 ページの『第 16 章 UNIX のもとでのコンパイル』
427 ページの『第 23 章 UNIX のもとでの COBOL プログラムの実行』

関連参照

303 ページの『cob2』
323 ページの『DBCS』

324 ページの『DLL』
343 ページの『RENT』
352 ページの『THREAD』

第 18 章 コンパイラー・オプション

コンパイルは、次の 2 つの方法で指示および制御することができます。

- コンパイラー・オプションを使用する。
- コンパイラー指示ステートメント (コンパイル指示) を使用する。

コンパイラー・オプションは、次の表にリストされているプログラムの局面に影響を与えます。 各オプションに結び付けられている情報は、そのオプションを指定するための構文を与えるもので、オプション、そのパラメーター、および他のパラメーターとの相互作用を説明しています。

プログラムの局面	コンパイラー・オプション	デフォルト	省略形
ソース言語	316 ページの『ARITH』	ARITH(COMPAT)	AR(C)、AR(E)
	343 ページの『QUOTE/APOST』	QUOTE	Q|APOST
	317 ページの『CICS』	NOCICS	なし
	320 ページの『CURRENCY』	NOCURRENCY	CURR|NOCURR
	318 ページの『CODEPAGE』	CODEPAGE(01140)	CP(*ccsid*)
	323 ページの『DBCS』	DBCS	なし
	332 ページの『LIB』	LIB	なし
	335 ページの『NSYMBOL』	NSYMBOL(NATIONAL)	NS(DBCS|NAT)
	336 ページの『NUMBER』	NONUMBER	NUM|NONUM
	345 ページの『SEQUENCE』	SEQUENCE	SEQ|NOSEQ
	347 ページの『SQL』	NOSQL	なし
	357 ページの『WORD』	NOWORD	WD|NOWD
日付処理	322 ページの『DATEPROC』	NODATEPROC、または DATEPROC(FLAG,NOTRIG) (DATEPROC だけが指定された場合)	DP|NODP
	330 ページの『INTDATE』	INTDATE(ANSI)	なし
	358 ページの『YEARWINDOW』	YEARWINDOW(1900)	YW
マップおよびリスト	331 ページの『LANGUAGE』	LANGUAGE(ENGLISH)	LANG(EN|UE|JA|JP)
	332 ページの『LINECOUNT』	LINECOUNT(60)	LC
	333 ページの『LIST』	NOLIST	なし
	333 ページの『MAP』	NOMAP	なし
	338 ページの『OFFSET』	NOOFFSET	OFF|NOOFF
	346 ページの『SOURCE』	SOURCE	S|NOS
	347 ページの『SPACE』	SPACE(1)	なし
	349 ページの『TERMINAL』	NOTERMINAL	TERM|NOTERM
	356 ページの『VBREF』	NOVBREF	なし
	357 ページの『XREF』	XREF	X|NOX

© Copyright IBM Corp. 1991, 2001

プログラムの局面	コンパイラー・オプション	デフォルト	省略形
オブジェクト・デックの生成	319 ページの『COMPILE』	NOCOMPILE(S)	C|NOC
	323 ページの『DECK』	NODECK	D|NOD
	334 ページの『NAME』	NONAME、または NAME(NOALIAS) (NAME だけが指定された場合)	なし
	338 ページの『OBJECT』	OBJECT	OBJ|NOOBJ
	340 ページの『PGMNAME』	PGMNAME(COMPAT)	PGMN(CO|LU|LM)
オブジェクト・コード制御	315 ページの『ADV』	ADV	なし
	316 ページの『AWO』	NOAWO	なし
	324 ページの『DLL』	NODLL	なし
	327 ページの『EXPORTALL』	NOEXPORTALL	EXP|NOEXP
	327 ページの『FASTSRT』	NOFASTSRT	FSRT|NOFSRT
	336 ページの『NUMPROC』	NUMPROC(NOPFD)	なし
	339 ページの『OPTIMIZE』	NOOPTIMIZE	OPT|NOOPT
	340 ページの『OUTDD』	OUTDD(SYSOUT)	OUT
	353 ページの『TRUNC』	TRUNC(STD)	なし
	359 ページの『ZWB』	ZWB	なし
仮想記憶域の使用方法	317 ページの『BUFSIZE』	4096	BUF
	346 ページの『SIZE』	SIZE(MAX)	SZ
	321 ページの『DATA』	DATA(31)	なし
	326 ページの『DYNAM』	NODYNAM	DYN|NODYN
	343 ページの『RENT』	RENT	なし
	344 ページの『RMODE』	AUTO	なし
デバッグと診断	324 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR, NODTR
	325 ページの『DUMP』	NODUMP	DU|NODU
	328 ページの『FLAG』	FLAG(I,I)	F|NOF
	329 ページの『FLAGSTD』	NOFLAGSTD	なし
	349 ページの『TEST』	NOTEST	なし
	348 ページの『SSRANGE』	NOSSRANGE	SSR|NOSSR
Other	315 ページの『ADATA』	NOADATA	なし
	327 ページの『EXIT』	NOEXIT	EX(INX,LIBX,PRTX,ADX)
	352 ページの『THREAD』	NOTHREAD	なし

インストール先デフォルト: コンパイラーがインストールされたときにセットアップされたデフォルト・オプションは、他のオプションでオーバーライドしない限り、プログラムでは有効になります。(インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないように固定としてセットアップされます。問題がある場合は、システム管理担当者に連絡してください。) 有効なデフォルトのコンパイラー・オプションを見つけるためには、オプションを指定しないで、テスト・コンパイルを実行してください。出力リストには、インストール先によって指定されたデフォルト・オプションがリストされます。

オーバーライド不可能なオプション: インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないようにセットアップされます。問題がある場合は、システム管理担当者に連絡してください。

パフォーマンスについての考慮事項: DYNAM、FASTSRT、NUMPROC、OPTIMIZE、RENT、SSRANGE、TEST、および TRUNC コンパイラー・オプションはすべて、実行時パフォーマンスに影響を及ぼす可能性があります。

関連参照
『矛盾するコンパイラー・オプション』
360 ページの『コンパイラー指示ステートメント』
『COBOL 85 標準に準拠するオプション設定値』

関連タスク
263 ページの『第 15 章 z/OS のもとでのコンパイル』
276 ページの『TSO のもとでのコンパイル』
299 ページの『第 16 章 UNIX のもとでのコンパイル』
591 ページの『第 32 章 プログラムの調整』

COBOL 85 標準に準拠するオプション設定値

COBOL 85 標準の仕様に準拠するには、以下のコンパイラー・オプションが必要です。

ADV	NAME(ALIAS) または NAME(NOALIAS)
NOCICS	NUMPROC(NOPFD) または NUMPROC(MIG)
NODATEPROC	PGMNAME(COMPAT) または PGMNAME(LONGUPPER)
NODLL	QUOTE
DYNAM	NOTHREAD
NOEXPORTALL	TRUNC(STD)
NOFASTSRT	NOWORD
LIB	ZWB

COBOL 85 標準に準拠するには、以下のランタイム・オプションが必要です。

- AIXBLD
- CBLQDA(ON)
- TRAP(ON)

関連参照
言語環境プログラム プログラミング・リファレンス

矛盾するコンパイラー・オプション

Enterprise COBOL コンパイラーが矛盾するコンパイラー・オプションを検出するのは、次の 2 つの場合です。

- 1 つのコンパイラー・オプションの肯定形式と否定形式の両方が、優先順位の階層の中で同じレベルで指定されている (たとえば、PROCESS (または CBL) ステートメントに DECK と NODECK の両方が指定されている)。

矛盾するオプションを階層の同じレベルで指定した場合は、最後に指定したオプションの方が有効になります。

- 互いに排他的なオプションが階層の同じレベルで指定されている (指定順序は関係ありません)。

  互いに排他的なオプションを指定すると、矛盾するオプションの指定はすべて無視されます。たとえば、PROCESS ステートメントで OFFSET と LIST の両方を指定すると、指定した順序に関係なく、OFFSET が有効になり、LIST は無視されます。

  しかし、高レベルの優先順位でコーディングされたオプションは、低レベルの優先順位で指定されたオプションをオーバーライドします。たとえば、JCL ステートメントでは OFFSET をコーディングし、PROCESS ステートメントでは LIST をコーディングすると、PROCESS ステートメントでコーディングされたオプションと、PROCESS ステートメントで強制的にオンにされたオプションの優先順位の方が高いため、LIST が有効になります。

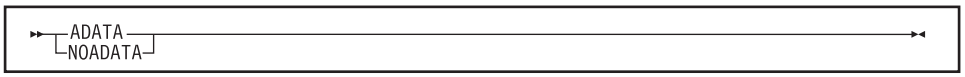
互いに排他的なコンパイラー・オプションを、以下の表にリストしています。

指定される	無視される[1]	強制的にオンにされる[1]
CICS	NOLIB DYNAM NORENT	LIB NODYNAM RENT
DLL	DYNAM NORENT	NODYNAM RENT
EXIT	DUMP	NODUMP
EXPORTALL	NODLL DYNAM NORENT	DLL NODYNAM RENT
FLAGSTD	DBCS	NODBCS
NSYMBOL(NATIONAL)	NODBCS	DBCS
OFFSET	LIST	NOLIST
SQL	NOLIB	LIB
TEST TEST(ALL) TEST(STMT) TEST(PATH) TEST(BLOCK)	NOOBJECT OPT(STD) または OPT(FULL)	OBJECT NOOPTIMIZE
THREAD	NORENT	RENT
WORD	FLAGSTD	NOFLAGSTD
1. 固定されているインストール先デフォルト・オプションと対立する場合を除きます。		

関連タスク

285 ページの『z/OS のもとでのコンパイラー・オプションの指定』
291 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』
300 ページの『UNIX のもとでのコンパイラー・オプションの指定』

ADATA



デフォルト: NOADATA

省略形: なし

ADATA は、コンパイラーに追加のコンパイル情報のレコードを収めた SYSADATA ファイルを作成させる場合に使用します。このファイルは、IBM VisualAge COBOL を使用してリモート・コンパイルを行う場合に必要になります。z/OS では、このファイルは DD 名 SYSADATA に書き込まれます。このファイルのサイズは、通常、関連するプログラムのサイズに比例します。

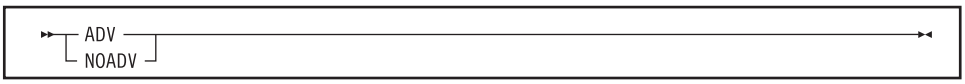
ADATA は、PROCESS (CBL) ステートメントでは指定できません。次のいずれかの方法でのみ、指定できます。

- オプション・リストを使用するコンパイラーの起動時
- JCL の PARM フィールドで
- コマンド・オプションとして
- インストール先デフォルト値として

関連参照

679 ページの『付録 G. COBOL SYSADATA ファイル内容』

ADV



デフォルト: ADV

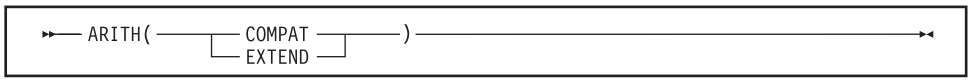
省略形: なし

ADV が意味を持つのは、ソース・コードで WRITE ... ADVANCING を使用する場合だけです。

ADV が有効になっていると、コンパイラーは、印刷制御文字を入れるための 1 バイトをレコード長に追加します。

すでに印刷制御文字のための 1 バイトを組み込むようにレコード長を調整している場合は、NOADV を使用してください。

ARITH



デフォルト: ARITH(COMPAT)

省略形: AR(C)、AR(E)

ARITH(EXTEND) を指定すると、次のようになります。

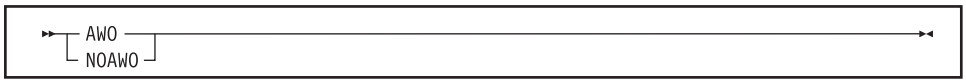
- パック 10 進数、ゾーン 10 進数、および数字編集データ項目に関して、PICTURE 文節で指定できる桁数の最大数が、18 から 31 に上がります。
- 固定小数点数値リテラルに指定できる桁数の最大数が、18 から 31 に上がります。以下の場所を含め、数値リテラルが現在許可されているところであればどこでも、長精度の数値リテラルを使用することができます。
  - PROCEDURE DIVISION ステートメントのオペランド
  - VALUE 文節 (長精度 PICTURE を含む数値データ項目に関する)
  - 条件名の値 (長精度 PICTURE を含む数値データ項目に関する)
- NUMVAL および NUMVAL-C への引き数の中で指定できる桁数の最大数が、18 から 31 に上がります。
- FACTORIAL 関数への整数引き数の最大値は、29 です。
- 算術ステートメントの中間結果は、*拡張モード* を使用します。

ARITH(COMPAT) を指定すると、次のようになります。

- パック 10 進数、ゾーン 10 進数、および数字編集データ項目に関して、PICTURE 文節で指定できる桁数の最大数は、18 です。
- 固定小数点数値リテラルに指定できる桁数の最大数は、18 です。
- NUMVAL および NUMVAL-C への引き数の中で指定できる桁数の最大数は、18 です。
- FACTORIAL 関数への整数引き数の最大値は、28 です。
- 算術ステートメントの中間結果は、*互換モード* を使用します。

関連概念
619 ページの『付録 A. 中間結果と算術精度』

AWO



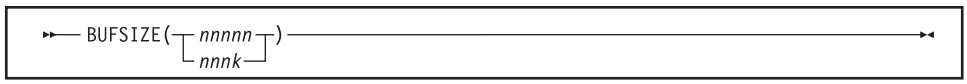
デフォルト: NOAWO

省略形: なし

AWO を指定すると、暗黙の APPLY WRITE-ONLY 文節が、プログラム内のこの文節に適格なすべてのファイルに対して活動化されます。適格にするためには、ファイルを物理順次編成にして、ブロック化可変長レコードを指定しなければなりません。

関連タスク
14 ページの『バッファーおよび装置スペースの最適化』

BUFSIZE



デフォルト: 4096

省略形: BUF

nnnnn は 10 進数で、少なくとも 256 でなければなりません。

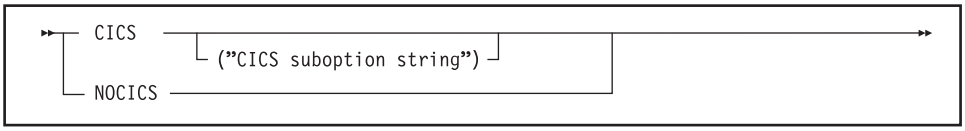
*nnn*K は、1K 単位で 10 進数を指定します。

BUFSIZE は、コンパイラー作業データ・セットごとに、ある容量の主記憶域をバッファーに割り振るために使用します (ここで、1K は 10 進数の 1024 バイト)。通常、バッファー・サイズを大きくすると、コンパイラーのパフォーマンスが向上します。

BUFSIZE と SIZE の両方を使用している場合、バッファーに割り振られる容量は、SIZE オプションによってコンパイルに使用可能にされる主記憶域の容量に含まれます。

BUFSIZE は、使用される装置のトラック容量を超えてはならず、また、データ管理サービスで許可される最大量を超えてはなりません。

CICS



デフォルト: NOCICS

省略形: なし

CICS コンパイラー・オプションを指定すると、組み込みの CICS トランスレーター機能が使用可能になり、CICS の各オプションを指定できるようになります。COBOL ソース・プログラムに EXEC CICS ステートメントや EXEC DLI ステートメントが含まれている場合、そのソース・プログラムが分離型の CICS トランスレーターで処理されていないときは、CICS オプションを指定する必要があります。

CICS オプションは、CICS プログラムをコンパイルする場合にのみ使用してください。CICS オプションを指定してコンパイルされたプログラムは、非 CICS 環境では実行することができません。

コンパイラーは、CICS Transaction Server バージョン 2 以上にアクセスする必要があります。

NOCICS オプションを指定した場合、ソース・プログラム内で検出された CICS ステートメントはすべて診断され、破棄されます。

CICS サブオプションのストリングを区切るときは、引用符またはアポストロフィを使用してください。

長いサブオプション・ストリングを、複数のサブオプション・ストリングに分割して、複数の CBL ステートメントに置くことができます。それぞれの CICS サブオプションは、指定された順に連結されます。以下に例を示します。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='CICS("string1")'
//COBOL.SYSIN DD *
       CBL CICS('string2')
       CBL CICS("string3")
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRIVER1.
       . . .
```

コンパイラーは、以下のサブオプションを組み込みの CICS トランスレーターに渡します。

```
"string1 string2 string3"
```

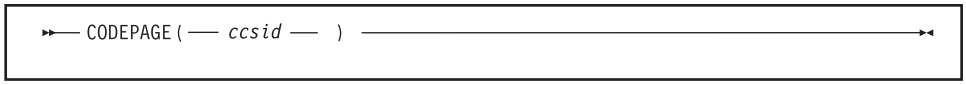
ここに示すように、連結されたストリングは 1 つのスペースで区切られます。同じ CICS オプションの複数インスタンスが見つかった場合は、各オプションで最後に指定されたものが使用されます。コンパイラーは、連結 CICS サブオプション・ストリングの長さを 4K バイトに限定しています。

関連概念
413 ページの『組み込みの CICS トランスレーター』

関連タスク
411 ページの『CICS オプションを使用したコンパイル』

関連参照
313 ページの『矛盾するコンパイラー・オプション』

CODEPAGE



デフォルト: CODEPAGE(1140)

省略形: CP(*ccsid*)

CODEPAGE オプションは、以下のエンコードに使用されるコード・ページを指定します。

- 実行時の英数字および DBCS データ項目の内容
- COBOL ソース・プログラム内の英数字リテラル、国別リテラル、および DBCS リテラル

CODEPAGE オプションは、英数字 XML 文書の構文解析に使用するデフォルトのコード・ページも指定します。

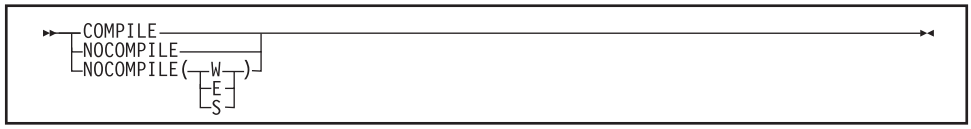
ccsid は、有効なコード化文字セット ID (CCSID) 番号を表す整数でなければなりません。

デフォルトの CCSID 1140 は、CCSID 37 (EBCDIC Latin-1、USA) と等価ですが、ユーロ記号が含まれる点が異なります。

重要: Unicode 変換サービス実装の一環として構成された変換イメージは、指定された CCSID と CCSID 1200 との間の変換をサポートしている必要があります。オブジェクト指向構文を使用するアプリケーションの場合は、指定された CCSID から CCSID 1208 への変換もサポートしなければなりません。

関連タスク
Enterprise COBOL カスタマイズ・ガイド

COMPILE



デフォルト: NOCOMPILE(S)

省略形: C|NOC

COMPILE オプションは、重大エラーがあっても完全コンパイルを強制的に行う場合に限り、使用してください。すべての診断およびオブジェクト・コードが生成されます。コンパイルの結果として重大エラーが発生した場合は、生成されたオブジェクト・コードを実行しないでください。実行した場合の結果は保証されず、異常終了する場合があります。

NOCOMPILE にサブオプションを指定しないで使用すると、構文検査を要求します (診断だけが作成され、オブジェクト・コードは生成されません)。

NOCOMPILE に W、E、または S を付けて使用すると、条件付き完全コンパイルを行います。コンパイラーが指定されたレベルのエラーを見つけると、完全コンパイル (診断およびオブジェクト・コード) は停止し、構文検査だけを継続します。

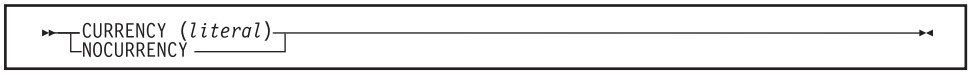
無条件 NOCOMPILE を要求すると、オブジェクト・コードが作成されないので、次のオプションは効力を持ちません。

- LIST

- SSRANGE
- OPTIMIZE
- OBJECT
- TEST

関連参照
294 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

CURRENCY



デフォルト: NOCURRENCY

デフォルトの通貨記号はドル記号 ($) です。CURRENCY オプションを使用すれば、COBOL プログラムで使用する代替のデフォルト通貨記号を指定することができます。

NOCURRENCY を指定すると、代替のデフォルト通貨記号が使用されません。

デフォルト通貨記号を変更するには、CURRENCY(*literal*) オプションを使用します。ここで、*literal* は、単一文字を表す有効な COBOL 英数字リテラル (16 進数リテラルを含む) であり、以下を指定することはできません。

- 数値 0 ～ 9
- 英大文字 A B C D E G N P R S V X Z またはその英小文字
- スペース
- 特殊文字 * + - / , .; ( ) “ = ’
- 形象定数
- ヌル終了リテラル
- DBCS リテラル
- 国別リテラル

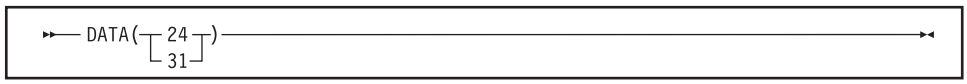
プログラムが 1 つの通貨タイプしか処理しない場合には、CURRENCY SIGN 文節の代わりに CURRENCY オプションを使用して、プログラムの PICTURE 文節で使用する通貨記号を選択できます。プログラムで複数の通貨タイプを処理する場合は、CURRENCY SIGN 文節と WITH PICTURE SYMBOL 句を併用して、異なる通貨記号タイプを指定しなければなりません。

CURRENCY オプションと CURRENCY SIGN 文節の両方をプログラムで使用した場合は、CURRENCY オプションの方が無視されます。CURRENCY SIGN 文節で指定した通貨記号を、PICTURE 文節で使用することができます。

NOCURRENCY オプションが有効なときに、CURRENCY SIGN 文節を省略すると、通貨記号の PICTURE 記号としてドル記号が使用されます。

区切り文字についての注: CURRENCY オプション・リテラルは、QUOTE|APOST コンパイラー・オプションの設定に関係なく、引用符やアポストロフィで区切ることができます。

DATA



デフォルト: DATA(31)

省略形: なし

16MB 境界より上 (DATA(31)) であるか下 (DATA(24)) であるかに関係なく、動的データ域に使用するストレージおよびその他の動的ランタイム・ストレージの位置に影響します。 再入可能プログラムの場合は、DATA(24|31) コンパイラー・オプションおよび HEAP ランタイム・オプションによって、動的データ域のストレージ (WORKING-STORAGE や FD レコード域など) を 16MB 境界より下から獲得するか、制限のないストレージから獲得するかが制御されます。(DATA オプションは LOCAL-STORAGE データの位置に影響を与えません。代わりに、STACK ランタイム・オプションが、プログラムの AMODE とともに、その位置を制御します。このオプションは、ローカルの WORKING-STORAGE データの位置に影響を及ぼします。)

ランタイム・オプション HEAP(,,BELOW) を指定すると、DATA(24|31) コンパイラー・オプションは効力を持ちません。動的データ域のストレージはすべて、16MB 境界より下から割り振られます。しかし、HEAP(,,ANYWHERE) をランタイム・オプションとして指定すると、動的データ域のストレージは、プログラムを DATA(24) コンパイラー・オプションを使用してコンパイルした場合は、16MB 境界より下から割り振られ、DATA(31) コンパイラー・オプションを使用してコンパイルした場合は、制限のないストレージから割り振られます。

31 ビット・アドレッシング・モードで実行されているプログラムで、24 ビット・アドレッシング・モードのプログラムにデータ引き数を渡す場合は、DATA(24) コンパイラー・オプションを指定します。そうすると、データは必ず呼び出されたプログラムからアドレス可能になります。

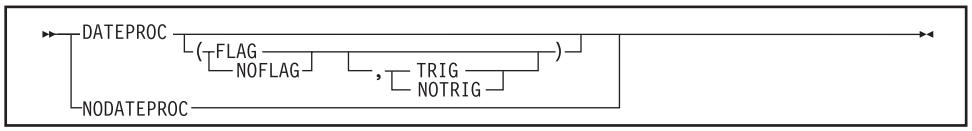
外部データおよび QSAM バッファー: DATA コンパイラー・オプションは、ストレージおよびそのアドレス可能度に影響を与える他のコンパイラー・オプションおよびランタイム・オプションと相互作用します。詳細については、以下の関連情報を参照してください。

関連概念
38 ページの『ストレージとそのアドレス可能度』

関連タスク
ランタイム・オプションの使用 (*言語環境プログラム プログラミング・ガイド*)

関連参照
157 ページの『QSAM ファイル用のバッファーの割り振り』

DATEPROC



デフォルト: NODATEPROC、または DATEPROC(FLAG,NOTRIG) (DATEPROC だけが指定された場合)

省略形: DP|NODP

DATEPROC オプションは、COBOL コンパイラーの 2000 年言語拡張を使用可能にする場合に使用します。

DATEPROC(FLAG)

DATEPROC(FLAG) を指定すると、2000 年言語拡張が使用可能になり、言語エレメントが拡張機能を使用するか、または言語エレメントが拡張機能の影響を受けるたびに、コンパイラーは診断メッセージを作成します。このメッセージは通常は通知レベルまたは警告レベルのメッセージであり、日付依存型処理に関連するステートメントを識別します。日付構造のエラーまたは矛盾の可能性を識別する追加のメッセージが生成されることもあります。

診断メッセージの作成とソース・リスト内またはソース・リストの後にそれらのメッセージが示されるかどうかは、FLAG コンパイラー・オプションの設定に左右されます。

DATEPROC(NOFLAG)

DATEPROC(NOFLAG) を指定すると、2000 年言語拡張は有効になりますが、COBOL ソースにエラーまたは矛盾がない限り、コンパイラーは関連メッセージを作成しません。

DATEPROC(TRIG)

DATEPROC(TRIG) を指定すると、2000 年言語拡張が使用可能になり、コンパイラーがウィンドウ化日付フィールドの操作に適用する自動ウィンドウ操作では、日付フィールドや、ウィンドウ化日付フィールドに保管されたり比較されたりするその他の非日付フィールドにおいて、特定のトリガー値または限界値が考慮されるようになります。これらの特殊値は、テストの対象になったり、上限値または下限値として使ったりする無効日付を表します。

パフォーマンス: DATEPROC(TRIG) オプションを使用すると、ウィンドウ化日付の比較処理が遅くなります。

DATEPROC(NOTRIG)

DATEPROC(NOTRIG) を指定すると、2000 年言語拡張が使用可能になり、コンパイラーがウィンドウ化日付の操作に適用する自動ウィンドウ操作では、オペランド内に特定のトリガー値があっても認識されません。日付の年部分の値だけが自動ウィンドウ操作の対象になります。

パフォーマンス: DATEPROC(NOTRIG) オプションは、ウィンドウ化日付フィールドにおける有効な日付値を想定するパフォーマンス・オプションです。

NODATEPROC

NODATEPROC は、このコンパイル単位に関して拡張機能を使用可能にしないことを示します。これは、日付関連プログラム構造に、次のような影響を与えます。

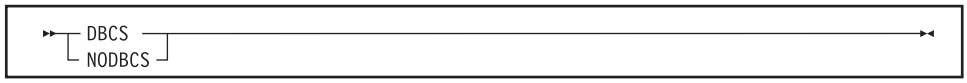
- DATE FORMAT 文節は構文検査されますが、プログラムの実行には影響しなくなります。
- DATEVAL と UNDATE 組み込み関数は無効です。すなわち、組み込み関数によって戻される値は引き数の値と同じになります。
- YEARWINDOW 組み込み関数は値 0 を戻します。

使用上の注意: FLAG|NOFLAG サブオプションおよび TRIG|NOTRIG サブオプションは、任意の順序で指定できます。どちらのサブオプションも省略した場合のデフォルトは、現行の設定になります。

関連参照

328 ページの『FLAG』

DBCS



デフォルト: DBCS

省略形: なし

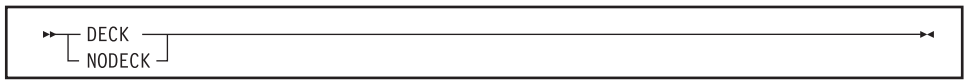
DBCS を使用すると、コンパイラーは、X'0E' (SO) および X'0F' (SI) を英数字リテラルの 2 バイト部分のシフト・コードとして認識するようになります。

DBCS を選択すると、リテラルの 2 バイト部分が構文検査されるようになり、リテラルは英数字のカテゴリーのままになります。

関連参照

313 ページの『矛盾するコンパイラー・オプション』

DECK

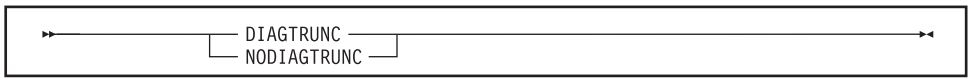


デフォルト: NODECK

省略形: D|NOD

DECK は、オブジェクト・コードを 80 桁のカード・イメージ形式で作成する場合に使用します。DECK オプションを使用する場合は、コンパイル用の JCL で SYSPUNCH を必ず定義するようにしてください。

DIAGTRUNC



デフォルト: NODIAGTRUNC

省略形: DTR、NODTR

DIAGTRUNC を使用すると、受け取り側が数値である MOVE ステートメントの場合に、受け取りデータ項目の整数桁数が送り出しデータ項目またはリテラルよりも少ないときには、コンパイラーは、重大度 4 (警告) の診断メッセージを出します。複数の受け取り側があるステートメントでは、切り捨てられる可能性があるそれぞれの受け取り側ごとにメッセージが出されます。

診断メッセージは、次のようなステートメントに関連した暗黙の移動の場合にも出されます。

- INITIALIZE
- READ ... INTO
- RELEASE ... FROM
- RETURN ... INTO
- REWRITE ... FROM
- WRITE ... FROM

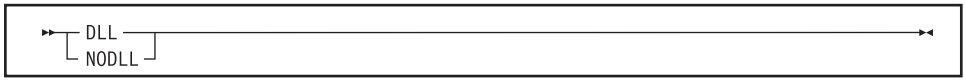
送り出しフィールドが参照変更である場合を除いて、英数字データ名またはリテラルの送り出し側から数値の受け取り側への移動についても、診断が出されます。

TRUNC(BIN) オプションを指定した場合は、COMP-5 の受け取り側についても、2 進数の受け取り側についても診断は行われません。

関連概念
46 ページの『数値データの形式』
102 ページの『参照修飾子』

関連参照
353 ページの『TRUNC』

DLL



デフォルト: NODLL

省略形: なし

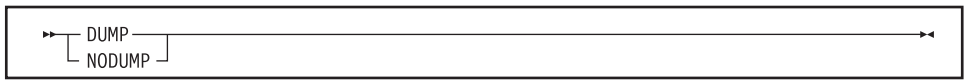
DLL は、ダイナミック・リンク・ライブラリー (DLL) サポートで使用可能なオブジェクト・モジュールを生成するようコンパイラーに指示する場合に使用します。DLL を使用可能にする必要があるのは、プログラムが DLL の一部である場合、プログラムが DLL を参照する場合、あるいはプログラムに INVOKE ステートメントやクラス定義などのオブジェクト指向構文が含まれている場合です。

DLL オプションを指定する場合は、RENT リンケージ・エディター・オプションまたはバインダー・オプションも併せて指定する必要があります。

NODLL は、DLL として使用できないオブジェクト・モジュールを生成するようにコンパイラーに指示します。

関連参照
313 ページの『矛盾するコンパイラー・オプション』

DUMP



デフォルト: NODUMP

省略形: DU|NODU

通常は不使用: DUMP オプションは、IBM 担当員から依頼があったときにのみ使用してください。

DUMP は、コンパイル時に内部コンパイラー・エラーに関するシステム・ダンプを作成する場合に使用します。

ダンプは、コンパイラーのレジスターのリストとストレージ・ダンプで構成され、主として診断を行う担当者がコンパイラーのエラーを判別するために使用するものです。

DUMP オプションを使用する場合は、コンパイル時に DD ステートメントを組み込んで、SYSABEND、SYSUDUMP、または SYSMDUMP を定義してください。

DUMP を指定すると、コンパイラーは、異常終了処理の前に診断メッセージを出しません。その代わりに、ユーザー異常終了コードは IGY*ppnnnn* で出されます。一般に、メッセージ IGY*ppnnnn* はコンパイル時のユーザー異常終了コード *nnnn* に対応しています。ただし、IGY*pp*5*nnn* メッセージと IGY*pp*1*nnn* メッセージはどちらも、ユーザー異常終了 1*nnn* を生成します。NODUMP を使用して再コンパイルすれば、そのメッセージが実際は 5*nnn* と 1*nnn* のどちらなのかを区別することができます。

次のものを含め、通常の終了処理を行いたい場合には、NODUMP を使用してください。

- コンパイル中でそれまでに作成された診断メッセージ。
- エラーの記述。

- 現在実行中のコンパイラー・フェーズの名前。
- エラー検出時に処理されていた COBOL ステートメントの行番号 (OPTIMIZE を使用してコンパイルした場合は、行番号が正しく示されないことがあります。エラーによっては、プログラムの最後の行が示される場合があります)。
- 汎用レジスターの内容。

DUMP および OPTIMIZE コンパイラー・オプションを一緒に使用すると、コンパイラーは、次の最適化プログラム・メッセージの代わりにシステム・ダンプを作成します。

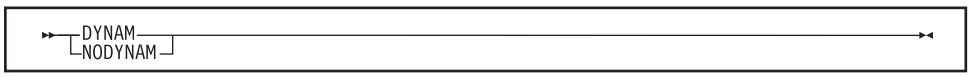
```
"IGYOP3124-W  このステートメントは、実行時にプログラム例外を引き起こす可能性があります。"
```

この状況はコンパイラー・エラーではありません。NODUMP オプションを使用すると、コンパイラーはメッセージ IGYOP3124-W を出して、処理を継続することができます。

関連タスク
異常終了コードの理解 (*言語環境プログラム デバッグのガイド*)

関連参照
313 ページの『矛盾するコンパイラー・オプション』

DYNAM



デフォルト: NODYNAM

省略形: DYN|NODYN

DYNAM を使用すると、CALL *literal* ステートメントにより呼び出された、ネストされていない、別々にコンパイルされたプログラムを実行時に動的にロードしたり (CALL の場合)、削除したり (CANCEL の場合) することができます。CALL *identifier* ステートメントを使用した場合、ターゲット・プログラムは必ず実行時にロードされるので、このオプションによる影響はありません。DYNAM コンパイラー・オプションは、CICS トランスレーターで変換されるプログラムで使用してはなりません。

COBOL プログラムがダイナミック・リンク・ライブラリー (DLL) としてリンクされているプログラムを呼び出す場合は、DYNAM オプションを使用してはなりません。代わりに、NODYNAM および DLL オプションを使用してそのプログラムをコンパイルするようにしてください。

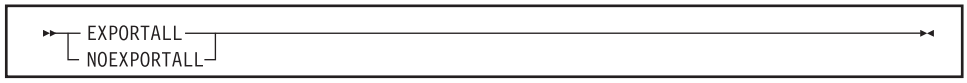
関連参照
445 ページの『静的呼び出しと動的呼び出しの両方の作成』
313 ページの『矛盾するコンパイラー・オプション』

EXIT

EXIT コンパイラー・オプションの詳細については、以下の最初の関連情報を参照してください。

関連参照
657 ページの『付録 E. EXIT コンパイラー・オプション』
313 ページの『矛盾するコンパイラー・オプション』

EXPORTALL



デフォルト: NOEXPORTALL

省略形: EXP|NOEXP

EXPORTALL を使用すると、オブジェクト・デックをリンク・エディットして DLL を形成するときに、PROGRAM-ID 名および各代替入り口点名をそれぞれのプログラム定義から自動的にエクスポートするようコンパイラーに指示することができます。

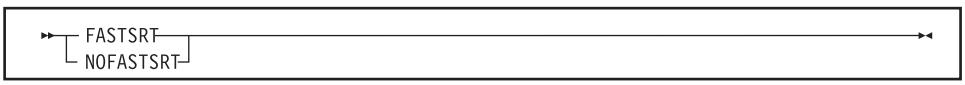
これらの記号が DLL からエクスポートされた場合、エクスポートされたプログラム名と入り口点名は、同じ DLL にリンクされたプログラムからだけでなく、アプリケーション内のルート・ロード・モジュールまたはその他の DLL ロード・モジュールのプログラムから呼び出すことができるようになります。

EXPORTALL オプションを指定する場合は、RENT リンカー・オプションも併せて指定する必要があります。

NOEXPORTALL は、記号をエクスポートしないようにコンパイラーに指示します。この場合、プログラムには、この COBOL プログラム定義と一緒に同じロード・モジュールにリンク・エディットされた他のルーチンからしかアクセスすることができません。

関連参照
313 ページの『矛盾するコンパイラー・オプション』

FASTSRT



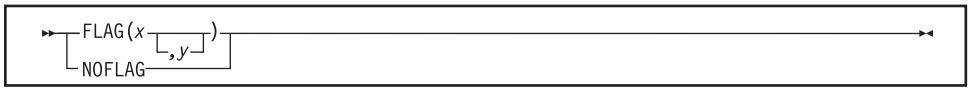
デフォルト: NOFASTSRT

省略形: FSRT|NOFSRT

FASTSRT は、IBM DFSORT またはそれと同等のものが COBOL の代わりに入出力を実行することを許可します。

関連タスク
217 ページの『FASTSRT を使ってのソートのパフォーマンスの向上』

FLAG



デフォルト: FLAG(I,I)

省略形: F|NOF

x および *y* は、I、W、E、S、U のいずれかになります。

FLAG(*x*) を使用すると、重大度レベル *x* 以上のエラーに関して診断メッセージをソース・リストの終わりに作成することができます。

FLAG(*x*,*y*) を使用すると、重大度レベル *x* 以上のエラーに関して診断メッセージをソース・リストの終わりに作成し、重大度レベル *y* 以上のエラーに関してはエラー・メッセージをソース・リストに直接組み込むことができます。*y* に指定する重大度は、*x* に指定する重大度より低い値にすることができません。FLAG(*x*,*y*) を使用する場合は、一緒に SOURCE コンパイラー・オプションも指定しなければなりません。

ソース・リスト内のエラー・メッセージは、メッセージ・コードを指す矢印の中にステートメント番号を埋め込むことによって、強調されます。次に、メッセージ・コードの後にメッセージ・テキストが続きます。たとえば、次のようになります。

```
  000413     MOVE CORR WS-DATE TO HEADER-DATE

==000413==>    IGYPS2121-S       " WS-DATE " was not defined as a data-name.  . . .
```

FLAG(*x*,*y*) を選択すると、重大度 *y* 以上のメッセージが、リスト内でそのメッセージの原因となった行の後に組み込まれるようになります (例外については、以下の組み込みメッセージに関する注を参照)。

エラーのフラグ付けを抑止する場合は、NOFLAG を使用してください。NOFLAG を使用しても、コンパイラー・オプションのエラー・メッセージは抑止されません。

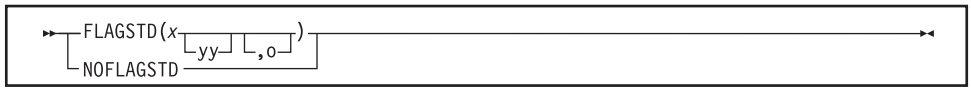
組み込みメッセージ:

- レベル U のメッセージの組み込みの指定は受け入れられますが、ソース・リスト内には何のメッセージも作成されません。レベル U メッセージを組み込みに指定するのはお勧めできません。
- FLAG オプションは、コンパイラー・オプションの処理前に作成された診断メッセージには影響しません。
- コンパイラー・オプション、CBL ステートメントおよび PROCESS ステートメント、または BASIS、COPY、および REPLACE の各ステートメントの処理中に生成された診断メッセージがソース・リストに組み込まれることはありません。このようなメッセージはすべて、コンパイラー出力の先頭に表示されます。

- *CONTROL (*CBL) ステートメントの処理中に作成されたメッセージは、ソース・リストに組み込まれません。

関連参照
294 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

FLAGSTD



デフォルト: NOFLAGSTD

x は、以下に示す、規格準拠と見なされる COBOL 85 標準のレベルまたはサブセットを表します。

M 最小サブセットからのものではない 言語エレメントに、「規格準拠外の標準」というフラグを付けます。

I 最小サブセットまたは中間サブセットからのものではない 言語エレメントに、「規格準拠外の標準」というフラグを付けます。

H 高位サブセットが使用されており、言語エレメントにはサブセットによってフラグが付けられません。さらに、IBM 拡張カテゴリーのエレメントには、「規格準拠外の標準、IBM 拡張」というフラグが付けられます。

yy は、単一文字または 2 文字の組み合わせによって、サブセットに組み込むオプショナル・モジュールを指定します。

D デバッグ・モジュール・レベル 1 のエレメントには、「規格準拠外の標準」というフラグを*付けません*。

N 分割モジュール・レベル 1 のエレメントには、「規格準拠外の標準」というフラグを*付けません*。

S 分割モジュール・レベル 2 のエレメントには、「規格準拠外の標準」というフラグを*付けません*。

S を指定すると、N が含められます (N は S のサブセットです)。

O は、廃止された言語エレメントに「廃止」のフラグを付けることを表します。

FLAGSTD を使用すると、プログラムに含まれている COBOL 85 標準エレメントに関する通知メッセージを入手することができます。フラグ付け処理には、次の項目のどれかを指定することができます。

- 連邦情報処理標準 (FIPS) COBOL の選択されたサブセット
- オプショナル・モジュールのいずれか
- 廃止された言語エレメント
- サブセットとオプショナル・モジュールの任意の組み合わせ
- サブセットと廃止されたエレメントの任意の組み合わせ

- IBM 拡張 (IBM 拡張にフラグが付けられるのは、FLAGSTD が指定され、かつ、「規格準拠外の非標準」として識別された場合です。)

通知メッセージはソース・プログラム・リストに表示され、次のような情報を含んでいます。

- エレメントを、「廃止」、「規格準拠外の標準」、または「規格準拠外の非標準」として識別します (廃止になり、しかも規格準拠外の言語エレメントには廃止のフラグだけを立てます)。
- そのエレメントが含まれている文節、ステートメント、またはヘッダーを識別します。
- そのエレメントが含まれる文節、ステートメント、またはヘッダーのソース・プログラム行および開始位置を識別します。
- そのエレメントが属するサブセットまたはオプショナル・モジュールを識別します。

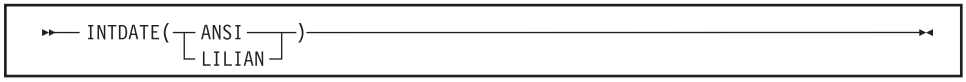
FLAGSTD には、予約語の標準セットが必要です。

次の例では、メッセージ・コードとテキストとともに、フラグ付き文節、ステートメントまたはヘッダーが出てきた行番号と桁が示されています。最下部には、フラグ付けされた項目の合計とそれらのタイプがまとめられています。

```
  LINE.COL CODE       FIPS MESSAGE TEXT

           IGYDS8211  Comment lines before "IDENTIFICATION DIVISION":
                      nonconforming nonstandard, IBM extension to
                      ANS/ISO 1985.

     11.14 IGYDS8111  "GLOBAL clause":  nonconforming standard, ANS/ISO
                      1985 high subset.

     59.12 IGYPS8169  "USE FOR DEBUGGING statement":  obsolete element
                      in ANS/ISO 1985.


FIPS MESSAGES TOTAL             STANDARD      NONSTANDARD      OBSOLETE

                3                    1              1              1
```

関連参照
313 ページの『矛盾するコンパイラー・オプション』

INTDATE



デフォルト: INTDATE(ANSI)

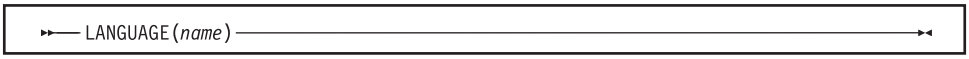
INTDATE(ANSI) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に ANSI COBOL 標準の開始日を使用するように指示します。日付 1 は、1601 年 1 月 1 日です。

INTDATE(LILIAN) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に言語環境プログラムのリリアン開始日を使用するように指示します。日付 1 は、1582 年 10 月 15 日です。

INTDATE(LILIAN) を使用すると、日付組み込み関数は、言語環境プログラムの日付呼び出し可能サービスと互換性のある結果を戻します。

使用上の注意: INTDATE(LILIAN) が有効なときは、CEECBLDY は使用不能になります。これは、組み込み関数または呼び出し可能サービスを使用して ANSI 整数を意味のある日付に変換する方法がないためです。INTDATE(LILIAN) が有効になっている呼び出しのターゲットとして、CEECBLDY を指定した CALL *literal* ステートメントをコーディングすると、コンパイラーはこれを診断し、呼び出しターゲットを CEEDAYS に変換します。

LANGUAGE



デフォルト: LANGUAGE(ENGLISH)

省略形: LANG(EN|UE|JA|JP)

LANGUAGE オプションは、コンパイラー出力の印刷に使用する言語を選択する場合に使用します。選択された言語で印刷される情報には次のものがあります。診断メッセージ、ソース・リストのページ・ヘッダーとスケール・ヘッダー、FIPS メッセージ・ヘッダー、メッセージ要約ヘッダー、コンパイル要約、および特定のコンパイラー・オプション (MAP、XREF、VBREF、および FLAGSTD) を選択した結果として生じるヘッダーと表記が含まれます。

name は、コンパイラー出力メッセージに使用する言語を指定します。LANGUAGE オプションに使用できる可能な値を、以下の表に示します。

言語名	省略形[1]	出力言語
ENGLISH	EN	英大 / 小文字混合 (デフォルト)
JAPANESE	JA、JP	日本語 (日本語文字セットを使用)
UENGLISH[2]	UE	英大文字

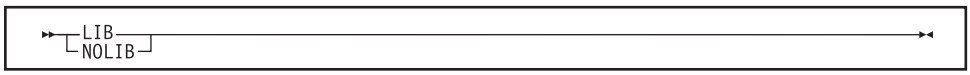
1. インストール先のシステム・プログラマーが、ここに説明されている以外の言語を提供した場合は、その言語名の少なくとも最初の 2 文字を指定しなければなりません。
2. UENGLISH 以外の言語を指定するには、該当する言語の機能をインストールしなければなりません。

コンパイル時に (CBL または PROCESS ステートメントを使用して) LANGUAGE オプションが変更された場合は、一部の初期テキストは、コンパイラーの開始時に有効であった言語を使用して印刷されます。

NATLANG: NATLANG ランタイム・オプションを使用すると、エラー・メッセージ、月名、および曜日名を含む、ランタイム環境で使用される各国語を制御することがで

きます。LANGUAGE コンパイラー・オプションと NATLANG ランタイム・オプションは互いに独立して働きます。それらのどちらかに優先権を与えずに同時に使用することができます。

LIB



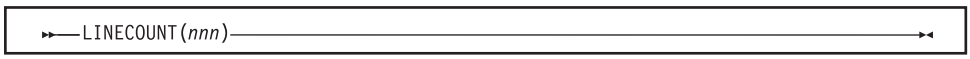
デフォルト: LIB

省略形: なし

プログラムで COPY、BASIS、または REPLACE ステートメントを使用する場合は、LIB コンパイラー・オプションを有効にする必要があります。COPY ステートメントと BASIS ステートメントの場合は、さらに、コピーされたコードをコンパイラーが獲得するライブラリー (複数も可) を定義する必要があります。ライブラリーは、環境に応じて、DD ステートメント、ALLOCATE コマンド、または環境変数を使用して定義してください。JCL を使用する場合は、SYSUT5 を割り振るための DD ステートメントも含める必要があります。

関連参照
360 ページの『コンパイラー指示ステートメント』
313 ページの『矛盾するコンパイラー・オプション』

LINECOUNT



デフォルト: LINECOUNT(60)

省略形: LC

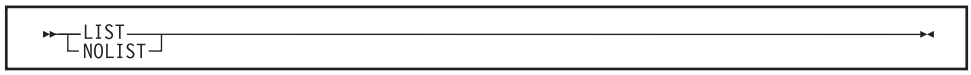
nnn は、10 ～ 255 の整数か、0 でなければなりません。

LINECOUNT(*nnn*) は、コンパイル・リストの各ページに印刷する行数を指定する場合に使用します。ページ編集を抑止する場合には、LINECOUNT(0) を使用してください。

LINECOUNT(0) を指定すると、コンパイル・リストではページ替えが行われません。

コンパイラーは、タイトル用に *nnn* のうちの 3 行を使用します。たとえば、LINECOUNT(60) を指定すると、57 行のソース・コードが出力リストの各ページに印刷されます。

LIST



デフォルト: NOLIST

省略形: なし

LIST コンパイラー・オプションは、ソース・コードのアセンブラー言語拡張のリストを作成する場合に使用します。

さらに、出力リストでは次のものを入手することができます。

- グローバル・テーブル
- リテラル・プール
- WORKING-STORAGE に関する情報
- プログラムの WORKING-STORAGE のサイズ、およびプログラムが NORENT オプションを使用してコンパイルされた場合には、オブジェクト・コード内でのその位置

出力が生成されるのは、以下の場合です。

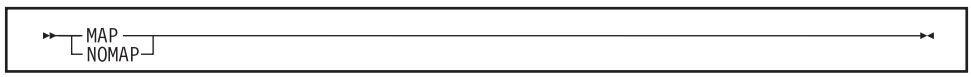
- COMPILE オプションを指定している (または NOCOMPILE(*x*) オプションが有効であり、エラー・レベル *x* 以上が発生していない)。
- OFFSET オプションを指定していない。

アセンブラー・リスト出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL LIST または NOLIST ステートメントを使用します。*CONTROL NOLIST の後のソース・ステートメントは、*CONTROL LIST ステートメントによって出力が通常の LIST 形式に戻されない限り、リストには含められません。

関連タスク
377 ページの『リストの入手』

関連参照
313 ページの『矛盾するコンパイラー・オプション』
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 言語解説書*)

MAP



デフォルト: NOMAP

省略形: なし

MAP コンパイラー・オプションを使用すると、DATA DIVISION で定義した項目のリストを作成することができます。出力には、以下のものが含まれます。

- DATA DIVISION のマップ

- グローバル・テーブル
- リテラル・プール
- ネストされたプログラム構造マップ、およびプログラム属性
- プログラムの作業用ストレージのサイズ、およびオブジェクト・コードにおけるその位置 (NORENT オプションを指定してプログラムをコンパイルする場合)

MAP 出力を制限したい場合は、DATA DIVISION で *CONTROL MAP または NOMAP ステートメントを使用してください。*CONTROL NOMAP の後のソース・ステートメントは、*CONTROL MAP ステートメントによって出力が通常の MAP 形式に戻らない限り、リストには含められません。たとえば、次のようになります。

```
*CONTROL NOMAP            *CBL NOMAP
    01  A                     01  A
    02  B                     02  B
*CONTROL MAP              *CBL MAP
```

MAP オプションを選択すると、組み込み MAP 報告書もソース・コード・リストに印刷することができます。圧縮 MAP 情報は、DATA DIVISION の FILE SECTION、WORKING-STORAGE SECTION、および LINKAGE SECTION のデータ名定義の右側に印刷されます。XREF データと組み込み MAP 要約の両方が同じ行にあるときは、組み込み要約の方が先に印刷されます。

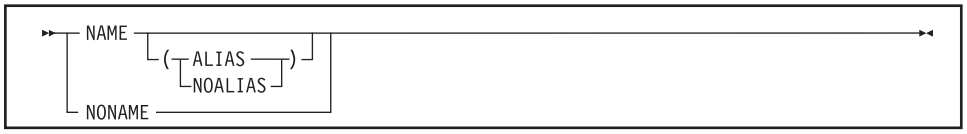
382 ページの『例: MAP 出力』

関連概念
365 ページの『第 19 章 デバッグ』

関連タスク
377 ページの『リストの入手』

関連参照
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 言語解説書*)

NAME



デフォルト: NONAME、または NAME(NOALIAS) (NAME だけが指定された場合)

省略形: なし

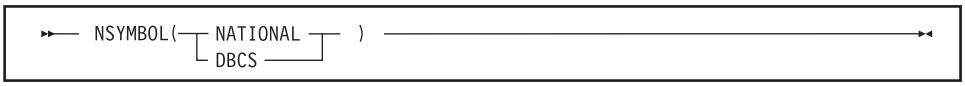
NAME は、各オブジェクト・モジュールについてのリンク・エディット NAME カードを生成する場合に使用します。NAME は、バッチ・コンパイルを行う際に、各ロード・モジュールの名前を生成する目的でも使用できます。NAME を指定すると、作成される各オブジェクト・モジュールに NAME カードが追加されます。ロード・モジュール名は、PROGRAM-ID ステートメントからモジュール名を形成する際の規則を使用して作成されます。

NAME(ALIAS) を指定したとき、プログラムに ENTRY ステートメントが含まれていると、ENTRY ステートメントごとにリンク・エディット ALIAS カードが生成されます。

NAME または NAME(ALIAS) オプションは、言語環境プログラムのプリリンカーでプリリンクされるプログラムをコンパイルするときには使用できません。

関連参照
PROGRAM-ID 段落 (*Enterprise COBOL 言語解説書*)

NSYMBOL



デフォルト: NSYMBOL(NATIONAL)

省略形: NS(NAT|DBCS)

NSYMBOL オプションは、リテラルおよび PICTURE 文節で使用される N 記号の解釈を制御し、国別処理や DBCS 処理が必要かどうかを指示します。

NSYMBOL(NATIONAL) を指定した場合:

- 記号 N のみからなる PICTURE 文節を指定し、USAGE 文節を指定せずに定義されたデータ項目は、USAGE NATIONAL 文節が指定されたものとして扱われます。
- N". . ." または N'. . .' の形式のリテラルは、国別リテラルとして扱われます。

NSYMBOL(DBCS) を指定した場合:

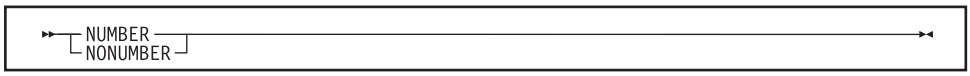
- 記号 N のみからなる PICTURE 文節を指定し、USAGE 文節を指定せずに定義されたデータ項目は、USAGE DISPLAY-1 文節が指定されたものとして扱われます。
- N". . ." または N'. . .' の形式のリテラルは、DBCS リテラルとして扱われます。

NSYMBOL(DBCS) オプションは、前のリリースの IBM COBOL との互換性を提供します。NSYMBOL(NATIONAL) オプションにより、前述の言語エレメントがこの点に関して 200x COBOL 標準の草案に準拠するようになります。

NSYMBOL(NATIONAL) は、Unicode データや Java とのインターオペラビリティーのためのオブジェクト指向構文を使用するアプリケーションの場合の推奨オプションです。

関連参照
313 ページの『矛盾するコンパイラー・オプション』

NUMBER



デフォルト: NONUMBER

省略形: NUM|NONUM

NUMBER コンパイラー・オプションは、ソース・コードの中に行番号があり、それらの番号がエラー・メッセージと SOURCE、MAP、LIST、および XREF のリストで必要な場合に使用してください。

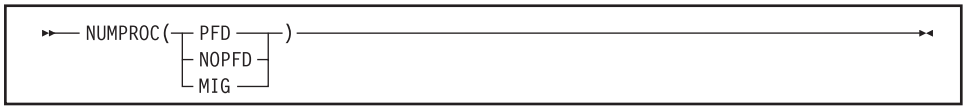
NUMBER を要求すると、コンパイラーは、桁 1 ～ 6 に数字だけが含まれているかどうか、および番号が数字の照合シーケンスになっているかどうかを検査します (これに反して、SEQUENCE を使用すると、これらの桁の文字が EBCDIC 照合シーケンスになっているかどうかが検査されます)。行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の行番号を割り当てます。コンパイラーは、新規の値に 2 つのアスタリスクでフラグを立て、シーケンス・エラーを示すメッセージをリストに組み込みます。シーケンス検査は、先行の行の新しく割り当てられた値に基づいて、次のステートメントから継続されます。

COPY ステートメントを使用する場合、NUMBER が有効なときは、ソース・プログラムの行番号とコピーブックの行番号が対応している必要があります。

バッチ・コンパイルを行っており、LIB と NUMBER が有効な場合は、バッチ・コンパイルのすべてのプログラムが 1 つの入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

ソース・コードの中に行番号がない場合や、コンパイラーにソース・コードの行番号を無視させる場合には、NONUMBER を使用してください。NONUMBER が有効であると、コンパイラーは、ソース・ステートメントの行番号を生成し、それらの番号をリストで参照として使用します。

NUMPROC



デフォルト: NUMPROC(NOPFD)

省略形: なし

NUMPROC(NOPFD) は、コンパイラーに無効符号処理を実行させる場合に使用します。このオプションは NUMPROC(PFD) ほど有効ではありません。オブジェクト・コードのサイズが大きくなり、符号付きデータをすべて妥当性検査するために実行時オーバーヘッドが増加する可能性があります。

NUMPROC(PFD) は、無効符号処理をバイパスするために使用できるパフォーマンス・オプションです。このオプションは、プログラム・データが以下の IBM システム標準と正確に一致する場合にのみ 使用してください。

外部 10 進数、符号なし: 符号バイトの高位 4 ビットには、X'F' が入ります。

外部 10 進数、符号付きオーバーパンチ: 符号バイトの高位 4 ビットには、その数値が正または 0 であれば X'C' が入り、そうでなければ X'D' が入ります。

外部 10 進数、分離符号: 分離符号には、その数値が正または 0 であれば文字 '+' が入り、そうでなければ '-' が入ります。

内部 10 進数、符号なし: 最低位バイトの下位 4 ビットには、X'F' が入ります。

内部 10 進数、符号付き: 最低位バイトの下位 4 ビットには、その数値が正または 0 であれば X'C' が入り、そうでなければ X'D' が入ります。

COBOL 算術ステートメントが作成するデータは、上記の IBM システム標準に適合します。しかし、REDEFINES を使用したり、グループ移動を行うと、データが変更されて、この標準に適合しなくなることがあります。NUMPROC(PFD) を使用する場合は、グループ移動を行うのではなく、INITIALIZE ステートメントを使用してデータ・フィールドを初期化しなければなりません。

NUMPROC(PFD) を使用すると、数値データのクラス・テストに影響を与えることがあります。PL/I または FORTRAN で書かれたプログラムを COBOL プログラムで呼び出す場合は、NUMPROC(NOPFD) または NUMPROC(MIG) を使用しなければなりません。

符号表示は、NUMPROC オプションばかりでなく、インストール時のオプション NUMCLS の影響も受けます。

NUMPROC(MIG) は、OS/VS COBOL プログラムを Enterprise COBOL に移行する際の援助手段として使用してください。NUMPROC(MIG) が有効であると、次の処理が行われます。

- MOVE ステートメントと算術演算の出力の場合のみ、優先符号が作成されます。
- 入力では明示符号修復は行われません。
- 変換の際に暗黙符号修復が行われることがあります。
- 数値比較は、論理比較ではなく、10 進比較によって行われます。

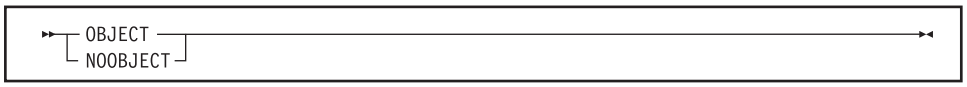
関連タスク

53 ページの『非互換データの検査 (数値のクラス・テスト)』

関連参照

52 ページの『符号表現および処理』

OBJECT



デフォルト: OBJECT

省略形: OBJ|NOOBJ

OBJECT を使用すると、生成されたオブジェクト・コードをディスクまたはテープに保管し、リンケージ・エディターまたはバインダーへの入力として後で使用することができます。

OBJECT を指定する場合は、コンパイル用の JCL の中に SYSLIN DD ステートメントを組み込んでください。

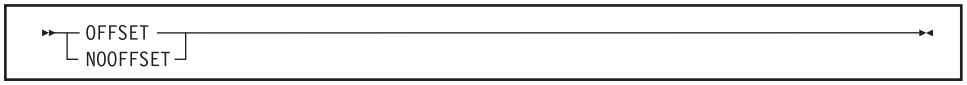
DECK と OBJECT の相違点は、データ・セットの経路指定だけです。

- DECK 出力は、SYSPUNCH の DD 名に関連するデータ・セットに送られます。
- OBJECT 出力は、SYSLIN の DD 名に関連するデータ・セットに送られます。

ご使用のシステムの指針に従ってオプションを使用してください。

関連参照
313 ページの『矛盾するコンパイラー・オプション』

OFFSET



デフォルト: NOOFFSET

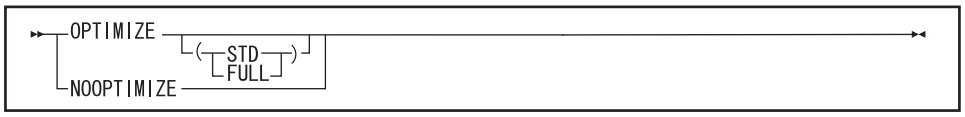
省略形: OFF|NOOFF

OFFSET は、PROCEDURE DIVISION の圧縮リストを作成する場合に使用します。
OFFSET を使用した場合、PROCEDURE DIVISION の圧縮リストには、行番号、ステートメント参照、および各ステートメントに対して生成された最初の命令の位置が入れられます。さらに、以下のものが作成されます。

- グローバル・テーブル
- リテラル・プール
- プログラムの作業用ストレージのサイズ、およびオブジェクト・コードにおけるその位置 (NORENT オプションを指定してプログラムをコンパイルする場合)

関連参照
313 ページの『矛盾するコンパイラー・オプション』

OPTIMIZE



デフォルト: NOOPTIMIZE

省略形: OPT|NOOPT

OPTIMIZE は、オブジェクト・プログラムの実行時間を短縮するために使用します。最適化によって、オブジェクト・プログラムが使用するストレージの量を減らすこともできます。OPTIMIZE を使用すると、コンパイル時間が長くなり、プログラム内のステートメントの順序が変わることがあるので、デバッグの際にはこのオプションを使用しないでください。

サブオプションを付けないで OPTIMIZE を指定すると、OPTIMIZE(STD) が有効になります。

FULL サブオプションは、OPT(STD) の下で実行される最適化に加えて、コンパイラーが DATA DIVISION から未参照データ項目を破棄し、さらにこれらのデータ項目をその VALUE 文節に初期化するためのコードの生成を抑止するように要求します。OPT(FULL) が有効であると、未参照の 77 レベル項目および基本 01 レベル項目がすべて破棄されます。さらに、どの従属項目も参照されなければ、01 レベルのグループ項目も破棄されます。削除された項目はリストの中で示されます。MAP オプションが有効であれば、データ・マップ情報内の XXXX の BL 番号は、そのデータ項目が破棄されたことを示します。

未使用データ項目

プログラムで未使用データ項目を意図的に利用している場合は、OPT(FULL) を使用しないでください。次のような 2 つの方法が以前は広く使用されていました。

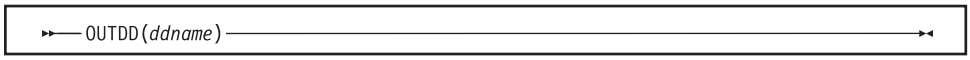
- OS/VS COBOL プログラムで時々使用される方法で、参照されたテーブルの後に未参照テーブルを置き、2 番目のテーブルにアクセスするために最初のテーブルの範囲外添え字を使用する。プログラムにこの問題があるかどうかを調べるには、SSRANGE コンパイラー・オプションと CHECK(ON) ランタイム・オプションを一緒に使用してください。この問題を処理するには、大きいテーブルをコーディングする COBOL の機能を使用し、テーブルを 1 つだけ使用します。
- 目印となるデータ項目を WORKING-STORAGE SECTION に置いて、プログラム・データの始めと終わりを識別できるようにする、あるいはそのデータを使用するライブラリー・ツール用のプログラムのコピーに印を付けてプログラムのバージョンを識別できるようにする方法。この問題を解決するには、これらの項目を VALUE 文節ではなく、PROCEDURE DIVISION ステートメントで初期化します。この方法を使用すると、コンパイラーはそれらの項目を使用されているものと見なし、削除しません。

重大レベル以上のエラーが起こった場合、OPTIMIZE オプションはオフにされます。

関連概念
599 ページの『最適化』

関連参照
313 ページの『矛盾するコンパイラー・オプション』

OUTDD



デフォルト: OUTDD(SYSOUT)

省略形: OUT

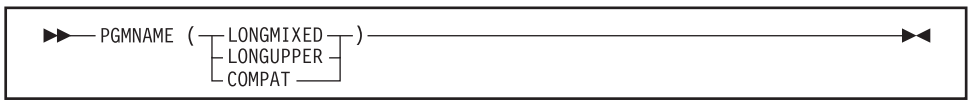
OUTDD を使用すると、システム論理出力装置に送られる DISPLAY 出力を特定の *ddname* に送るように指定することができます。OUTDD で指名する DD 名として、階層ファイル・システム (HFS) 内のファイルを指定することができます。この DD 名が割り振られなかった場合のデフォルト値および動作については、DISPLAY ステートメントの説明を参照してください。

MSGFILE ランタイム・オプションを使用すると、RPTOPTS および RPTSTG ランタイム・オプションによって生成されるすべての実行時診断および報告書が書き込まれるファイルの DD 名を指定することができます。IBM 提供のデフォルト値は MSGFILE(SYSOUT) です。OUTDD コンパイラー・オプションと MSGFILE ランタイム・オプションで同じ DD 名が指定された場合は、システム論理出力装置に送られる DISPLAY 出力とエラー・メッセージ情報は同じ宛先に経路指定されます。

関連タスク
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

関連参照
MSGFILE (*言語環境プログラム プログラミング・リファレンス*)

PGMNAME



デフォルト: PGMNAME(COMPAT)

省略形: PGMN(LM|LU|CO)

LONGUPPER は、UPPER、LU、または U と省略することができ、LONGMIXED は、MIXED、LM、または M と省略することができます。

PGMNAME オプションは、以下のコンテキストで使用される名前の取り扱いを制御します。

- PROGRAM-ID 段落で定義されたプログラム名
- ENTRY ステートメントのプログラム入り口点名
- 以下におけるプログラム名参照:
  - ネストされたプログラムの呼び出し
  - 別個にコンパイルされたプログラムへの静的呼び出し
  - 静的 SET *procedure-pointer* TO ENTRY *literal* ステートメント
  - 静的 SET *function-pointer* TO ENTRY *literal* ステートメント
  - ネストされたプログラムの CANCEL

PGMNAME(COMPAT)

PGMNAME(COMPAT) を使用すると、プログラムは、COBOL コンパイラーの旧バージョンと互換性のある方法で処理されます。すなわち、次のようになります。

- プログラム名の長さは最高 30 文字までである。
- 名前で使用される文字はすべて英字、数字、またはハイフンでなければならず (ただし、プログラム名がリテラル形式で入力されており、最も外側のプログラムにある場合を除く)、リテラルには拡張文字の @、#、および $ を含めることができる。
- 少なくとも 1 文字は英字でなければならない。
- ハイフンは最初の文字または最後の文字として使用できない。

外部プログラム名はコンパイラーによって次のように処理されます。

- 大文字に変換される。
- 8 文字に切り捨てられる。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でない場合は、次のように変換される。
  - 1 ～ 9 は A ～ I に変換される。
  - その他はすべて J に変換される。

PGMNAME(LONGUPPER)

PGMNAME(LONGUPPER) を使用する場合、PROGRAM-ID 段落で COBOL ユーザー定義語として指定されるプログラム名は、次のようなユーザー定義語に関する通常の COBOL 規則に従っていなければなりません。

- プログラム名の長さは最高 30 文字までである。
- 名前で使用される文字はすべて英字、数字、またはハイフンでなければならない。
- 少なくとも 1 文字は英字でなければならない。
- ハイフンは最初の文字または最後の文字として使用できない。

定義または参照のいずれかで、プログラム名をリテラルとして指定する場合は、次のようになります。

- プログラム名の長さは最高 160 文字までである。
- 名前で使用される文字はすべて英字、数字、またはハイフンでなければならない。

- 少なくとも 1 文字は英字でなければならない。
- ハイフンは最初の文字または最後の文字として使用できない。

外部プログラム名はコンパイラーによって次のように処理されます。

- 大文字に変換される。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でない場合は、次のように変換される。
  - 1 ～ 9 は A ～ I に変換される。
  - その他はすべて J に変換される。

ネストされたプログラムはコンパイラーによって大文字に変換されますが、それ以外はそのまま処理され、切り捨ても変換も行われません。

PGMNAME(LONGMIXED)

PGMNAME(LONGMIXED) を使用する場合、プログラム名は、切り捨てられたり、変換されたり、大文字に変換されることなく、現状のまま処理されます。

PGMNAME(LONGMIXED) を使用する場合、すべてのプログラム名定義は、プログラム名のリテラル形式を使用して、PROGRAM-ID 段落または ENTRY ステートメントで指定する必要があります。

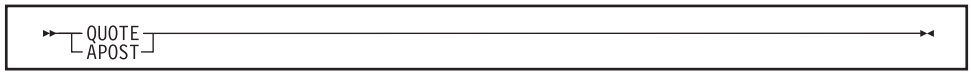
(PGMNAME オプションの影響を受ける、上記にリストされたコンテキストの中で) プログラム名に使用されるリテラルには、X'41'-X'FE' の範囲の文字を含めることができます。

使用上の注意

- 以下のものは、PGMNAME オプションの影響を受けません。
  - クラス名およびメソッド名。
  - システム名 (SELECT . . . ASSIGN 内の割り当て名、および COPY ステートメントのテキスト名またはライブラリー名)。
  - 動的呼び出し。動的呼び出しはターゲット・プログラム名で解決されます。このとき、ターゲット・プログラム名は 8 文字に切り詰められ、大文字に変換されて、組み込みハイフンや先行桁は変換されます。
  - ネストされていないプログラムの CANCEL。ネーム・レゾリューションは動的呼び出しのメカニズムと同じものを使用します。
- PGMNAME オプションは、ネストされたプログラムの呼び出し、および呼び出し側と一緒にリンクされるプログラムへの静的呼び出しには影響を与えません。
- プログラム名が 8 バイト以下ですべて大文字でない限り、PGMNAME(LONGMIXED) または PGMNAME(LONGUPPER) オプションでコンパイルされた COBOL プログラムへの動的呼び出しは許可されません。さらに、プログラムの名前はそれが入っているモジュールの名前に対して固有でなければなりません。
- PGMNAME(LONGMIXED) によってサポートされる拡張文字セットを使用するときは、名前の解決に使用されるメカニズムに応じて、リンケージ・エディター、バインダー、プリリンカー、またはシステム規則のうちの該当するものに適合する名前を使用してください。

コンマや括弧などの文字は使用しないでください。これらの文字は、リンケージ・エディターやバインダーの制御ステートメントで使用されます。

QUOTE/APOST



デフォルト: QUOTE

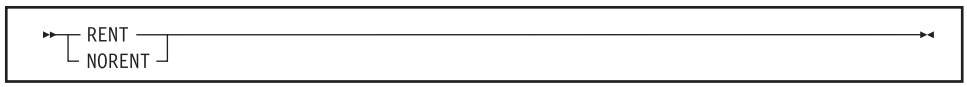
省略形: Q|APOST

QUOTE は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つまたは複数の引用符 (“) 文字を表すようにする場合に使用します。

APOST は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つまたは複数のアポストロフィ (’) 文字を表すようにする場合に使用します。

区切り文字: APOST または QUOTE オプションのどちらが有効であるかに関係なく、引用符またはアポストロフィのいずれかをリテラル区切り文字として使用することができます。リテラルの左区切り文字として使用される区切り文字は、そのリテラルの右区切り文字としても使用しなければなりません。

RENT



デフォルト: RENT

省略形: なし

RENT としてコンパイルされたプログラムは再入可能オブジェクト・プログラムとして生成され、NORENT としてコンパイルされたプログラムは再入不可オブジェクト・プログラムとして生成されます。どちらもメインプログラムまたはサブプログラムとして呼び出すことができます。

DATA および RMODE 設定: RENT オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラー・オプションと相互作用します。 再入可能プログラムを拡張アドレス方式で実行する場合は、DATA(24|31) オプションを使用して、動的データ域を制限のないストレージで割り振るのか 16MB より下から獲得されたストレージで割り振るのかを制御することができます。プログラムが 16MB より上の仮想記憶アドレスで拡張アドレス方式で実行される場合は、プログラムを RENT または RMODE(ANY) でコンパイルしなければなりません。

RENT は、生成されたオブジェクト・プログラムの RMODE (常駐モード) にも影響を与えます。Enterprise COBOL のプログラムはすべて、AMODE ANY になります。

DATA: NORENT を指定してコンパイルされたプログラムは、DATA オプションの設定による影響を受けません。

CICS: CICS のもとで実行するプログラムには RENT を使用しなければなりません。

UNIX: UNIX 環境のもとで実行するプログラムには RENT を使用しなければなりません。

DB2: DB2 ストアード・プロシージャー (特に、メインプログラムとして指定されたプログラム) の場合、推奨オプションは RENT です。

リンク・エディットについての考慮事項: ロード・モジュール内のすべてのプログラムが RENT を指定してコンパイルされる場合は、RENT リンケージ・エディター・オプションまたはバインダー・オプションを使用して、そのロード・モジュールをリンク・エディットすることをお勧めします (逐次に再使用可能な非 COBOL プログラムがロード・モジュールに含まれている場合は、RENT ではなく、REUS リンケージ・エディター・オプションまたはバインダー・オプションを使用してください)。

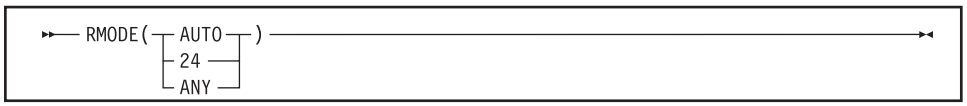
ロード・モジュール内のいずれかのプログラムが NORENT でコンパイルされた場合には、そのロード・モジュールは RENT または REUS リンク・エディット属性を使用してリンク・エディットしてはなりません。CANCEL ステートメントがその後の CALL においてプログラムの最新コピーを保証するようにするためには、NOREUS リンケージ・エディター・オプションを使用する必要があります。

関連概念
38 ページの『ストレージとそのアドレス可能度』

関連タスク
再入可能コードの使用 (*IBM DB2 アプリケーション・プログラミングおよび SQL ガイド*)

関連参照
313 ページの『矛盾するコンパイラー・オプション』

RMODE



デフォルト: AUTO

省略形: なし

RMODE オプションの設定は、生成されたオブジェクト・プログラムの RMODE (常駐モード) にも影響を与えます。 RMODE(AUTO) オプションでコンパイルされたプログラムは、NORENT が指定された場合は RMODE(24) になり、RENT が指定された場合は RMODE(ANY) になります。

RMODE(24) オプションでコンパイルされたプログラムは、NORENT または RENT のいずれが指定されたかに関係なく、RMODE(24) になります。

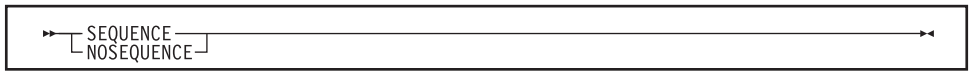
RMODE(ANY) オプションでコンパイルされたプログラムは、NORENT または RENT のいずれが指定されたかに関係なく、RMODE(ANY) になります。

DATA および RENT: RMODE オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラー・オプションおよびランタイム・オプションにも影響を与えます。モードの異なるプログラム間でのデータの受け渡しに関する情報については、以下のセクションを参照してください。

関連概念
38 ページの『ストレージとそのアドレス可能度』

関連参照
157 ページの『QSAM ファイル用のバッファーの割り振り』
313 ページの『矛盾するコンパイラー・オプション』

SEQUENCE



デフォルト: SEQUENCE

省略形: SEQ|NOSEQ

SEQUENCE を使用すると、コンパイラーはソース・ステートメントの桁 1 ～ 6 を調べ、各ステートメントが EBCDIC 照合シーケンスに従って昇順に並んでいるかどうかを検査します。昇順になっていないステートメントがあると、コンパイラーは診断メッセージを出します (桁 1 ～ 6 がブランクのソース・ステートメントはこのシーケンス検査には関与せず、したがってメッセージは出されません)。

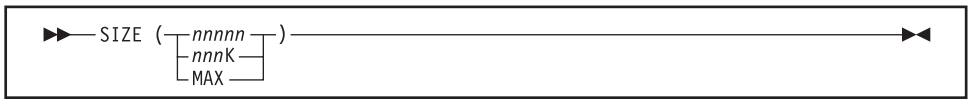
COPY ステートメントを使用する場合、SEQUENCE が有効なときは、ソース・プログラムのシーケンス・フィールドとコピーブックのシーケンス・フィールドが対応している必要があります。

NUMBER と SEQUENCE を使用すると、シーケンス検査は、EBCDIC 照合シーケンスではなく、数字に従って行われます。

バッチ・コンパイルを行っている場合は、LIB と SEQUENCE が有効であると、バッチ・コンパイルのすべてのプログラムが 1 つの入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

この検査と診断メッセージを抑止する場合は、NOSEQUENCE を使用してください。

SIZE



デフォルト: SIZE(MAX)

省略形: SZ

nnnnn は 10 進数で、少なくとも 851968 でなければなりません。

*nnn*K は、1K 単位で 10 進数を指定します。受け入れられる最小値は 832K です。

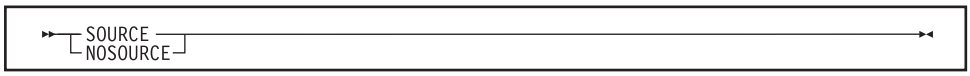
MAX は、ユーザー領域で使用可能な最大のストレージ・ブロックをコンパイル時に使用することを要求します。

SIZE は、コンパイルに使用できる主記憶域の量を示すために使用します (1K は 10 進数の 1024 バイト)。

コンパイラーを呼び出すとき、ユーザー領域に特定量の未使用ストレージを残す必要がある場合には、SIZE(MAX) を使用してはなりません。31 ビット・モードでコンパイルする場合、SIZE(MAX) を指定したときは、コンパイラーは以下の順序でストレージを使用します。

- 16MB より上 - ユーザー領域のすべてのストレージ
- 16MB より下 - 以下のためのストレージ:
  - 作業ファイル・バッファー
  - 16MB 境界より下にロードしなければならないコンパイラー・モジュール

SOURCE



デフォルト: SOURCE

省略形: S|NOS

SOURCE は、ソース・プログラムのリストを入手する場合に使用します。このリストには、PROCESS または COPY ステートメントによって組み込まれたすべてのステートメントが入ります。

ソース・リストに組み込みメッセージが必要な場合は、SOURCE を必ず指定します。

コンパイラー出力リストにソース・コードを出したくない場合は、NOSOURCE を使用してください。

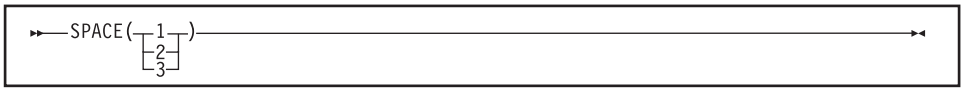
SOURCE 出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL SOURCE または NOSOURCE ステートメントを使用してください。*CONTROL NOSOURCE の後のソース・

ステートメントは、*CONTROL SOURCE ステートメントによって出力が通常の SOURCE 形式に戻されない限り、リストには含められません。

382 ページの『例: MAP 出力』

関連参照
*CONTROL (*CBL) ステートメント (*Enterprise COBOL 言語解説書*)

SPACE



デフォルト: SPACE(1)

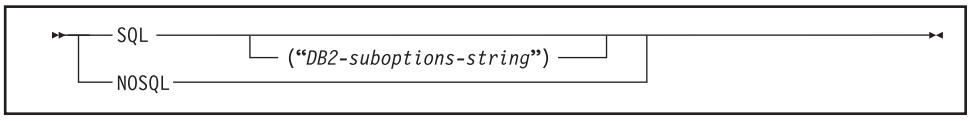
省略形: なし

SPACE は、ソース・コード・リストで 1 行送り、2 行送り、または 3 行送りを選択するために使用します。

SPACE が意味を持つのは、SOURCE コンパイラー・オプションが有効な場合だけです。

関連参照
346 ページの『SOURCE』

SQL



デフォルト: NOSQL

省略形: なし

SQL コンパイラー・オプションを使用すると、DB2 coprocessor 機能を使用可能にし、DB2 サブオプションを指定できるようになります。COBOL ソース・プログラムに SQL ステートメントが含まれており、それが DB2 プリコンパイラーで処理されていない場合には、SQL オプションを必ず指定しなければなりません。

SQL オプションを使用すると、コンパイラーは、データベース要求モジュール (DBRM) を DD 名 DBRMLIB に書き込みます。コンパイラーでは DB2 バージョン 7 またはそれ以降のリリースにアクセスする必要があります。

NOSQL オプションを指定した場合は、ソース・プログラム内で検出された SQL ステートメントは診断され、破棄されます。

DB2 サブオプションのストリングは、引用符またはアポストロフィを使用して区切ってください。

長いサブオプション・ストリングを、複数のサブオプション・ストリングに分割して、複数の CBL ステートメントに置くことができます。それぞれの DB2 サブオプションは、指定された順に連結されます。たとえば、次のようになります。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='SQL("string1")'
//COBOL.SYSIN DD *
       CBL SQL("string2")
       CBL SQL("string3")
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DRIVER1.
       . . .
```

コンパイラーは、以下のサブオプション・ストリングを DB2 coprocessor に渡します。

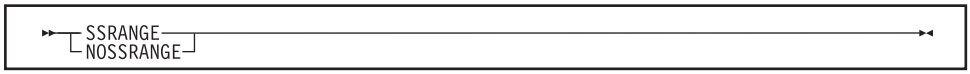
```
"string1 string2 string3"
```

ここに示すように、連結されたストリングは 1 つのスペースで区切られます。同じ DB2 オプションの複数インスタンスが見つかった場合は、各オプションで最後に指定されたものが使用されます。コンパイラーは、連結 DB2 サブオプション・ストリングの長さを 4K バイトに限定しています。

関連タスク
421 ページの『SQL オプションを使用したコンパイル』

関連参照
313 ページの『矛盾するコンパイラー・オプション』

SSRANGE



デフォルト: NOSSRANGE

省略形: SSR|NOSSR

SSRANGE を使用すると、添え字（ALL 添え字を含む）または指標がテーブルの領域外の区域を参照しようとしているかどうかを検査するコードを生成することができます。それぞれの添え字または指標は、個別に妥当性を検査されるわけではありません。むしろ、テーブルの領域外の区域を参照しないようにするために、有効アドレスが検査されます。定義された最大長の範囲内で参照を行うようにするために、可変長項目も検査されます。

次の点を確認するために、参照変更式が検査されます。

- 参照変更開始位置が 1 以上である。
- 参照変更開始位置が、サブジェクト・データ項目の現在の長さより大きくない。
- 参照変更長の値 (指定されている場合) が 1 以上である。

- 参照変更開始位置と長さの値 (指定されている場合) がサブジェクト・データ項目の終わりを超えた区域を参照していない。

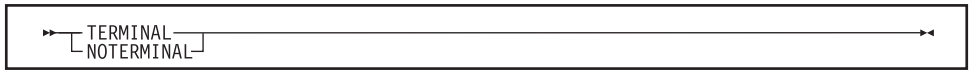
SSRANGE がコンパイル時に有効であると、範囲検査コードが生成されます。ランタイム・オプションとして CHECK(OFF) を指定すれば、範囲検査を抑止することができます。こうすれば、範囲検査コードはオブジェクト・コードで休止状態になります。このように範囲検査コードを任意指定で使用して、予期しないエラーを再コンパイルせずに解決するようにして、役立てることもできます。

範囲外条件が検出されると、エラー・メッセージが表示され、プログラムは終了します。

覚え書き: 範囲検査が行われるのは、プログラムを SSRANGE オプションを指定してコンパイルし、かつ CHECK(ON) ランタイム・オプションを指定して実行した場合だけです。

関連概念
102 ページの『参照修飾子』

TERMINAL



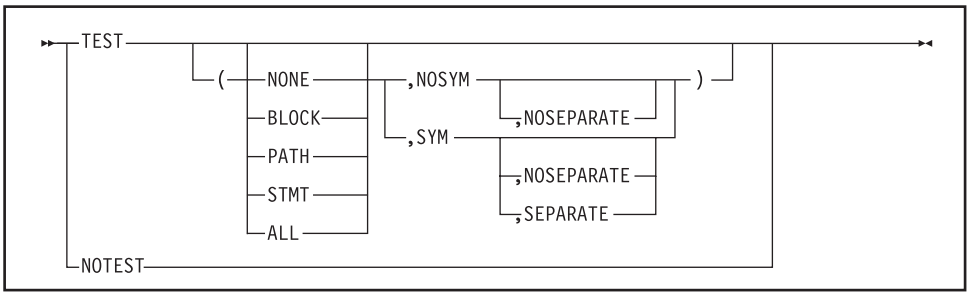
デフォルト: NOTERMINAL

省略形: TERM|NOTERM

TERMINAL を使用すると、進行メッセージと診断メッセージを SYSTERM データ・セットに送ることができます。

この追加出力が不要な場合は NOTERMINAL を使用してください。

TEST



デフォルト: NOTEST

省略形: SEPARATE の場合は SEP、NOSEPARATE の場合は NOSEP

TEST を使用すると、デバッグ・ツールがバッチ・デバッグおよび対話式デバッグを実行できるようにするオブジェクト・コードを作成することができます。使用できるデバッグ・サポートの量は、どの TEST サブオプションを使用するかによって決まります。また、TEST オプションを使用して、言語環境プログラムによって作成される定様式ダンプに記号変数を組み込むように要求することもできます。

デバッグ情報とともにオブジェクト・コードを生成したくない場合、および定様式ダンプに記号変数を組み込みたくない場合は、NOTEST を使用します。

TEST には、次の 3 つのサブオプションがあります。

1. 最初のサブオプションは、コンパイラーがコンパイル時組み込みフックを生成するかどうかを指定します。
2. 2 番目は、記号情報を生成するかどうかを指定します。
3. 3 番目は、記号情報をオブジェクト・プログラムの一部にするのか、それとも別個のファイルに入れるのかを指定します。

   1 つ、2 つ、またはすべてのサブオプションを任意に組み合わせて指定できますが、SEPARATE を指定できるのは、SYM が有効な場合だけです。

以下のフック位置サブオプションの 1 つを選択できます。

NONE フックは生成されません。

BLOCK フックはすべてのプログラム入り口点および出口点で生成されます。

PATH フックは、すべてのプログラム入り口点および出口点と、すべてのパス点で生成されます。パス点とは、ロジック・フローが必ずしも順次でなくてもよいか、あるいはロジック・フローを変更できるプログラム内の任意の場所です。パス点の例としては、IF-THEN-ELSE 構成、PERFORM ループ、ON SIZE ERROR 句、および CALL ステートメントなどがあります。

STMT フックは、すべてのステートメントおよびラベルと、すべてのプログラム入り口点および出口点で生成されます。さらに、DATEPROC オプションが有効であると、すべての日付処理ステートメントでフックが生成されます。

ALL フックは、すべてのステートメント、すべてのパス点、およびすべてのプログラム入り口点と出口点 (最外部プログラムと含まれているプログラムの両方) で生成されます。さらに、DATEPROC オプションが有効であると、すべての日付処理ステートメントでフックが生成されます。

SYM サブオプションを設定するときに、任意指定のサブオプション SEPARATE を指定して、デバッグ・ツール用の記号情報テーブルをオブジェクト・モジュールとは別個のデータ・セットに生成するように指示することができます。SEPARATE サブオプションを指定しないと、記号情報はオブジェクト・モジュールに組み込まれます。

JCL または TSO から COBOL コンパイラーを呼び出すときに、TEST(...,SYM,SEPARATE) オプションを指定すると、記号デバッグ情報テーブルは、SYSDEBUG DD ステートメントで指定されたデータ・セットに書き込まれます。SYSDEBUG DD ステートメントでは、順次データ・セットの名前、PDS または PDSE メンバーの名前、あるいは HFS パスを指定する必要があります。データ・セットの LRECL は 80 以上、1024 以下でなければなりません。SYSDEBUG のデフォルトの LRECL は 1024 です。データ・セットの RECFM は、F または FB にすることが

できます。ブロック・サイズは、DCB パラメーターの BLKSIZE サブパラメーターを使用して設定するか、またはシステム決定のデフォルトのブロック・サイズをシステムに設定させることができます。

UNIX シェルから COBOL コンパイラーを呼び出すときに、TEST(...,SYM,SEPARATE) オプションを指定すると、記号デバッグ情報テーブルは、現行ディレクトリー内のファイル *file*.dbg に書き込まれます (ここで、*file* は COBOL ソース・ファイルの名前です)。

実動デバッグ: 最新レベルのデバッグ・ツールでは、コンパイル時組み込みフックがなくても、*オーバーレイ・フック* と呼ばれるデバッグ・ツール機能を使用してプログラムをステップスルー・デバッグすることができます。この機能を利用するには、NOOPTIMIZE および TEST(NONE) でコンパイルしてください。この方法は、*実動デバッグ* または*実動レベル・デバッグ* とも呼ばれます。デバッグ・ツール機能の大部分は使用可能でありながら、コンパイル時組み込みフックのパフォーマンス・オーバーヘッドを回避できるからです。実動デバッグは SEPARATE サブオプションの使用を指すこともあります。このサブオプションを使用すると、ロード・モジュール・サイズを小さくしたまま、デバッグ・ツールおよび言語環境プログラムの定様式ダンプ機能性を維持することができます。

実動デバッグでオーバーレイ・フックを使用する場合には、デバッグ機能に関する次のような制約があります。

- GO TO デバッグ・ツール・コマンドはサポートされません。

TEST オプションが言語環境プログラムの定様式ダンプを見やすくできる方法としては、次の 2 つがあります。

- TEST の SYM サブオプションを使用すると、単なるオフセットではなく、失敗したステートメントを示す行番号がダンプに入れられます。
- TEST の SYM サブオプションを使用すると、プログラム変数の値がダンプ内にリストされます。

NOTEST で作成したダンプの場合は、プログラム変数も失敗したステートメントの行番号も示されません。

Enterprise COBOL は、言語環境プログラムが提供するダンプ・サービスを使用して、言語環境プログラム準拠のその他のメンバー言語によって生成されるダンプの内容および形式と一致するダンプを生成します。処理されない条件に対して言語環境プログラムがダンプを作成するかどうかは、ランタイム・オプション TERMTHDACT の設定値によって決まります。TERMTHDACT(DUMP) を指定した場合は、重大度 2 以上の条件が処理されないと、ダンプが生成されます。

関連概念

TERMTHDACT オプションの設定についての考慮事項 (*言語環境プログラム デバッグのガイド*)

関連タスク

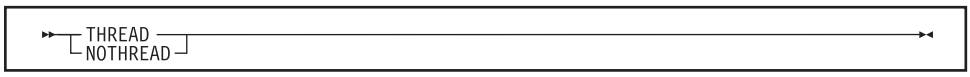
285 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』

ダンプの生成 (*言語環境プログラム デバッグのガイド*)

Debug Tool User's Guide and Reference の『Invoking Debug Tool using the run-time TEST option』の項

関連参照
313 ページの『矛盾するコンパイラー・オプション』
TEST | NOTEST (*言語環境プログラム プログラミング・リファレンス*)

THREAD



デフォルト : NOTHREAD

省略形: なし

THREAD を指定すると、COBOL プログラムを、複数の POSIX スレッドまたは PL/I タスクを含む言語環境プログラムのエンクレーブで実行できるようになります。

THREAD オプションを指定してコンパイルされたプログラムは、スレッド化されていないアプリケーションでも使用することができます。しかし、スレッド化されたアプリケーションで COBOL プログラムを実行する場合は、言語環境プログラムのエンクレーブ内のすべての COBOL プログラムが THREAD オプションを指定してコンパイルされている必要があります。

NOTHREAD を指定した場合、その COBOL プログラムは、複数の POSIX スレッドまたは PL/I タスクを含むエンクレーブで実行することができません。

Enterprise COBOL より前のコンパイラーでコンパイルされたプログラムは、NOTHREAD を指定してコンパイルされたものとして処理されます。

THREAD オプションが有効な場合、以下の言語エレメントはサポートされません。これらの言語エレメントが検出された場合は、エラーとして診断されます。

- ALTER ステートメント
- DEBUG-ITEM 特殊レジスター
- プロシージャー名が指定されていない GO TO ステートメント
- RERUN
- STOP リテラル・ステートメント
- 分割モジュール
- USE FOR DEBUGGING ステートメント
- PROGRAM-ID 文節の INITIAL 句
- ネストされたプログラム
- SORT または MERGE ステートメント

また、スレッド化されている場合とスレッド化されていない場合では、一部の言語構成要素のセマンティクスが異なります。

スレッド化されたアプリケーションの場合、プログラミングおよび環境に関するさまざまな制約が適用されますが、スレッド化されていないアプリケーションでのプログラムの使用についてはあまり制約がありません。たとえば、THREAD オプション

を指定してコンパイルされたプログラムは、実行時に複数の POSIX スレッドまたは PL/I タスクがアプリケーションに含まれていない限り、依然として CICS 環境および IMS 環境で実行することができ、AMODE 24 で実行することも、マルチスレッド化をサポートしない他のプログラムとの間で呼び出しを行うこともできます。

THREAD オプションを指定してコンパイルされたプログラムは、言語環境プログラムの事前初期設定ルーチン CEEPIPI を呼び出すことにより作成された再使用可能環境でサポートされます。しかし、IGZERRE または ILBOSTP0 を呼び出すことにより作成された再使用可能環境や、RTEREUS ランタイム・オプションを使用して作成された再使用可能環境は、THREAD オプションを指定してコンパイルされたプログラムをサポートしません。

パフォーマンス: THREAD オプションを使用する場合は、自動的に生成される逐次化ロジックのオーバーヘッドが原因で、実行時のパフォーマンスが多少低下することがあります。

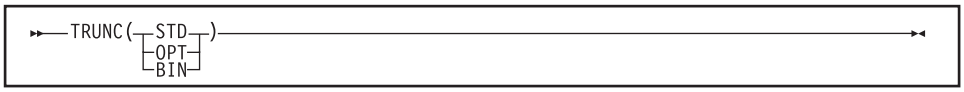
関連タスク

481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照

313 ページの『矛盾するコンパイラー・オプション』

TRUNC



デフォルト: TRUNC(STD)

省略形: なし

TRUNC は、COMP-5 データ項目には効力を持ちません。COMP-5 項目は、TRUNC サブオプションの指定に関係なく、TRUNC(BIN) が有効である場合と同様に処理されます。

TRUNC(STD)

MOVE 操作および算術演算の間の算術フィールドの切り捨て方法を制御する場合に、TRUNC(STD) を使用します。TRUNC(STD) は、MOVE ステートメントおよび算術式の中の USAGE BINARY 受け取りフィールドにのみ適用されます。TRUNC(STD) が有効であると、算術式の最終結果または MOVE 中の送り出しフィールドは、BINARY 受け取りフィールドの PICTURE 文節の桁数に切り捨てられます。

TRUNC(OPT)

TRUNC(OPT) はパフォーマンス・オプションです。TRUNC(OPT) が有効であると、コンパイラーは、データが MOVE ステートメントおよび算術式にある USAGE BINARY 受け取りフィールドの PICTURE の指定に従うものと想定します。結果は最適な方法で処理され、PICTURE 文節の中の桁数か、またはストレージ内の 2 進数フィールドのサイズ (ハーフワード、フルワード、またはダブルワード) に切り捨てられます。

ヒント:

- TRUNC(OPT) オプションを使用するのは、2 進数区域に移動されるデータが、2 進数項目に対する PICTURE 文節で定義された値よりも高い精度の値にならないことが確実である場合に限定してください。そうしないと、結果は予測できません。この切り捨ては、想定される最も効果的な方法で実行されます。そのため、結果は、生成される特定のコード・シーケンスに左右されます。特定のステートメントに対して生成されたコード・シーケンスを見なければ、切り捨ての予想は不可能です。
- Enterprise COBOL のもとで TRUNC(OPT) オプションを指定してコンパイルされたプログラムの結果が、同じプログラムを OS/VS COBOL のもとで NOTRUNC を指定してコンパイルした場合の結果と異なる場合があります。非ゼロの高位桁を実際に失わなければ、この違いは現れません。高位桁が消失すると Enterprise COBOL と OS/VS COBOL の間で結果が異なる可能性のあるステートメントについては、Enterprise COBOL から診断メッセージが出されます。このメッセージを受け取った場合は、送り出し項目に大きな数値が入らないようにするか、または PICTURE 文節で定義された受け取り側が最大送り出しデータ項目を処理できる十分な桁数を持つかどうかを確認してください。

TRUNC(BIN)

TRUNC(BIN) オプションは、USAGE BINARY データを処理するすべての COBOL 言語に適用されます。TRUNC(BIN) が有効な場合、すべての 2 進数項目 (USAGE COMP、COMP-4、または BINARY) は、固有ハードウェア 2 進数項目として、すなわち、それぞれが個々に USAGE COMP-5 と宣言されたものとして処理されます。

- BINARY 受け取りフィールドは、ハーフワード、フルワード、またはダブルワード境界でのみ切り捨てられます。
- BINARY 送り出しフィールドは、受け取り側が数値であれば、ハーフワード、フルワード、またはダブルワードとして処理されます。受け取り側が数値でない場合、TRUNC(BIN) は効力を持ちません。
- フィールドの全 2 進内容が重要です。
- DISPLAY は切り捨てを行わずに、2 進フィールドの内容全体を変換します。

推奨事項: 他のプロダクトによって設定される 2 進値を使用するプログラムの場合、推奨オプションは TRUNC(BIN) です。他のプロダクト (IMS、DB2、C/C++、FORTRAN、および PL/I など) は、COBOL の 2 進数データ項目に、データ項目の PICTURE 文節に従わない値を入れることがあります。データが BINARY データ項目用の PICTURE 文節に矛盾しない場合は、CICS プログラムで TRUNC(OPT) を使用することができます。

インストール先デフォルトとして TRUNC(BIN) を使用することによるパフォーマンスのオーバーヘッドは、非 COBOL プログラムまたは他のプロダクトやサブシステムに渡される個々の 2 進数データ項目に対して COMP-5 を使用すれば、回避することができます。COMP-5 の使用は、どの TRUNC サブオプションが有効であっても影響を受けません。

VALUE 文節における大きなリテラル: コンパイラー・オプション TRUNC(BIN) を使用する場合、2 進数データ項目 (COMP、COMP-4、または BINARY) 用の VALUE 文節に指定された数字リテラルは、PICTURE 文節の 9 の数により暗黙指定される値に制限されることはなく、通常、固有 2 進表現 (2、4、または 8 バイト) の容量までの大きさの値を持つことができます。

TRUNC の例 1

```
01  BIN-VAR     PIC 99 USAGE BINARY.
. . .
    MOVE 123451 to BIN-VAR
```

次の表に、MOVE 後のデータ項目の値を示しています。

データ項目	10 進数	16 進数	Display の結果
送り出し側	123451	00|01|E2|3B	123451
受け取り側 TRUNC(STD)	51	00|33	51
受け取り側 TRUNC(OPT)	-7621	E2|3B	2J
受け取り側 TRUNC(BIN)	-7621	E2|3B	762J

ハーフワードのストレージが BIN-VAR に割り振られます。プログラムが TRUNC(STD) オプションでコンパイルされた場合は、この MOVE ステートメントの結果は 51 で、フィールドは、PICTURE 文節に適合するように切り捨てられます。

プログラムが TRUNC(BIN) オプションでコンパイルされた場合、MOVE ステートメントの結果は -7621 です。このような異常に見える結果になるのは、非ゼロの高位桁が切り捨てられたためです。ここでは、生成されたコード・シーケンスは、下位のハーフワード量 X'E23B' を受け取り側に移動させるだけです。切り捨てられた新しい値はオーバーフローして 2 進数ハーフワードの符号ビットになるため、値が負の数になります。

123451 は BIN-VAR の PICTURE 文節より高い精度を持つため、この MOVE ステートメントを TRUNC(OPT) オプションでコンパイルしてはなりません。TRUNC(OPT) を使用した場合も、結果は -7621 になります。これは、10 進数の切り捨てを行わないことによって、最高のパフォーマンスが得られたからです。

TRUNC の例 2

```
01  BIN-VAR     PIC 9(6)  USAGE BINARY
. . .
    MOVE 1234567891 to BIN-VAR
```

次の表に、MOVE 後のデータ項目の値を示しています。

データ項目	10 進数	16 進数	Display の結果
送り出し側	1234567891	49|96|02|D3	1234567891

データ項目	10 進数	16 進数	Display の結果
受け取り側 TRUNC(STD)	567891	00|08|AA|53	567891
受け取り側 TRUNC(OPT)	567891	53|AA|08|00	567891
受け取り側 TRUNC(BIN)	1234567891	49|96|02|D3	1234567891

TRUNC(STD) を指定すると、送り出しデータは BINARY 受け取り側の PICTURE 文節に適合するように、6 桁の整数に切り捨てられます。

TRUNC(OPT) を指定すると、コンパイラーは送り出しデータの精度が BINARY 受け取り側の PICTURE 文節の精度よりも大きくないと想定します。この場合、最も効率のよいコード・シーケンスは、TRUNC(STD) が指定されているものとして切り捨てを行うことです。

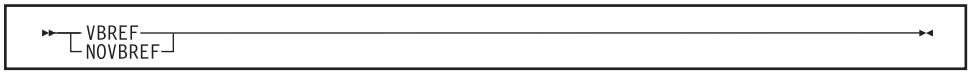
TRUNC(BIN) を指定すると、BIN-VAR に割り振られた 2 進数フルワードにすべての送り出しデータが収まるため、切り捨ては行われません。

関連概念
46 ページの『数値データの形式』

関連タスク
411 ページの『CICS オプションを使用したコンパイル』

関連参照
VALUE 文節 (*Enterprise COBOL 言語解説書*)

VBREF



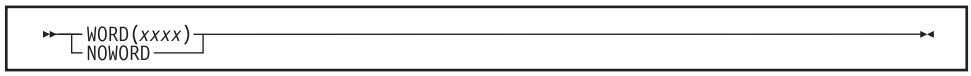
デフォルト: NOVBREF

省略形: なし

VBREF は、ソース・プログラムの中で使用されるすべての動詞タイプ、およびこれらの動詞が使用されている行番号の相互参照を入手するために使用します。また、VBREF はプログラムの中でそれぞれの動詞が使用された回数の合計も出します。

コンパイルの効率を高める場合は、NOVBREF を使用してください。

WORD



デフォルト: NOWORD

省略形: WD|NOWD

xxxx には、コンパイルで使用する予約語テーブル (IGYC*xxxx*) の名前の後半の文字を指定します。IGYC は名前の最初の部分の標準 4 文字で、*xxxx* は 1 ～ 4 文字の長さにすることができます。

WORD(*xxxx*) は、コンパイル時に代替予約語テーブルを使用することを指定するために使用します。

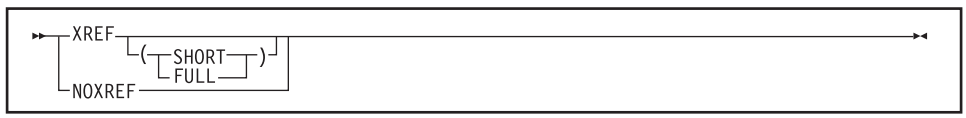
代替予約語テーブルを使用して、IBM 提供のデフォルトの予約語テーブルに対する変更を行います。システム・プログラマーが、設置場所に対して 1 つまたは複数の代替予約語テーブルを作成している可能性があります。代替予約語テーブルの名前については、システム・プログラマーにお問い合わせください。

Enterprise COBOL には、CICS アプリケーション専用の代替予約語テーブル (IGYCCICS) が用意されています。エラー・メッセージを使用して、CICS でサポートされない COBOL のワードにフラグを付けるようにセットアップされています。コンパイル時にこの CICS 予約語テーブルを使用したい場合は、コンパイラー・オプション WORD(CICS) を指定してください。

関連タスク
411 ページの『CICS オプションを使用したコンパイル』

関連参照
313 ページの『矛盾するコンパイラー・オプション』

XREF



デフォルト: XREF

省略形: X|NOX

XREF、XREF(FULL)、または XREF(SHORT) を選択できます。

XREF は、ソート済みの相互参照を入手するために使用します。EBCDIC データ名とプロシージャー名は英数字順にリストされます。DBCS データ名とプロシージャー名は、プログラムでの物理的順序に基づいてリストされ、DBCSXREF インストール・オプションが日本語配列プログラムで選択された場合を除いて、EBCDIC データ名

とプロシージャー名の前に表示されます。DBCSXREF オプションが選択された場合は、DBCS データ名とプロシージャー名は、日本語配列プログラムで指定されたとおりに順序付けられます。

プログラム内で参照されるすべてのプログラム名、およびそれらが定義されている行番号をリストしているセクションも含まれます。外部プログラム名はそれ自体で識別されます。

XREF と SOURCE を使用した場合は、相互参照がリスト中の元のソースと同じ行に印刷されます。行番号参照またはその他の情報は、リスト・ページの右側に表示されます。組み込み関数を参照するソース行の右側には、IFN という文字と、その関数の引き数が定義されている行番号が示されます。組み込み参照に含められた情報によって、ID が未定義であるか、それとも 2 回以上定義されているか (UND または DUP が印刷される)、特殊レジスターや形象定数のように項目が暗黙定義であるかどうか (IMP)、およびプログラム名が外部であるかどうか (EXT) がわかります。

XREF と NOSOURCE を使用すると、ソート済みの相互参照リストだけが得られます。

XREF(SHORT) は、相互参照リスト内の明示的に参照された変数だけを印刷します。XREF(SHORT) は、DBCS データ名とプロシージャー名、および EBCDIC 名に適用されます。

NOXREF を使用すると、このリストは抑止されます。

注意:

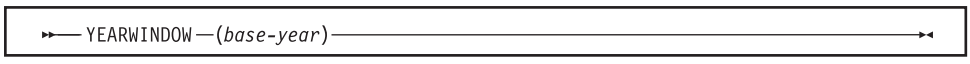
- MOVE CORRESPONDING ステートメントで使用されるグループ名は、XREF リストに入れられます。さらに、それらのグループの基本名もリストされています。
- データ名の XREF リストでは、文字 M が前に付いている行番号は、そのデータ項目がその行のステートメントによって明示的に変更されたことを示しています。
- XREF リストは追加のストレージを使用します。

関連概念
365 ページの『第 19 章 デバッグ』

関連タスク
377 ページの『リストの入手』

関連参照
COBOL コンパイラー・オプション (*言語環境プログラム デバッグのガイド*)

YEARWINDOW



デフォルト: YEARWINDOW(1900)

省略形: YW

YEARWINDOW オプションを使用すると、COBOL コンパイラーによるウィンドウ化日付フィールド処理に適用する 100 年ウィンドウ (*世紀ウィンドウ*) の最初の年を指定することができます。

base-year は、100 年ウィンドウの最初の年を表し、次のいずれかとして指定しなければなりません。

- 1900 ～ 1999 の間の符号なし 10 進数

  これは固定ウィンドウの開始年号を指定します。たとえば、YEARWINDOW(1930) は 1930 ～ 2029 年の世紀ウィンドウを指定します。
- -1 ～ -99 の負の整数

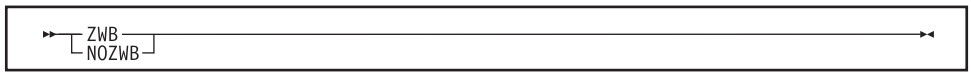
  これは、スライディング・ウィンドウを指定するもので、ウィンドウの最初の年号は現在の実行時日付から計算されます。この数値が現在の年号から減算され、世紀ウィンドウの開始年号が与えられます。たとえば、YEARWINDOW(-80) は、世紀ウィンドウの最初の年がプログラム実行時点の年号より 80 年前であることを指定します。

使用上の注意

- YEARWINDOW オプションは、DATEPROC オプションも有効でない限り、効力を持ちません。
- 実行時には、次の 2 つの条件が真でなければなりません。
  - 世紀ウィンドウの開始年号が 1900 年代の年号である。
  - 現在の年号がコンパイル単位の世紀ウィンドウ内にある。

  たとえば、現在の年号が 2001 年で、DATEPROC オプションが有効な場合に、YEARWINDOW(1900) というオプションを指定すると、プログラムは終了し、エラー・メッセージが出されます。

ZWB



デフォルト: ZWB

省略形: なし

ZWB を指定すると、コンパイラーは、実行時に符号付き外部 10 進数 (DISPLAY) フィールドを英数字基本フィールドと比較するとき、この外部 10 進数フィールドから符号を取り除きます。

外部 10 進数項目がスケーリングされた項目 (その PICTURE 文字ストリングに記号 P が入っている) の場合は、それを比較に使用しても ZWB の影響を受けません。そのような項目では常に、英数字フィールドとの比較が行われる前に符号が除去されます。

ZWB はプログラムの実行方法に影響を与えます。オプションの設定によっては、同じ COBOL ソース・プログラムが別の結果を生じさせることがあります。

NOZWB は、入力数字フィールドで SPACES をテストする場合に使用します。

コンパイラー指示ステートメント

プログラムのコンパイルを指示するのに役立つステートメントがいくつかあります。

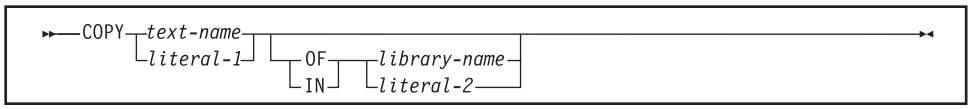
BASIS ステートメント

この拡張ソース・プログラム・ライブラリー・ステートメントは、コンパイルのソースとして、完全な COBOL プログラムを提供します。形式化および処理の規則については、COPY ステートメントの text-name の説明を参照してください。

***CONTROL (*CBL) ステートメント**

このコンパイラー指示ステートメントは、出力の作成を抑制するかまたは可能にするかを選択します。名前の *CONTROL と *CBL は同義語です。

COPY ステートメント



このライブラリー・ステートメントは、事前に作成されたテキストを COBOL プログラムに入れます。ユーザー定義語を *text-name* または *library-name* と同じにすることができます。*text-name* および *library-name* の固有性は、システム依存名の形式化および変換の規則が適用された後で決められます。*library-name* を省略した場合は、SYSLIB と見なされます。

JCL によってコンパイルするとき:

text-name、*library-name*、および *literal* は、次のように処理されます。

- 名前 (1 文字から 30 文字までの長さ) は 8 文字に切り捨てられます。*text-name* および *library-name* の最初の 8 文字だけが、識別名として使用されます。これらの 8 文字は、1 つの COBOL ライブラリー内では固有でなければなりません。
- 名前は大文字に変換されます。
- 先頭文字でも最後の文字でもないハイフンはゼロ (0) に変換され、警告メッセージが出されます。
- 先頭文字が数字であると、1 ～ 9 の文字は A ～ I に変換され、警告メッセージが出されます。先頭文字または最後の文字が、ハイフンであると (リテラルの場合は、ハイフン、@、#、または $ であると)、エラー・メッセージが出されます。

たとえば、次のようになります。

```
COPY INVOICES1Q
COPY "Company-#Employees" IN Personellib
```

IN/OF 句の中の *library-name* は、コピー元の区分データ・セットを識別する DD 名です。次の例に示すように、DD ステートメントを使用して、*library-name* を定義します。

```
//COPYLIB     DD    DSNAME=ABC.COB,VOLUME=SER=111111,
//                  DISP=SHR,UNIT=3380
```

複数のコピー・ライブラリーを指定するには、JCL または JCL と IN/OF 句の組み合わせのいずれかを使用してください。JCL だけを使用する場合は、SYSLIB に対する DD ステートメント上でデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントに IN/OF 句を組み込みます。

コピー・ライブラリーの最大ブロック・サイズは、データ・セットが常駐している装置によって決まります。

UNIX システム・サービス・シェルでコンパイルするとき:

cob2 コマンドを使用してコンパイルすると、コピーブックは HFS から取り込まれます。*text-name*、*library-name*、および *literal* は、次のように処理されます。

- ユーザー定義語は大文字に変換されます。リテラルは変換されません。UNIX は大文字小文字を区別するので、ファイル名が小文字であるか大文字小文字混合である場合は、それをリテラルとして指定します。
- *text-name* がリテラルのとき、*library-name* が省略されていると、*text-name* は直接的に使用されます。すなわち、ファイル名、相対パス名、または絶対パス名 (先頭文字が / の場合) として使用されます。たとえば、次のようになります。

  ```
  COPY "MyInc"
  COPY "x/MyInc"
  COPY "/u/user1/MyInc"
  ```

- *text-name* がユーザー定義語で、その名前の環境変数が定義されていると、その環境変数の値が、コピーブックを含むファイルの名前として使用されます。

  その名前の環境変数が定義されていない場合には、コピーブックは、与えられた順序で、以下の名前として探索されます。

  1. *text-name*.cpy
  2. *text-name*.CPY
  3. *text-name*.cbl
  4. *text-name*.CBL
  5. *text-name*.cob
  6. *text-name*.COB
  7. *text-name*

- *library-name* がリテラルである場合は、それはコピー・ファイル *text-name* までの実際のパス (相対または絶対) として扱われます。
- *library-name* がユーザー定義語であると、それは環境変数として扱われます。その環境変数の値がパスとして使用されます。環境変数が設定されていないと、エラーとなります。
- *library-name* と *text-name* の両方が指定された場合、コンパイラーは、*library-name* と *text-name* の間にパス区切り文字 (/) を入れて 2 つの値

を連結することによって、コピーブックのパス名を形成します。たとえば、COPY MYCOPY OF MYLIB が次のように設定されているとします。

```
export MYCOPY=mystuff/today.cpy
export MYLIB=/u/user1
```

この設定の結果、次のようになります。

```
/u/user1/mystuff/today.cpy
```

library-name が環境変数で、この環境変数の値がコピーブックのコピー元のパスを識別する場合は、次の例に示すように、export コマンドを使用して *library-name* を定義します。

```
export COPYLIB=/u/mystuff/copybooks
```

環境変数の名前は大文字でなければなりません。複数のコピー・ライブラリーを指定するには、環境変数を、: (コロン) で区切られたマルチパス名に設定してください。

library-name が省略されたときに、*text-name* が絶対パス名でないと、コピーブックは、次の順序で探索されます。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

DELETE ステートメント
この拡張ソース・ライブラリー・ステートメントは、BASIS ソース・プログラムから COBOL ステートメントを除去します。

EJECT ステートメント
このコンパイラー指示ステートメントは、次のソース・ステートメントが次のページの最上部に印刷されるように指定します。

ENTER ステートメント
コンパイラーはこのステートメントをコメントとして処理します。

INSERT ステートメント
このライブラリー・ステートメントは、COBOL ステートメントを BASIS ソース・プログラムに追加します。

PROCESS (CBL) ステートメント
このステートメントは、最も外側の IDENTIFICATION DIVISION ヘッダーの前に置かれるもので、プログラムのコンパイル時にどのコンパイラー・オプションが使用されるのかを示します。

REPLACE ステートメント
このステートメントは、ソース・プログラム・テキストを置き換えるために使用されます。

SERVICE LABEL ステートメント
このステートメントは、制御の流れを示すために CICS トランスレーターによって生成されます。一般的に使用されるものではありません。

SKIP1/2/3 ステートメント
このステートメントは、ある行がソース・リストでスキップされることを示します。

TITLE ステートメント

このステートメントは、ソース・リストの各ページの最上部に表題 (ヘッダー) が印刷されるように指定します。

USE ステートメント

USE ステートメントは、以下のものを指定するための宣言 を提供します。

- エラー処理プロシージャー: EXCEPTION/ERROR
- ユーザー・ラベル処理プロシージャー: LABEL
- デバッグ行およびセクション: DEBUGGING

関連タスク

9 ページの『ソース・リストのヘッダーの変更』
609 ページの『反復コーディングの除去』
285 ページの『z/OS のもとでのコンパイラー・オプションの指定』
300 ページの『UNIX のもとでのコンパイラー・オプションの指定』
299 ページの『UNIX のもとでの環境変数の設定』

関連参照

303 ページの『cob2』
Enterprise COBOL 言語解説書

第 19 章 デバッグ

アプリケーションのプログラム動作における問題の原因を判別するには、ソース言語デバッグと対話式デバッガーの 2 つの方法を使用することができます。

ソース言語デバッグの場合、COBOL は、デバッグを容易にするいくつかの言語エレメント、コンパイラー・オプション、およびリスト出力を提供します。

プログラムの問題が容易に検出されず、利用できるデバッガーがない場合、プログラムのストレージ・ダンプを分析する必要があります。

COBOL に固有の機能を使用するほか、デバッグ・ツール (このコンパイラーの全機能オファリングで使用可能) を利用することもできます。

デバッグ・ツールは、次のような生産性拡張機能を提供します。

- 対話式デバッグ (フルスクリーン・モードまたは行モード) またはバッチ・モード

  対話式フルスクリーン・モード・セッション中に、デバッグ・ツールのフルスクリーン・サービスと 3270 装置上のセッション・パネル・ウィンドウを使用して、実行中のプログラムをデバッグすることができます。
- COBOL 類似コマンド

  サポートされる高水準言語ごとに、ブレークポイントで取るべきアクションを指定するコマンドが、そのプログラム言語に類似した構文で提供されます (この機能はワークステーション・ユーザーには利用不能です)。
- 混合言語デバッグ

  異なる言語で作成されたプログラムが入っているアプリケーションをデバッグすることができます。デバッグ・ツールが、実行中のプログラムまたはサブプログラムの言語を自動的に判別します。
- COBOL-CICS デバッグ

  デバッグ・ツールは、対話式モードとバッチ・モードの両方で CICS アプリケーションのデバッグをサポートします。
- リモート・デバッグのサポート

  ワークステーション・ユーザーは、IBM VisualAge COBOL 製品を使用して、z/OS に常駐するプログラムをデバッグすることができます。VisualAge COBOL は、別個の製品として、またはこのコンパイラーのエンタープライズ・ワークステーション機能として使用できます。

関連タスク

366 ページの『ソース言語によるデバッグ』
370 ページの『コンパイラー・オプションを使用したデバッグ』
377 ページの『リストの入手』
402 ページの『デバッガーの使用準備』
デバッグ・ツール ユーザーズ・ガイドおよびリファレンス

関連参照

Debug Tool Reference Manual and Messages

© Copyright IBM Corp. 1991, 2001

システム・ダンプのフォーマット設定および分析 (*言語環境プログラム デバッグのガイド*)
COBOL プログラムのデバッグ例 (*言語環境プログラム デバッグのガイド*)

ソース言語によるデバッグ

さまざまな COBOL 言語機能を使用して、プログラムの障害の原因を正確に示すことができます。プログラムがすでに実動中の大規模なアプリケーションの一部である場合 (ソース更新を除外する) は、プログラムの障害部分をシミュレートするような小さいテスト・ケースを作成してください。テスト・ケースでは、以下の問題の検出に役立つようなデバッグ機能をコーディングしてください。

- プログラム・ロジックのエラー
- 入出力エラー
- データ型のミスマッチ
- 初期化されていないデータ
- プロシージャーの問題

関連タスク
『プログラム・ロジックのトレース』
367 ページの『入出力エラーの検出および処理』
367 ページの『データの妥当性検査』
368 ページの『初期化されていないデータの検出』
368 ページの『プロシージャーに関する情報の生成』

関連参照
ソース言語のデバッグ (*Enterprise COBOL 言語解説書*)

プログラム・ロジックのトレース

プログラムの論理は、DISPLAY ステートメントを追加することによってトレースしてください。たとえば、問題が EVALUATE ステートメントまたは 1 組のネストされた IF ステートメントにあると判断した場合は、それぞれのパスで DISPLAY ステートメントを使用して、ロジック・フローを調べます。問題の原因が数値の計算方法にあると判断した場合は、DISPLAY ステートメントを使用して、いくつかの中間結果の値を検査することができます。

プログラムの中で明示的範囲終了符号を使用してステートメントを終了させていた場合は、プログラムの論理はより明確であり、したがってトレースしやすくなります。

たとえば、特定のルーチンが開始して終了したかどうかを判別する場合は、プログラムに次のようなコードを挿入してみてください。

```
DISPLAY "ENTER CHECK PROCEDURE"
    .
    .  (checking procedure routine)
    .
DISPLAY "FINISHED CHECK PROCEDURE"
```

ルーチンが正しく作動していることを確認したら、次のいずれかの方法で DISPLAY ステートメントを使用不可にします。

- 各 DISPLAY ステートメントの行の 7 桁目にアスタリスクを置き、コメント行に変換する。
- 各 DISPLAY ステートメントの 7 桁目に D を置き、コメント行に変換する。これらのステートメントを再活動化したい場合は、ENVIRONMENT DIVISION に WITH DEBUGGING MODE 文節を含めると、7 桁目の D は無視され、DISPLAY ステートメントが実施されます。

プログラムを実動に移す前に、使用したすべてのデバッグ・エイドを削除するか使用不可にしてから、プログラムを再コンパイルします。プログラムはより効果的に実行され、使用するストレージは小さくなります。

関連概念
24 ページの『範囲終了符号』

関連参照
DISPLAY ステートメント (*Enterprise COBOL 言語解説書*)

入出力エラーの検出および処理

ファイル状況キーは、プログラムのエラーが、ストレージ・メディアで起こっている入出力エラーによるものかどうかを判別するのに役立ちます。

ファイル状況キーをデバッグ・エイドとして使用するためには、各入出力ステートメントの後にテストを組み込み、状況キーの値が 0 以外かどうか検査します。値が 0 以外である (エラー・メッセージで報告される) 場合には、プログラム内の入出力プロシージャーのコーディングを調べる必要があります。状況キーの値に基づいてエラーを訂正するためのプロシージャーを組み込むこともできます。

問題がプログラムの入出力プロシージャーにあると判断した場合は、USE EXCEPTION/ERROR 宣言を組み込んで、問題のデバッグに役立てることができます。その後、ファイルのオープンに失敗すると、適切な EXCEPTION/ERROR 宣言が実行されます。適切な宣言とは、ファイルに固有なもの、あるいはオープン属性 (INPUT、OUTPUT、I-O、または EXTEND) 用に提供されたものです。

それぞれの USE AFTER STANDARD ERROR ステートメントは、PROCEDURE DIVISION の DECLARATIVE SECTION キーワードの直後のセクションにコーディングします。

関連タスク
253 ページの『ERROR 宣言のコーディング』

関連参照
状況キーの値と意味 (*Enterprise COBOL 言語解説書*)
状況キー (*Enterprise COBOL 言語解説書*)

データの妥当性検査

プログラムが非数値データに算術演算を実行しようとしているか、または入力レコードで間違った型のデータを受け取ろうとしていると思われる場合は、クラス・テストを使用して、データの型を妥当性検査してください。クラス・テストは、データが英字、英小文字、英大文字、DBCS、KANJI、または数字であるかどうかを検査します。

関連参照
クラス条件 (*Enterprise COBOL 言語解説書*)

初期化されていないデータの検出

問題の原因がテーブルまたは変数のフィールドに残された残余データにあると考えられるときは、INITIALIZE または SET ステートメントを使用して、テーブルまたは変数を初期化してください。

問題が起きたり起きなかったりし、しかも同一のデータで起きるとは限らない場合、その問題の原因は、スイッチが初期化されていないが多くの場合正しい値 (0 または 1) に偶然に設定されることであると考えられます。SET ステートメントを組み込んでスイッチを初期化するようにすれば、初期化されていないスイッチが問題であると判断できるか、考えられる原因からそのスイッチを除外することができます。

関連参照
INITIALIZE ステートメント (*Enterprise COBOL 言語解説書*)
SET ステートメント (*Enterprise COBOL 言語解説書*)

プロシージャーに関する情報の生成

プログラムまたはテスト・ケースに関する情報、およびその実行方法に関する情報は、USE FOR DEBUGGING 宣言を使用して生成してください。この宣言を使用すると、ステートメントをプログラムに組み込んで、プログラムの実行時にいつそれらのステートメントを実行しなければならないかを指示することができます。

たとえば、プロシージャーの実行回数を調べたい場合は、USE FOR DEBUGGING 宣言にデバッグ・プロシージャーを組み込んで、カウンターを使用して、制御権が何回そのプロシージャーに渡されたかを記録しておくことができます。カウンター技法を使用して、次のような項目を検査できます。

- PERFORM が実行される回数。特定のルーチンが使用されているかどうか、および制御構造が正しいかどうか。
- ループ・ルーチンが実行される回数。ループが実行されているかどうか、およびループの回数が正確かどうか。

プログラムに、デバッグ行かデバッグ・ステートメント、またはその両方を入れることができます。

デバッグ行

デバッグ行とは、7 桁目に D があることで識別されるステートメントのことです。プログラムのデバッグ行をアクティブにするには、ENVIRONMENT DIVISION の SOURCE-COMPUTER 行に WITH DEBUGGING MODE 文節を含める必要があります。この文節が含まれていないと、デバッグ行はコメントとしてしか扱われません。

デバッグ・ステートメント

デバッグ・ステートメントとは、PROCEDURE DIVISION の DECLARATIVES SECTION にコーディングされたステートメントです。それぞれの USE FOR DEBUGGING 宣言は別個のセクションにコーディングしなければなりません。 デバッグ・ステートメントは、次のようにコーディングしてください。

- DECLARATIVES SECTION 内のみ。
- ヘッダー USE FOR DEBUGGING の後に置く。
- 最外部のプログラムだけに置く (ネストされたプログラムでは無効です)。デバッグ・ステートメントが、ネストされたプログラムに含まれているプロシージャーによって開始されることはありません。

プログラムでデバッグ・ステートメントを使用するには、WITH DEBUGGING MODE 文節を指定し、さらに、DEBUG ランタイム・オプションを使用する必要があります。ただし、THREAD オプションを指定してコンパイルするプログラムでは、USE FOR DEBUGGING 宣言を使用できません。

WITH DEBUGGING MODE 文節と TEST コンパイラー・オプション (NONE 以外のサブオプション値が指定されたもの) は互いに排他的です。両方が存在する場合は、WITH DEBUGGING MODE 文節が優先されます。

『例: USE FOR DEBUGGING』

関連参照
デバッグ行 (*Enterprise COBOL 言語解説書*)
デバッグ・セクションのコーディング (*Enterprise COBOL 言語解説書*)
DEBUGGING 宣言 (*Enterprise COBOL 言語解説書*)

例: USE FOR DEBUGGING

以下のプログラム・セグメントは、DISPLAY ステートメントと USE FOR DEBUGGING 宣言を使ってプログラムをテストする際に必要となるステートメントの種類を示しています。DISPLAY ステートメントは、端末上または出力データ・セットに情報を生成します。USE FOR DEBUGGING 宣言は、ルーチンが実行される回数を示すカウンターと一緒に使用されます。

```
Environment Division.
   .
   .
   .
Data Division.
   .
   .
   .
Working-Storage Section.
   .
   .   (other entries your program needs)
   .
01  Trace-Msg    PIC X(30) Value "  Trace for Procedure-Name : ".
01  Total        PIC 9(9)  Value 1.
   .
   .
   .
Procedure Division.
Declaratives.
Debug-Declaratives Section.
    Use For Debugging On Some-Routine.
Debug-Declaratives-Paragraph.
    Display Trace-Msg, Debug-Name, Total.
End Declaratives.

Main-Program Section.
   .
   .  (source program statements)
   .
  Perform Some-Routine.
   .
   .  (source program statements)
   .
  Stop Run.
Some-Routine.
   .
   .  (whatever statements you need in this paragraph)
   .
  Add 1 To Total.
Some-Routine-End.
```

この例では、プロシージャー Some-Routine が実行されるたびに、DECLARATIVES SECTION の DISPLAY ステートメントがこのメッセージを出します。

```
Trace For Procedure-Name : Some-Routine 22
```

メッセージの終わりにある番号 22 は、データ項目 Total の累算された値で、Some-Routine が実行された回数を示しています。デバッグ宣言内のステートメントは、名前を指定されたプロシージャーが実行される前に実行されます。

DISPLAY ステートメントを使用して、プログラムの実行をトレースし、プログラム中のフローを示すこともできます。これを行うには、DISPLAY ステートメントから Total を除去し、DECLARATIVES SECTION の USE FOR DEBUGGING を次のように変更します。

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

これで、最外部プログラムのそれぞれの非デバッグ・プロシージャーが実行される前にメッセージが表示されるようになります。

コンパイラー・オプションを使用したデバッグ

特定のコンパイラー・オプションを使用すると、プログラムの次のようなエラーを検出するのに役立つ情報を生成することができます。

- 重複データ名のような構文エラー (NOCOMPILE)
- セクションの欠落 (SEQUENCE)
- 無効な添え字値 (SSRANGE)

さらに、一部のコンパイラー・オプションを使用すると、プログラムの次のようなエレメントを見つけやすくなります。

- エラー・メッセージとエラーの発生場所 (FLAG)
- プログラム・エンティティー定義および参照 (XREF)
- DATA DIVISION に定義された項目 (MAP)
- 動詞参照 (VBREF)
- アセンブラー言語拡張 (LIST)

ソースのコピー (SOURCE コンパイラー・オプションを使用して) または生成されたコードのリスト (LIST コンパイラー・オプションを使用して) を入手することができます。

プログラムをデバッグ用に準備するために使用する必要のあるコンパイラー・オプション (TEST) もあります。

関連タスク
373 ページの『診断するエラーのレベルの選択』
『コーディング・エラーの検出』
372 ページの『行シーケンス問題の検出』
372 ページの『有効範囲の検査』
375 ページの『プログラム・エンティティー定義および参照の検出』
376 ページの『データ項目のリスト』
377 ページの『リストの入手』
402 ページの『デバッガーの使用準備』

関連参照
319 ページの『COMPILE』
345 ページの『SEQUENCE』
348 ページの『SSRANGE』
328 ページの『FLAG』
357 ページの『XREF』
333 ページの『MAP』
356 ページの『VBREF』
333 ページの『LIST』
349 ページの『TEST』

コーディング・エラーの検出

NOCOMPILE コンパイラー・オプションは、条件付きコンパイルまたは構文検査のみを行う場合に使用します。SOURCE オプションと一緒に使用されると、このオプションは、COBOL コーディングの間違い (欠落している定義、正しく定義されていないデータ項目、重複するデータ名など) を見つけるのに役立つリストを作成します。

TSO フォアグラウンドでコンパイルしている場合は、データ・セットを SYSTERM データ・セットとして定義し、コンパイル時に TERM オプションを使用することによって、メッセージを画面に送ることができます。

構文検査のみ

パラメーターなしで NOCOMPILE を使用すると、コンパイラーは、ソース・プログラムの構文検査だけを行い、オブジェクト・コードを作成しません。一緒に SOURCE オプションを指定すると、コンパイラーは、構文検査の完了後にリストを作成します。

以下のコンパイラー・オプションは、パラメーターなしで NOCOMPILE を使用すると抑止されます。DECK、OFFSET、LIST、OBJECT、OPTIMIZE、SSRANGE、および TEST。

条件付きコンパイル

NOCOMPILE(*x*) (*x* はエラーの重大度レベルの 1 つ) を使用すると、プログラムがコンパイルされるのは、すべてのエラーの重大度が *x* レベルより低い場合です。使用できる重大度レベル (最高位から最低位へ) は、S (重大)、E (エラー)、および W (警告) です。

x レベルまたはそれより高いレベルのエラーが発生すると、コンパイルは停止し、プログラムの構文検査のみが行われます。SOURCE オプションを指定していた場合には、ソース・リストを受け取ります。

関連参照

319 ページの『COMPILE』

行シーケンス問題の検出

順序どおりになっていないステートメントを見つけるには、SEQUENCE コンパイラー・オプションを使用してください。このようなシーケンス中断は、ソース・プログラムのセクションが移動または削除されたことを表します。

SEQUENCE を使うと、コンパイラーはユーザー提供のソース・ステートメント番号を検査し、昇順になっているかどうかを調べます。順序どおりになっていないステートメント番号の横には、2 つのアスタリスクが入れられます。さらに、これらのステートメントの合計数がソース・リストに続く診断の最初の行として印刷されます。

関連参照

345 ページの『SEQUENCE』

有効範囲の検査

以下の範囲を検査するには、SSRANGE コンパイラー・オプションを使用してください。

- 添え字付きまたは指標付きデータ参照

  所要のエレメントの有効アドレスが指定されたテーブルの最大境界内にあるかどうか。
- 可変長データ参照（OCCURS DEPENDING ON 文節を含むデータ項目への参照）

  実際の長さが正であるかどうか、そしてグループ・データ項目に対して定義された最大長より短いかどうか。
- 参照変更データ参照

オフセットと長さが正であるかどうか。オフセットと長さの合計がデータ項目の最大長より短いかどうか。

SSRANGE オプションが指定された場合、以下の両方が真であれば、実行時に検査が行われることになります。

- 指標付き、添え字付き、可変長、または参照変更データ項目が含まれている COBOL ステートメントが実際に実行される。
- CHECK ランタイム・オプションが ON である。

検査で、参照されたデータが入っているデータ項目のアドレス範囲外にあるアドレスが生成されていることが検出されると、エラー・メッセージが生成され、プログラムは実行を停止します。エラー・メッセージでは、参照されたテーブルまたは ID、およびエラーが発生した行番号が識別されます。エラーの原因となった参照のタイプによっては、追加情報が提供されます。

すべての添え字、指標、または参照修飾子が特定のデータ参照でリテラルであり、データ項目の外側を参照する結果になる場合は、SSRANGE コンパイラー・オプションの設定値に関係なく、エラーはコンパイル時に診断されます。

パフォーマンスの考慮: SSRANGE が指定されると、プログラムのパフォーマンスは少し低下することがあります。これは、それぞれの添え字付きまたは指標付き項目を検査するために必要なオーバーヘッドが余分にかかるためです。

関連参照
348 ページの『SSRANGE』
603 ページの『パフォーマンスに関連するコンパイラー・オプション』

診断するエラーのレベルの選択

FLAG コンパイラー・オプションを使用すると、コンパイル時に診断するエラーのレベルを選択すること、および構文エラー・メッセージをリストに組み込みかどうかを指示することができます。プログラム中のすべてのエラーが通知されるようにするには、FLAG(I) または FLAG(I,I) を使ってください。

最初のパラメーターには、発行される構文エラー・メッセージのうち最も重大度レベルの低いものを指定してください。オプションとして、2 番目のパラメーターには、ソース・リストに組み込む構文メッセージのうち最も重大度レベルの低いものを指定します。この重大度レベルは、最初のパラメーターのレベルと同じかそれ以上でなければなりません。両方のパラメーターを指定する場合は、一緒に SOURCE コンパイラー・オプションも指定する必要があります。

重大度レベル	対応するレベルを指定したときに入手できるもの
U (回復不能)	U (メッセージのみ)
S (重大)	すべての S および U メッセージ
E (エラー)	すべての E、S、および U メッセージ
W (警告)	すべての W、E、S、および U メッセージ
I (通知)	すべてのメッセージ

2 番目のパラメーターを指定すると、コンパイラーがエラーを検出するために利用できる情報が十分ある時点で、構文エラー・メッセージ (U レベルのメッセージを除く) がソース・リストに組み込まれます。ライブラリー・コンパイラー・フェーズで出されるメッセージ以外のすべての組み込みメッセージは、それらが参照するステートメントの直後に続きます。エラーがあるステートメントの番号もメッセージに示されます。組み込みメッセージは、ソース・リストの終わりに出される残りのソース・リストで繰り返されます。

NOSOURCE コンパイラー・オプションを指定した場合は、構文エラー・メッセージはリストの終わりにだけ入れられます。回復不能エラーに関するメッセージはソース・リストに組み込まれません。この重大度のエラーはコンパイルを終了させるからです。

『例: 組み込みメッセージ』

関連概念
296 ページの『コンパイル・エラー・メッセージの重大度コード』

関連タスク
294 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照
328 ページの『FLAG』
294 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

例: 組み込みメッセージ

次の例は、FLAG オプションに 2 番目のパラメーターを指定することによって生成される組み込みメッセージを示しています。要約の中のメッセージのいくつかは複数の COBOL ステートメントに適用されることに注意してください。

```
DATA VALIDATION AND UPDATE PROGRAM               IGYTCARA  Date 10/11/2001  Time 12:26:53    Page    27
  LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+  Map and Cross Reference
.
.
.
  090671**             /
  090672**             ****************************************************************
  090673**             ***         I N I T I A L I Z E   P A R A G R A P H          **
  090674**             ***  Open files. Accept date, time and format header lines.    **
  090675**             ***   Load location-table.                                     **
  090676**             ****************************************************************
  090677**                1OO-initialize-paragraph.
  090678**                  move spaces to ws-transaction-record                                IMP 331
  090679**                  move spaces to ws-commuter-record                                   IMP 307
  090680**                  move zeroes to commuter-zipcode                                     IMP 318
  090681**                  move zeroes to commuter-home-phone                                  IMP 319
  090682**                  move zeroes to commuter-work-phone                                  IMP 320
  090683**                  move zeroes to commuter-update-date                                 IMP 324
  090684**                  open input update-transaction-file                                  204
==090684==> IGYPS2052-S An error was found in the definition of file "LOCATION-FILE".  The
                        reference to this file was discarded.
  090685**                       location-file                                                  193
  090686**                       i-o commuter-file                                              181
  090687**                       output print-file                                              217
  090688**                  if commuter-file-status not = "00" and not = "97"                   241
  090689**    1               display "1OO-OPEN"
  090690**    1               move 1OO to comp-code                                           231
  090691**    1               perform 5OO-vsam-error                                          91069
  090692**    1               perform 9OO-abnormal-termination                                91114
  090693**                  end-if
  090694**                  accept ws-date from date                                            UND
==090694==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.
  090695**                  move corr ws-date to header-date                                    UND 455
==090695==> IGYPS2121-S "WS-DATE" was not defined as a data-name.  The statement was discarded.
  090696**                  accept ws-time from time                                            UND
==090696==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.
  090697**                  move corr ws-time to header-time                                    UND 449
==090697==> IGYPS2121-S "WS-TIME" was not defined as a data-name.  The statement was discarded.
  090698**                  read location-file                                                  193
==090698==> IGYPS2053-S An error was found in the definition of file "LOCATION-FILE".  This
                        input/output statement was discarded.
  090699**                    at end
  09070O**    1                   set location-eof to true                                    256
  09070I**                  end-read
.
.
.
DATA VALIDATION AND UPDATE PROGRAM               IGYTCARA  Date 10/11/2001  Time 12:26:53   Page    76
LineID  Message code  Message text
          IGYSC0090-W   1700 sequence errors were found in this program.
          IGYSC3002-I   A severe error was found in the program.  The "OPTIMIZE" compiler option was cancelled.
     160  IGYDS1089-S   "ASSIGNN" was invalid.  Scanning was resumed at the next area "A" item, level-number, or
                        the start of the next clause.
     193  IGYGR1207-S   The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file "LOCATION-FILE".
                        The file definition was discarded.
     269  IGYDS1066-S   "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data item.
                        The "REDEFINES" clause was discarded.
   90602  IGYPS2052-S   An error was found in the definition of file "LOCATION-FILE".  The reference to this file
                        was discarded. Same message on line:  90684
   90694  IGYPS2121-S   "WS-DATE" was not defined as a data-name.  The statement was discarded.
                        Same message on line:  90695
   90696  IGYPS2121-S   "WS-TIME" was not defined as a data-name.  The statement was discarded.
                        Same message on line:  90697
   90698  IGYPS2053-S   An error was found in the definition of file "LOCATION-FILE".  This input/output statement
                        was discarded. Same message on line:  90709
Messages    Total    Informational    Warning    Error    Severe    Terminating
Printed:       13            1              1                11
* Statistics for COBOL program IGYTCARA:
*    Source records = 1735
*    Data Division statements = 287
*    Procedure Division statements = 471
End of compilation 1,  program IGYTCARA,  highest severity 12.
Return code 12
```

プログラム・エンティティー定義および参照の検出

XREF(FULL) コンパイラー・オプションを使用すると、データ名、プロシージャー名、またはプログラム名が定義および参照されている場所を見つけることができます。ソート済み相互参照には、そのエンティティーが定義されている行番号およびそのエンティティーへのすべての参照の行番号が入れられます。

明示的に参照されている変数だけを含める場合は、XREF(SHORT) オプションを使用します。

XREF (FULL または SHORT を指定した) と SOURCE オプションの両方を使用すると、変更された相互参照がソース・リストの右側に印刷されます。この組み込み相互参照は、データ名またはプロシージャー名が定義されている行番号を示します。

プログラムに DBCS ユーザー定義語が含まれている場合、それらのユーザー定義語は、EBCDIC ユーザー定義語のアルファベット順リストの前にリストされます。

z/OS 環境では、DBCSXREF インストール・オプションで日本語配列プログラムが指定された場合には、DBCS ユーザー定義語は、指定された DBCS 照合シーケンスに従ってリストされます。それ以外の場合、DBCS ユーザー定義語は、ソース・プログラムでの表示順に従った物理的な順序でリストされます。

MOVE CORRESPONDING ステートメントのグループ名が、XREF リストに表示されます。相互参照リストには、その移動に含まれるグループ名とすべての基本名が含まれます。

399 ページの『例: XREF 出力 - データ名相互参照』
400 ページの『例: XREF 出力 - プログラム名相互参照』
400 ページの『例: 組み込み相互参照』

関連タスク
377 ページの『リストの入手』

関連参照
357 ページの『XREF』

データ項目のリスト

MAP コンパイラー・オプションを使用すると、DATA DIVISION に定義した項目および暗黙に宣言されているすべての項目のリストを作成することができます。システム・ダンプのデータ項目の内容を突き止める場合に、MAP 出力を使用してください。

さらに、MAP オプションを使用すると、組み込み MAP 要約 (圧縮データ MAP 情報が含まれている) が、COBOL ソース・データ宣言の右側に生成されます。XREF データと組み込み MAP 要約の両方が同じ行にあるときは、組み込み要約の方が先に印刷されます。

MAP リストおよび組み込み MAP 要約の各部分は、ソース全体に散在した *CONTROL MAP または *CONTROL NOMAP ステートメント (*CBL MAP または *CBL NOMAP ステートメント) を使用して、選択したり禁止することができます。たとえば、次のようになります。

```
*CONTROL NOMAP            *CBL NOMAP
    01  A                     01  A
    02  B                     02  B
*CONTROL MAP              *CBL MAP
```

382 ページの『例: MAP 出力』

関連タスク
『リストの入手』

関連参照
333 ページの『MAP』

リストの入手

コンパイラー・オプションを使用して適切なコンパイラー・リストを要求することによって、デバッグに必要な情報を入手してください。

重要: コンパイラーによって作成されるリストは、プログラミング・インターフェースではなく、容易に変更できるものです。

用途	リスト	内容	コンパイラー・オプション
コンパイルに関する診断メッセージ、プログラムに有効なオプションのリスト、およびプログラムの内容に関する統計を検査する。	短縮リスト	コンパイルに関する診断メッセージ[1] (378 ページ)。プログラムに有効なオプションのリスト。プログラムの内容に関する統計。	NOSOURCE、NOXREF、NOVBREF、NOMAP、NOOFFSET、NOLIST
プログラムのテストおよびデバッグを援助する。プログラムのデバッグ後にレコードを得る。	ソース・リスト	ソースのコピー	346 ページの『SOURCE』
ストレージ・ダンプ内で特定のデータ項目を見つける。再入可能性または最適化を考慮した後の最終ストレージ割り振りを調べる。プログラムが定義されている場所を見つけ、その属性を検査する。	DATA DIVISION 項目のマップ	すべての DATA DIVISION 項目および暗黙的に宣言されたすべての変数	333 ページの『MAP』[2]
		組み込みマップ要約 (DATA DIVISION 内の、データ宣言が含まれている行のリストの右マージン)	
		ネストされたプログラム・マップ (ネストされたプログラムが含まれているプログラム)	
名前が定義、参照、または変更されている場所を調べる。プロシージャーが参照されているコンテキスト (たとえば、動詞が PERFORM ブロックで使用されたかどうか) を判別する。	名前のソート済み相互参照リスト	データ名、プロシージャー名、プログラム名。これらの名前への参照。	357 ページの『XREF』[2] [3] (378 ページ)
		組み込みの変更済み相互参照。この組み込み相互参照は、データ名またはプロシージャー名が定義されている行番号を渡す。	
プログラム内の障害のある動詞を見つける。または、プログラムの実行時に移動されたデータ項目のストレージのアドレスを調べる。	コンパイラーによって生成される PROCEDURE DIVISION コードおよびアセンブラー・コード[3] (378 ページ)	生成されたコード	333 ページの『LIST』[2] [4] (378 ページ)

用途	リスト	内容	コンパイラー・オプション
PROCEDURE DIVISION セクションを移動または追加した後でも有効な論理パスがまだあるか検査する。	圧縮 PROCEDURE DIVISION リスト	圧縮された動詞リスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラル	338 ページの『OFFSET』
特定の動詞のインスタンスを見つける。	アルファベット順の動詞	使用されたそれぞれの動詞、各動詞が使用された回数、各動詞が使用された行番号	356 ページの『VBREF』

1. メッセージを除去するには、コンパイル診断情報のレベルを左右するオプション (たとえば、FLAG) をオフにしてください。
2. コンパイル済みプログラムの行番号を使用するには、NUMBER コンパイラー・オプションを使用してください。コンパイラーは、ステートメントが読み込まれるときに、桁 1 ～ 6 にあるソース・ステートメント行番号のシーケンスを検査します。行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の番号を割り当てます。新しい値には、2 つのアスタリスクのフラグが付けられます。シーケンス・エラーを示す診断メッセージがコンパイル・リストに入れられます。
3. プロシージャー参照のコンテキストは、行番号の前の文字で示されます。
4. ソースに *CONTROL LIST および *CONTROL NOLIST ステートメント (*CBL LIST および *CBL NOLIST) を入れることによって、生成されるオブジェクト・コードの選択的リストを制御することができます。*CONTROL ステートメントは PROCESS (または CBL) ステートメントとは異なることに注意してください。

   出力が生成されるのは、以下の場合です。

   - COMPILE オプションを指定している (または NOCOMPILE(*x*) オプションが有効であり、エラー・レベル *x* 以上が発生していない)。
   - OFFSET オプションを指定していない。OFFSET と LIST は互いに排他的なオプションで、OFFSET の方が優先されます。

379 ページの『例: 短縮リスト』
381 ページの『例: SOURCE および NUMBER 出力』
382 ページの『例: MAP 出力』
383 ページの『例: 組み込みマップ要約』
386 ページの『例: ネストされたプログラム・マップ』
399 ページの『例: XREF 出力 - データ名相互参照』
400 ページの『例: XREF 出力 - プログラム名相互参照』
400 ページの『例: 組み込み相互参照』
401 ページの『例: OFFSET コンパイラー出力』
402 ページの『例: VBREF コンパイラー出力』

関連タスク

294 ページの『コンパイル・エラー・メッセージのリストの生成』
386 ページの『LIST 出力の読み取り』
COBOL プログラムのデバッグ (*言語環境プログラム デバッグのガイド*)

関連参照

294 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

例: 短縮リスト

リストの後の説明で使用されている番号は、リストに付けられている番号に対応しています。診断メッセージが出される原因となったエラーのいくつかは、説明を行うために故意に挿入されたものです。

```
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0 Date 10/11/2001 Time 13:25:42 Page 1 (1)
Invocation parameters:      (2)
NOSOURCE,OPTIMIZE(FULL)
PROCESS(CBL) statements:
CBL TEST(ALL)          (3)
                            (4)
        IGYOS4022-W   The "OPTIMIZE" option was discarded due to option conflict resolution.
                      The "TEST" option from "PROCESS/CBL" statement took precedence.

Options in effect:          (5)
    NOADATA
      ADV
      QUOTE
      ARITH(COMPAT)
    NOAWO
      BUFSIZE(4096)
    NOCICS
      CODEPAGE(1140)
    NOCOMPILE(S)
    NOCURRENCY
      DATA(31)
    NODATEPROC
      DBCS
    NODECK
    NODIAGTRUNC
    NODLL
    NODUMP
    NODYNAM
    NOEXIT
    NOEXPORTALL
    NOFASTSRT
      FLAG(I,I)
    NOFLAGSTD
      INTDATE(ANSI)
      LANGUAGE(EN)
    NOLIB
      LINECOUNT(60)
    NOLIST
    NOMAP
    NONAME
      NSYMBOL(NATIONAL)
    NONUMBER
      NUMPROC(NOPFD)
      OBJECT
    NOOFFSET
    NOOPTIMIZE
      OUTDD(SYSOUT)
      PGMNAME(COMPAT)
      RENT
      RMODE(AUTO)
      SEQUENCE
      SIZE(MAX)
    NOSOURCE
      SPACE(1)
    NOSQL
    NOSSRANGE
    NOTERM
      TEST(ALL,SYM,NOSEPARATE)
    NOTHREAD
      TRUNC(STD)
    NOVBREF
    NOWORD
      XREF
      YEARWINDOW(1900)
      ZWB
DATA VALIDATION AND UPDATE PROGRAM IGYTCARA Date 10/11/2001 Time 13:25:42 Page 3 (6)
```

```
  LineID  Message code  Message text   (7)
          IGYDS0139-W   Diagnostic messages were issued during processing of compiler options.
                        These messages are located at the beginning of the listing.
          IGYSC0090-W   3 sequence errors were found in this program.
     160  IGYDS1089-S   "ASSIGNN" was invalid.  Scanning was resumed at the next area "A" item,
                        level-number,or the start of the next clause.
     193  IGYGR1207-S   The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file
                        "LOCATION-FILE".  The file definition was discarded.
     269  IGYDS1066-S   "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data item.
                        The "REDEFINES" clause was discarded.
     901  IGYPS2052-S   An error was found in the definition of file "LOCATION-FILE".  The reference to
                        this file was discarded. Same message on line:    983
     993  IGYPS2121-S   "WS-DATE" was not defined as a data-name.  The statement was discarded.
                        Same message on line:    994
     995  IGYPS2121-S   "WS-TIME" was not defined as a data-name.  The statement was discarded.
                        Same message on line:    996
     997  IGYPS2053-S   An error was found in the definition of file "LOCATION-FILE".  This input/output
                        statement was discarded. Same message on line:   1008
  Messages    Total    Informational    Warning    Error    Severe    Terminating   (8)
Printed:      14                3                 11
  * Statistics for COBOL program IGYTCARA:    (9)
*    Source records = 1735
*    Data Division statements = 287
*    Procedure Division statements = 471
End of compilation 1,  program IGYTCARA,  highest severity 12.  (10)
Return code 12
```

(1) COBOL のデフォルトのページ・ヘッダー (LVLINFO インストール時コンパイラー・オプションからのコンパイラー・レベルの情報を含みます)。

(2) コンパイラー呼び出し時にコンパイラーに渡されたオプションに関するメッセージ。このメッセージは、オプションが渡されていない場合には表示されません。

NOSOURCE
SOURCE をオフにすると、COBOL ソース・コードが COBOL リストから除去されます。

(3) PROCESS (または CBL) ステートメントの中にコーディングされたオプション。

TEST(ALL)
プログラムはデバッグ・ツールでの使用を目的にコンパイルされました。

(4) コンパイラー入力パラメーター・リストに OPTIMIZE オプションを指定することによって、故意にオプションの矛盾を引き起こしています。OPTIMIZE と CBL ステートメントで指定された TEST(ALL) オプションは互いに排他的です。その結果、OPTIMIZE オプションは無視されます。

(5) このコンパイルの開始時のオプションの状況。

(6) COBOL プログラムの TITLE ステートメントから生じたカスタマイズ済みページ・ヘッダー。

(7) プログラム診断メッセージ。最初のメッセージは、ライブラリー・フェーズ診断 (ある場合) に言及しています。ライブラリー・フェーズの診断は、リストの先頭に示されます。

(8) このプログラムの診断メッセージのカウントで、重大度レベルによってグループ化されたもの。

(9) プログラム IGYTCARA のプログラム統計。

(10) コンパイル単位のプログラム統計。バッチ・コンパイルを実行する場合、戻りコードは、コンパイル全体について最高レベルのメッセージ重大度です。

例: SOURCE および NUMBER 出力

次に示されているリストの部分では、プログラマーは 2 つのステートメントに順序どおりでない番号を付けています。

```
DATA VALIDATION AND UPDATE PROGRAM (1)                          IGYTCARA  Date 10/11/2001  Time 12:26:53   Page    22
  LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Cross-Reference (2)
    (3)    (4)    (5)
                087000*****************************************************************  *
                087100***                D O   M A I N   L O G I C                      *  *
                087200***                                                               *  *
                087300*** Initialization. Read and process update transactions until *  *
                087400*** EOE. Close files and stop run.                              *  *
                087500*****************************************************************  *
                087600 procedure division.
                087700   000-do-main-logic.
                087800     display "PROGRAM IGYTCARA - Beginning"
                087900     perform 050-create-vsam-master-file.                                 90633
                088150     display "perform 050-create-vsam-master finished".
 088151**       088125     perform 100-initialize-paragraph                                     90677
                088200     display "perform 100-initialize-paragraph finished"
                088300     read update-transaction-file into ws-transaction-record              204 331
                088400         at end
              1 088500       set transaction-eof to true                                      254
                088600     end-read
                088700     display "READ completed"
                088800     perform until transaction-eof                                        254
              1 088900       display "inside perform until loop"
              1 089000       perform 200-edit-update-transaction                                90733
              1 089100       display "After perform 200-edit   "
              1 089200       if no-errors                                                       365
              2 089300         perform 300-update-commuter-record                               90842
              2 089400         display "After perform 300-update "
              1 089650       else
 089651**     2 089600         perform 400-print-transaction-errors                             90995
              2 089700         display "After perform 400-errors "
              1 089800       end-if
              1 089900       perform 410-re-initialize-fields                                   91056
              1 090000       display "After perform 410-reinitialize"
              1 090100       read update-transaction-file into ws-transaction-record            204 331
              1 090200           at end
              2 090300         set transaction-eof to true                                      254
              1 090400       end-read
              1 090500       display "After '2nd READ'   "
                090600     end-perform
```

(1) COBOL プログラムの TITLE ステートメントから生じたカスタマイズ済みページ・ヘッダー。

(2) スケール行では、区域 A、区域 B、およびソース・コード列番号にラベルを付けます。

(3) コンパイラーが割り当てるソース・コード行番号。

(4) プログラム (PL) とステートメント (SL) のネスト・レベル。

(5) プログラムの第 1 ～ 6 桁 (シーケンス番号域)。

例: MAP 出力

次の例は、MAP オプションからの出力を示しています。その下の説明で使用されている番号は、出力に付けられている番号と対応しています。

```
Data Division Map

(1)
Data Definition Attribute codes (rightmost column) have the following meanings:
    D = Object of OCCURS DEPENDING    G = GLOBAL                          S = Spanned file
    E = EXTERNAL                      O = Has OCCURS clause               U = Undefined format file
    F = Fixed-length file             OG= Group has own length definition V = Variable-length file
    FB= Fixed-length blocked file     R = REDEFINES                       VB= Variable-length blocked file

(2)      (3) (4)                       (5)     (6)      (7)
                                                                    (8)       (9)      (10)
Source   Hierarchy and                   Base       Hex-Displacement  Asmblr Data                 Data Def
LineID   Data Name                       Locator    Blk   Structure   Definition    Data Type     Attributes
     4  PROGRAM-ID IGYTCARA----------------------------------------------------------------------------------*
   181   FD COMMUTER-FILE . . . . . .                                              VSAM           F
   183   1  COMMUTER-RECORD . . . . . .  BLF=0000   000              DS 0CL80      Group
   184     2  COMMUTER-KEY. . . . . . .  BLF=0000   000   0 000 000  DS 16C        Display
   185     2  FILLER. . . . . . . . . .  BLF=0000   010   0 000 010  DS 64C        Display
   187   FD COMMUTER-FILE-MST . . . . .                                            VSAM           F
   189   1  COMMUTER-RECORD-MST . . . . .  BLF=0001   000              DS 0CL80      Group
   190     2  COMMUTER-KEY-MST. . . . .  BLF=0001   000   0 000 000  DS 16C        Display
   191     2  FILLER. . . . . . . . . .  BLF=0001   010   0 000 010  DS 64C        Display
   193   FD LOCATION-FILE . . . . . .                                              QSAM           FB
   198   1  LOCATION-RECORD . . . . . .  BLF=0002   000              DS 0CL80      Group
   199     2  LOC-CODE. . . . . . . . .  BLF=0002   000   0 000 000  DS 2C         Display
   200     2  LOC-DESCRIPTION . . . . .  BLF=0002   002   0 000 002  DS 20C        Display
   201     2  FILLER. . . . . . . . . .  BLF=0002   016   0 000 016  DS 58C        Display
   204   FD UPDATE-TRANSACTION-FILE . . .                                          QSAM           FB
   209   1  UPDATE-TRANSACTION-RECORD . .  BLF=0003   000              DS 80C        Display
   217   FD PRINT-FILE. . . . . . . . .                                            QSAM           FB
   222   1  PRINT-RECORD. . . . . . . .  BLF=0004   000              DS 121C       Display
   229   1  WORKING-STORAGE-FOR-IGYCARA .  BLW=0000   000              DS 1C         Display
   231   77 COMP-CODE . . . . . . . . .  BLW=0000   008              DS 2C         Binary
   232   77 WS-TYPE . . . . . . . . . .  BLW=0000   010              DS 3C         Display
   235   1  I-F-STATUS-AREA . . . . . .  BLW=0000   018              DS 0CL2       Group
   236     2  I-F-FILE-STATUS . . . . .  BLW=0000   018   0 000 000  DS 2C         Display
   237     88 I-O-SUCCESSFUL. . . . . .
   240   1  STATUS-AREA . . . . . . . .  BLW=0000   020              DS 0CL8       Group
   241     2  COMMUTER-FILE-STATUS. . . .  BLW=0000   020   0 000 000  DS 2C         Display
   242     88 I-O-OKAY. . . . . . . . .
   243     2  COMMUTER-VSAM-STATUS. . . .  BLW=0000   022   0 000 002  DS 0CL6       Group
   244       3  VSAM-R15-RETURN-CODE. . .  BLW=0000   022   0 000 002  DS 2C         Binary
   245   77 UNUSED-DATA-ITEM. . . . . .  BLW=XXXX   022              DS 10C        Display  (11)
```

(1) データ定義属性コードの説明。

(2) データ項目が定義されたソース行番号。

(3) レベル定義または番号。コンパイラーは、次の方法でこの番号を生成します。

- 階層の第 1 レベルは常に 01 です。02 ～ 49 としてコーディングされた項目のレベルごとに 1 を加えます。
- レベル番号の 66、77、および 88、そして標識 FD と SD は変更されません。

(4) ソース・モジュールでソース順序で使用されるデータ名。

(5) このデータ項目に使用されるベース・ロケーター。

(6) ベース・ロケーター値の先頭からの 16 進変位。

(7) この項目を含む構造の先頭からの 16 進変位。

(8) データが定義されている方法を示す、疑似アセンブラー・コード。構造に可変長フィールドが含まれている場合、構造の最大長が示されます。

(9) データ型および使用法。

(10) データ定義属性コード。定義は DATA DIVISION マップの先頭で説明されています。

(11) UNUSED-DATA-ITEM は PROCEDURE DIVISION で参照されませんでした。OPTIMIZE(FULL) が指定されていたため、UNUSED-DATA-ITEM は削除され、その結果、ベース・ロケーターは XXXX に設定されました。

関連参照

384 ページの『MAP 出力で使用される用語』
385 ページの『LIST および MAP 出力で使用される記号』

例: 組み込みマップ要約

次の例は、MAP オプションによって作成される組み込みマップ要約を示しています。この要約は、DATA DIVISION の、データ宣言を含む行のリストの右マージンに現れます。

```
000002    Identification Division.
 000003
 000004    Program-id.    IGYTCARA.
  . . .
 000177    Data division.
 000178    File section.
 000179
 000180
 000181    FD  COMMUTER-FILE
 000182        record 80 characters.                                    (1)    (2)  (3)          (4)
  . . .
 000222      01 print-record                  pic x(121).               BLF=0004+000           121C
  . . .
 000228    Working-storage section.
 000229      01 Working-storage-for-IGYCARA    pic x.                    BLW=0000+000           1C
 000230
 000231      77 comp-code                     pic S9999 comp.           BLW=0000+008           2C
 000232      77 ws-type                       pic x(3)   value spaces.  BLW=0000+010           3C
 000233
 000234
 000235      01 i-f-status-area.                                         BLW=0000+018           0CL2
 000236        05 i-f-file-status             pic x(2).                 BLW=0000+018,0000000   2C
 000237          88 i-o-successful            value zeroes.
 000238
 000239
 000240      01 status-area.                                             BLW=0000+020           0CL8
 000241        05 commuter-file-status        pic x(2).                 BLW=0000+020,0000000   2C
 000242          88 i-o-okay                  value zeroes.
 000243        05 commuter-vsam-status.                                  BLW=0000+022,0000002   0CL6
 000244          10 vsam-r15-return-code      pic 9(2) comp.            BLW=0000+022,0000002   2C
 000245          10 vsam-function-code        pic 9(1) comp.            BLW=0000+024,0000004   2C
 000246          10 vsam-feedback-code        pic 9(3) comp.            BLW=0000+026,0000006   2C
 000247
 000248      77 update-file-status            pic xx.                   BLW=0000+028           2C
 000249      77 loccode-file-status           pic xx.                   BLW=0000+030           2C
 000250      77 updprint-file-status          pic xx.                   BLW=0000+038           2C
 000251
 000252      01 flags.                                                  BLW=0000+040           0CL3
 000253        05 transaction-eof-flag        pic x value space.        BLW=0000+040,0000000   1C
 000254            88 transaction-eof         value "Y".
 000255        05 location-eof-flag           pic x value space.        BLW=0000+041,0000001   1C
 000256            88 location-eof            value "Y".
 000257        05 transaction-match-flag      pic x.                    BLW=0000+042,0000002   1C
  . . .
 000876    procedure division.
 000877      000-do-main-logic.
 000878        display "PROGRAM IGYTCARA - Beginning"
 000879        perform 050-create-vsam-master-file.
```

(1) このデータ項目に使用されるベース・ロケーター。

(2) ベース・ロケーター値の先頭からの 16 進変位。

(3) 収容構造の先頭からの 16 進変位。

(4) データが定義されている方法を示す、疑似アセンブラー・コード。

MAP 出力で使用される用語

次の表は、MAP オプションによって作成されるリストで使用される用語を説明しています。

用語	定義	説明
GROUP	DS 0CLn[1]	固定長グループ・データ項目
ALPHABETIC	DS nC	英字データ項目 (PICTURE A)
ALPHA-EDIT	DS nC	英字編集データ項目
DISPLAY	DS nC	英数字データ項目 (PICTURE X)
AN-EDIT	DS nC	英数字編集データ項目
GRP-VARLEN	DS 0LCn[1]	可変長グループ・データ項目
NUM-EDIT	DS nC	数字編集データ項目
DISP-NUM	DS nC	外部 10 進データ項目 (USAGE DISPLAY)
BINARY	DS 1H[2]、1F[2]、2F[2]、2C、4C、または 8C	2 進データ項目 (USAGE BINARY または USAGE COMPUTATIONAL または USAGE COMPUTATIONAL-5)
COMP-1	DS 4C	単精度浮動小数点データ項目 (USAGE COMPUTATIONAL-1)
COMP-2	DS 8C	倍精度浮動小数点データ項目 (USAGE COMPUTATIONAL-2)
PACKED-DEC	DS nP	内部 10 進データ項目 (USAGE PACKED-DECIMAL または USAGE COMPUTATIONAL-3)
DBCS	DS nC	DBCS データ項目 (USAGE DISPLAY-1)
DBCS-EDIT	DS nC	DBCS 編集済みデータ項目 (USAGE DISPLAY-1)
INDX-NAME		指標名
INDEX		指標データ項目 (USAGE INDEX)
NATIONAL	DS nC	国別データ項目 (USAGE NATIONAL)
OBJECT-REF		オブジェクト・リファレンス・データ項目 (USAGE OBJECT-REFERENCE)
POINTER		ポインター・データ項目 (USAGE POINTER)
FD		ファイル定義
VSAM, QSAM, LINESEQ		ファイル処理方式
1 ～ 49、77		データ記述に関するレベル番号
66		RENAMES に関するレベル番号
88		条件名に関するレベル番号
SD		ソート・ファイル定義
1. *n* は、固定長グループのバイト単位のサイズ、および可変長グループのバイト単位のサイズです。 2. SYNCHRONIZED 文節が表示されると、これらのフィールドが使用されます。		

LIST および MAP 出力で使用される記号

次の表は、LIST または MAP オプションによって作成されるリストで使用される記号を説明しています。

記号	定義
APBdisp=n[1]	すべての添え字パラメーター・ブロックの変位
AVN=n[1]	ALTER ステートメントの変数名セル
BL=n[1]	特殊レジスターのベース・ロケーター
BLA=n[1]	英数字一時記憶域[4] のベース・ロケーター
BLF=n[1]	ファイルのベース・ロケーター
BLK=n[1]	LOCAL-STORAGE のベース・ロケーター
BLL=n[1]	LINKAGE SECTION のベース・ロケーター
BLM=n[1]	ファクトリー・データのベース・ロケーター
BLO=n[1]	オブジェクト・インスタンス・データのベース・ロケーター
BLS=n[1]	ソート項目のベース・ロケーター
BLT=n[1]	XML-TEXT および XML-NTEXT のベース・ロケーター
BLV=n[1]	位置が変更できるデータのベース・ロケーター
BLW=n[1]	WORKING-STORAGE のベース・ロケーター
BLX=n[1]	外部データのベース・ロケーター
CBL=n[1]	定数グローバル・テーブル (CGT) のベース・ロケーター
CLLE=@=n[1]	TGT におけるロード・リスト入り口アドレス
CLO=n[1]	クラス・オブジェクト・セル
DOV=n[1]	DSA オーバーフロー・セル
EVALUATE=n[1]	ブール・セルを評価する
FCB=n[1]	ファイル制御ブロック (FCB) のアドレス
GN=n(hhhhh)[2.]	生成されたプロシージャー名と、16 進数のそのオフセット
IDX=n[1]	指標名のベース・ロケーター
IDX=n[1]	指標セル番号
ILS=n[1]	LOCAL-STORAGE テーブルまたはインスタンス変数の指標セル
ODOSAVE=n[1]	ODO 保管セル番号
OPT=nnnn[3]	最適化プログラムの一時記憶セル
PBL=n[1]	プロシージャー・コードのベース・ロケーター
PFM=n[1]	PERFORM n 回セル
PGMLIT AT + nnnn[3]	リテラル・プールの先頭からのプログラム・リテラルの変位
PSV=n[1]	保管セル番号を実行する
PVN=n[1]	PERFORM ステートメントの変数名セル
RBKST=n[1]	レジスター逆保管セル
SFCB=n[1]	外部ファイルの 2 次ファイル制御ブロック
SYSLIT AT + nnnn[3]	システム・リテラル・プールの先頭からのシステム・リテラルの変位
TGT FDMP TEST INFO.AREA + nnnn[3]	FDUMP/TEST 情報域
TGTFIXD + nnnn[3]	タスク・グローバル・テーブル (TGT) の固定部分の先頭からのオフセット

記号	定義
TOV=n[1]	TGT オーバーフロー・セル番号
TS1=aaaa	サブプール 1 の一時記憶セル番号
TS2=aaaa	サブプール 2 の一時記憶セル番号
TS3=aaaa	サブプール 3 の一時記憶セル番号
TS4=aaaa	サブプール 4 の一時記憶セル番号
V (ルーチン名)	外部ルーチンのアセンブラー VCON
VLC=n[1]	可変長名セル番号 (ODO)
VNI=n[1]	変数名初期化
WHEN=n[1]	WHEN セル番号を評価する

1. n は記入項目の番号です。ベース・ロケーターの場合は XXXX になることもあります。これは、OPTIMIZE(FULL) 処理によって削除されたデータ項目を表します。
2. (hhhhh) は 16 進数のプログラム・オフセットです。
3. nnnn は、記入項目の先頭からの 10 進数のオフセットです。
4. 英数字一時記憶域は、英数字組み込み関数および英数字 EVALUATE ステートメントのサブジェクトを処理する際に使用される一時データ値です。

例: ネストされたプログラム・マップ

この例は、MAP コンパイラー・オプションによって作成される、ネストされたプロシージャーのマップを示しています。

```
Nested Program Map
Program Attribute codes (rightmost column) have the following meanings:
    C = COMMON
    I = INITIAL (1)
    U = PROCEDURE DIVISION USING...                          (5)
Source Nesting                                               Program
LineID Level   Program Name from PROGRAM-ID paragraph        Attributes
     2     0   NESTMAIN. . . . . . . . . . . . . . . . . . . U
   120     1 (4) SUBPRO1 . . . . . . . . . . . . . . . . . . I,C,U
(2)199     2       NESTED1 . . . . . . . . . . . . . . . . . I,C,U
   253     1     SUBPRO2 . . . . . . . . . . . . . . . . . . U
   335     2       NESTED2 . . . . . . . . . . . . . . . . . C,U
         (3)
```

(1) プログラム属性コードの説明。

(2) プログラムが定義されたソース行番号。

(3) プログラムのネストの深さ。

(4) プログラム名。

(5) プログラム属性コード。

LIST 出力の読み取り

LIST コンパイラー出力の各部分は、プログラムのデバッグに有用である場合があります。LIST で作成される出力を理解するために、アセンブラー言語でプログラミングができる必要はありません。アセンブラー・コードの大部分を伴うコメントは、コードが実行する機能を概念的に理解するのに役立ちます。

LIST コンパイラー・オプションは、次の 7 つの出力を作成します。

- プログラムの初期化コードのアセンブラー・リスト (プログラム・シグニチャー情報バイト)。このリストからは、次のようなプログラム特性を検査することができます。
  - 有効なコンパイラー・オプション
  - 存在するデータ項目のタイプ
  - PROCEDURE DIVISION で使用されている動詞
- プログラムのソース・コードのアセンブラー・リスト

  異常終了発生時に実行されていた命令のストレージ内のアドレスから、その命令に対応する COBOL 動詞を見つけることができます。障害のある命令のアドレスが見つかったら、アセンブラー・リストに進み、その命令が生成される対象となった動詞を調べてください。
- オブジェクト・モジュールでのコンパイラー生成テーブルの位置
- タスク・グローバル・テーブル (TGT) のマップ (プログラム・グローバル・テーブル (PGT) および定数グローバル・テーブル (CGT) に関する情報を含む)

  TGT を使用すると、プログラムが実行されている環境に関する情報を見つけることができます。
- WORKING-STORAGE および制御ブロックの位置とサイズに関する情報

  LIST 出力の WORKING-STORAGE 部分を使用すると、WORKING-STORAGE で定義されているデータ項目の位置がわかります (RENT オプションを指定してコンパイルしたプログラムの場合は、WORKING STORAGE の先頭位置は示されません)。
- 動的保管域 (DSA) のマップ

  DSA (スタック・フレームとも呼ばれる) のマップには、別個のコンパイル済みプロシージャーに入るたびに獲得されたストレージの内容に関する情報が入れられます。
- 動的ストレージの使用についてのリテラルとコードの位置に関する情報

『例: プログラム初期化コード』
395 ページの『例: ソース・コードから生成されるアセンブラー・コード』
396 ページの『例: TGT メモリー・マップ』
397 ページの『例: WORKING-STORAGE の位置とサイズ』
397 ページの『例: DSA メモリー・マップ』

関連タスク
Principles of Operation の『Interpret a particular instruction』の項

関連参照
スタック・ストレージの概要 (*言語環境プログラム プログラミング・ガイド*)

例: プログラム初期化コード

プログラム初期化コードのリストは、COBOL ソース・プログラムの特性に関する情報を提供します。プログラムの特性を検査するには、プログラム・シグニチャー情報バイトを解釈してください。

```
IBM Enterprise for z/OS and OS/390 3.1.0           IMIN      Date 08/21/2001  Time 10:48:16   Page     5
      (1)   (2)                    (3)                          (4)
     000000           IMIN         DS   0H                      PROGRAM: IMIN
                                   USING *,15
     000000  47F0 F028             B    40(,15)                 BYPASS CONSTANTS. BRANCH TO @STM
     000004  00                    DC   AL1(0)                   ZERO NAME LENGTH FOR DUMPS
     000005  C3C5C5                DC   CL3'CEE'                 CEE EYE CATCHER                  (5)
     000008  000000E8              DC   X'000000E8'              STACK FRAME SIZE
     00000C  00000014              DC   A(@PPA1-IMIN)            OFFSET TO PPA1 FROM PRIMARY ENTRY
     000010  47F0 F001             B    1(,15)                  RESERVED
     000014                 @PPA1  DS   0H                      PPA1 STARTS HERE
     000014  98                    DC   X'98'                   OFFSET TO LENGTH OF NAME FROM PPA1
     000015  CE                    DC   X'CE'                   CEL SIGNATURE
     000016  AC                    DC   X'AC'                   CEL FLAGS: '10101100'B
     000017  00                    DC   X'00'                   MEMBER FLAGS FOR COBOL
     000018  000000B6              DC   A(@PPA2)                ADDRESS OF PPA2
     00001C  00000000              DC   F'0'                    OFFSET TO THE BDI (NONE)
     000020  00000000              DC   F'0'                    ADDRESS OF ENTRY POINT DESCRIPTORS
     000024  0000                  DC   X'0000'                 RESERVED
     000026  00                    DC   X'00'                   DSA FPR 8-15 SAVE AREA OFFSET/16
     000027  00                    DC   X'00'                   DSA FPR 8-15 SAVE AREA BIT MASK
     000028                 @STM   DS   0H                      STM STARTS HERE
     000028  90EC D00C              STM  14,12,12(13)    @STM:   SAVE CALLER'S REGISTERS
     00002C  4110 F038              LA   1,56(,15)               GET ADDRESS OF PARMLIST INTO R1
     000030  98EF F04C              LM   14,15,76(15)            LOAD ADDRESSES FROM @BRVAL
     000034  07FF                   BR   15                      DO ANY NECESSARY INITIALIZATION
     000036  0000                   DC   AL2'0'                  AVAILABLE HALF-WORD
     000038                 @MAINENT DS   0H                     PRIMARY ENTRY POINT ADDRESS
     000038  00000000              DC   A(IMIN)         @PARMS: 1) PRIMARY ENTRY POINT ADDRESS
     00003C  00000000              DC   AL4'0'                  2) Available
     000040  000002C8              DC   A(TGT)                  3) TGT ADDRESS                   (6)
     000044  000000AE              DC   A(@EPNAM)               4) ENTRY POINT NAME ADDRESS
     000048  00000000              DC   A(IMIN)                 5) CURRENT ENTRY POINT ADDRESS
     00004C  000001A4              DC   A(START)        @BRVAL: 6) PROCEDURE CODE ADDRESS
     000050  00000000              DC   V(IGZCBS0)              7) INITIALIZATION ROUTINE
     000054  000000CA              DC   A(@CEEPARM)             8) ADDRESS OF PARM LIST FOR CEEINT
     000058  00104001              DC   X'00104001'             DSA WORD 0 CONSTANT
     00005C  00000000              DC   AL4'0'                  AVAILABLE WORD
     000060  00000000              DC   AL4'0'                  AVAILABLE WORD
     000064  00000000              DC   AL4'0'                  AVAILABLE WORD
     000068  F2F0F0F1              DC   CL4'2001'     @TIMEVRS: YEAR OF COMPILATION              (7)
     00006C  F0F8F2F1              DC   CL4'0821'               MONTH/DAY OF COMPILATION         (8)
     000070  F1F0F4F8              DC   CL4'1048'               HOURS/MINUTES OF COMPILATION     (9)
     000074  F1F6                  DC   CL2'16'                 SECONDS FOR COMPILATION DATE
     000076  F0F3F0F1F0F0          DC   CL6'030100'             VERSION/RELEASE/MOD LEVEL OF PROD (10)
     00007C  0474                  DC   X'0474'                 UNSIGNED BINARY CODE PAGE CCSID VALUE (11)
     000080  0000                  DC   AL2'0'                  AVAILABLE HALF-WORD
     000082  0000                  DC   X'0000'                 INFO. BYTES 28-29                (12)
     000084  076C                  DC   X'076C'                 SIGNED BINARY YEARWINDOW OPTION VALUE
     000086  E0E83D440000          DC   X'E0E83D440000'         INFO. BYTES 1-6
     00008C  000000004009          DC   X'000000004009'         INFO. BYTES 7-12
     000092  000000000800          DC   X'000000000800'         INFO. BYTES 13-18                (12)
     000098  0000000004            DC   X'0000000004'           INFO. BYTES 19-23
     00009D  00                    DC   X'00'                   COBOL SIGNATURE LEVEL
     00009E  00000007              DC   X'00000007'             # DATA DIVISION STATEMENTS       (13)
     0000A2  00000004              DC   X'00000004'             # PROCEDURE DIVISION STATEMENTS  (14)
     0000A6  000000                DC   X'000000'               INFO. BYTES 24-26                (12)
     0000A9  00                    DC   X'00'                   INFO. BYTE 27
     0000AA  40404040              DC   C'    '                 USER LEVEL INFO (LVLINFO)        (15)
     0000AE  0004                  DC   X'0004'                 LENGTH OF PROGRAM NAME
     0000B0                 @EPNAM   DS   0H                    ENTRY POINT NAME
     0000B0  C9D4C9D540404040      DC   C'IMIN    '             PROGRAM NAME                     (16)
     0000B8                 @PPA2    DS   0H                    PPA2 STARTS HERE
     0000B8  05                    DC   X'05'                   CEL MEMBER IDENTIFIER
     0000B9  00                    DC   X'00'                   CEL MEMBER SUB-IDENTIFIER
     0000BA  00                    DC   X'00'                   CEL MEMBER DEFINED BYTE
     0000BB  01                    DC   X'01'                   CONTROL LEVEL OF PROLOG
     0000BC  00000000              DC   V(CEESTART)             VCON FOR LOAD MODULE
     0000C0  00000000              DC   F'0'                    OFFSET TO THE CDI (NONE)
     0000C4  FFFFFFB2              DC   A(@TIMEVRS-@PPA2)       OFFSET TO TIMESTAMP/VERSION INFO
     0000C8  00000000              DC   A(IMIN)                 ADDRESS OF CU PRIMARY ENTRY POINT
     0000CC                 @CEEPARM DS   0H                    PARM LIST FOR CEEINT
     0000CC  00000038              DC   A(@MAINENT)             POINTER TO PRIMARY ENTRY PT ADDR
     0000D0  00000008              DC   A(@PARMCEE-@CEEPARM)    OFFSET TO PARAMETERS FOR CEEINT
     0000D4                 @PARMCEE DS   0H                    PARAMETERS FOR CEEINT
     0000D4  00000006              DC   F'6'                    1) NUMBER OF ENTRIES IN PARM LIST
     0000D8  00000038              DC   A(@MAINENT)             2) POINTER TO PRIMARY ENTRY PT ADDR
     0000DC  00000000              DC   V(CEESTART)             3) ADDRESS OF CEESTART
     0000E0  00000000              DC   V(CEEBETBL)             4) ADDRESS OF CEEBETBL
     0000E4  00000005              DC   F'5'                    5) CEL MEMBER IDENTIFIER
     0000E8  00000000              DC   F'0'                    6) FOR CEL MEMBER USE
     0000EC  00000000              DC   AL4'0'                  AVAILABLE WORD
     0000F0  00000000              DC   AL4'0'                  AVAILABLE WORD
     0000F4  00000000              DC   AL4'0'                  AVAILABLE WORD
     0000F8  00000000              DC   AL4'0'                  AVAILABLE WORD
     0000FC  0000                  DC   AL2'0'                  AVAILABLE HALF-WORD
```

(1) COBOL プログラムの先頭からのオフセット。

(2) アセンブラー命令の 16 進表記。

(3) COBOL プログラムに対して生成される疑似アセンブラー・コード。

(4) アセンブラー・コードを説明するコメント。

(5) このプログラムをコンパイルするために使用される COBOL コンパイラーを示す目印。

(6) タスク・グローバル・テーブル (TGT) のアドレスか、またはプログラムが再入可能な場合は、動的アクセス・ブロック (DAB) のアドレス。

(7) プログラムがコンパイルされたときの 4 桁の年。

(8) プログラムがコンパイルされたときの月および日。

(9) プログラムがコンパイルされたときの時刻。

(10) このプログラムをコンパイルするために使用された COBOL コンパイラーのバージョン、リリース、および修正レベル (それぞれが 2 桁で表される)。

(11) コード・ページの CCSID 値 (CODEPAGE コンパイラー・オプションより取得)。

(12) プログラム・シグニチャー情報バイト。これらは、このプログラムの以下のものに関する情報を提供します。

- コンパイラー・オプション
- DATA DIVISION (データ部)
- ENVIRONMENT DIVISION (環境部)
- PROCEDURE DIVISION (手続き部)

(13) DATA DIVISION の中のステートメントの数。

(14) PROCEDURE DIVISION の中のステートメントの数。

(15) 4 バイトのユーザー制御レベル情報フィールド。このフィールドの値は LVLINFO によって制御されます。

(16) プログラムの IDENTIFICATION DIVISION で使用されるプログラム名。

関連参照

『シグニチャー情報バイト: コンパイラー・オプション』
391 ページの『シグニチャー情報バイト: DATA DIVISION』
392 ページの『シグニチャー情報バイト: ENVIRONMENT DIVISION』
392 ページの『シグニチャー情報バイト: PROCEDURE DIVISION 動詞』
394 ページの『シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目』

シグニチャー情報バイト: コンパイラー・オプション

次の表は、LIST コンパイラー・オプションが使用されたときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

バイト	ビット	オン	オフ
1	0	ADV	NOADV
	1	APOST	QUOTE
	2	DATA(31)	DATA(24)
	3	DECK	NODECK
	4	DUMP	NODUMP
	5	DYNAM	NODYNAM
	6	FASTSRT	NOFASTSRT
	7	予約済み	
2	0	LIB	NOLIB
	1	LIST	NOLIST
	2	MAP	NOMAP
	3	NUM	NONUM
	4	OBJ	NOOBJ
	5	OFFSET	NOOFFSET
	6	OPTIMIZE	NOOPTIMIZE
	7	OUTDD オプションで指定された DD 名が使用される	OUTDD(SYSOUT) が有効
3	0	NUMPROC(PFD)	NUMPROC(NOPFD)
	1	RENT	NORENT
	2	予約済み	
	3	SEQUENCE	NOSEQUENCE
	4	SIZE(MAX)	SIZE(*value*)
	5	SOURCE	NOSOURCE
	6	SSRANGE	NOSSRANGE
	7	TERM	NOTERM
4	0	TEST	NOTEST
	1	TRUNC(STD)	TRUNC(OPT)
	2	WORD オプションが指定された	NOWORD
	3	VBREF	NOVBREF
	4	XREF	NOXREF
	5	ZWB	NOZWB
	6	NAME	NONAME
5	0	NUMPROC(MIG)	
	1	NUMCLS(ALT)	NUMCLS(PRIM)
	2	DBCS	NODBCS
	3	AWO	NOAWO
	4	TRUNC(BIN)	TRUNC(BIN) ではない
	6	CURRENCY	NOCURRENCY
	7	コンパイル単位はクラス	コンパイル単位はプログラム

バイト	ビット	オン	オフ
26	0	RMODE(ANY)	RMODE(24)
	1	TEST(STMT)	TEST(STMT) ではない
	2	TEST(PATH)	TEST(PATH) ではない
	3	TEST(BLOCK)	TEST(BLOCK) ではない
	4	OPT(FULL)	OPT(STD) または NOOPT
	5	INTDATE(LILIAN)	INTDATE(ANSI)
	6	TEST(SEPARATE)	TEST(SEPARATE) ではない
27	0	PGMNAME(LONGUPPER)	PGMNAME(LONGUPPER) ではない
	1	PGMNAME(LONGMIXED)	PGMNAME(LONGMIXED) ではない
	2	DLL	NODLL
	3	EXPORTALL	NOEXPORTALL
	4	DATEPROC	NODATEPROC
	5	ARITH(EXTEND)	ARITH(COMPAT)
	6	THREAD	NOTHREAD

シグニチャー情報バイト: DATA DIVISION

次の表は、LIST コンパイラー・オプションが使用されたときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

バイト	ビット	項目
6	0	QSAM ファイル記述子
	1	VSAM 順次ファイル記述子
	2	VSAM 索引付きファイル記述子
	3	VSAM 相対ファイル記述子
	4	ファイル記述子内の CODE-SET 文節 (ASCII ファイル)
	5	スパン・レコード
	6	PIC G または PIC N (DBCS データ項目)
	7	データ記述記入項目の OCCURS DEPENDING ON 文節
7	0	データ記述記入項目の SYNCHRONIZED 文節
	1	データ記述記入項目の JUSTIFIED 文節
	2	USAGE IS POINTER 項目
	3	複合 OCCURS DEPENDING ON 文節
	4	DATA DIVISION の外部浮動小数点項目
	5	DATA DIVISION の内部浮動小数点項目
	6	行順次ファイル
	7	USAGE IS PROCEDURE-POINTER 項目または FUNCTION-POINTER 項目

関連参照
333 ページの『LIST』

シグニチャー情報バイト: ENVIRONMENT DIVISION

次の表は、LIST コンパイラー・オプションが使用されたときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

バイト	ビット	項目
8	0	FILE-CONTROL 段落の FILE STATUS 文節
	1	INPUT-OUTPUT SECTION の I-O-CONTROL 段落の RERUN 文節
	2	SPECIAL-NAMES 段落で定義された UPSI スイッチ

シグニチャー情報バイト: PROCEDURE DIVISION 動詞

次の表は、LIST コンパイラー・オプションが使用されたときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

バイト	ビット	項目
9	0	ACCEPT
	1	ADD
	2	ALTER
	3	CALL
	4	CANCEL
	6	CLOSE
10	0	COMPUTE
	2	DELETE
	4	DISPLAY
	5	DIVIDE
11	1	END-PERFORM
	2	ENTER
	3	ENTRY
	4	EXIT
	5	EXEC
	6	GO TO
	7	IF
12	0	INITIALIZE
	1	INVOKE
	2	INSPECT
	3	MERGE
	4	MOVE
	5	MULTIPLY
	6	OPEN
	7	PERFORM

バイト	ビット	項目
13	0	READ
	2	RELEASE
	3	RETURN
	4	REWRITE
	5	SEARCH
	7	SET
14	0	SORT
	1	START
	2	STOP
	3	STRING
	4	SUBTRACT
	7	UNSTRING
15	0	USE
	1	WRITE
	2	CONTINUE
	3	END-ADD
	4	END-CALL
	5	END-COMPUTE
	6	END-DELETE
	7	END-DIVIDE
16	0	END-EVALUATE
	1	END-IF
	2	END-MULTIPLY
	3	END-READ
	4	END-RETURN
	5	END-REWRITE
	6	END-SEARCH
	7	END-START
17	0	END-STRING
	1	END-SUBTRACT
	2	END-UNSTRING
	3	END-WRITE
	4	GOBACK
	5	EVALUATE
	7	SERVICE
18	0	END-INVOKE
	1	END-EXEC
	2	XML
	3	END-XML

戻りコードの検査: コンパイラーから 4 より大きな戻りコードが戻された場合は、プログラム中にあるように情報バイトで示される一部の動詞が、エラーのために破棄された可能性があることを意味します。

シグニチャー情報バイト: 多数の PROCEDURE DIVISION 項目

次の表は、LIST コンパイラー・オプションが使用されたときに提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報を示しています。

バイト	ビット	項目
21	0	16 進数リテラル
	1	更新 GO TO
	2	I-O ERROR 宣言
	3	LABEL 宣言
	4	DEBUGGING 宣言
	5	プログラムのセグメント化
	6	OPEN . . . EXTEND
	7	EXIT PROGRAM
22	0	CALL リテラル
	1	CALL ID
	2	CALL . . . ON OVERFLOW
	3	CALL . . . LENGTH OF
	4	CALL . . . ADDRESS OF
	5	CLOSE . . . REEL/UNIT
	6	使用されている指数
	7	使用されている浮動小数点項目
23	0	COPY
	1	BASIS
	2	プログラムの DBCS 名
	3	プログラムのシフトアウトとシフトイン
	4 〜 7	ASM2 モジュール IGYBINIT への入り口での最高レベルのエラー重大度
24	0	DBCS リテラル
	1	REPLACE
	2	参照変更が使用された
	3	ネストされたプログラム
	4	INITIAL
	5	COMMON
	6	SELECT . . . OPTIONAL
	7	EXTERNAL

バイト	ビット	項目
25	0	GLOBAL
	1	RECORD IS VARYING
	2	LABEL 宣言で使用される ACCEPT FROM SYSIPT
	3	LABEL 宣言で使用される DISPLAY UPON SYSLST
	4	LABEL 宣言で使用される DISPLAY UPON SYSPCH
	5	組み込み関数が使用された
29	0	プログラムにおける Java ベースのオブジェクト指向構文
	1	プログラムで使用される FUNCTION RANDOM
	2	プログラムで使用される NATIONAL データ

関連参照

333 ページの『LIST』

例: ソース・コードから生成されるアセンブラー・コード

次の例は、LIST コンパイラー・オプションを使用したときに、ソース・コードから生成されるアセンブラー・コードのリストを示しています。このリストを使って、失敗した命令に対応する COBOL 動詞を見つけることができます。

```
DATA VALIDATION AND UPDATE PROGRAM    IGYTCARA  Date 09/30/2001 Time 12:26:53 Page  52
000433  MOVE
000435  READ
000436  SET   (1)

    (2)          (3)              (5)                        (6)
    000F26  92E8 A00A              MVI  10(10),X'E8'          LOCATION-EOF-FLAG
    000F2A                GN=13    EQU  *
    000F2A  47F0 B426              BC   15,1062(0,11)         GN=75(000EFA)
    000F2E                GN=74    EQU  *
000439  IF
    000F2E  95E8 A00A              CLI  10(10),X'E8'          LOCATION-EOF-FLAG
    000F32  4780 B490              BC   8,1168(0,11)          GN=14(000F64)
000440  DISPLAY
    000F36  5820 D05C              L    2,92(0,13)            TGTFIXD+92
    000F3A  58F0 202C              L    15,44(0,2)            V(IGZCDSP )
    000F3E  4110 97FF              LA   1,2047(0,9)           PGMLIT AT +1999
    000F42  05EF                   BALR 14,15
000443  CALL
    000F44  4130 A012              LA   3,18(0,10)            COMP-CODE
    000F48  5030 D21C              ST   3,540(0,13)           TS2=4
    000F4C  9680 D21C              OI   540(13),X'80'         TS2=4
    000F50  4110 D21C              LA   1,540(0,13)           TS2=4
    000F54  58F0 9000              L    15,0(0,9)             V(ILBOABN0)
    000F58  05EF                   BALR 14,15
    000F5A  50F0 D078              ST   15,120(0,13)          TGTFIXD+120
    000F5E  BF38 D089              ICM  3,8,137(13)           TGTFIXD+137
    000F62  0430                   SPM  3,0
    000F64        (4)     GN=14    EQU  *
    000F64  5820 D154              L    2,340(0,13)           VN=3
    000F68  07F2                   BCR  15,2
```

(1) ソース行番号と、COBOL 動詞、段落名、またはセクション名。

行 000436 の SET は COBOL 動詞です。名前の前のアスタリスク (*) は、その名前が段落名またはセクション名であることを示しています。

(2) モジュール内でのオブジェクト・コード命令の相対位置 (16 進表記)。

(3) オブジェクト・コード命令 (16 進表記)。

16 進数字の最初の 2 桁または 4 桁が命令で、残りの桁は命令オペランドです。2 つのオペランドのある命令もあります。

(4) コード・シーケンス用のコンパイラー生成名 (GN)。

(5) アセンブラー言語に非常に類似した形式のオブジェクト・コード命令。

(6) オブジェクト・コード命令に関するコメント。

- マシン命令で使用される 1 つまたは 2 つのオペランドは、右側に表示されます。2 つ以上の構造で定義される (このようにして、ソース・プログラム内で修飾によって固有にされた) データ名には、その直後にアスタリスクが置かれます。
- オペランドとして出てくる生成ラベルの相対位置は、括弧内に表示されます。

関連参照

385 ページの『LIST および MAP 出力で使用される記号』

例: TGT メモリー・マップ

次の例は、タスク・グローバル・テーブル (TGT) の LIST 出力を示しています。この出力には、プログラムが実行される環境に関する情報が伴います。

```
DATA VALIDATION AND UPDATE PROGRAM    IGYTCARA  Date 09/30/2001 Time 12:26:53 Page  132
                  *** TGT MEMORY MAP ***
            (1)      (2)       (3)
           PGMLOC   TGTLOC

           005160   000000   RESERVED - 72 BYTES
           0051A8   000048   TGT IDENTIFIER
           0051AC   00004C   RESERVED - 4 BYTES
           0051B0   000050   TGT LEVEL INDICATOR
           0051B1   000051   RESERVED - 3 SINGLE BYTE FIELDS
           0051B4   000054   32 BIT SWITCH
           0051B8   000058   POINTER TO RUNCOM
           0051BC   00005C   POINTER TO COBVEC
           0051C0   000060   POINTER TO PROGRAM DYNAMIC BLOCK TABLE
           0051C4   000064   NUMBER OF FCB'S
           0051C8   000068   WORKING-STORAGE LENGTH
           0051CC   00006C   RESERVED - 4 BYTES
           0051D0   000070   ADDRESS OF IGZESMG WORK AREA
           0051D4   000074   ADDRESS OF 1ST GETMAIN BLOCK (SPACE MGR)
           0051D8   000078   RESERVED - 2 BYTES
           0051DA   00007A   RESERVED - 2 BYTES
           0051DC   00007C   RESERVED - 2 BYTES
           0051DE   00007E   MERGE FILE NUMBER
           0051E0   000080   ADDRESS OF CEL COMMON ANCHOR AREA
           0051E4   000084   LENGTH OF TGT
           0051E8   000088   RESERVED - 1 SINGLE BYTE FIELD
           0051E9   000089   PROGRAM MASK USED BY THIS PROGRAM
           0051EA   00008A   RESERVED - 2 SINGLE BYTE FIELDS
           0051EC   00008C   NUMBER OF SECONDARY FCB CELLS
           0051F0   000090   LENGTH OF THE ALTER VN(VNI) VECTOR
           0051F4   000094   COUNT OF NESTED PROGRAMS IN COMPILE UNIT
           0051F8   000098   DDNAME FOR DISPLAY OUTPUT
           005200   0000A0   RESERVED - 8 BYTES
           005208   0000A8   POINTER TO COM-REG SPECIAL REGISTER
           00520C   0000AC   CALC ROUTINE REGISTER SAVE AREA
           005240   0000E0   ALTERNATE COLLATING SEQUENCE TABLE PTR.
           005244   0000E4   ADDRESS OF SORT G.N. ADDRESS BLOCK
           005248   0000E8   ADDRESS OF PGT
           00524C   0000EC   CURRENT INTERNAL PROGRAM NUMBER
           005250   0000F0   POINTER TO 1ST IPCB
           005254   0000F4   ADDRESS OF THE CLLE FOR THIS PROGRAM
           005258   0000F8   POINTER TO ABEND INFORMATION TABLE
```

```
00525C  0000FC  POINTER TO TEST INFO FIELDS IN THE TGT
005260  000100  ADDRESS OF START OF COBOL PROGRAM
005264  000104  POINTER TO ALTER VNI'S IN CGT
005268  000108  POINTER TO ALTER VN'S IN TGT
00526C  00010C  POINTER TO FIRST PBL IN THE PGT
005270  000110  POINTER TO FIRST FCB CELL
005274  000114  WORKING-STORAGE ADDRESS
005278  000118  POINTER TO FIRST SECONDARY FCB CELL
00527C  00011C  POINTER TO STATIC CLASS INFO BLOCK

        *** VARIABLE PORTION OF TGT ***

005280  000120  BASE LOCATORS FOR SPECIAL REGISTERS
005288  000128  BASE LOCATORS FOR WORKING-STORAGE  (4)
005290  000130  BASE LOCATORS FOR LINKAGE-SECTION
005294  000134  BASE LOCATORS FOR FILES
0052A8  000148  BASE LOCATORS FOR ALPHANUMERIC TEMPS
0052AC  00014C  CLLE ADDR. CELLS FOR CALL LIT. SUB-PGMS.
0052C8  000168  INDEX CELLS
0052EC  00018C  FCB CELLS
005300  0001A0  ALL PARAMETER BLOCK
005364  000204  INTERNAL PROGRAM CONTROL BLOCKS
```

(1) COBOL プログラムの先頭からの TGT フィールドの 16 進オフセット。RENT オプションを使用してコンパイルされたプログラムの場合には、示されません。

(2) TGT の先頭からの TGT フィールドの 16 進オフセット。

(3) TGT フィールドの内容の説明。

(4) COBOL データ域のベース・ロケーターの TGT フィールド。

例: WORKING-STORAGE の位置とサイズ

次の例は、NORENT プログラムの WORKING-STORAGE に関する LIST 出力を示しています。

```
   (1)              (2)          (3)

WRK-STOR LOCATED AT 0066D8 FOR 00001598 BYTES
```

(1) WORKING-STORAGE の識別。

(2) COBOL プログラムの先頭からの WORKING-STORAGE の 16 進オフセット。

(3) WORKING-STORAGE の長さ (16 進表記)。

関連概念

38 ページの『ストレージとそのアドレス可能度』

例: DSA メモリー・マップ

次の例は、動的保管域 (DSA) の LIST 出力を示しています。DSA には、別個にコンパイルされたプロシージャーに入ったときに獲得されたストレージの内容に関する情報が含まれています。

```
DATA VALIDATION AND UPDATE PROGRAM    IGYTCARA  Date 09/30/2001 Time 12:26:53 Page  139
          *** DSA MEMORY MAP ***
          (1)     (2)
          DSALOC

          000000  REGISTER SAVE AREA
          00004C  STACK NAB (NEXT AVAILABLE BYTE)
          00005C  ADDRESS OF TGT
          000058  ADDRESS OF INLINE-CODE PRIMARY DSA
          000080  PROCEDURE DIVISION RETURNING VALUE

          *** VARIABLE PORTION OF DSA ***
          000084  BACKSTORE CELLS FOR SYMBOLIC REGISTERS
          00009C  VARIABLE-LENGTH CELLS
          0000A8  ODO SAVE CELLS
          0000B4  VARIABLE NAME (VN) CELLS FOR PERFORM
          000124  PERFORM SAVE CELLS
          000250  TEMPORARY STORAGE-1
          000260  TEMPORARY STORAGE-2
          000430  OPTIMIZER TEMPORARY STORAGE
```

(1) DSA の先頭からの動的保管域 (DSA) フィールドの 16 進オフセット。

(2) DSA フィールドの内容の説明。

例: XREF 出力 - データ名相互参照

次の例は、データ名のソート済み相互参照を示しています。これは、XREF コンパイラー・オプションによって作成されます。

```
An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

    (1)        (2)                            (3)

 Defined   Cross-reference of data names   References

     264   ABEND-ITEM1
     265   ABEND-ITEM2
     347   ADD-CODE . . . . . . . . . . .  1126 1192
     381   ADDRESS-ERROR. . . . . . . . .  M1156
     280   AREA-CODE. . . . . . . . . . .  1266 1291 1354 1375
     382   CITY-ERROR . . . . . . . . . .  M1159

       (4)

Context usage is indicated by the letter preceding a procedure-name reference.
These letters and their meanings are:
    A = ALTER (procedure-name)
    D = GO TO (procedure-name) DEPENDING ON
    E = End of range of (PERFORM) through (procedure-name)
    G = GO TO (procedure-name)
    P = PERFORM (procedure-name)
    T = (ALTER) TO PROCEED TO (procedure-name)
    U = USE FOR DEBUGGING (procedure-name)

    (5)       (6)                             (7)

 Defined   Cross-reference of procedures   References

     877   000-DO-MAIN-LOGIC
     943   050-CREATE-VSAM-MASTER-FILE. .  P879
     995   100-INITIALIZE-PARAGRAPH . . .  P881
    1471   1100-PRINT-I-F-HEADINGS. . . .  P926
    1511   1200-PRINT-I-F-DATA. . . . . .  P928
    1573   1210-GET-MILES-TIME. . . . . .  P1540
    1666   1220-STORE-MILES-TIME. . . . .  P1541
    1682   1230-PRINT-SUB-I-F-DATA. . . .  P1562
    1706   1240-COMPUTE-SUMMARY . . . . .  P1563
    1052   200-EDIT-UPDATE-TRANSACTION. .  P890
    1154   210-EDIT-THE-REST. . . . . . .  P1145
    1189   300-UPDATE-COMMUTER-RECORD . .  P893
    1237   310-FORMAT-COMMUTER-RECORD . .  P1194 P1209
    1258   320-PRINT-COMMUTER-RECORD. . .  P1195 P1206 P1212 P1222
    1318   330-PRINT-REPORT . . . . . . .  P1208 P1232 P1286 P1310 P1370 P1395 P1399
    1342   400-PRINT-TRANSACTION-ERRORS .  P896
```

データ名の相互参照:

(1) その名前が定義されている行番号。

(2) データ名。

(3) その名前が使用されている行番号。M が行番号の前に置かれている場合は、データ項目がその位置で明示的に変更されたことを意味します。

プロシージャー参照の相互参照:

(4) プロシージャー参照のコンテキスト取扱コードの説明。

(5) そのプロシージャー名が定義されている行番号。

(6) プロシージャー名。

(7) そのプロシージャーが参照されている行番号およびそのプロシージャーのコンテキスト取扱コード。

例: XREF 出力 - プログラム名相互参照

次の例は、プログラム名のソート済み相互参照を示しています。これは、XREF コンパイラー・オプションによって作成されます。

```
   (1)       (2)                          (3)

Defined   Cross-reference of programs     References


EXTERNAL   EXTERNAL1. . . . . . . . . . . 25
      2   X. . . . . . . . . . . . . . .  41
     12   X1 . . . . . . . . . . . . . .  33 7
     20   X11. . . . . . . . . . . . . .  25 16
     27   X12. . . . . . . . . . . . . .  32 17
     35   X2 . . . . . . . . . . . . . .  40 8
```

(1) そのプログラム名が定義されている行番号。プログラムが外部の場合は、定義行番号の代わりに EXTERNAL という語が表示されます。

(2) プログラム名。

(3) そのプログラムが参照されている行番号。

例: 組み込み相互参照

次の例は、ソース・リストに組み込まれた変更済み相互参照を示しています。これは、XREF コンパイラー・オプションによって作成されます。

```
LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  Map and Cross Reference
  . . .
000878               procedure division.
000879                 000-do-main-logic.
000880                    display "PROGRAM IGYTCARA - Beginning".
000881                    perform 050-create-vsam-master-file.                              932 (1)
000882                    perform 100-initialize-paragraph.                                 984
000883                    read update-transaction-file into ws-transaction-record           204 340
000884                        at end
000885      1                   set transaction-eof to true                               254
000886                    end-read.
  . . .
000984                 100-initialize-paragraph.
000985                    move spaces to ws-transaction-record                           IMP 340 (2)
000986                    move spaces to ws-commuter-record                              IMP 316
000987                    move zeroes to commuter-zipcode                                IMP 327
000988                    move zeroes to commuter-home-phone                             IMP 328
000989                    move zeroes to commuter-work-phone                             IMP 329
000990                    move zeroes to commuter-update-date                            IMP 333
000991                    open input update-transaction-file                             204
000992                        location-file                                              193
000993                        i-o commuter-file                                          181
000994                        output print-file                                          217
  . . .
001442               1100-print-i-f-headings.
001443
001444                    open output print-file.                                        217
001445
001446                    move function when-compiled to when-comp.                      IFN 698 (2)
001447                    move when-comp (5:2) to compile-month.                         698 640
001448                    move when-comp (7:2) to compile-day.                           698 642
001449                    move when-comp (3:2) to compile-year.                          698 644
001450
001451                    move function current-date (5:2) to current-month.             IFN 649
001452                    move function current-date (7:2) to current-day.               IFN 651
001453                    move function current-date (3:2) to current-year.              IFN 653
001454
001455                    write print-record from i-f-header-line-1                      222 635
001456                          after new-page.                                          138
  . . .
```

(1) プログラム内のデータ名またはプロシージャー名の定義の行番号。

(2) 特殊定義記号:

UND ユーザー名が未定義です。

DUP ユーザー名が 2 回以上定義されています。

IMP 暗黙的に定義された名前 (特殊レジスターや形象定数など)。

IFN 組み込み関数参照。

EXT 外部参照。

***** NOCOMPILE オプションが有効なため、プログラム名が未解決です。

例: OFFSET コンパイラー出力

次の例は、圧縮された動詞のリスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラルが含まれているリストを示しています。これは、OFFSET コンパイラー・オプションからの出力です。

```
DATA VALIDATION AND UPDATE PROGRAM    IGYTCARA  Date 09/30/2001  Time 12:26:53  Page  54
  . . .
  (1)     (2)    (3)
LINE #  HEXLOC  VERB         LINE #  HEXLOC  VERB         LINE #  HEXLOC  VERB
000880 0026F0 DISPLAY        000881 002702 PERFORM        000933 002702 OPEN
000934 002722 IF             000935 00272C DISPLAY        000936 002736 PERFORM
001389 002736 DISPLAY        001390 002740 DISPLAY        001391 00274A DISPLAY
001392 002754 DISPLAY        001393 00275E DISPLAY        001394 002768 DISPLAY
001395 002772 DISPLAY        000937 00277C PERFORM        001434 00277C DISPLAY
001435 002786 STOP           000939 0027A2 MOVE           000940 0027AC WRITE
000941 0027D6 IF             000942 0027E0 DISPLAY        000943 0027EA PERFORM
001389 0027EA DISPLAY        001390 0027F4 DISPLAY        001391 0027FE DISPLAY
001392 002808 DISPLAY        001393 002812 DISPLAY        001394 00281C DISPLAY
001395 002826 DISPLAY        000944 002830 DISPLAY        000945 00283A PERFORM
001403 00283A DISPLAY        001404 002844 DISPLAY        001405 00284E DISPLAY
001406 002858 DISPLAY        001407 002862 CALL           000947 002888 CLOSE
```

(1) 行番号。ユーザーの行番号またはコンパイラー生成の行番号がリストされます。

(2) この動詞用に生成されたコードの、プログラムの先頭からのオフセット (16 進表記)。

動詞は、発生順に使用されるたびに 1 度ずつリストされます。

(3) 使用される動詞。

関連参照

338 ページの『OFFSET』

例: VBREF コンパイラー出力

次の例は、プログラム内のすべての動詞のアルファベット順のリストと、各動詞が参照されている場所を示しています。このリストは、VBREF コンパイラー・オプションによって作成されます。

```
(1)      (2)                        (3)

2        ACCEPT . . . . . .  101  101
2        ADD. . . . . .  129  130
1        CALL . . . . . .  140
5        CLOSE. . . . . .  90  94  97  152  153
20       COMPUTE. . . . . .  150  164  164  165  166  166  166  166  167  168  168  169  169  170  171  171
                             171  172  172  173
2        CONTINUE . . . . . .  106  107
2        DELETE . . . . . .  96  119
47       DISPLAY. . . . . .  88  90  91  92  92  93  94  94  94  95  96  96  97  99  99  100  100  100  100
                             103  109  117  117  118  119  138  139  139  139  139  139  140  140  140
                             140  143  148  148  149  149  149  152  152  152  153  162
2        EVALUATE . . . . . .  116  155
47       IF . . . . . .  88  90  93  94  94  95  96  96  97  99  100  103  105  105  107  107  107  109
                             110  111  111  112  113  113  113  113  114  114  115  115  116  118  119  124
                             124  126  127  129  132  133  134  135  136  148  149  152  152
183      MOVE . . . . . .  90  93  95  98  98  98  98  98  99  100  101  101  102  104  105  105  106  106
                             107  107  108  108  108  108  108  108  109  110  111  112  113  113  113  114
                             114  114  115  115  116  116  117  117  117  118  118  118  119  119  120  121
                             121  121  121  121  121  121  121  121  121  122  122  122  122  122  123  123
                             123  123  123  123  123  124  124  124  125  125  125  125  125  125  125  126
                             126  126  126  126  127  127  127  127  128  128  129  129  130  130  130  130
                             131  131  131  131  131  132  132  132  132  132  132  133  133  133  133  133
                             134  134  134  134  134  135  135  135  135  135  135  136  136  137  137  137
                             137  137  138  138  138  138  141  141  142  142  144  144  144  144  145  145
                             145  145  146  149  150  150  150  151  151  155  156  156  157  157  158  158
                             159  159  160  160  161  161  162  162  162  168  168  168  169  169  170  171
                             171  172  172  173  173
5        OPEN . . . . . .  93  95  99  144  148
62       PERFORM. . . . . .  88  88  88  88  89  89  89  91  91  91  91  93  93  94  94  95  95  95  95  96
                             96  96  97  97  97  100  100  101  102  104  109  109  111  116  116  117  117
                             117  118  118  118  118  119  119  119  120  120  124  125  127  128  133  134
                             135  136  136  137  150  151  151  153  153
8        READ . . . . . .  88  89  96  101  102  108  149  151
1        REWRITE. . . . . .  118
4        SEARCH . . . . . .  106  106  141  142
46       SET. . . . . .  88  89  101  103  104  105  106  108  108  136  141  142  149  150  151  152  154
                             155  156  156  156  156  157  157  157  157  158  158  158  158  159  159  159
                             159  160  160  160  160  161  161  161  161  162  162  164  164
2        STOP . . . . . .  92  143
4        STRING . . . . . .  123  126  132  134
33       WRITE. . . . . .  94  116  129  129  129  129  129  130  130  130  130  145  146  146  146  146  147
                             147  151  165  165  166  166  167  174  174  174  174  174  174  174  175  175
```

(1) その動詞がプログラムで使用されている回数

(2) 動詞

(3) その動詞が使用されている行番号

デバッガーの使用準備

デバッガーで使用するように実行可能な COBOL プログラムを準備する場合は、TEST オプションを使用してください。

デバッグ・ツールのデバッガーは、Enterprise COBOL コンパイラーの Full Function フィーチャーにより提供されます。デバッグ・ツールを使用してプログラムの実行をステップスルーする場合は、TEST コンパイラー・オプションを指定してください。リモート・デバッグの場合は、分散デバッガーが、デバッグ情報に対するクライアント・グラフィカル・ユーザー・インターフェースを提供します。このデバッグ情報は、z/OS または UNIX 上で実行するデバッグ・ツール・エンジンによって提供されます。

TEST のサブオプション SEPARATE を指定すると、デバッグ・ツール用の記号情報テーブルをオブジェクト・モジュールとは別個のデータ・セットに生成することができます。また、コンパイル時組み込みフック (TEST オプションがオフであっても、パフォーマンスの低下が起こる) ではなく、オーバーレイ・フック (*実動デバッグ*) を使用して、デバッグ用の COBOL プログラムを使用可能にすることができます。

関連タスク

Debug Tool User's Guide and Reference の『Preparing your program for debugging』の項

関連参照

349 ページの『TEST』

第 3 部 特定の環境に合わせた COBOL プログラムの目標

第 20 章 COBOL プログラムの開発 (CICS の場合) 407
CICS のもとで実行する COBOL プログラムのコーディング 408
ファイル入出力のコーディング 408
システム日付および時刻の検索 408
データ項目の内容の表示. 409
COBOL プログラムとの間の呼び出し 409
ネストされたプログラムのコーディング . . . 410
16MB 境界より上で実行する COBOL プログラムのコーディング 410
ECI 呼び出しの成否の判断 411
CICS オプションを使用したコンパイル. 411
一連のプログラムのコンパイル 412
CICS サブオプションの分離 412
組み込みの CICS トランスレーター. 413
分離型の CICS トランスレーターの使用 414
CICS 予約語テーブル. 416
CICS HANDLE を使用したエラー処理 416
例: CICS HANDLE を使用したエラー処理 . . 417

第 21 章 DB2 環境用のプログラミング 419
SQL ステートメントのコーディング. 419
DB2 coprocessor を用いた SQL INCLUDE の使用 420
2 進数項目の使用 420
SQL ステートメントの成否の判断 420
SQL オプションを使用したコンパイル 421
バッチでのコンパイル 421
DB2 サブオプションの分離. 422
DB2 coprocessor 422

第 22 章 IMS のもとでの COBOL プログラムの実行 425
IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク 425

第 23 章 UNIX のもとでの COBOL プログラムの実行 427
UNIX 環境での実行 427
環境変数の設定およびアクセス 429
実行に影響を与える環境変数の設定 429
COBOL プログラムに重要な環境変数 . . . 429
環境変数のリセット 429
環境変数へのアクセス 429
例: 環境変数へのアクセス 430
UNIX/POSIX API の呼び出し 430
fork、exec および spawn. 431
サンプル 431
メインプログラム・パラメーターへのアクセス . . 432
例: メインプログラム・パラメーターへのアクセス 433

© Copyright IBM Corp. 1991, 2001

第 20 章 COBOL プログラムの開発 (CICS の場合)

CICS 用に作成された COBOL プログラムは、CICS Transaction Server のもとで実行することができます。 CICS サービスを使用する CICS COBOL アプリケーション・プログラムは、CICS のコマンド・レベル・インターフェースを使用する必要があります。

CICS コンパイラー・オプションが指定されていると、Enterprise COBOL コンパイラーは、ソース・プログラム内のネイティブ COBOL ステートメントと組み込みの CICS ステートメントの両方を処理します。COBOL (OS/390 および VM 版) バージョン 2 リリース 2 より前のコンパイラーの場合、EXEC CICS コマンドを COBOL コードに変換するには、分離型の変換ステップが必要です。組み込みの CICS ステートメントは、分離型の変換も引き続き可能ですが、組み込みの CICS トランスレーターを使用することを推奨します。使用する CICS トランスレーターによっては、コーディングの変更が必要になる場合があります。どのトランスレーターを使用する場合でも、EXEC CICS コマンドは 1 つまたは複数の COBOL ステートメントに置き換えられます (そのうちの 1 つは CALL ステートメントです)。

組み込みの CICS トランスレーターを使用するには、CICS Transaction Server バージョン 2 が必要です。

プログラムをコンパイルし、リンク・エディットした後、CICS テーブルの更新などその他のステップを実行する必要があります。これらのステップを実行しないと、CICS のもとで COBOL プログラムを実行することはできません。しかし、これらの CICS トピックは、本書の説明範囲を超えています。詳細については、以下の CICS 資料を参照してください。

実行時エラーの処理方法を判別するには、CBLPSHPOP ランタイム・オプションを設定します。詳細については、以下の CICS HANDLE および CBLPSHPOP に関する情報を参照してください。

関連概念
413 ページの『組み込みの CICS トランスレーター』

関連タスク
408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
411 ページの『CICS オプションを使用したコンパイル』
414 ページの『分離型の CICS トランスレーターの使用』
416 ページの『CICS HANDLE を使用したエラー処理』
CICS のもとでの CBLPSHPOP ランタイム・オプションの使用 (*言語環境プログラム プログラミング・ガイド*)

関連参照
317 ページの『CICS』
CICS アプリケーション・プログラミング・ガイド

© Copyright IBM Corp. 1991, 2001

CICS のもとで実行する COBOL プログラムのコーディング

プログラムをコーディングするには、PROCEDURE DIVISION での CICS コマンドのコーディング方法を知る必要があります。次のような基本形式をとります。

```
EXEC CICS command name and command options
END-EXEC
```

EXEC コマンド内では、スペースをワード分離文字として使用してください。コンマやセミコロンは使用しないでください。

CICS のもとで実行するプログラムをコーディングするときは、以下のことを行わないでください。

- FILE-CONTROL 記入項目を ENVIRONMENT DIVISION で使用する (ただし、その FILE-CONTROL 記入項目が SORT ステートメントで使用される場合を除く)
- DATA DIVISION の FILE SECTION を使用する (ただし、FILE SECTION が SORT ステートメントに使用される場合を除く)
- メインプログラムに対してユーザー指定のパラメーターを使用する
- USE 宣言部分 (USE FOR DEBUGGING を除く) を使用する
- 次の COBOL 言語ステートメントを使用する

  ACCEPT 形式 1: データ転送 (システムの日付と時刻を取得するには、形式 2 の ACCEPT を使用できます)
  CLOSE
  DELETE
  DISPLAY UPON SYSPUNCH
  DISPLAY UPON CONSOLE
  MERGE
  OPEN
  READ
  RERUN
  REWRITE
  START
  STOP *literal*
  WRITE

分離型の CICS トランスレーターを使用する場合、EXEC コマンドを含む REPLACE ステートメントはすべて、プログラムの PROCEDURE DIVISION ステートメントの後に置く必要があります。そうしないと、これらのステートメントは変換されません。

ファイル入出力のコーディング

ほとんどの入出力処理には CICS コマンドを使用しなければなりません。したがって、ファイルを記述したり、OPEN、CLOSE、READ、START、REWRITE、WRITE、または DELETE ステートメントをコーディングすることはありません。かわりに、CICS コマンドを使用してデータの検索、更新、挿入、および削除を行います。

システム日付および時刻の検索

CICS プログラムでは、形式 1 の ACCEPT ステートメントは使用できません。

以下の形式 2 の ACCEPT ステートメントを CICS 環境で使用すると、システム日付を入手することができます。

- ACCEPT *identifier-2* FROM DATE
- ACCEPT *identifier-2* FROM DATE YYYYMMDD
- ACCEPT *identifier-2* FROM DAY
- ACCEPT *identifier-2* FROM DAY YYYYDDD
- ACCEPT *identifier-2* FROM DAY-OF-WEEK

次に示す形式 2 の ACCEPT ステートメントを CICS 環境で使用すると、システム時刻を入手することができます。

- ACCEPT *identifier-2* FROM TIME

システムの日付と時刻に関する情報を取得するには、ACCEPT ステートメントを使用することをお勧めします。このステートメントは、CICS 環境でも非 CICS 環境でも有効です。

データ項目の内容の表示

CICS のもとでは、システム論理出力装置 (SYSOUT、SYSLIST、SYSLST) に対する DISPLAY がサポートされます。DISPLAY 出力は、言語環境プログラムのメッセージ・ファイル (一時データ・キュー CESE) に書きこまれます。

ただし、DISPLAY . . . UPON CONSOLE と DISPLAY . . . UPON SYSPUNCH はサポートされません。

COBOL プログラムとの間の呼び出し

CALL ステートメントを使用すると、VS COBOL II、COBOL (MVS および VM 版)、COBOL (OS/390 および VM 版)、および Enterprise COBOL のプログラムの間で呼び出しを行うことができます。しかし、これらのプログラムから OS/VS COBOL プログラムを呼び出したり、OS/VS COBOL からこれらのプログラムを呼び出す場合には、CALL ステートメントを使用することはできません。代わりに、EXEC CICS LINK を使用してください。 分離型の CICS トランスレーターまたは組み込みの CICS トランスレーターで処理された、個別にコンパイルされた COBOL プログラムを呼び出す場合は、DFHEIBLK および DFHCOMMAREA を、CALL ステートメントの 1 番目と 2 番目のパラメーターとして渡す必要があります。

CALL *ID* を NODYNAM コンパイラー・オプションと一緒に使用すると、プログラムを動的に呼び出すことができます。呼び出されるプログラムには、その言語に関して CICS がサポートする機能を入れることができます。動的に呼び出されるプログラムは、CICS 自動インストールを使用していない場合は、CICS プログラム処理テーブル (PPT) に定義されていなければなりません。

他の高水準言語との言語間通信 (ILC) がサポートされます。ILC がサポートされていない場合は、代わりに CICS の LINK、XCTL、および RETURN を使用できます。

次の表に、COBOL プログラムとアセンブラー言語プログラム間の呼び出し関係を示しています。この表では、*言語環境プログラム プログラミング・ガイド* で解説されているインターフェースに準拠しているアセンブラー言語プログラムを *LE 準拠* アセンブラー・プログラムと呼んでいます。このインターフェースに準拠してい

ないアセンブラー言語プログラムは、*非 LE 準拠* アセンブラー・プログラムと呼んでいます。

COBOL プログラムとアセンブラー・プログラム間の呼び出し	LE 準拠アセンブラー・プログラム	非 LE 準拠アセンブラー・プログラム
Enterprise COBOL プログラムからアセンブラー・プログラムを呼び出せるか	はい	はい
アセンブラー・プログラムから Enterprise COBOL を呼び出せるか	はい (ただし、アセンブラー・プログラムがメインプログラムでない場合)	いいえ

ネストされたプログラムのコーディング

組み込みの CICS トランスレーターを使用してコンパイルすると、トランスレーターでは、最外部のプログラムの GLOBAL 文節を使用して、DFHEIBLK 制御ブロックと DFHCOMMAREA 制御ブロックを生成します。したがって、ネストされたプログラムをコーディングするときは、ネストされたプログラムの呼び出しに対する引き数として、これらの制御ブロックを渡す必要がありません。

ネストされたプログラムをコーディングするときに、分離型の CICS トランスレーターを使用する予定の場合は、DFHEIBLK および DFHCOMMAREA を、EXEC コマンドを含むネストされたプログラムに対して、パラメーターとして渡すか、EIB (EXEC インターフェース・ブロック) への参照として渡します。制御階層内で、このようなネストされたプログラムとその最上位のプログラムとの間にあるプログラムにも、同じパラメーターを渡す必要があります。

16MB 境界より上で実行する COBOL プログラムのコーディング

Enterprise COBOL のもとでは、16MB 境界より上で実行する COBOL プログラムをコーディングする場合、以下の制約事項が適用されます。

- IMS/ESA バージョン 6 (またはそれ以降) を DBCTL なしで使用しているときは、DL/I CALL ステートメントがサポートされるのは、呼び出しで渡されるすべてのデータが 16MB 境界より下に常駐している場合だけです。このため、DATA(24) コンパイラー・オプションを指定する必要があります。しかし、IMS/ESA バージョン 6 (またはそれ以降) を DBCTL とともに使用している場合は、DATA(31) コンパイラー・オプションを使用して、16MB より上に常駐するデータを渡すことができます。

  DL/I CALL ステートメントではなく EXEC DLI を使用する場合は、IMS 製品のレベルに関係なく、DATA(31) を指定することができます。
- 受け取りプログラムが AMODE 31 でリンク・エディットされている場合、渡されるアドレスは、31 ビットまたは 24 ビットの長さで、左端バイトが 0 に設定されていなければなりません。
- 受け取りプログラムが AMODE 24 でリンク・エディットされている場合、渡されるアドレスは 24 ビットの長さでなければなりません。

ECI 呼び出しの成否の判断

外部 CICS インターフェース (ECI) の呼び出しの後で、RETURN-CODE 特殊レジスターの内容は予期できない値に設定されます。このため、外部 CICS インターフェースを正常に使用した後、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了する可能性があります。終了時に意味のある戻りコードが渡されるようにするには、プログラムを終了する前に、RETURN-CODE 特殊レジスターを設定します。

CICS への最後の呼び出しに関する状況をジョブの戻りコードに反映させるには、外部 CICS インターフェースへの最後の呼び出しで戻された応答コードに基づいて RETURN-CODE 特殊レジスターを設定します。

関連タスク

259 ページの『プログラム呼び出し時のエラーの処理』

CICS のもとでの ILC (*言語環境プログラム ILC (言語間通信) アプリケーションの作成*)

関連参照

CICS External Interfaces Guide

CICS オプションを使用したコンパイル

CICS コンパイラー・オプションを使用すると、組み込みの CICS トランスレーターが使用可能になり、CICS サブオプションを指定することができます。NOCICS オプションを指定場合、コンパイラーはソース・プログラムで検出した CICS ステートメントをすべて診断し破棄します。これまで分離型の CICS トランスレーターを使用していた場合は、NOCICS コンパイラー・オプションを指定する必要があります。

CICS オプションは、任意のコンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS ステートメントまたは CBL ステートメント、あるいはインストール・デフォルト) で指定することができます。CICS オプションが COBOL インストール・デフォルトである場合、CICS サブオプションを指定することはできません。ただし、組み込みの CICS トランスレーターで行った変更は非 CICS アプリケーションには適切でないので、CICS オプションをインストール・デフォルトとして使用することは推奨しません。

Enterprise COBOL の規則に従い、CBL ステートメントや PROCESS ステートメントはすべて、コメント行より前に置く必要があります。

CICS コンパイラー・オプションで指定された CICS サブオプション・ストリングは、組み込みの CICS トランスレーターで使用可能になります。このストリングの内容は、組み込みの CICS トランスレーターに対してのみ有効です。

組み込みの CICS トランスレーターを使用するときは、以下のオプションを指定してコンパイルする必要があります。

コンパイラー・オプション	説明
CICS	NOLIB、DYNAM、または NORENT を指定した場合、コンパイラーでは、LIB、NODYNAM、および RENT を強制します。

コンパイラー・オプション	説明
LIB	CICS とともに、このオプションを有効にする必要があります。
NODYNAM	CICS とともに、このオプションを有効にする必要があります。
RENT	CICS とともに、このオプションを有効にする必要があります。
SIZE(xxx)	xxx は、組み込みの CICS トランスレーターのための十分なストレージをユーザー領域に残すようなサイズ値 (MAX ではない) に設定する必要があります。

さらに、CICS のもとでサポートされない言語エレメントに対してコンパイラーでフラグが付けられるように、コンパイラー・オプション WORD(CICS) を使用することをお勧めします。

COBOLによって提供される標準の JCL プロシージャー・ステートメントを使用すると、組み込みの CICS トランスレーターを使用してプログラムをコンパイルすることができます。前述のコンパイラー・オプションを指定することに加え、JCL を変更して、COBOL ステップ用の STEPLIB オーバーライドを指定し、組み込みの CICS トランスレーター・サービスが含まれたデータ・セットを追加する必要があります (これらのサービスがリンク・リストに含まれている場合を除く)。CICS Transaction Server V2R2 用のデータ・セットのデフォルト名は CICSTS22.CICS.SDFHLOAD ですが、インストール先によってはこの名前が変更されている場合があります。たとえば、以下の行が JCL に含まれている場合があります。

```
//STEPLIB  DD  DSN=CICSTS22.CICS.SDFHLOAD,DISP=SHR
```

COBOL コンパイラー・リストには、組み込みの CICS トランスレーターが生成するエラー診断 (CICS ステートメントの構文エラーなど) が含まれます。このリストは入力ソースを反映するものであり、組み込みの CICS トランスレーターが生成する COBOL ステートメントは含まれていません。

一連のプログラムのコンパイル

CICS オプションを使用して、一連の COBOL プログラムを含むソース・ファイルをコンパイルするときは、オプションの優先順位は次のようになります。

1. コンパイル単位を開始する CBL カードまたは PROCESS カードに指定されたオプション
2. コンパイラーを開始するときに指定されたオプション
3. CICS デフォルト・オプション

CICS サブオプションの分離

複数の CICS オプションの指定は連結されるので、CICS サブオプションが単一の CBL ステートメントに収まらない場合は、複数の CBL ステートメントに分けることができます。

サブオプション・ストリングに組み込まれる CICS サブオプションは累積されます。コンパイラーは、複数のソースからのこれらのサブオプションを、指定された順に連結します。たとえば、JCL ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//PARM.COBOL="CICS("FLAG(I)")"
//COBOL.SYSIN DD *
  CBL CICS("DEBUG")
  CBL CICS("LINKAGE")
  IDENTIFICATION DIVISION.
  PROGRAM-ID. COBOL1.
```

コンパイル時に、コンパイラーは次のサブオプション・ストリングを組み込みの CICS トランスレーターに渡します。

```
"FLAG(I) DEBUG LINKAGE"
```

連結ストリングは、グループの前後に引用符またはアポストロフィを付けて、単一のスペースで区切ります。コンパイラーが同じ CICS サブオプションの複数インスタンスを検出した場合は、連結ストリングの中で最後に指定されたサブオプションが有効になります。コンパイラーでは、連結された CICS サブオプション・ストリングの長さは 4KB に制限されます。

関連概念
『組み込みの CICS トランスレーター』

関連タスク
408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

関連参照
317 ページの『CICS』
CICS アプリケーション・プログラミング・ガイド

組み込みの CICS トランスレーター

CICS コンパイラー・オプションが指定されている場合、COBOL コンパイラーは組み込みの CICS トランスレーターと連動して、ソース・プログラム内のネイティブ COBOL ステートメントと組み込みの CICS ステートメントの両方を処理します。コンパイラーは、CICS ステートメントを検出したとき、およびソース・プログラム内の重要な地点で、組み込みの CICS トランスレーターとインターフェースをとります。トランスレーターは CICS ステートメントに対して適切な処置を行ってから、通常は、生成するネイティブ言語ステートメントを指示してコンパイラーに制御を戻します。ただし、CICS Transaction Server 1.3 を使用している場合は、分離型のトランスレーターを引き続き使用する必要があります。

分離型の CICS トランスレーターの使用は引き続きサポートされますが、組み込みの CICS トランスレーターを使用することをお勧めします。分離型のトランスレーターを使用する場合に適用される一部の制約は、組み込みのトランスレーターを使用する場合には適用されません。

- デバッグ・ツールを使用して、分離型の CICS トランスレーターによって提供される拡張ソースではなく、元のソースをデバッグすることができます。

- コピーブック内の EXEC CICS ステートメントや EXEC DLI ステートメントを個別に変換する必要がありません。
- 変換済みで未コンパイル・バージョンのソース・プログラムに対応する中間データ・セットが不要です。
- 出力リストは 2 つではなく 1 つだけです。
- EXEC CICS が含まれているネストされたプログラムの使用が簡単です。最外部のプログラムの GLOBAL 属性を使用して DFHCOMAREA および DFHEIBLK が生成されます。ネストされたプログラムの呼び出し時にこれらを引き数として指定する必要はなく、ネストされたプログラムの PROCEDURE DIVISION USING ステートメントで指定する必要もありません。
- EXEC CICS ステートメントが含まれているネストされたプログラムを個別のファイルに保管し、COPY ステートメントを使用してこれを組み込むことができます。
- REPLACE ステートメントを EXEC CICS ステートメントに影響させることができます。
- CICS 制御ブロックの 2 進数フィールドは BINARY ではなく COMP-5 の形式で生成されるようになったので、TRUNC コンパイラー・オプションの設定との依存関係がありません。CICS アプリケーションの TRUNC については、アプリケーション・ロジックおよびユーザー定義の 2 進数フィールドの使用に関する要件に従っている限り、任意の設定値を使用することができます。

関連タスク

408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
411 ページの『CICS オプションを使用したコンパイル』

関連参照

353 ページの『TRUNC』

分離型の CICS トランスレーターの使用

CICS のもとで COBOL プログラムを実行するために、分離型の CICS トランスレーターを使用して CICS コマンドを COBOL ステートメントに変換してから、プログラムをコンパイルして実行可能モジュールを作成することができます。しかし、分離型のトランスレーターよりも、Enterprise COBOL コンパイラーの組み込みの CICS トランスレーターを使用することをお勧めします。

CICS ステートメントを個別に変換するには、COBOL3 トランスレーター・オプションを使用します。このオプションを使用すると、次の行が挿入されます。

```
CBL RENT,NODYNAM,LIB
```

CICS トランスレーター・オプション NOCBLCARD を使用すれば、CBL ステートメントの挿入を抑止することができます。

CICS は、COBOL 85 標準で導入された以下の言語機能をサポートする ANSI85 トランスレーター・オプション を提供します。

- リテラルの間にあるブランク行
- 任意の文字を含むシーケンス番号
- すべての COBOL ワードでサポートされる小文字

- REPLACE ステートメント
- バッチ・コンパイル
- ネストされたプログラム
- 参照変更
- GLOBAL 変数
- コンマ、セミコロン、およびスペースの交換可能性
- シンボリック文字定義

分離型の CICS トランスレーターを使用したときは、プログラムのコンパイル時に以下のコンパイラー・オプションを使用してください。

必要なコンパイラー・オプション	条件
RENT	
NODYNAM	プログラムが CICS トランスレーターによって変換される。
LIB	プログラムに COPY または BASIS ステートメントが含まれている。

さらに、CICS のもとでサポートされない言語エレメントに対してコンパイラーでフラグが付けられるように、コンパイラー・オプション WORD(CICS) を使用することをお勧めします。

TRUNC コンパイラー・オプションに関する以下の推奨は、2 進数データ項目の予期値に基づいています。

推奨コンパイラー・オプション	条件
TRUNC(OPT)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 文節に準拠している。
TRUNC(BIN)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 文節に準拠しているわけではない。

たとえば、分離型の CICS トランスレーターを使用する場合、データ項目を、8 桁より大きな値を受け取る可能性のある PIC S9(8) BINARY として定義するときは、TRUNC(BIN) コンパイラー・オプションを使用するか、項目を USAGE COMP-5 に変更するか、PICTURE 文節を変更してください。

また、効果のない以下のオプションの使用を避けることもできます。

ADV
FASTSRT
OUTDD

コンパイラーの入力データ・セットは、変換の結果として受け取ったデータ・セットであり、デフォルトでは SYSPUNCH です。

関連概念
413 ページの『組み込みの CICS トランスレーター』

関連タスク
411 ページの『CICS オプションを使用したコンパイル』

CICS 予約語テーブル

COBOL は、CICS アプリケーション・プログラム用に代替予約語テーブル (IGYCCICS) を用意しています。コンパイラー・オプション WORD(CICS) が指定されると、コンパイラーは、CICS のもとでサポートされない COBOL 語にフラグを立て、エラー・メッセージを出します。

IBM 提供のデフォルトの予約語テーブルにより制約を受ける COBOL ワード以外に、IBM 提供の CICS 予約語テーブルも次の COBOL ワードに制約を与えます。

CLOSE	I-O-CONTROL	REWRITE
DELETE	MERGE	**SD**
FD	OPEN	**SORT**
FILE	READ	START
FILE-CONTROL	RERUN	WRITE
INPUT-OUTPUT		

CICS のもとで SORT ステートメントを使用したい場合 (COBOL は CICS のもとで SORT ステートメントのインターフェースをサポートします) は、使用する前に CICS 予約語テーブルを変更しなければなりません。上記のワードのうち、**太字**で示されたものは、制限されたものとしてマークされたワードのリストから削除してください。これらのワードは、SORT 機能で必要になります。

関連タスク
411 ページの『CICS オプションを使用したコンパイル』

関連参照
357 ページの『WORD』

CICS HANDLE を使用したエラー処理

CBLPSHPOP ランタイム・オプションの設定は、プログラムが COBOL サブプログラムを呼び出すときの HANDLE 指定の状態に影響を与えます。

CBLPSHPOP が ON の場合は、COBOL サブプログラム (ネストされたプログラムではないもの) が呼び出されると、以下のことが起こります。

- プログラム初期化の一部として、実行時に、呼び出しプログラムの HANDLE 指定が延期されます (EXEC CICS PUSH HANDLE を使用して)。
- 呼び出されたプログラムが独自の HANDLE コマンドを出すまで、HANDLE のデフォルトのアクションが適用されます。
- プログラム終了の一部として、実行時に、呼び出しプログラムの HANDLE 指定が回復されます (EXEC CICS POP HANDLE を使用して)。

CBLPSHPOP が OFF であると、実行時に、COBOL サブプログラムへの呼び出しで CICS PUSH または POP は実行されません。

呼び出される COBOL サブプログラムのいずれかが以下の CICS コマンドを 1 つまたは複数使用している場合は、CBLPSHPOP(ON) を指定して実行してください。

```
CICS HANDLE CONDITION
CICS HANDLE AID
CICS HANDLE ABEND
CICS IGNORE CONDITION
CICS PUSH HANDLE
CICS POP HANDLE
```

CICS HANDLE CONDITION または HANDLE AID コマンドを使用する場合は、CICS HANDLE コマンドで指定された LABEL が、CICS HANDLE ラベルへの分岐を引き起こす CICS コマンドと同じ PROCEDURE DIVISION に入っていなければなりません。LABEL オプションを指定した CICS HANDLE コマンドでは、COBOL CALL ステートメントを使用して呼び出された別のプログラムによって引き起こされた条件、援助機能、および異常終了を処理することはできません。LABEL オプションを指定した CICS HANDLE コマンドを使用してプログラム間分岐を実行しようとすると、トランザクションが異常終了します。

ネストされたプログラム内で条件、援助機能、または異常終了が発生する場合、その条件、援助機能、または異常終了の LABEL は、同じネストされたプログラム内に存在しなければなりません。さもないと、予測不能な結果が生じます。

『例: CICS HANDLE を使用したエラー処理』

例: CICS HANDLE を使用したエラー処理

次のサンプル・コードは、COBOL プログラムにおける CICS HANDLE の使用を図示しています。プログラム A には CICS HANDLE CONDITION コマンドがありますが、プログラム B には CICS HANDLE コマンドはありません。プログラム A はプログラム B を呼び出します。プログラム A はネストされたプログラム A1 も呼び出します。

以下は、3 つのシナリオで、条件がどのように処理されるかを示します。

(1) CBLPSHPOP(ON): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に CICS PUSH HANDLE が実行されたため、HANDLE 指定が延期されたことが原因です)。この条件はトランザクション異常終了となります。

(2) CBLPSHPOP(OFF): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に、LABEL オプションを指定した CICS HANDLE コマンドを使用して、プログラム間分岐の試みを診断します)。この条件はトランザクション異常終了となります。

(3) ネストされたプログラム A1 の CICS READ コマンドによって条件が引き起こされた場合、制御のフローはラベル ERR-1 に移り、予測できない結果が生じます。

```
************************************************************
*  Program A                                               *
************************************************************
```

```
 ID DIVISION.
 PROGRAM-ID. A.
 . . .
 PROCEDURE DIVISION.
     EXEC CICS HANDLE CONDITION
               ERROR(ERR-1)
               END-EXEC.
     CALL 'B' USING DFHEIBLK DFHCOMMAREA.
     CALL 'A1'.
     . . .
 THE-END.
     EXEC CICS RETURN END-EXEC.
 ERR-1.
 . . .
* Nested program A1.
 ID DIVISION.
 PROGRAM-ID. A1.
 PROCEDURE DIVISION.
     EXEC CICS READ             (3)
               FILE('LEDGER')
               INTO(RECORD)
               RIDFLD(ACCTNO)
               END-EXEC.
 END PROGRAM A1.
 END PROGRAM A.
*
*************************************************************
*  Program B                                                *
*************************************************************
 ID DIVISION.
 PROGRAM-ID. B.
 . . .
 PROCEDURE DIVISION.
     EXEC CICS READ             (1)  (2)
               FILE('MASTER')
               INTO(RECORD)
               RIDFLD(ACCTNO)
               END-EXEC.
     . . .
 END PROGRAM B.
```

第 21 章 DB2 環境用のプログラミング

一般に、COBOL プログラムのコーディングは、プログラムを DB2 データベースにアクセスさせるかどうかに関係なく同じになります。しかし、DB2 データの検索、更新、挿入、および削除を行い、さらにその他の DB2 サービスを使用するためには、SQL ステートメントを使用しなければなりません。

DB2 と通信するためには、次のことを行わなければなりません。

- EXEC SQL および END-EXEC ステートメントで区切って、必要なすべての SQL ステートメントをコーディングします。
- DB2 プリコンパイラーを使用するか、または DB2 (OS/390 用) バージョン 7 またはそれ以降を使用している場合は、SQL コンパイラー・オプションを使用してコンパイルします。

制約事項:

- 国別データ項目を組み込み SQL ステートメントのホスト変数として使用することはできません。
- 英数字または DBCS のホスト変数を、DB2 によるその他の文字表現に変更する場合、CODEPAGE コンパイラー・オプションの設定は効果がありません。変換が必要な場合は、SQL DECLARE VARIABLE ステートメントで明示的な CCSID を指定するなど、既存の DB2 機能を使用して、特定のホスト変数に対応するコード・ページを指定します。

関連概念

422 ページの『DB2 coprocessor』

関連タスク

『SQL ステートメントのコーディング』
421 ページの『SQL オプションを使用したコンパイル』
COBOL アプリケーションでの SQL ステートメントのコーディング (*IBM DB2 アプリケーション・プログラミングおよび SQL ガイド*)

関連参照

IBM DB2 SQL 解説書

SQL ステートメントのコーディング

SQL ステートメントは、EXEC SQL および END-EXEC ステートメントで区切らなければなりません。さらに、以下の特別のステップを行う必要があります。

- SQL 連絡域 (SQLCA) を WORKING-STORAGE SECTION に宣言する。
- SQL ステートメントで使用するすべてのホスト変数 WORKING-STORAGE または LINKAGE SECTION に宣言する。

© Copyright IBM Corp. 1991, 2001

DB2 coprocessor を用いた SQL INCLUDE の使用

SQL コンパイラー・オプションを使用する場合、SQL INCLUDE ステートメントは、ネイティブの COBOL COPY ステートメントと同じように扱われます。したがって、次の 2 つの行は同様に扱われます。

```
EXEC SQL INCLUDE name
COPY name
```

SQL INCLUDE ステートメントの *name* は、COPY *text-name* の場合と同じ規則に従い、REPLACING 文節を持たない COPY *text-name* と同様に処理されます。

SQL INCLUDE ステートメントのライブラリー探索順序は、ライブラリー名を指定しない COBOL COPY ステートメントを解決するためにコンパイラーが使用するのと同じ SYSLIB 連結です。

2 進数項目の使用

SQL ステートメントに指定する 2 進数データ項目には、以下のいずれかの技法を使用してください。

- それらを USAGE COMP-5 として宣言する。
- USAGE BINARY、COMP、または COMP-4 が指定される場合は、TRUNC(BIN) オプションを使用する (これは、個々のデータ項目に USAGE COMP-5 を使用する場合よりも、パフォーマンスに大きな影響を与える可能性があります)。

USAGE BINARY、COMP、または COMP-4 項目を指定したときに、オプション TRUNC(OPT) または TRUNC(STD) (またはその両方) が有効であると、コンパイラーはその項目を受け入れますが、10 進数切り捨て規則のため、そのデータは無効なことがあります。切り捨てがデータの妥当性に影響を与えないようにしなければなりません。

SQL ステートメントの成否の判断

DB2 は SQL ステートメントの実行を終了すると、DB2 は SQLCA に入れて戻りコードを送り (例外が 1 つあります)、操作が成功したか失敗したかを示します。プログラムは戻りコードをテストし、必要なアクションを取らなければなりません。

例外が起こるのは、プログラムが DSN のもとで、TSO バッチ・モード・モジュール IKJEFT01 の代替入り口点の 1 つ (IKJEFT1A または IKJEFT1B) から実行された場合です。この場合、戻りコードはレジスター 15 に入れて渡されます。

SQL ステートメントの実行後、RETURN-CODE 特殊レジスターの内容が有効でなくなる場合があります。このため、SQL ステートメントが正常に実行され、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了することがあります。終了時に意味のある戻りコードが渡されるようにするには、プログラムを終了する前に、RETURN-CODE 特殊レジスターを設定します。

関連概念

46 ページの『数値データの形式』

SQL オプションを使用したコンパイル

z/OS のもとで SQL コンパイラー・オプションを使用すると、DB2 coprocessor が使用可能になり、DB2 サブオプションを指定できるようになります。SQL コンパイラー・オプションが組み込み SQL ステートメントに作用するのは、コンパイラーが DB2 (OS/390 用) バージョン 7 またはそれ以降へのアクセス権を持っている場合だけです。

SQL オプションは、コンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS または CBL ステートメント、あるいはインストール・デフォルト) のいずれにでも指定することができます。SQL オプションが COBOL インストール・デフォルトであるときは、DB2 サブオプションを指定することはできません。ただし、DB2 プロダクトのインストール・デフォルトをカスタマイズすることによって、デフォルトの DB2 サブオプションを指定することができます。

SQL コンパイラー・オプションで指定された DB2 サブオプション・ストリングは、DB2 coprocessor で使用可能になります。ストリングの内容を見るのは、DB2 coprocessor だけです。

DB2 coprocessor を使用するときは、以下のオプションを使用してコンパイルする必要があります。

コンパイラー・オプション	説明
SQL	一緒に NOLIB を使用すると、LIB が強制的にオンにされます。
LIB	SQL と一緒に指定する必要があります。
SIZE(*xxx*)	*xxx* は、DB2 coprocessor サービスのための十分なストレージをユーザー領域に残すようなサイズ値 (MAX ではない) です。

標準の JCL プロシージャー・ステートメントを使用して、プログラムを DB2 coprocessor でコンパイルすることができます。上記のコンパイラー・オプションを指定する以外に、以下の項目を JCL に指定する必要があります。

- 生成済みデータベース要求モジュール (DBRM) の位置を指定した DBRMLIB DD ステートメント。
- DB2 coprocessor サービスが含まれたデータ・セットを追加する COBOL ステップ用の STEPLIB オーバーライド (これらのサービスが LNKLST に含まれている場合を除く)。通常、このデータ・セットは DSN710.SDSNLOAD ですが、インストール先で名前を変更している場合もあります。

たとえば、以下の行が JCL に含まれる場合があります。

```
//DBRMLIB  DD  DSN=PAYROLL.MONTHLY.DBRMLIB.DATA(MASTER),DISP=SHR
//STEPLIB  DD  DSN=DSN710.SDSNLOAD,DISP=SHR
```

バッチでのコンパイル

SQL オプションを使用して、一連の COBOL プログラム (バッチ・コンパイル・シーケンス) が含まれているソース・ファイルをコンパイルするときは、バッチ・シーケンスの最初のプログラムに対してオプションが有効でなければなりません。SQL

オプションが CBL または PROCESS カードに指定された場合、CBL または PROCESS カードは、バッチ・コンパイル・シーケンスの最初のプログラムより前に置かれなければなりません。

DB2 サブオプションの分離

複数の SQL オプション指定は連結されるため、DB2 サブオプション (単一の CBL ステートメントに収まらないもの) を複数の CBL ステートメントに分けることができます。

サブオプション・ストリングに組み込まれる DB2 サブオプションは累積されます。コンパイラーは、複数のソースからのこれらのサブオプションを、指定された順に連結します。たとえば、ソース・ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//  PARM.COBOL='SQL("DATABASE xxxx")'
//COBOL.SYSIN DD *
    CBL SQL("PACKAGE USER xxxx")
    CBL SQL("USING xxxx")
    IDENTIFICATION DIVISION.
    PROGRAM-ID. DRIVER1.
```

コンパイル時、コンパイラーは次のサブオプション・ストリングを DB2 coprocessor に渡します。

```
"DATABASE xxxx PACKAGE USER xxxx USING xxxx"
```

連結ストリングは 1 つのスペースで区切られます。コンパイラーが同じ DB2 サブオプションの複数インスタンスを検出した場合は、連結ストリングの中で最後に指定されたサブオプションが有効になります。コンパイラーは、連結 DB2 サブオプション・ストリングの長さを 4 KB に限定しています。

関連概念
『DB2 coprocessor』

関連参照
347 ページの『SQL』
IBM DB2 コマンド解説書

DB2 coprocessor

DB2 coprocessor (DB2 では *SQL ステートメント coprocessor* と呼ばれる) を使用すると、別個のプリコンパイル・ステップを使用しなくても、組み込み SQL ステートメントが含まれたソース・プログラムをコンパイラーが処理するようになります。コンパイラーは、ソース・プログラム内の重要なポイントで SQL ステートメントを検出すると、DB2 coprocessor とインターフェースします。この coprocessor が、SQL ステートメントに基づいて適切なアクションを取り、それらのために生成する固有 COBOL ステートメントをコンパイラーに指示します。

別個のプリコンパイル・ステップの使用も引き続きサポートされますが、coprocessor の使用をお勧めします。coprocessor を使用すると、リストには、生成された

COBOL ソースではなく、SQL ステートメントが出力されるので、デバッグ・ツールを使用した対話式デバッグが強化されます。ただし、このためには、DB2 for OS/390 バージョン 7 以上が必要です。

DB2 coprocessor を使用してコンパイルすると、オブジェクト・モジュールやリストのような通常の COBOL コンパイラー出力と一緒に、DB2 データベース要求モジュール (DBRM) が生成されます。DBRM は、COBOL コンパイル・ステップに関して JCL 内の `DBRMLIBB DD` ステートメントで指定されたデータ・セットに書き出します。DBRM データ・セットには、DB2 バインド・プロセスへの入力として、プログラム内の SQL ステートメントおよびホスト変数に関する情報が入ります。

COBOL コンパイラー・リストには、DB2 coprocessor が生成するエラー診断 (SQL ステートメントの構文エラーなど) が含まれます。

プリコンパイル・ステップを使用するときに適用される COBOL 言語の使用に関する制約事項は、DB2 coprocessor を使用するときには適用されません。

- ネストされたプログラムのいずれでも SQL ステートメントを使用することができます (プリコンパイラーの場合、SQL ステートメントの使用は一番外側のプログラムに制限されます)。
- コピーブックで SQL ステートメントを使用することができます。
- `REPLACE` ステートメントは SQL ステートメントに影響を及ぼします。

関連タスク

421 ページの『SQL オプションを使用したコンパイル』

第 22 章 IMS のもとでの COBOL プログラムの実行

COBOL プログラムのコーディングは、IMS のもとで実行する場合とほとんど同じですが、以下の推奨事項と制約事項に留意してください。

COBOL では、IMS メッセージ処理プログラム (MPP) は、IMS 以外の入力ステートメントまたは出力ステートメント (READ、WRITE、REWRITE、OPEN、および CLOSE など) を使用しません。

Enterprise COBOL では、次のインターフェースを使用して IMS 機能を呼び出すことができます。

- CBLTDLI 呼び出し
- 言語環境プログラム呼び出し可能サービス CEETDLI

CEETDLI への呼び出しは、CBLTDLI への呼び出しと同様にコーディングされます。CEETDLI は基本的には CBLTDLI と同じことを行います。

IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク

IMS 環境で最適なパフォーマンスを得るためには、RENT コンパイラー・オプションを使用してください。このオプションを使用すると、COBOL で再入可能コードが生成されます。その後、アプリケーション・プログラムを*プリロード・モード* (プログラムが常にストレージにある) または*非プリロード・モード*のいずれでも実行することができます。別のオプションで再コンパイルする必要はありません。

IMS を使用すると、COBOL プログラムをプリロードできます。このプリロードによってパフォーマンスが向上します。これは、プログラムがすでにストレージにあると (必要が生じるたびにライブラリーから取り出すよりも) プログラムに対する後続の要求をより速く処理できるためです。

RENT コンパイラー・オプションを使用して、プリロードして実行するプログラム、またはプリロードおよび非プリロードの両方で実行するプログラムをコンパイルする必要があります。さらに、COBOL プログラムを含むロード・モジュールをプリロードするときは、そのロード・モジュール内のすべての COBOL プログラムが RENT オプションを指定してコンパイルされていなければなりません。

IMS プログラムについては、RENT コンパイラー・オプションを使用することをお勧めします。

RENT オプションを指定してコンパイルされたプログラムは、z/OS リンク・パック域に入れることができます。そこでは、IMS 従属領域間でプログラムを共用することができます。

16MB 境界より上で実行するには、RENT または NORENT RMODE(ANY) を指定してアプリケーション・プログラムをコンパイルする必要があります。

© Copyright IBM Corp. 1991, 2001

IMS アプリケーション・プログラムのデータは 16MB 境界より上に常駐させることができ、IMS サービスを使用するプログラムでは DATA(31) RENT または RMODE(ANY) NORENT を指定することができます。

IMS のもとで COBOL プログラムを正しく実行するために推奨されるリンク・エディット属性は、次のとおりです。

- RENT コンパイラー・オプションを指定してコンパイルされた COBOL プログラムのみを含む RENT ロード・モジュールとしてリンクする。
- COBOL RENT プログラムとその他のプログラムが混在するロード・モジュールをリンクするために、他のプログラムについて推奨されるリンク・エディット属性を使用する。

関連概念

38 ページの『ストレージとそのアドレス可能度』

関連タスク

IMS のもとでの協調条件処理 (*言語環境プログラム プログラミング・ガイド*)

関連参照

321 ページの『DATA』
343 ページの『RENT』

IMS についての考慮事項 (*Enterprise COBOL コンパイラーおよび実行時プログラム移行ガイド*)

第 23 章 UNIX のもとでの COBOL プログラムの実行

COBOL プログラムを UNIX 環境で実行するには、Enterprise COBOL コンパイラーまたは COBOL (OS/390 および VM 版) コンパイラーを使用してコンパイルする必要があります。これらのプログラムは再入可能でなければなりません。したがって、コンパイラーおよびリンカー・オプション RENT を使用してください。HFS から実行する予定であれば、リンカー・オプション AMODE 31 を使用してください。UNIX アプリケーション内から呼び出す AMODE 24 プログラムは、MVS PDS または PDSE に常駐させる必要があります。

UNIX のもとでの実行には、以下の制約事項が適用されます。

- SORT および MERGE ステートメントはサポートされません。
- 事前初期設定用として旧バージョンの COBOL インターフェース (ランタイム・オプション RTEREUS と関数 IGZERRE および ILBOSTP0) を使用した場合、UNIX 環境で再使用可能な環境を確立することはできません。
- NOTHREAD オプションを指定してコンパイルした COBOL プログラムを複数のスレッドで実行することはできません。2 番目のスレッドで COBOL アプリケーションを開始すると、COBOL 実行時に、ソフトウェア条件を受け取ります。NOTHREAD を指定してコンパイルした COBOL プログラムは、初期プロセス・スレッド (IPT) か、C または PL/I のルーチンから作成する 1 つの非 IPT で実行することができます (アプリケーション内のすべての COBOL プログラムが THREAD オプションを指定してコンパイルされている場合は、複数のスレッドで COBOL プログラムを実行できます)。
- デバッグ・ツールによる UNIX プログラムのデバッグは、IBM VisualAge COBOL からのリモート・デバッグ・セッションを使用してのみ行うことができます。

関連タスク

299 ページの『第 16 章 UNIX のもとでのコンパイル』
『UNIX 環境での実行』
429 ページの『環境変数の設定およびアクセス』
430 ページの『UNIX/POSIX API の呼び出し』
432 ページの『メインプログラム・パラメーターへのアクセス』
言語環境プログラム プログラミング・ガイド

関連参照

343 ページの『RENT』

UNIX 環境での実行

UNIX COBOL プログラムは、以下の UNIX 実行環境のいずれかで実行することができます。

- 次のいずれかの UNIX シェルから。
  - OMVS シェル (OMVS)
  - ISPF シェル (ISHELL)

© Copyright IBM Corp. 1991, 2001

シェル・プロンプトでプログラム名を入力してください。

プログラムは、現行ディレクトリーまたは探索パスに入っている必要があります。UNIX プログラムを実行するときは、プログラムを開始する前に環境変数 _CEE_RUNOPTS を設定することによってのみ、ランタイム・オプションを指定することができます。

カタログ式 MVS データ・セットに常駐する MVS 実行可能プログラムは、tso ユーティリティーを使用して、シェルから実行することができます。たとえば、次のようになります。

```
tso "call 'my.loadlib(myprog)'"
```

ISPF シェルは、stdout および stderr のみを、端末ではなく HFS ファイルに送信します。

- 次のいずれかのシェルの外側から。
  - TSO/E
  - バッチ

  HFS ファイルに常駐する UNIX COBOL プログラムを TSO/E 作動可能プロンプトから呼び出すには、BPXBATCH ユーティリティーを使用するか、REXX exec で spawn() syscall を使用してください。

  HFS ファイルに常駐する UNIX COBOL プログラムを JCL EXEC ステートメントを使用して呼び出すには、BPXBATCH ユーティリティーを使用してください。

関連タスク

429 ページの『環境変数の設定およびアクセス』
430 ページの『UNIX/POSIX API の呼び出し』
432 ページの『メインプログラム・パラメーターへのアクセス』
UNIX シェルを使用した POSIX 可能プログラムの実行 (*言語環境プログラム プログラミング・ガイド*)
UNIX シェルの外側からの POSIX 可能プログラムの実行 (*言語環境プログラム プログラミング・ガイド*)
150 ページの『QSAM ファイルの定義および割り振り』
197 ページの『行順次ファイルの定義および割り振り』
189 ページの『VSAM ファイルの割り振り』
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

関連参照

349 ページの『TEST』
BPXBATCH ユーティリティー (*UNIX システム・サービス ユーザーズ・ガイド*)
言語環境プログラム プログラミング・リファレンス

環境変数の設定およびアクセス

UNIX COBOL プログラムの環境変数は、次のいずれかの方法で設定することができます。

- シェルからコマンド export および set を使用して
- プログラムから

プログラムの実行を開始する前にシェルから環境変数を設定およびリセットするのが一般的な手順ですが、実行中にプログラムから環境変数を設定、リセット、およびアクセスすることができます。

BPXBATCH を使用してプログラムを実行している場合は、STDENV DD ステートメントを使用して環境変数を設定することができます。

実行に影響を与える環境変数の設定

シェルから環境変数を設定するには、export または set コマンドを使用します。たとえば、環境変数 MYFILE を設定するには、次のように指定します。

```
export MYFILE=/usr/mystuff/notes.txt
```

プログラムから環境変数を設定するには、POSIX 関数 setenv() および putenv() を呼び出してください。

COBOL プログラムに重要な環境変数

_CEE_RUNOPTS
ランタイム・オプション。

LIBPATH
ダイナミック・リンク・ライブラリーのディレクトリー・パス。

_IGZ_SYSOUT
DISPLAY 出力の宛先の stdout または stderr。使用できる値はこの 2 つだけです。

STEPLIB
LNKLST に含まれていないプログラムの位置。

環境変数のリセット

環境変数は、シェルまたはプログラムからリセットすることができます。環境変数を元の設定どおりにリセットするには、UNIX シェル・コマンド unset を使用してください。COBOL プログラムから環境変数をリセットするには、setenv() 関数を呼び出してください。

環境変数へのアクセス

すべての環境変数の値を調べるには、export コマンドをパラメーターを付けずに指定することができます。

COBOL プログラムから環境変数の値にアクセスするには、getenv() 関数を呼び出してください。

430 ページの『例: 環境変数へのアクセス』

関連タスク

427 ページの『UNIX 環境での実行』
『UNIX/POSIX API の呼び出し』
432 ページの『メインプログラム・パラメーターへのアクセス』
34 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

関連参照

言語環境プログラム プログラミング・リファレンス
MVS JCL 解説書

例: 環境変数へのアクセス

次の例は、標準の POSIX 関数 getenv() および putenv() を呼び出すことによって、COBOL プログラムから環境変数にアクセスする方法と環境変数を設定する方法を示しています。

これらは C 関数なので、BY VALUE によって引き数を渡さなければなりません。文字ストリングは、ヌル終了ストリングを指し示す BY VALUE ポインターとして渡さなければなりません。

```
 CBL pgmname(longmixed)
 Identification division.
 Program-id. "envdemo".
 Data division.
 Working-storage section.
 01 P pointer.
 01 PATH pic x(5) value Z"PATH".
 01 var-ptr pointer.
 01 var-len pic 9(4) binary.
 01 putenv-arg pic x(14) value Z"MYVAR=ABCDEFG".
 01 rc pic 9(9) binary.
 Linkage section.
 01 var pic x(5000).
 Procedure division.
* Retrieve and display the PATH environment variable
     Set P to address of PATH
     Call "getenv" using by value P returning var-ptr
     If var-ptr = null then
         Display "PATH not set"
     Else
         Set address of var to var-ptr
         Move 0 to var-len
         Inspect var tallying var-len
           for characters before initial X"00"
         Display "PATH = " var(1:var-len)
     End-if
* Set environment variable MYVAR to ABCDEFG
     Set P to address of putenv-arg
     Call "putenv" using by value P returning rc
     If rc not = 0 then
         Display "putenv failed"
         Stop run
     End-if
     Goback.
```

UNIX/POSIX API の呼び出し

標準の UNIX/POSIX 関数は、UNIX プログラムおよび従来の MVS COBOL プログラムから呼び出すことができます。これらの関数は、言語環境プログラムの一部です。

これらは C 関数なので、BY VALUE によって引き数を渡さなければなりません。文字ストリングは、ヌル終了ストリングを指し示す BY VALUE ポインターとして渡さなければなりません。これらの関数を呼び出すプログラムをコンパイルするときは、コンパイラー・オプション NODYNAM および PGMNAME(LONGMIXED) を使用する必要があります。

fork、exec および spawn

fork、exec、および spawn 関数は、COBOL プログラムから、または COBOL プログラムと同じプロセスに含まれている非 COBOL プログラムから呼び出すことができます。ただし、以下の制約事項に留意しなければなりません。

- fork されたプロセスから、fork を出したときにオープンされた COBOL 順次ファイル、索引付きファイル、または相対ファイルにアクセスすることはできません。 このようなアクセス (CLOSE、READ、WRITE、REWRITE、DELETE、または START) を試みると、ファイル状況コード 92 を受け取ります。fork を出したときにオープンされた行順次ファイルにはアクセスできます。
- 以下のいずれかが真であるようなプロセスでは、fork 関数を使用できません。
  - COBOL の SORT または MERGE が実行されている。
  - 宣言が実行されている。
  - プロセスに、複数の言語環境プログラム・エンクレーブ (COBOL 実行単位) が含まれている。
  - プロセスで、いずれかの COBOL 再使用可能環境インターフェースが使用されている。
  - プロセスで、OS/VS COBOL または VS COBOL II プログラムを実行している。
- 1 つの例外を除いて、MVS DD 割り振りは親プロセスから子プロセスへ継承されません。例外はローカル spawn で、この場合は、親プロセスと同じアドレス・スペース内に子プロセスが作成されます。ローカル spawn は、spawn 関数を呼び出す前に、環境変数 _BPX_ SHAREAS=YES を設定することによって要求します。

exec および spawn 関数は、新規プロセス内で新規の言語環境プログラム・エンクレーブを開始します。したがって、exec または spawn のターゲット・プログラムはメインプログラムであり、プロセス内のすべての COBOL プログラムは、すべてのファイルをクローズした初期状態で開始します。

サンプル

ある種の POSIX 関数を呼び出す製品には、サンプル・コードが用意されています。サンプル・ソース・コードは、SIGYSAMP データ・セットに入っています。

目的	サンプル	使用される関数
一部のファイルおよびディレクトリー・ルーチンの使用法を示す	IGYTFL1	getcwd mkdir rmdir access
iconv ルーチンを使用してデータを変換する方法を示す	IGYTCNV	iconv_open iconv iconv_close

exec ルーチンを使用して、他のプロセス関連ルーチンと一緒に新規プログラムを実行する方法を示す	IGYTEXC IGYTEXC1	fork getpid getppid execl perror wait
errno 値の入手方法を示す	IGYTERNO IGYTGETE	perror fopen
プロセス間通信メッセージ・ルーチンの使用法を示す	IGYTMSQ IGYTMSQ2	ftok msgget msgsndd perror fopen fclose msgrcv msgctl perror

関連タスク

427 ページの『UNIX 環境での実行』
429 ページの『環境変数の設定およびアクセス』
『メインプログラム・パラメーターへのアクセス』
言語環境プログラム プログラミング・ガイド

関連参照

C/C++ ランタイム・ライブラリー・リファレンス
UNIX システム・サービス プログラミング: アセンブラー呼び出し可能サービス解説書

メインプログラム・パラメーターへのアクセス

UNIX シェル・コマンド行から、あるいは exec または spawn 関数を使用して COBOL プログラムを呼び出す場合、パラメーター・リストは参照によって渡される 3 つのパラメーターから構成されます。

引き数カウント

2 番目および 3 番目のパラメーターとして渡されるそれぞれの配列のエレメントの数が含まれている 2 進数のフルワード整数。

引き数の長さのリスト

ポインターの配列。配列内の *n* 番目の記入項目は、引き数リスト内の *n* 番目の記入項目の長さを含むフルワードの 2 進整数のアドレスです。

引き数リスト

ポインターの配列。配列内の *n* 番目の記入項目は、spawn 関数、exec 関数、またはコマンド呼び出しで引き数として渡される *n* 番目の文字ストリングのアドレスです。各文字ストリングは、ヌル終了文字ストリングです。

この配列は空になることはありません。最初の引き数は、開始するプロセスと関連付けられたファイルの名前を表す文字ストリングです。

標準の COBOL コーディングを使用してこれらのパラメーターにアクセスすることができます。

『例: メインプログラム・パラメーターへのアクセス』

関連タスク

427 ページの『UNIX 環境での実行』
429 ページの『環境変数の設定およびアクセス』
430 ページの『UNIX/POSIX API の呼び出し』

例: メインプログラム・パラメーターへのアクセス

次の例は、参照によって渡される 3 つのパラメーターを示しています。

```
 Identification division.
 Program-id. "EXECED".
*****************************************************************
* This sample program displays arguments received via exec()   *
* function of UNIX System Services                             *
*****************************************************************
 Data division.
 Working-storage section.
 01 curr-arg-count pic 9(9) binary value zero.
 Linkage section.
 01 arg-count pic 9(9) binary.                        (1)
 01 arg-length-list.                                  (2)
     05 arg-length-addr pointer occurs 1 to 99999
           depending on curr-arg-count.
 01 arg-list.                                         (3)
     05 arg-addr pointer occurs 1 to 99999
           depending on curr-arg-count.
 01 arg-length pic 9(9) binary.
 01 arg pic X(65536).
 Procedure division using arg-count arg-length-list arg-list.
*****************************************************************
* Display number of arguments received                          *
*****************************************************************
     Display "Number of arguments received: " arg-count
*****************************************************************
* Display each argument passed to this program                  *
*****************************************************************
     Perform arg-count times
       Add 1 to curr-arg-count
* *******************************************************
* * Set address of arg-length to address of current     *
* * argument length and display                         *
* *******************************************************
       Set Address of arg-length
         to arg-length-addr(curr-arg-count)
       Display
         "Length of Arg " curr-arg-count " = " arg-length
* *******************************************************
* * Set address of arg to address of current argument   *
* * and display                                         *
* *******************************************************
       Set Address of arg to arg-addr(curr-arg-count)
       Display "Arg " curr-arg-count " = " arg (1:arg-length)
     End-Perform
     Display "Display of arguments complete."
     Goback.
```

(1) このカウントには、2 番目および 3 番目のパラメーターとして渡されるそれぞれの配列のエレメントの数が含まれています。

(2) この配列には、引き数リスト内の *n* 番目の記入項目の長さへのポインターが含まれています。

(3) この配列には、spawn 関数、exec 関数、またはコマンド呼び出しで引き数として渡される *n* 番目の文字ストリングへのポインターが含まれています。

第 4 部 複雑なアプリケーションの構造

第 24 章 サブプログラムの使用 437
メインプログラム、サブプログラム、および呼び出し 438
メインプログラムまたはサブプログラムの終了と再入 439
別のプログラムへの制御権の移動. 440
静的呼び出しの作成 441
動的呼び出しの作成 442
サブプログラムの取り消し 443
動的呼び出しの使用時期. 443
静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項 444
静的呼び出しと動的呼び出しの両方の作成. . . 445
例: 静的および動的 CALL ステートメント . . 445
ネストされた COBOL プログラムの呼び出し 447
ネストされたプログラム. 448
例: ネストされたプログラムの構造 449
名前の有効範囲. 450
再帰呼び出しの作成 451
オブジェクト指向プログラムとの間での呼び出し 451
プロシージャー・ポインターと関数ポインターの使用 452
どのタイプのポインターを使用するかの決定 . . 453
C 関数ポインターの呼び出し 453
代替入り口点への呼び出し 453
プログラムを再入可能にする 454

第 25 章 データの共用 455
データの受け渡し 455
呼び出しプログラムの中での引き数の記述. . . 456
呼び出されるプログラムの中でのパラメーターの記述 457
OMITTED 引き数に関するテスト. 457
LINKAGE SECTION のコーディング 458
引き数を受け渡すための PROCEDURE DIVISION のコーディング. 458
受け渡されるデータのグループ化. 459
ヌル終了ストリングの処理 459
チェーン・リストを使用するためのポインターの使用 460
例: チェーン・リストを処理するためのポインターの使用 461
戻りコード情報の受け渡し 463
RETURN-CODE 特殊レジスターの理解. . . . 463
PROCEDURE DIVISION RETURNING . . . の使用 463
CALL . . . RETURNING の指定 464
EXTERNAL 文節によるデータの共用 464
プログラム間でのファイルの共用 (外部ファイル) 464
例: 外部ファイルの使用 465

第 26 章 DLL または DLL アプリケーションの作成 469
ダイナミック・リンク・ライブラリー (DLL) . . . 469
DLL を作成するためのプログラムのコンパイル 470
DLL のリンク 471
特定の DLL のプリリンク 472
例: プロシージャー型 DLL アプリケーションのサンプル JCL 473
DLL での CALL ID の使用 474
HFS での DLL の探索順序. 475
DLL リンケージと動的呼び出しの併用 475
DLL でのプロシージャー・ポインターまたは関数ポインターの使用 476
非 DLL からの DLL の呼び出し. 477
例: 非 DLL からの DLL の呼び出し 478
C/C++ プログラムでの COBOL DLL の使用 . . . 479
OO COBOL アプリケーションでの DLL の使用 480

第 27 章 マルチスレッド化のための COBOL プログラムの準備 481
マルチスレッド化 482
マルチスレッド化サポートのための THREAD の選択 483
マルチスレッド化による制御権移動 484
スレッド化プログラムでの取り消しの使用. . . 484
プログラムの終了 484
COBOL 環境の事前初期設定 484
マルチスレッド化によるファイルの処理 485
ファイル定義ストレージ. 485
ファイル・アクセスの場合の推奨使用法 . . . 485
例: マルチスレッド化によるファイル入出力の使用パターン 486
マルチスレッド化による COBOL 制限の処理. . . 487

© Copyright IBM Corp. 1991, 2001

第 24 章 サブプログラムの使用

多くのアプリケーションは、一緒にリンクされた別々にコンパイルされた複数のプログラムから構成されます。*実行単位* (言語環境プログラムのエンクレーブ と同義の COBOL 用語) には 1 つまたは複数のオブジェクト・プログラムが含まれており、他の言語環境プログラムのメンバー言語で書かれたオブジェクト・プログラムが含まれることもあります。

言語環境プログラムには言語間サポートが用意されており、これを使用すると、Enterprise COBOL プログラムは、言語環境プログラムの要件に適合するプログラムとの間で呼び出しを行うことができます。

名前の接頭部についての注: IBM 製品で使用されている接頭部で始まるプログラム名は使用しないでください。名前が以下のいずれかで始まるプログラムの使用を試みると、CALL ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラー・ルーチンに解決されることがあります。

AFB	AFH	CBC	CEE	EDC
IBM	IFY	IGY	IGZ	ILB

マルチスレッド化: マルチスレッド化された COBOL プログラムは、マルチスレッド実行のために適切にコーディングされた C プログラムや言語環境プログラム対応アセンブラー・プログラムと結合して、1 つの実行単位にすることができます。

PL/I タスク: 複数の PL/I タスクを含むアプリケーションに COBOL プログラムを組み込む場合の指針を以下に示します。

- 複数の PL/I タスクで実行する COBOL プログラムはすべて、THREAD オプションを指定してコンパイルします。NOTHREAD オプションを指定してコンパイルされた COBOL プログラムが 1 つでもあると、どの COBOL プログラムも単一の PL/I タスクでしか実行できなくなります。
- THREAD オプションを指定してコンパイルされた COBOL プログラムは、1 つまたは複数の PL/I タスクから呼び出すことができます。しかし、PL/I プログラムからの COBOL プログラムの呼び出しでは、TASK オプションや EVENT オプションを指定することができません。PL/I タスク呼び出しでは、最初に PL/I のプログラムまたは関数を呼び出し、そこから COBOL プログラムを呼び出す必要があります。このような間接呼び出しが必要になるのは、TASK オプションまたは EVENT オプションを含む PL/I の CALL ステートメントのターゲットとして COBOL プログラムを指定することができないからです。
- COBOL プログラムから STOP RUN ステートメントを発行したり、PL/I プログラムから STOP ステートメントを発行すると、言語環境プログラムのエンクレーブ全体 (すべてのタスク実行を含む) が終了してしまいます。
- PL/I タスクを含む実行単位では、明示的な POSIX スレッド化 (pthread_create() の呼び出し) をコーディングしないでください。

© Copyright IBM Corp. 1991, 2001

関連概念
『メインプログラム、サブプログラム、および呼び出し』

関連タスク
439 ページの『メインプログラムまたはサブプログラムの終了と再入』
440 ページの『別のプログラムへの制御権の移動』
451 ページの『再帰呼び出しの作成』
451 ページの『オブジェクト指向プログラムとの間での呼び出し』
452 ページの『プロシージャー・ポインターと関数ポインターの使用』
454 ページの『プログラムを再入可能にする』
言語環境プログラム ILC アプリケーションの作成

関連参照
レジスター規則 (*言語環境プログラム プログラミング・ガイド*)

メインプログラム、サブプログラム、および呼び出し

COBOL プログラムが実行単位の最初のプログラムであれば、その COBOL プログラムはメインプログラム です。それ以外の場合は、そのプログラムも実行単位内の他のすべての COBOL プログラムも、サブプログラム です。特定のソース・コードステートメントまたはオプションが、COBOL プログラムをメインプログラムまたはサブプログラムとして識別することはありません。

COBOL プログラムがメインプログラムであるかサブプログラムであるかは、次の 2 つの理由により重要になることもあります。

- プログラム終了処理ステートメントの影響
- 戻った後に再入する際のそのプログラムの状態

PROCEDURE DIVISION では、あるプログラムから別のプログラム (通常 COBOL 用語ではサブプログラムと呼ばれる) を呼び出すことができ、この呼び出されたプログラム自体からも別のプログラムを呼び出せます。別のプログラムを呼び出すプログラムは*呼び出し* プログラムと呼ばれ、そのプログラムが呼び出すプログラムは*呼び出される* プログラムと呼ばれます。呼び出されるプログラムの処理が完了すると、そのプログラムは制御権を呼び出しプログラムに戻すか、実行単位を終了することができます。

呼び出される COBOL プログラムは、PROCEDURE DIVISION の先頭で実行を開始します。

関連タスク
439 ページの『メインプログラムまたはサブプログラムの終了と再入』
451 ページの『再帰呼び出しの作成』
440 ページの『別のプログラムへの制御権の移動』

関連参照
言語環境プログラム プログラミング・ガイド

メインプログラムまたはサブプログラムの終了と再入

メインプログラムまたはサブプログラムで 3 つの終了ステートメントのいずれを使用することもできますが、次の表に示すように、その影響がそれぞれ異なります。

終了 ステートメント	メインプログラム	サブプログラム
EXIT PROGRAM	処置は取られません。	実行単位を終了せずに呼び出しプログラムに戻ります。呼び出されるプログラムの中に次に実行可能なステートメントがない場合、暗黙の EXIT PROGRAM ステートメントが生成されます。 スレッド化された環境では、そのプログラムがスレッド内の最初の (最も古い) プログラムでない限り、スレッドは終了しません。
STOP RUN	呼び出しプログラムに戻ります。[1] (オペレーティング・システムに戻る可能性があり、ジョブは終了します。) STOP RUN は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、言語環境プログラムのエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。	メインプログラムを呼び出したプログラムに直接戻ります。[1] (オペレーティング・システムに戻る可能性があり、ジョブは終了します。) スレッド化された環境では、言語環境プログラムのエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。
GOBACK	効果は STOP RUN と同じであり、呼び出しプログラムに戻ります。[1] (オペレーティング・システムに戻る可能性があり、ジョブは終了します。) スレッド化された環境では、スレッドが終了します。[2]	呼び出しプログラムに戻ります。 スレッド化された環境では、そのプログラムがスレッド内の最初のプログラムであると、スレッドが終了します。[2]

1. メインプログラムが、言語環境プログラムのリンケージ規約に従っていない別の言語で書かれたプログラムによって呼び出された場合は、この呼び出しプログラムに戻ります。
2. スレッドがエンクレーブ内の最初の実行スレッドである場合は、エンクレーブが終了します。

サブプログラムは、EXIT PROGRAM または GOBACK で終了されると、*最後に使われた状態* で残されるのが普通です。それが実行単位の中で次に呼び出されると、

PERFORM ステートメントの戻り値が初期値にリセットされることを除けば、その内部値は終了したときのままになります (それに対して、メインプログラムは呼び出されるたびに初期化されます)。

プログラムが初期状態になるのは、次のような場合です。

- 動的に呼び出され、その後取り消されるサブプログラムは、次に呼び出されると、初期状態になります。
- INITIAL 属性を持つプログラムは、呼び出されるたびに初期状態になります。
- LOCAL-STORAGE SECTION で定義されたデータは、最外部の収容プログラムが呼び出されるたびに初期状態になります (ネストされたプログラムの場合、LOCAL-STORAGE は、ネストされたプログラムが呼び出されるたびに初期状態になります)。

関連概念

18 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』
スレッドの終了 (*言語環境プログラム プログラミング・ガイド*)

関連タスク

447 ページの『ネストされた COBOL プログラムの呼び出し』
451 ページの『再帰呼び出しの作成』

別のプログラムへの制御権の移動

次のようなさまざまな方法を使って、別のプログラムに制御権を移動することができます。

- 静的呼び出し
- 動的呼び出し
- ネストされたプログラムの呼び出し
- ダイナミック・リンク・ライブラリー (DLL) への呼び出し

すべての環境 (CICS を含む) において、Enterprise COBOL プログラム相互間での呼び出しだけでなく、Enterprise COBOL プログラムとその他の古いバージョンのコンパイラーでコンパイルされたプログラムとの間で、静的呼び出しおよび動的呼び出しを行うことができます。

OS/VS COBOL を Enterprise COBOL と併用する場合、非 CICS と CICS との間では、サポートに違いがあります。

非 CICS 環境の場合

Enterprise COBOL プログラムとその他の COBOL プログラムとの間で、静的呼び出しおよび動的呼び出しを行うことができます。

例外: UNIX 環境では、VS COBOL II プログラムや OS/VS COBOL プログラムを呼び出すことはできません。

CICS 環境の場合

CICS 環境では、OS/VS COBOL プログラムを呼び出すことはできません。OS/VS COBOL プログラムとその他の COBOL プログラムとの間で制御権を移動するには、EXEC CICS LINK を使用する必要があります。

ネストされたプログラムを呼び出すことによって、構造化プログラミング技法を使用して、アプリケーションを作成することができます。また、データ項目を不注意で変更しないようにするためには、PERFORM プロシージャーの代わりに、ネストされたプログラムを使用することができます。ネストされたプログラムの呼び出しには、CALL *literal* ステートメントまたは CALL *identifier* ステートメントを使用します。

ダイナミック・リンク・ライブラリー (DLL) の呼び出しは、COBOL の動的 CALL の代替方法として使用することができ、オブジェクト指向 COBOL アプリケーション、UNIX プログラム、および C/C++ との相互協調処理を行うアプリケーションでの使用に適しています。

z/OS では、2 つのロード・モジュールをリンクすると、論理的には単一のプログラムになり、そのプログラムには 1 次入り口点と代替入り口点があり、その入り口点にはそれぞれ独自の名前があります。サブプログラムを動的に呼び出すのに使われる個々の名前は、システムに認識されていなければなりません。このような個々の名前は、そのサブプログラムを含むロード・モジュールの NAME または ALIAS として、リンケージ・エディターまたはバインダーの制御ステートメントで指定する必要があります。

関連概念
448 ページの『ネストされたプログラム』

関連タスク
『静的呼び出しの作成』
442 ページの『動的呼び出しの作成』
445 ページの『静的呼び出しと動的呼び出しの両方の作成』
447 ページの『ネストされた COBOL プログラムの呼び出し』

静的呼び出しの作成

NODYNAM および NODLL コンパイラー・オプションを使用してコンパイルされたプログラム内で CALL *literal* ステートメントを使用すると、静的呼び出しが行われます。これらのオプションが使用されていると、CALL *literal* 形式の呼び出しはすべて、静的呼び出しとして処理されます。

静的 CALL ステートメントでは、COBOL プログラムと呼び出されるプログラムはすべて同一のロード・モジュールの一部になります。呼び出されるプログラムに制御権が移動すると、そのプログラムはすでにストレージにあるので、そのプログラムに対する分岐が行われます。それ以後 CALL ステートメントを実行すると、呼び出されるプログラムは最後に使われた状態で使えます (呼び出されるプログラムに INITIAL 属性がある場合を除く)。呼び出されるプログラムに INITIAL 属性が設定されていると、呼び出されるプログラムおよびそのプログラムに直接または間接に含まれているプログラムは、呼び出されるプログラムが実行単位内で呼び出されるたびに初期状態になります。

代替入り口点を指定すると、静的 CALL ステートメントはいずれかの代替入り口点を使って、呼び出されるサブプログラムに入ることができます。

445 ページの『例: 静的および動的 CALL ステートメント』

関連概念
444 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

関連タスク
451 ページの『オブジェクト指向プログラムとの間での呼び出し』
『動的呼び出しの作成』
445 ページの『静的呼び出しと動的呼び出しの両方の作成』

関連参照
326 ページの『DYNAM』
324 ページの『DLL』
CALL ステートメント (*Enterprise COBOL 言語解説書*)

動的呼び出しの作成

DYNAM および NODLL コンパイラー・オプションを使用してコンパイルされたプログラム内で CALL *literal* ステートメントを使用するか、または NODLL コンパイラー・オプションを使用してコンパイルされたプログラム内で CALL *identifier* ステートメントを使用すると、動的呼び出しが行われます。PROGRAM-ID 段落または ENTRY ステートメントのプログラム名は、そのプログラムを含むロード・モジュールの対応するロード・モジュール名またはロード・モジュール別名と同一でなければなりません。

この形式の CALL ステートメントでは、呼び出される COBOL サブプログラムはメインプログラムとはリンク・エディットされず、代わりに個別のロード・モジュールにリンク・エディットされ、実行時には、必要な場合 (つまり呼び出された場合) にだけロードされます。

動的 CALL ステートメントで呼び出される各サブプログラムは、別のロード・モジュール (システム・リンク・ライブラリーまたはユーザー指定の専用ライブラリー) の一部である場合があります。その場合、サブプログラムは z/OS ロード・ライブラリーに置かれていなければならず、階層ファイル・システムに常駐させることはできません。動的 CALL ステートメントが、ストレージに常駐していないサブプログラムを呼び出すと、そのサブプログラムは、2 次ストレージから、メインプログラムを含む領域または区画にロードされ、そのサブプログラムへの分岐が実行されます。

実行単位内でサブプログラムへの最初の動的呼び出しが行われると、そのサブプログラムの新規コピーが獲得されます。それ以後この同じサブプログラムを (元の呼び出し側または同じ実行単位内の他のサブプログラムによって) 呼び出すと、サブプログラムの同一コピーに最後に使われた状態で分岐されます (このサブプログラムが INITIAL 属性を処理する場合を除きます)。したがって、次の項目のどちらかを再初期化する必要があります。

- 更新済みの GO TO ステートメント
- データ項目

同じ COBOL プログラムが異なる実行単位のもとで呼び出された場合は、実行単位ごとに作業用ストレージの別個のコピーが割り振られます。

制約事項: 以下への動的呼び出しは許可されていません。

- COBOL DLL プログラム
- PGMNAME(LONGMIXED) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下ですべて大文字の場合は可能)。
- PGMNAME(LONGUPPER) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下の場合は可能)。
- 同じ COBOL プログラム内の複数の入り口点 (その間で CANCEL ステートメントが実行された場合は可能)

サブプログラムの取り消し

サブプログラムに対して CANCEL ステートメントを発行すると、そのサブプログラムが占有していたストレージが解放され、そのサブプログラムの後続の呼び出しは、最初の呼び出しと同じように機能します。当初の呼び出し元以外のプログラムから、サブプログラムの取り消しを行うことができます。

呼び出されるサブプログラムに複数の入り口点がある場合は、その間で CANCEL ステートメントが実行されない限り、別個の入り口点を動的 CALL ステートメントに指定することはできません。

含まれているプログラムが動的に呼び出され、それに対して CANCEL ステートメントが処理されると、そのプログラムは最初に使われた状態になります。しかし、そのプログラムは初期呼び出しでロードされず、プログラムが取り消されてもストレージは解放されません。

動的呼び出しの使用時期

動的 CALL ステートメントは、次のような場合に使用してください。

- 動的に呼び出したいロード・モジュールが、階層ファイル・システムではなく z/OS ロード・ライブラリーに入っている場合。
- 保守を容易にしたい場合。動的に呼び出されたサブプログラムが変更されても、アプリケーションを再度リンク・エディットする必要はありません。
- 呼び出されるサブプログラムが、頻繁に使用されないか、非常に大きい場合。

  サブプログラムが呼び出される条件が少ない場合、動的呼び出しは、必要な時にのみサブプログラムを作動させることができます。

  サブプログラムが非常に大きいか、あるいはその数が多い場合、静的呼び出しを使用すると、あまりにも多くの主記憶域を必要とする場合があります。使用する合計ストレージをより少なくするためには、2 つのサブプログラムを両方とも静的に呼び出すよりも、あるサブプログラムを呼び出してから取り消し、次に別のサブプログラムを呼び出してそれを取り消す必要があります。
- 未使用状態のサブプログラムを呼び出そうとしていて、INITIAL 属性を使えない場合。

  INITIAL 属性を使ってサブプログラムが呼び出されるときに必ず未使用状態にすることができない場合は、動的 CALL と CANCEL ステートメントの組み合わせを使用して未使用状態を設定できます。最初に COBOL プログラムによって呼び出されたサブプログラムを取り消すと、そのサブプログラムは次の呼び出しで未使用状態に再初期化されます。

CANCEL ステートメントを使って、非 COBOL プログラムによって動的にロードして分岐したサブプログラムを明示的に取り消しても、そのサブプログラムのストレージを解放したり、そのサブプログラムを削除するための処置は取られません。

- 31 ビットのアドレッシング・モードで実行する必要がある Enterprise COBOL と同じ実行単位内に OS/VS COBOL プログラムまたはその他の AMODE 24 プログラムがある場合。COBOL の動的呼び出し処理には、AMODE 31 プログラムを呼び出す AMODE 24 プログラム (および AMODE 24 プログラムを呼び出す AMODE 31 プログラム) のための AMODE 切り替えが含まれます。この暗黙の AMODE 切り替えを実行するためには、言語環境プログラムのランタイム・オプション ALL31(OFF) を使用する必要があります。ALL31(ON) が設定されると、AMODE 切り替えは実行されません。

  AMODE 切り替えが実行されると、制御権は呼び出し側から言語環境プログラムのライブラリー・ルーチンに渡されます。切り替えの実行後、制御権は呼び出されたプログラムに移動します。ライブラリー・ルーチンの保管域は、呼び出し側プログラムの保管域と呼び出されたプログラムの保管域の間に位置付けられます。
- 呼び出されるプログラム名が実行時までわからない場合。 この場合は、CALL *identifier* の形式を使ってください (*identifier* は、呼び出されるプログラムの名前が実行時に入れられるデータ項目です)。実際のアプリケーションでは、呼び出されるプログラムが、プログラムの条件付き処理によって変わるときには、CALL *identifier* を使用することもできます。NODYNAM を使用する場合でも、CALL *identifier* は常に動的です。

445 ページの『例: 静的および動的 CALL ステートメント』

関連概念
『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』

関連タスク
445 ページの『静的呼び出しと動的呼び出しの両方の作成』

関連参照
326 ページの『DYNAM』
CALL ステートメント (*Enterprise COBOL 言語解説書*)
ENTRY ステートメント (*Enterprise COBOL 言語解説書*)
言語環境プログラム プログラミング・リファレンス

静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項

静的に呼び出されるプログラムは、呼び出しプログラムと同じロード・モジュールにリンク・エディットされるので、静的呼び出しは動的呼び出しより高速です。動的呼び出しのサービスがアプリケーションで必要ない場合は、静的呼び出しの方が効果的です。

静的に呼び出されたプログラムは（CANCEL を使用して）削除できないので、静的呼び出しを使用すると、より多くの主記憶域を使用することになる場合があります。ストレージを重要視する場合には、動的呼び出しの使用を考えてください。呼び出しで使うストレージは、次の要因によって異なります。

- サブプログラムが 2、3 回しか呼び出されないかどうか。サブプログラムが呼び出されるかどうかに関係なく、静的に呼び出されるプログラムはストレージにロードされます。動的に呼び出されるプログラムは、サブプログラムが呼び出される場合に限りロードされます。
- 後で CANCEL ステートメントを使って、動的に呼び出されたサブプログラムを取り消すかどうか。

  静的に呼び出されたプログラムは削除できませんが、動的に呼び出されたプログラムは削除することができます。動的呼び出しを行った後で CANCEL ステートメントを使用して、動的に呼び出されたプログラムを (呼び出し後ではなく、アプリケーションで不要になった後で) 削除すると、必要なストレージは静的呼び出しを使う場合よりも少なくて済みます。

関連タスク

441 ページの『静的呼び出しの作成』
442 ページの『動的呼び出しの作成』

静的呼び出しと動的呼び出しの両方の作成

NODYNAM コンパイラー・オプションを使用してコンパイルされたプログラムでは、静的と動的の両方の CALL ステートメントを同一のプログラムに指定することができます。この場合、CALL *literal* ステートメントを使用すると、呼び出されるサブプログラムはメインプログラムとリンク・エディットされ、1 つのロード・モジュールになります。CALL *identifier* ステートメントを実行すると、別個のロード・モジュールが動的に呼び出されます。

1 つのプログラム内から同一のサブプログラムに対して動的 CALL ステートメントと静的 CALL ステートメントが出されると、そのサブプログラムの 2 つ目のコピーがストレージにロードされます。このことにより、サブプログラムが最後に使われた状態のままになるとは限らないので、結果が予測できなくなることがあります。

関連参照

326 ページの『DYNAM』

例: 静的および動的 CALL ステートメント

この例は次の 3 つの部分から構成されます。

- 静的呼び出しを使用してサブプログラムを呼び出すコード
- 動的呼び出しを使用して同じサブプログラムを呼び出すコード
- この 2 種類の呼び出しによって呼び出されるサブプログラム

次の例は、静的呼び出しをコーディングする方法を示しています。

```
PROCESS NODYNAM NODLL
IDENTIFICATION DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RECORD-2                   PIC X.                    (6)
01  RECORD-1.                                            (2)
    05  PAY                    PICTURE S9(5)V99.
    05  HOURLY-RATE            PICTURE S9V99.
    05  HOURS                  PICTURE S99V9.
. . .
```

```
PROCEDURE DIVISION.
    CALL "SUBPROG" USING RECORD-1.                          (1)
    CALL "PAYMASTR" USING RECORD-1 RECORD-2.                (5)
    STOP RUN.
```

次の例は、動的呼び出しをコーディングする方法を示しています。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  PGM-NAME                   PICTURE X(8).
01  RECORD-2                   PIC x.                      (6)
01  RECORD-1.                                              (2)
    05  PAY                    PICTURE S9(5)V99.
    05  HOURLY-RATE            PICTURE S9V99.
    05  HOURS                  PICTURE S99V9.
. . .
PROCEDURE DIVISION.
. . .
    MOVE "SUBPROG" TO PGM-NAME.
    CALL PGM-NAME USING RECORD-1.                           (1)
    CANCEL PGM-NAME.
    MOVE "PAYMASTR" TO PGM-NAME.                            (4)
    CALL PGM-NAME USING RECORD-1 RECORD-2.                  (5)
    STOP RUN.
```

次の例は、呼び出されるサブプログラムです。これは、前の 2 つの呼び出しプログラムのそれぞれによって呼び出されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
DATA DIVISION.
LINKAGE SECTION.
01  PAYREC.                                                (2)
    10  PAY             PICTURE S9(5)V99.
    10  HOURLY-RATE     PICTURE S9V99.
    10  HOURS           PICTURE S99V9.
77  PAY-CODE            PICTURE 9.                         (6)
PROCEDURE DIVISION USING PAYREC.   (1)
. . .
    EXIT PROGRAM.                                           (3)
    ENTRY "PAYMASTR" USING PAYREC PAY-CODE.                 (5)
    . . .
      GOBACK.                                               (7)
```

(1) 処理は呼び出しプログラムで開始されます。最初の CALL ステートメントが実行されると、SUBPROG (呼び出されるプログラム) の PROCEDURE DIVISION の先頭のステートメントに制御権が移動します。

各 CALL ステートメントでは、最初の USING オプションのオペランドが RECORD-1 として識別されます。

(2) SUBPROG が制御権を受け取ると、RECORD-1 内の値が SUBPROG で使用可能になります。ただし、SUBPROG では、この値は PAYREC として参照されます。

PAYREC および PAY-CODE 内の PICTURE 文字ストリングの文字数は、記述は同一ではありませんが、RECORD-1 および RECORD-2 の文字数と同じです。

(3) SUBPROG 中の処理が EXIT PROGRAM ステートメントに達すると、呼び出しプログラムに制御権が戻されます。このプログラムでの処理は、2 番目の CALL ステートメントが発行されるまで継続されます。

(4) 動的に呼び出されるプログラムの例では、2 番目の CALL ステートメントは SUBPROG 内の別の入り口点を参照するため、2 番目の CALL ステートメントの前に CANCEL ステートメントが出されます。

(5) 呼び出しプログラム中の 2 番目の CALL ステートメントによって制御権は再び SUBPROG に移動しますが、今回の処理は SUBPROG 中の ENTRY ステートメントの次のステートメントから開始されます。

(6) RECORD-1 内の値が再び PAYREC で使用可能になります。さらに、SUBPROG は対応する USING オペランド PAY-CODE を使って、RECORD-2 中の値も使えるようになります。

静的にリンクされたプログラムから 2 度目に制御権が移動すると、SUBPROG は最後に使われた状態で使えるようになります (したがって、最初の実行時に SUBPROG ストレージ内の任意の値が変更されていると、その変更された値は依然として有効です)。しかし、動的にリンクされたプログラムから制御権が移動すると、CANCEL ステートメントが実行されているため、SUBPROG は初期状態で使用可能になります。

(7) 処理が GOBACK ステートメントに達すると、呼び出しプログラム中の 2 番目の CALL ステートメントの直後のステートメントに制御権が戻されます。

呼び出されるプログラムと 2 つのいずれかの呼び出しプログラムの特定の実行において、最初の CALL と 2 番目の CALL の間で RECORD-1 内の値が変更された場合は、2 番目の CALL ステートメントの実行時に渡される値は、元の値ではなく、変更された値になります。元の値を使いたい場合は、その値を保管しなければなりません。

ネストされた COBOL プログラムの呼び出し

ネストされたプログラムを呼び出すことによって、構造化プログラミング技法を使用して、アプリケーションを作成することができます。また、データ項目を不注意で変更しないようにするためには、PERFORM プロシージャーの代わりに、ネストされたプログラムを使用することができます。

ネストされたプログラムの呼び出しには、CALL *literal* ステートメントまたは CALL *identifier* ステートメントを使用します。

含まれているプログラムは、それを直接収容しているプログラムからしか呼び出せません。ただし、含まれているプログラムを、その PROGRAM-ID 文節で COMMON として識別している場合は別です。その場合、*共通プログラム* は、共通プログラムと同じプログラムに (直接的または間接的に) 含まれている任意のプログラムから呼び出すことができます。COMMON として識別できるのは、含まれているプログラムだけです。再帰呼び出しはできません。

ネストされたプログラム構造を使用する際には、以下の指針に従ってください。

- それぞれのプログラムで IDENTIFICATION DIVISION を使用します。他の部はすべて任意指定です。
- 含まれているプログラムの名前を固有にします。プログラム名には、任意の有効な COBOL ワードか、英数字リテラルを使用します。
- 必要となる可能性のある CONFIGURATION SECTION オプションは、最外部のプログラムで設定します。含まれているプログラムが CONFIGURATION SECTION を持つことはできません。
- 含まれているプログラムはそれぞれ、収容しているプログラムの中で、End Program マーカーの直前に配置します。

- End Program マーカーを使用して、含まれているプログラムと収容しているプログラムを終了させます。

ネストされたプログラムを含むプログラムをコンパイルするときは、THREAD オプションを使用できません。

関連概念
『ネストされたプログラム』

関連参照
450 ページの『名前の有効範囲』
CALL ステートメント (*Enterprise COBOL 言語解説書*)

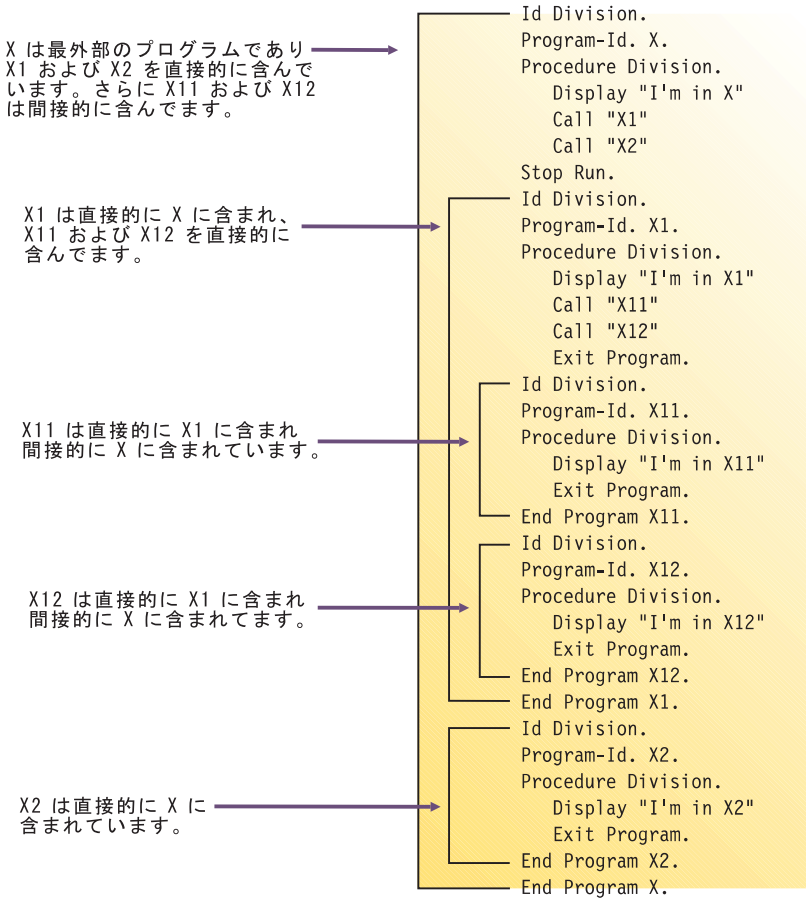
ネストされたプログラム

COBOL プログラムには、他の COBOL プログラムをネスト すること、つまり、他の COBOL プログラムを含めることができます。ネストされたプログラム自体に他のプログラムが含まれていても構いません。ネストされたプログラムは、プログラムに直接的に含めることも、間接的に含めることもできます。

呼び出されるプログラムをネストすることには、主に次の 4 つの長所があります。

1. ネストされたプログラムは、アプリケーション用のモジュラー機能を作成し、構造化プログラミング技法を保守する方法を提供します。これらは、PERFORM プロシージャーと同じようにして使うことができます。この場合、制御フローはより構造化されたものになり、ローカル・データ項目を保護することができます。
2. ネストされたプログラムは、それをアプリケーションに組み込む前にデバッグすることができます。
3. ネストされたプログラムを使うと、コンパイラーを 1 度起動するだけでアプリケーションをコンパイルできます。
4. COBOL CALL ステートメントのさまざまな形式の中で、ネストされたプログラムへの呼び出しは最高のパフォーマンスを発揮します。

次の例は、直接的および間接的に含まれたプログラムのあるネストされたプログラム構造を記述したものです。



『例: ネストされたプログラムの構造』

関連タスク

447 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照

450 ページの『名前の有効範囲』

例: ネストされたプログラムの構造

次の例は、一部の含まれているプログラムが COMMON として識別された、ネストされた構造を示しています。

```
Program-Id. A.
  Program-Id. A1.
    Program-Id. A11.
      Program-Id. A111.
      End Program A111.
    End Program A11.
    Program-Id. A12 is Common.
    End Program A12.
  End Program A1.
  Program-Id. A2 is Common.
  End Program A2.
  Program-Id. A3 is Common.
  End Program A13.
End Program A.
```

次の表に、上記の例で示した構造の呼び出し階層について説明しています。プログラム A12、A2、および A3 は COMMON として識別されており、それらに関連する呼び出しはそれぞれ異なります。

対象となるプログラム	対象プログラムが呼び出せるプログラム	対象プログラムを呼び出せるプログラム
A	A1、A2、A3	なし
A1	A11、A12、A2、A3	A
A11	A111、A12、A2、A3	A1
A111	A12、A2、A3	A11
A12	A2、A3	A1、A11、A111
A2	A3	A、A1、A11、A111、A12、A3
A3	A2	A、A1、A11、A111、A12、A2

次の点に注意してください。

- A2 は A1 を呼び出すことはできません。A1 は共通ではなく、A2 に含まれていないからです。
- A1 は A2 を呼び出すことができます。A2 は共通であるからです。

名前の有効範囲

ネストされた構造における名前は、ローカルとグローバルの 2 つのクラスに分けられます。名前を宣言しているプログラムの有効範囲を超えてその名前が知られているかどうかが、クラスによって判別されます。プログラム内で名前が参照された後で、特定の探索順序で名前の宣言を見つけます。

ローカル名: 他に宣言されていない限り、名前はローカルです (プログラム名を除く)。ローカル名は、それが宣言されているプログラム内からのみ見る、またはアクセスすることができます。含まれているプログラムおよび収容プログラムが、ローカル名を見たり、アクセスすることはできません。

グローバル名: グローバルである（GLOBAL 文節を使用して示された）名前は隠されず、その名前が宣言されているプログラムと、そのプログラムに直接的および間接的に含まれているすべてのプログラムからアクセスできます。したがって、含まれているプログラムは、単に項目の名前を参照するだけで、収容プログラムからの共通データおよびファイルを共用することができます。

グローバル項目に従属するすべての項目 (条件名と指標を含む) は、自動的にグローバルになります。

各宣言が異なるプログラムの中で現れるのであれば、GLOBAL 文節を使って同じ名前を何回も宣言することができます。同じ収容構造の異なるプログラムに同じ名前を持たせることによって、ネストされた構造における名前のマスキングや隠蔽が可能であることに注意してください。ただし、このマスキングによって、名前宣言の探索時に問題が生じることがあります。

名前の宣言の探索: プログラム内で名前が参照されると、その名前の宣言を見つける探索が行われます。探索は、参照が含まれるプログラムで始まり、一致する名前が見つかるまで、順番に収容プログラムへ移って外側へ続けられます。探索は次のプロセスに従います。

1. そのプログラム内の宣言が最初に探索されます。
2. 一致する名前が見つからない場合は、連続する外側の収容プログラムの中で、グローバル宣言だけが探索されます。
3. 最初の一致する名前が見つかった時点で探索は終了します。一致する名前が見つからない場合、エラーが存在しています。

探索はグローバル名に関するものであり、データ項目やファイル結合子などの名前に関連した特定の型のオブジェクトに関するものではありません。オブジェクトの型に関係なく、何らかの一致する名前が見つかると探索は停止します。宣言されたオブジェクトが予期されたものと違う場合は、エラー状態が存在します。

再帰呼び出しの作成

呼び出されるプログラムは、その呼び出し側を直接または間接に実行することができます。たとえば、プログラム X がプログラム Y を呼び出し、プログラム Y がプログラム Z を呼び出し、そして、プログラム Z がプログラム X を呼び出します。このような呼び出しを*再帰的* と言います。

再帰呼び出しを行うには、再帰的に呼び出されるプログラムの PROGRAM-ID 段落で RECURSIVE 文節をコーディングしなければなりません。PROGRAM-ID 段落で RECURSIVE 文節がコーディングされていない COBOL プログラムを再帰的に呼び出そうとすると、条件が通知されます。この条件を未処理のままにしておくと、実行単位が終了します。

関連タスク
8 ページの『プログラムを再帰的として識別する』

関連参照
RECURSIVE 属性 (*Enterprise COBOL 言語解説書*)
352 ページの『THREAD』

オブジェクト指向プログラムとの間での呼び出し

オブジェクト指向プログラムを含むアプリケーションを作成する場合、オブジェクト指向 COBOL プログラムは COBOL DLL プログラムなので、1 つまたは複数のダイナミック・リンク・ライブラリー (DLL) に置くことができます (ただし、各クラス定義は個別の DLL に置かれている必要があります)。 COBOL DLL プログラムとの間での呼び出しでは、DLL リンケージを使用するか、または静的 CALL のいずれかでなければなりません。COBOL DLL プログラムとの間での COBOL 動的呼び出しはサポートされていません。

COBOL の非 DLL プログラムから COBOL DLL プログラムを呼び出す必要がある場合は、DLL リンケージ・メカニズムに従っていることを保証する別の方法を使用することができます。

プロシージャー・ポインターと関数ポインターの使用

プロシージャー・ポインターとは、USAGE IS PROCEDURE-POINTER 文節によって定義されるデータ項目です。関数ポインターとは、USAGE IS FUNCTION-POINTER 文節によって定義されるデータ項目です。ここでは、「ポインター」は、プロシージャー・ポインター・データ項目または関数ポインター・データ項目のことを意味します。プロシージャー・ポインター・データ項目または関数ポインター・データ項目は、以下の入り口点の入り口アドレス (ポインター) が入るように設定することができます。

- ネストされていない別の COBOL プログラム。たとえば、例外条件が発生したときに、ユーザー作成エラー処理ルーチンに制御権を渡すためには、まずそのルーチンの入り口アドレスを CEEHDLR (条件管理言語環境プログラム呼び出し可能サービス) に渡して、それを登録させなければなりません。
- 別の言語で作成されているプログラム。たとえば、C 関数の入り口アドレスを受け取るためには、CALL ステートメントの CALL RETURNING 形式を使用して関数を呼び出してください。この結果、SET ステートメントの形式を使用して関数ポインターとして使用したりプロシージャー・ポインターに変換できるポインターが戻されます。
- 別の COBOL プログラムの代替入り口点 (ENTRY ステートメントで定義されたもの)。

プロシージャー・ポインター・データ項目および関数ポインター・データ項目の設定には、形式 6 の SET ステートメントのみを使用することができます。SET ステートメントは、コンパイラー・オプション DYNAM|NODYNAM および DLL|NODLL の設定に応じて、プログラムと同じロード・モジュール内の入り口点、個別のロード・モジュール、または DLL からエクスポートされた入り口点を参照するように、プロシージャー・ポインターを設定します。したがって、これらのポインター・データ項目を使用する場合は、以下の点を考慮する必要があります。

- NODYNAM オプションおよび NODLL オプションを指定してプログラムをコンパイルするときに、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、プログラムと同じロード・モジュール内の入り口点を参照している必要があります。そうしないと、参照は解決できません。
- NODLL オプションを指定してプログラムをコンパイルするときに、ポインター項目を実行時に入り口点の名前が入る ID に設定する場合、またはポインター項目をリテラルに設定し DYNAM オプションを指定してコンパイルする場合、ポインター項目は (リテラルであっても変数であっても) 別個のロード・モジュール内の入り口点を指している必要があります。入り口点は、1 次入り口点か、リンケージ・エディターまたはバインダーの ALIAS ステートメントで指定した代替入り口点になります。
- NODYNAM オプションおよび DLL オプションを指定してコンパイルするときに、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、プログラムと同じロード・モジュール内の入り口点か、DLL モジュールからエクスポートされた入り口点名を参照している必要があります。後者の場合は、プログラム・ロード・モジュールのリンク・エディットに、ターゲット DLL モジュールの DLL サイド・デックを含めなければなりません。
- NODYNAM オプションおよび DLL オプションを指定してコンパイルするときに、ポインター項目を ID (実行時に入り口点名が入るデータ項目) に設定する場合、そ

の ID 値は、DLL モジュールからエクスポートされた入り口点名を参照している必要があります。この場合、DLL モジュール名は、エクスポートされた入り口点の名前と一致しなければなりません。

ポインター項目を、動的に呼び出されるロード・モジュールの入り口アドレスに設定し、その後、動的に呼び出されたモジュールをプログラムで取り消すと、ポインター項目は未定義になります。その後その項目を参照しても、結果は信頼できないものになります。

どのタイプのポインターを使用するかの決定

プロシージャー・ポインターを使用して、他の COBOL プログラムを呼び出したり、言語環境プログラム呼び出し可能サービスを呼び出したりします。プロシージャー・ポインターは、COBOL-to-COBOL 呼び出しの関数ポインターよりも効率的で、言語環境プログラム条件処理サービスへの呼び出しには必須です。

関数ポインターを使用して、C/C++ プログラムや Java Native Interface が提供するサービスと通信します。

C 関数ポインターの呼び出し

C で書かれた多くの呼び出し可能サービスは、関数ポインターを戻します。以下に示すように COBOL 関数ポインターを使用して、COBOL プログラムからそのような C 関数ポインター を呼び出すことができます。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DEMO.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
******
 WORKING-STORAGE SECTION.
 01 FP USAGE FUNCTION-POINTER.
******
 PROCEDURE DIVISION.
     CALL "c-function" RETURNING FP.
     CALL FP.
 . . .
```

代替入り口点への呼び出し

呼び出されるプログラムの中で ENTRY ラベルを使用すると、プログラムが実行を開始する別の入り口点を指定することができます。しかし、構造化プログラムではこの方法はお勧めしません。

代替入り口点への静的呼び出しは制限なしで作動します。

代替入り口点への動的呼び出しには次のものが必要です。

- NAME または ALIAS リンケージ・エディターまたはバインダー制御ステートメント。
- NAME コンパイラー・オプション。
- 異なる入り口点での同じモジュールに対する動的呼び出しについては、その呼び出し間に CANCEL が介在すること。CANCEL を使用すると、プログラムは、新しい入り口点で呼び出されたとき初期状態で呼び出されます。

445 ページの『例: 静的および動的 CALL ステートメント』

関連タスク
476 ページの『DLL でのプロシージャー・ポインターまたは関数ポインターの使用』

関連参照
326 ページの『DYNAM』
334 ページの『NAME』
CANCEL ステートメント (*Enterprise COBOL 言語解説書*)
プロシージャー・ポインター (*Enterprise COBOL 言語解説書*)
関数ポインター (*Enterprise COBOL 言語解説書*)
ENTRY ステートメント (*Enterprise COBOL 言語解説書*)

プログラムを再入可能にする

複数のユーザーが同時にアプリケーション・プログラムを実行する場合 (たとえば、異なるアドレス・スペースのユーザーがリンク・パック域に常駐する同じプログラムにアクセスする場合) は、プログラムのコンパイル時に RENT オプションを使用してそのプログラムを*再入可能* にしなければなりません。プログラマーとしては、変数の複数コピーについて心配する必要はありません。コンパイラーが、オブジェクト・モジュールの中で必要な再入可能制御コードを作成します。

以下の Enterprise COBOL プログラムは再入可能でなければなりません。

- CICS で使用されるプログラム
- IMS でプリロードされるプログラム
- DB2 ストアード・プロシージャーとして使用されるプログラム
- UNIX 環境で実行されるプログラム

再入可能プログラムの場合は、DATA コンパイラー・オプションと HEAP および ALL31 ランタイム・オプションを使用して、WORKING-STORAGE などの動的データ域を 16MB 境界より下のストレージから獲得するか、または上のストレージから獲得するかを制御します。

関連概念
38 ページの『ストレージとそのアドレス可能度』

関連参照
343 ページの『RENT』
321 ページの『DATA』
ALL31 ランタイム・オプション (*言語環境プログラム プログラミング・リファレンス*)
HEAP ランタイム・オプション (*言語環境プログラム プログラミング・リファレンス*)

第 25 章 データの共用

実行単位が互いに呼び出す、別々にコンパイルされた複数のプログラムで構成される場合、プログラムは互いに通信できなければなりません。また、通常は共通データに対するアクセス権が必要です。

この章では、他のプログラムとデータを共用することができるプログラムの作成方法について説明します。説明の中でサブプログラム と呼んでいるのは、別のプログラムによって呼び出される任意のプログラムです。

関連タスク
『データの受け渡し』
458 ページの『LINKAGE SECTION のコーディング』
458 ページの『引き数を受け渡すための PROCEDURE DIVISION のコーディング』
463 ページの『戻りコード情報の受け渡し』
464 ページの『CALL . . . RETURNING の指定』
464 ページの『EXTERNAL 文節によるデータの共用』
464 ページの『プログラム間でのファイルの共用 (外部ファイル)』

データの受け渡し

プログラム間でのデータの受け渡しは、次の 3 つの方法で行うことができます。

BY REFERENCE
サブプログラムは、データのコピーを処理するのではなく、呼び出しプログラムのストレージ内のデータ項目を参照して処理します。

BY CONTENT
呼び出しプログラムは、*literal* または *identifier* の内容だけを渡します。CALL . . . BY CONTENT を使用すると、呼び出されるプログラムは、*literal* または *identifier* を受け取ったときの変数を変更する場合でも、呼び出しプログラム内の *literal* または *identifier* の値を変更することはできません。

BY VALUE
呼び出しプログラムまたはメソッドは、送り出しデータ項目を参照せず、*literal* または *identifier* の値を渡します。

呼び出されるプログラムまたはメソッドは、その中のパラメーターを変更できます。しかし、サブプログラムまたはメソッドは送り出しデータ項目の一時コピーへのアクセス権しか持っていないので、この変更内容は呼び出しプログラムの引き数には影響を与えません。

プログラムでどのようにデータを処理したいかに基づいて、上記 3 つのデータ受け渡し方法のどれを使用するかを決定してください。

© Copyright IBM Corp. 1991, 2001

コード	目的	コメント
CALL . . . BY REFERENCE *identifier*.	呼び出しプログラムの CALL ステートメントの引き数の定義と呼び出されるプログラムのパラメーターの定義に、同じメモリーを共用させる。	サブプログラムがパラメーターに対して行う変更は、呼び出しプログラムの引き数に影響を与えます。
CALL . . . BY REFERENCE *file-name*.	データ制御ブロック (DCB) をアセンブラー・プログラムに渡す。	ファイル名は QSAM 順次ファイルを参照しなければなりません。[1]
CALL . . . BY CONTENT ADDRESS OF *record-name*.	レコード域のアドレスを呼び出されるプログラムに渡す。	
CALL . . . BY CONTENT *identifier*.	ID のコピーをサブプログラムに渡す。サブプログラムによってパラメーターを変更しても、呼び出し側の ID には影響しません。	
CALL . . . BY CONTENT *literal*.	呼び出されるプログラムにリテラル値を渡す。	
CALL . . . BY CONTENT LENGTH OF *identifier*.	データ項目の長さを渡す。	呼び出しプログラムは、その LENGTH 特殊レジスターから *identifier* の長さを渡します。

次のような BY REFERENCE と BY CONTENT の組み合わせ:
`CALL 'ERRPROC'`
`  USING BY REFERENCE A`
`  BY CONTENT LENGTH OF A.`
	データ項目とその長さの両方をサブプログラムに渡す。	

CALL . . . BY VALUE *identifier*.	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。ID のコピーが渡されます。	
CALL . . . BY VALUE *literal*.	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。リテラルのコピーが渡されます。	
CALL . . . BY VALUE ADDRESS OF *identifier*	引き数を示すポインターを予期する C/C++ プログラムにデータを渡す。	
CALL . . . RETURNING	関数戻り値を使用して C/C++ 関数を呼び出す。	

1. CALL オペランドとしてのファイル名は、COBOL への IBM 拡張部分として許可されます。拡張部分の使用は一般的に、コンパイラーの特定の内部インプリメンテーションによって異なります。制御ブロック・フィールドの設定は、今後のリリースで変更される場合があります。制御ブロックに対して行われる変更は、各自の責任であって、IBM がサポートするものではありません。

呼び出しプログラムの中での引き数の記述

呼び出しプログラムでは、DATA DIVISION の他のデータ項目と同じ方法で、DATA DIVISION で引き数を記述します。呼び出しプログラムのデータ項目は、それが直接または間接的に呼び出すすべてのプログラムの LINKAGE SECTION の中で記述しなければなりません。

引き数のためのストレージは、最高位の一番外側のプログラムにおいてのみ割り振られます。たとえば、プログラム A がプログラム B を呼び出し、プログラム B がプログラム C を呼び出すとします。データ項目はプログラム A で割り振られ、プログラム B と C の LINKAGE SECTION で記述され、そのデータの集合を 3 つのすべてのプログラムで使用できます。

ファイルのデータを参照する場合、データが参照される時には、そのファイルはオープンされていなければなりません。

引き数を渡すためには、CALL ステートメントの USING 句をコーディングしてください。

16MB 境界より上のストレージで割り振られたパラメーターを AMODE 24 サブプログラムに渡してはなりません。RENT オプションが有効な場合は DATA(24) オプションを使用し、NORENT オプションが有効な場合は RMODE(24) オプションを使用してください。

呼び出されるプログラムの中でのパラメーターの記述

呼び出しプログラムから何が渡されるのかを理解し、それを呼び出されるプログラムの LINKAGE SECTION で記述しなければなりません。

パラメーターを受け取るためには、USING 句を PROCEDURE DIVISION ヘッダーの後にコーディングします。

OMITTED 引き数に関するテスト

CALL ステートメント内の引き数の代わりに OMITTED キーワードをコーディングして、1 つまたは複数の BY REFERENCE 引き数が、呼び出されるプログラムに渡されないように指定することができます。たとえば、プログラム sub1 を呼び出すときに、2 番目の引き数を省略するには、次のステートメントをコーディングします。

```
Call 'sub1' Using PARM1, OMITTED, PARM3
```

CALL ステートメントの USING 句の引き数は、数および位置において呼び出されるプログラムのパラメーターと一致しなければなりません。

呼び出されるプログラムで、対応するパラメーターのアドレスを NULL と比較して、引き数が OMITTED として渡されたかどうかをテストすることができます。以下にその例を示します。

```
Program-ID. sub1.
. . .
Procedure Division Using RPARM1, RPARM2, RPARM3.
    If Address Of RPARM2 = Null Then
        Display 'No 2nd argument was passed this time'
    Else
        Perform Process-Parm-2
    End-If
```

関連概念

38 ページの『ストレージとそのアドレス可能度』

関連タスク

464 ページの『CALL . . . RETURNING の指定』

464 ページの『EXTERNAL 文節によるデータの共用』
464 ページの『プログラム間でのファイルの共用 (外部ファイル)』

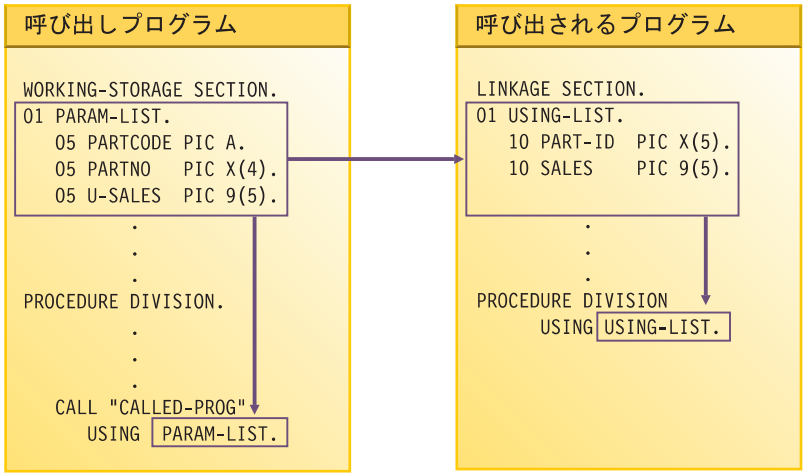
関連参照
CALL ステートメント (*Enterprise COBOL 言語解説書*)
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

LINKAGE SECTION のコーディング

呼び出されるプログラムの *identifier* リスト内の *data-names* と同じ数だけ、呼び出しプログラムの *identifier* リストに *data-names* をコーディングしなければなりません。コンパイラーは呼び出しプログラムの最初の *identifier* を呼び出されるプログラムの最初の *identifier* に渡し、以後も同様に渡していくので、それらの位置を同期させなければなりません。

呼び出されるプログラムの *identifier* リストの *data-names* の数が、呼び出しプログラムの *identifier* リストの *data-names* の数よりも多いと、エラーになります。コンパイラーは引き数とパラメーターの突き合わせを試行しません。

次の図は、あるプログラムから別のプログラムに渡されるデータ項目を示しています。



呼び出しプログラムでは、パーツ (PARTCODE) とパーツ番号 (PARTNO) のコードは別々に参照されます。それに対して、呼び出されるプログラムでは、パーツのコードとパーツ番号のコードが 1 つのデータ項目 (PART-ID) に結合されています。呼び出されるプログラムにおける PART-ID への参照は、これらの項目に対する唯一有効な参照です。

引き数を受け渡すための PROCEDURE DIVISION のコーディング

引き数を BY VALUE によって渡す場合は、サブプログラムの PROCEDURE DIVISION ヘッダーで USING BY VALUE 文節を使用してください。

```
PROCEDURE DIVISION USING BY VALUE
```

引き数を BY REFERENCE または BY CONTENT で渡す場合は、引き数の受け渡し方法を PROCEDURE DIVISION で指示する必要はありません。

ヘッダーは、次のいずれかの方法でコーディングすることができます。

```
PROCEDURE DIVISION USING
PROCEDURE DIVISION USING BY REFERENCE
```

関連参照
手続き部のヘッダー (*Enterprise COBOL 言語解説書*)

受け渡されるデータのグループ化

プログラム間で渡したいすべてのデータ項目をグループ化し、それらを 1 つのレベル 01 項目に入れることを考慮してください。こうすると、プログラム相互間で 1 個のレベル 01 レコードを渡すことができます。

レコード突き合わせの間違いの可能性をより少なくするためには、レベル 01 レコードをコピー・ライブラリーの中に置き、両方のプログラムでそれをコピーするようにします。すなわち、呼び出しプログラムの WORKING-STORAGE SECTION と呼び出されるプログラムの LINKAGE SECTION の中でコピーします。

関連タスク
458 ページの『LINKAGE SECTION のコーディング』

ヌル終了ストリングの処理

ヌル終了リテラル、16 進リテラル X'00'、およびストリング処理動詞を使用する場合、COBOL はヌル終了ストリングをサポートします。

ヌル終了ストリング (たとえば、C プログラムから渡された) は、次のコードのようなストリング処理メカニズムを使用して処理することができます。

```
01 L        pic X(20) value z'ab'.
01 M        pic X(20) value z'cd'.
01 N        pic X(20).
01 N-Length pic 99    value zero.
01 Y        pic X(13) value 'Hello, World!'.
```

ヌル終了ストリングの長さを決定してから、そのストリングの値を表示するには、次のように指定します。

```
Inspect N tallying N-length for characters before initial X'00'
Display 'N: ' N(1:N-length) ' Length: ' N-length
```

ヌル終了ストリングを英数字に移動し、ヌルを除去するには、次のように指定します。

```
Unstring N   delimited by X'00'
         into X
```

ヌル終了ストリングを作成するには、次のように指定します。

```
String Y     delimited by size
       X'00' delimited by size
       into N.
```

2 つのヌル終了ストリングを連結するには、次のように指定します。

```
String L     delimited by x'00'
       M     delimited by x'00'
       X'00' delimited by size
       into N.
```

関連タスク
100 ページの『ヌル終了ストリングの処理』

関連参照
ヌル終了英数字リテラル (*Enterprise COBOL 言語解説書*)

チェーン・リストを使用するためのポインターの使用

レコード域のアドレスを受け渡す場合には、データ・アドレスを保持するための特殊なデータ項目タイプである、ポインター・データ項目を処理することができます。ポインター・データ項目とは、USAGE IS POINTER 文節を使用して定義されているデータ項目、または ADDRESS 特殊レジスターであるデータ項目のいずれかです。ポインター・データ項目の代表的な適用は、チェーン・リスト (各レコードがそれぞれ次のレコードを指し示す一連のレコード) の処理です。

プログラム相互間のアドレスをチェーン・リストに入れて渡す場合は、NULL を使用して、無効なアドレスの値 (非数値 0) をポインター項目に割り当てることができます。次の 2 つの方法で、ポインター・データ項目に値 NULL を割り当てることができます。

- データ定義の中で VALUE IS NULL 文節を使用する。
- SET ステートメントの中で送り出しフィールドとして NULL を使用する。

最後のレコードのポインター・データ項目がヌル値を含んでいるチェーン・リストの場合、リストの終わりを検査するためのコードは次のとおりです。

```
IF PTR-NEXT-REC = NULL
. . .
 (logic for end of chain)
```

リストの終わりに達していない場合、プログラムはレコードを処理して次のレコードに移ることができます。

呼び出しプログラムから渡されるデータには、無視したいヘッダー情報が含まれていることがあります。ポインター・データ項目は数値ではないので、それらに対して直接に算術演算を行うことはできません。しかし、ヘッダー情報をバイパスするために、SET ステートメントを使用して、渡されたアドレスを増分することができます。

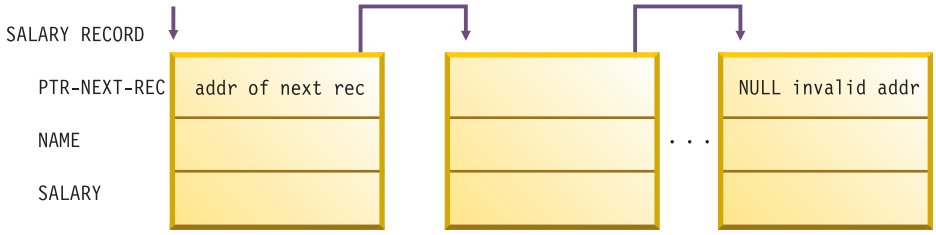
461 ページの『例: チェーン・リストを処理するためのポインターの使用』

関連タスク
458 ページの『LINKAGE SECTION のコーディング』
458 ページの『引き数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照
SET ステートメント (*Enterprise COBOL 言語解説書*)

例: チェーン・リストを処理するためのポインターの使用

この例では、個々の給与レコードで構成されるデータのチェーン・リストを描いています。次の図は、これらのレコードがストレージの中でどのようにリンクされているかを見やすく示したものです。



最後のレコードを除いて、各レコードの最初の項目は次のレコードを指し示しています。最後のレコードの最初の項目は、それが最後のレコードであることを示すために、アドレスではなくヌル値を含んでいます。

これらのレコードを処理するアプリケーションの高水準の論理は、次のようになります。

```
OBTAIN ADDRESS OF FIRST RECORD IN CHAINED LIST FROM ROUTINE
CHECK FOR END OF THE CHAINED LIST
DO UNTIL END OF THE CHAINED LIST
   PROCESS RECORD
   GO ON TO THE NEXT RECORD
END
```

次のコードは、チェーン・リストを処理するこの例で使用される、呼び出しプログラム LISTS の概要です。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. LISTS.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
******
 WORKING-STORAGE SECTION.
 77  PTR-FIRST         POINTER  VALUE IS NULL.                  (1)
 77  DEPT-TOTAL        PIC 9(4) VALUE IS 0.
******
 LINKAGE SECTION.
 01  SALARY-REC.
     02  PTR-NEXT-REC    POINTER.                               (2)
     02  NAME            PIC X(20).
     02  DEPT            PIC 9(4).
     02  SALARY          PIC 9(6).
 01  DEPT-X            PIC 9(4).
******
 PROCEDURE DIVISION USING DEPT-X.
******
* FOR EVERYONE IN THE DEPARTMENT RECEIVED AS DEPT-X,
* GO THROUGH ALL THE RECORDS IN THE CHAINED LIST BASED ON THE
* ADDRESS OBTAINED FROM THE PROGRAM CHAIN-ANCH
* AND CUMULATE THE SALARIES.
* IN EACH RECORD, PTR-NEXT-REC IS A POINTER TO THE NEXT RECORD
* IN THE LIST; IN THE LAST RECORD, PTR-NEXT-REC IS NULL.
* DISPLAY THE TOTAL.
******
     CALL "CHAIN-ANCH" USING PTR-FIRST                          (3)
     SET ADDRESS OF SALARY-REC TO PTR-FIRST                     (4)
******
```

```
     PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL (5)

      IF DEPT = DEPT-X
        THEN ADD SALARY TO DEPT-TOTAL
        ELSE CONTINUE
      END-IF
      SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC                  (6)

     END-PERFORM
******
     DISPLAY DEPT-TOTAL
     GOBACK.
```

(1) PTR-FIRST は、NULL の初期値を持つポインター・データ項目として定義されます。CHAIN-ANCH への呼び出しから正常に戻ると、PTR-FIRST には、チェーン・リスト内の最初のレコードのアドレスが入れられます。しかし、呼び出しで何か間違いが起こり、PTR-FIRST がチェーンの最初のレコードのアドレスの値を受け取っていないと、PTR-FIRST はヌル値のままであり、プログラムの論理に従って、レコードは処理されません。

(2) 呼び出しプログラムの LINKAGE SECTION には、チェーン・リストのレコードの記述が入っています。さらに、CALL ステートメントの USING 文節を使用して渡される部門コードの記述も含まれています。

(3) 最初の SALARY-REC レコード域のアドレスを取得するために、LISTSプログラムはプログラム CHAIN-ANCH を呼び出します。

(4) SET ステートメントのレコード記述 SALARY-REC の基礎となっているのは、PTR-FIRST に含まれているアドレスです。

(5) この例のチェーン・リストは、最後のレコードに無効アドレスが含まれるようにセットアップされます。チェーン・リスト終了に対するこの検査は、do-while 構造を使用して行われます (最後のレコードのポインター・データ項目には値 NULL が割り当てられる構造になっています)。

(6) LINKAGE-SECTION 内のレコードのアドレスは、SALARY-REC の最初のフィールドとして送信されるポインター・データ項目によって、次のレコードのアドレスに等しく設定されます。レコード処理ルーチンが繰り返され、チェーン・リスト内の次のレコードが処理されます。

別のプログラムから受け取ったアドレスを増分するには、LINKAGE SECTION および PROCEDURE DIVISION を次のようにセットアップすることができます。

```
LINKAGE SECTION.
01  RECORD-A.
    02  HEADER          PIC X(12).
    02  REAL-SALARY-REC PIC X(30).
. . .
01  SALARY-REC.
    02  PTR-NEXT-REC    POINTER.
    02  NAME            PIC X(20).
    02  DEPT            PIC 9(4).
    02  SALARY          PIC 9(6).
. . .
PROCEDURE DIVISION USING DEPT-X.
. . .
    SET ADDRESS OF SALARY-REC TO ADDRESS OF REAL-SALARY-REC
```

この時点では、SALARY-REC のアドレスは、REAL-SALARY-REC、または RECORD-A + 12 のアドレスを基底にしています。

関連タスク
460 ページの『チェーン・リストを使用するためのポインターの使用』

戻りコード情報の受け渡し

プログラム間で戻りコードを受け渡すには、RETURN-CODE 特殊レジスターを使用します。メソッドは RETURN-CODE 特殊レジスターに情報を戻しませんが、プログラムへの呼び出しの後でこのレジスターを検査できます。

また、メソッドの中の PROCEDURE DIVISION ヘッダーで RETURNING 句を使用して、起動プログラムまたはメソッドに情報を戻すこともできます。 PROCEDURE DIVISION . . . RETURNING で CALL . . . RETURNING を指定しても、RETURN-CODE レジスターは設定されません。

RETURN-CODE 特殊レジスターの理解

COBOL プログラムがその呼び出し側に戻ると、RETURN-CODE 特殊レジスターの内容がレジスター 15 に保管されます。制御が呼び出しから COBOL プログラムまたはメソッドに戻されると、レジスター 15 の内容は、呼び出しプログラムまたはメソッドの RETURN-CODE 特殊レジスターに保管されます。COBOL からオペレーティング・システムに制御が戻されると、特殊レジスターの内容はユーザー戻りコードとして戻されます。

非 COBOL プログラムから COBOL プログラムに制御が戻る場合には、このような RETURN-CODE の処理を考慮する必要が生じることがあります。非 COBOL プログラムが、戻りコードを戻すためにレジスター 15 を使用しない場合、COBOL プログラムの RETURN-CODE 特殊レジスターが無効値で更新されることがあります。Enterprise COBOL プログラムがオペレーティング・システムに戻る前に、この特殊レジスターを意味のある値に設定しない限り、無効な戻りコードがシステムに戻されます。

COBOL プログラムと C プログラムで同じように機能させるためには、COBOL プログラムが RETURNING オプションを使用して C プログラムを呼び出すようにしなければなりません。C プログラム (関数) が関数値を正しく宣言していれば、呼び出し COBOL プログラムの RETURNING 値が設定されます。

INVOKE ステートメントを使用して RETURN-CODE 特殊レジスターを設定することはできません。

PROCEDURE DIVISION RETURNING . . . の使用

呼び出しプログラムに情報を戻すには、プログラムの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用してください。

```
PROCEDURE DIVISION RETURNING dataname2
```

呼び出されたプログラムが正常にその呼び出し側に戻ると、*dataname2* の値が、CALL ステートメントの RETURNING 句で指定された ID に保管されます。

```
CALL . . . RETURNING dataname2
```

CEEPIPI: 言語環境プログラムの事前初期設定サービス (CEEPIPI) を使用して呼び出されるプログラムで PROCEDURE DIVISION RETURNING を指定すると結果は不定となります。

CALL . . . RETURNING の指定

C/C++ の関数または COBOL のサブルーチンへの呼び出しでは、CALL ステートメントで RETURNING 句を指定することができます。

形式は次のとおりです。

```
CALL . . . RETURNING dataname2
```

呼び出されるプログラムの戻り値は *dataname2* に保管されます。

dataname2 は、呼び出し COBOL プログラムの DATA DIVISION で定義しなければなりません。ターゲット関数で宣言される戻り値のデータ型は、*dataname2* のデータ型と同じでなければなりません。

EXTERNAL 文節によるデータの共用

EXTERNAL 文節を使用すると、別々にコンパイルされたプログラムやメソッド (バッチ・シーケンスのプログラムを含む) がデータ項目を共用できるようになります。

EXTERNAL は、プログラムまたはメソッドの WORKING-STORAGE SECTION の 01 レベルのデータ記述でコーディングします。以下の規則が適用されます。

- EXTERNAL グループ項目に従属する項目はそれ自体が EXTERNAL です。
- データ項目の名前を、同じプログラムの中で別の EXTERNAL 項目の名前として使用することはできません。
- VALUE 文節は、グループ項目または従属項目 (すなわち、EXTERNAL) には指定できません。

実行単位内で、ある COBOL プログラムまたはメソッドの項目のデータ記述が、その項目を含むプログラムのデータ記述と同じ場合は、そのプログラムまたはメソッドはそのデータ項目にアクセスして処理できます。たとえば、プログラム A に次のデータ記述があるとします。

```
01 EXT-ITEM1     EXTERNAL     PIC 99.
```

この場合、プログラム B は、WORKING-STORAGE SECTION に同じデータ記述を置くことによって、そのデータ項目にアクセスすることができます。

EXTERNAL データ項目にアクセス権を持っているプログラムはその値を変更できることを覚えておいてください。保護しなければならないデータ項目には、この文節を使用しないでください。

プログラム間でのファイルの共用 (外部ファイル)

ファイルに EXTERNAL 文節を使用すると、実行単位内の別々にコンパイルされたプログラムまたはメソッドが共通ファイルにアクセスできるようになります。以下の診断に従うことをお勧めします。

- ファイル状況コードを検査するすべてのプログラムの FILE STATUS 文節で、同じデータ名を使用する。
- 同じファイル状況フィールドを検査するすべてのプログラムについて、各プログラムのファイル状況フィールドのレベル 01 データ定義で EXTERNAL 文節をコーディングする。

外部ファイルを使用すると、次のような利点があります。

- メインプログラムに入出力ステートメントが含まれていなくても、メインプログラムからファイルのレコード域を参照することができます。
- それぞれのサブプログラムが、OPEN や READ のような単一の入出力機能を制御することができます。
- それぞれのプログラムがファイルにアクセスすることができます。

『例: 外部ファイルの使用』

関連参照
EXTERNAL 文節 (*Enterprise COBOL 言語解説書*)

例: 外部ファイルの使用

次の表に、以下の例で使用しているメインプログラムとサブプログラムを説明しています。

名前	機能
ef1	メインプログラム。すべてのサブプログラムを呼び出し、レコード域の内容を検査する。
ef1openo	出力用に外部ファイルをオープンし、ファイル状況コードを検査する。
ef1write	外部ファイルにレコードを書き込み、ファイル状況コードを検査する。
ef1openi	入力用に外部ファイルをオープンし、ファイル状況コードを検査する。
ef1read	外部ファイルからレコードを読み取り、ファイル状況コードを検査する。
ef1close	外部ファイルをクローズし、ファイル状況コードを検査する。

さらに、それぞれのサブプログラムにファイルの同じ記述が含まれていることを保証するために、COPY ステートメントを使用します。

例の各プログラムは、それぞれの WORKING-STORAGE SECTION に、EXTERNAL 文節を使用してデータ項目を宣言します。この項目は、ファイル状況コードの検査に使用されるもので、COPY ステートメントを使用しても設定されます。

各プログラムは、次の 3 つのコピーブックを使用します。

- 最初の名前は efselect であり、FILE-CONTROL 段落に設定されます。

```
Select ef1
Assign To ef1
File Status Is efs1
Organization Is Sequential.
```

- 2 番目の名前は effile であり、FILE SECTION に設定されます。

```
  Fd ef1 Is External
            Record Contains 80 Characters
            Recording Mode F.
  01  ef-record-1.
      02 ef-item-1     Pic X(80).
```

- 3 番目の名前は efwrkstg であり、WORKING-STORAGE SECTION に設定されます。

```
  01  efs1             Pic 99 External.
```

外部ファイルを使用する入出力

```
 Identification Division.
 Program-Id.
     ef1.
*
* This is the main program that controls the external file
* processing.
*
 Environment Division.
 Input-Output Section.
 File-Control.
     Copy efselect.
 Data Division.
 File Section.
     Copy effile.
 Working-Storage Section.
     Copy efwrkstg.
 Procedure Division.
     Call "ef1openo"
     Call "ef1write"
     Call "ef1close"
     Call "ef1openi"
     Call "ef1read"
     If ef-record-1 = "First record" Then
       Display "First record correct"
     Else
       Display "First record incorrect"
       Display "Expected: " "First record"
       Display "Found   : " ef-record-1
     End-If
     Call "ef1close"
     Goback.
 End Program ef1.
 Identification Division.
 Program-Id.
     ef1openo.
*
* This program opens the external file for output.
*
 Environment Division.
 Input-Output Section.
 File-Control.
     Copy efselect.
 Data Division.
 File Section.
     Copy effile.
 Working-Storage Section.
     Copy efwrkstg.
 Procedure Division.
     Open Output ef1
     If efs1 Not = 0
       Display "file status " efs1 " on open output"
       Stop Run
     End-If
     Goback.
 End Program ef1openo.
 Identification Division.
```

```
 Program-Id.
     ef1write.
*
* This program writes a record to the external file.
*
 Environment Division.
 Input-Output Section.
 File-Control.
     Copy efselect.
 Data Division.
 File Section.
     Copy effile.
 Working-Storage Section.
     Copy efwrkstg.
 Procedure Division.
     Move "First record" to ef-record-1
     Write ef-record-1
     If efs1 Not = 0
       Display "file status " efs1 " on write"
       Stop Run
     End-If
     Goback.
 End Program ef1write.
 Identification Division.
 Program-Id.
     ef1openi.
*
* This program opens the external file for input.
*
 Environment Division.
 Input-Output Section.
 File-Control.
     Copy efselect.
 Data Division.
 File Section.
     Copy effile.
 Working-Storage Section.
     Copy efwrkstg.
 Procedure Division.
     Open Input ef1
     If efs1 Not = 0
       Display "file status " efs1 " on open input"
       Stop Run
     End-If
     Goback.
 End Program ef1openi.
 Identification Division.
 Program-Id.
     ef1read.
*
* This program reads a record from the external file.
*
 Environment Division.
 Input-Output Section.
 File-Control.
     Copy efselect.
 Data Division.
 File Section.
     Copy effile.
 Working-Storage Section.
     Copy efwrkstg.
 Procedure Division.
     Read ef1
     If efs1 Not = 0
       Display "file status " efs1 " on read"
       Stop Run
     End-If
```

```
    Goback.
 End Program ef1read.
 Identification Division.
 Program-Id.
    ef1close.
*
* This program closes the external file.
*
 Environment Division.
 Input-Output Section.
 File-Control.
    Copy efselect.
 Data Division.
 File Section.
    Copy effile.
 Working-Storage Section.
    Copy efwrkstg.
 Procedure Division.
    Close ef1
    If efs1 Not = 0
      Display "file status " efs1 " on close"
      Stop Run
    End-If
    Goback.
 End Program ef1close.
```

第 26 章 DLL または DLL アプリケーションの作成

ダイナミック・リンク・ライブラリー (DLL) または DLL アプリケーションの作成は、通常の COBOL アプリケーションの作成と似ています。すなわち、ソース・コードを書き、コンパイルし、リンクします。

DLL または DLL アプリケーションを作成する際には、以下の特別な考慮事項が適用されます。

- ロード・モジュールまたはロード・モジュールの各部分を互いに関連付ける方法、または他の DLL と関連付ける方法を決定する。
- どんなリンクまたは呼び出しのメカニズムを使用するかを決定する。

DLL ロード・モジュールが必要であるのか、または別個の DLL を参照するロード・モジュールが必要であるのかによって、若干異なるコンパイラーとリンケージ・エディターまたはバインダー・オプションを使用する必要があります。

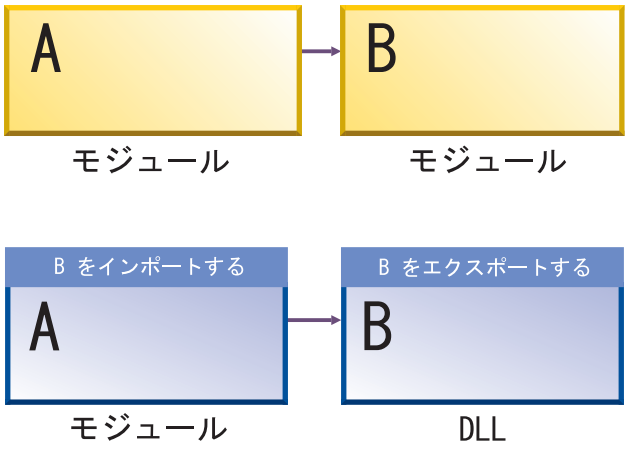
関連概念
ダイナミック・リンク・ライブラリー (DLL)

関連タスク
474 ページの『DLL での CALL ID の使用』
475 ページの『DLL リンケージと動的呼び出しの併用』
479 ページの『C/C++ プログラムでの COBOL DLL の使用』
480 ページの『OO COBOL アプリケーションでの DLL の使用』
470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』
452 ページの『プロシージャー・ポインターと関数ポインターの使用』

ダイナミック・リンク・ライブラリー (DLL)

DLLは、ロード・モジュールまたはプログラム・オブジェクトであり、他のロード・モジュールからアクセスすることができます。DLL は、プログラム、関数、または変数の定義を DLL、DLL アプリケーション、または非 DLL へエクスポートするという点で、従来のロード・モジュールとは異なります。このため、ターゲット・ルーチンを、参照ルーチンと同じロード・モジュールにリンクする必要はありません。アプリケーションが初めて別個の DLL を参照すると、システムは自動的にその DLL をメモリーにロードします。すなわち、DLL 内でプログラムを呼び出すことは、動的 CALL を使用してロード・モジュールを呼び出すことに似ています。

© Copyright IBM Corp. 1991, 2001



DLL アプリケーションは、プログラム、関数、または変数のインポートされた定義を参照するアプリケーションです。

z/OS DLL のいくつかの機能は COBOL の動的 CALL ステートメントで提供される機能とオーバーラップしますが、DLL の方が、次のように、通常の z/OS ロード・モジュールおよび動的呼び出しよりも利点があります。

- DLL は COBOL、および C/C++ 間で共通なので、複数のプログラム言語を使用するアプリケーションの相互協調処理が向上します。再入可能な COBOL および C/C++ DLL も、スムーズに相互協調処理することができます。
- 長いプログラム名を持つ、別個の DLL モジュールに入っているプログラムへの呼び出しを行うことができます。(動的呼び出し解決は、プログラム名を 8 文字に切り捨てます。) COBOL オプションの PGMNAME(LONGUPPER) または PGMNAME(LONGMIXED) と COBOL DLL サポートを使用すると、最高 160 文字までの名前を持つロード・モジュール間で呼び出しを行うことができます。

DLL は、z/OS DFSMS プログラム管理バインダーによって提供される機能に基づいて、IBM 言語環境プログラムによってサポートされます。DLL サポートは、z/OS のもとでバッチ環境で実行中のアプリケーションで、あるいは TSO、 CICS、UNIX、または IMS 環境で使用可能です。

関連参照
340 ページの『PGMNAME』
z/OS DFSMS: Program Management の『Binder support for DLLs』の項

DLL を作成するためのプログラムのコンパイル

DLL オプションを使用して COBOL プログラムをコンパイルすると、そのプログラムは DLL サポートに使用可能になります。DLL サポートを使用するアプリケーションは再入可能でなければなりません。このため、RENT コンパイラー・オプションを使用してコンパイルし、それらを RENT バインダー・オプションとリンクしなければなりません。

DLL サポートのあるアプリケーションでは、プログラムまたはクラスの場所に応じて以下のコンパイラー・オプションを使用してください。

プログラムまたはクラスの場所	使用するオプション
ルート・ロード・モジュール	DLL、RENT、NOEXPORTALL
他のロード・モジュールによって使用される DLL ロード・モジュール	DLL、RENT、EXPORTALL

DLL ロード・モジュールに、DLL モジュール内からだけ使用されるプログラムが含まれている場合は、NOEXPORTALL を使用してコンパイルすることによって、これらのルーチンを DLL の内部に隠すことができます。

473 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク

『DLL のリンク』
472 ページの『特定の DLL のプリリンク』
469 ページの『第 26 章 DLL または DLL アプリケーションの作成』

関連参照

324 ページの『DLL』
327 ページの『EXPORTALL』
343 ページの『RENT』

DLL のリンク

DLL 可能オブジェクト・モジュールを別個の DLL ロード・モジュールにリンクするか、またはそれらを静的にリンクすることができます。リンク時に、アプリケーションを 1 つのモジュールとしてパッケージするか、いくつかの DLL モジュールとしてパッケージするかを決定することができます。

DLL アプリケーションのリンクには、DFSMS バインダーで DLL サポートを使用することをお勧めします。DFSMS バインダーは COBOL コンパイラーからの出力を直接受け取ることができ、このため、プリリンク・ステップは必要ありません。しかし、DLL を PDS ロード・ライブラリーに常駐させる必要がある場合、または CICS Transaction Server 1.3 より下位レベルの CICS システムに DLL を配置することになる場合は、プリリンカーを使用する必要があります。

バインダー・ベースの DLL は、PDS ではなく、PDSE または HFS ファイルに常駐させなければなりません。

DFSMS バインダーを使用して DLL アプリケーションをリンクするときは、以下のバインダー・オプションを使用してください。

コードのタイプ	リンクに使用するバインダー・パラメーター
DLL アプリケーション	DYNAM(DLL)、RENT、COMPAT(PM3)、または COMPAT(CURRENT)
大文字小文字混合のエクスポート済みプログラム名を使用するアプリケーション クラス定義または INVOKE ステートメント	CASE(MIXED)

SYSDEFSD DD ステートメントを指定して、バインダーが DLL 定義サイド・ファイルを作成するデータ・セットを指示してください。このサイド・ファイルには、DLL によってエクスポートされた各記号についての IMPORT 制御ステートメントが入っています。バインダー SYSLIN 入力 (DLL コードを参照するバインディング・コード) には、リンクされるモジュールから参照される DLL のための DLL 定義サイド・ファイルを含めなければなりません。

DLL リンケージに使用させたくないプログラムがモジュールに含まれている場合には、定義サイド・ファイルを編集してそれらのプログラムを除去することができます。

DLL を PDS ロード・ライブラリーに常駐させる必要がある場合、または出力が CICS Transaction Server 1.3 より下位レベルの CICS システムに配置されることになる場合は、標準のリンケージ編集を行う前に、言語環境プログラムのプリリンカーを使用してアプリケーションをプリリンクしなければなりません。

473 ページの『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク

469 ページの『第 26 章 DLL または DLL アプリケーションの作成』
470 ページの『DLL を作成するためのプログラムのコンパイル』
『特定の DLL のプリリンク』
z/OS DFSMS: Program Management

関連参照

z/OS DFSMS: Program Management の『Binder support for DLLs』の項

特定の DLL のプリリンク

以下の場合には、標準のリンケージ編集を行う前に、言語環境プログラムのプリリンカーを使用する必要があります。

- DLL は、PDSE または HFS ファイルではなく、PDSE ロード・ライブラリーに常駐しなければならない場合。
- 出力が CICS Transaction Server 1.3 より下位レベルの CICS システムに配置されることになる場合。

ソース DLL をコンパイルした後、オブジェクト・モジュールをプリリンクして 1 つのオブジェクト・モジュールにします。

1. プリリンク・ステップに対する SYSDEFSD DD ステートメントを指定して、プリリンカーが DLL 用の DLL 定義サイド・ファイルを作成するデータ・セットを指示します。このサイド・ファイルには、DLL によってエクスポートされた各記号についての IMPORT プリリンカー制御ステートメントが入れられます。プリリンカーは、このサイド・ファイルを使用して、新規の DLL を参照する他のモジュールをプリリンクします。
2. DLLNAME(*xxx*) プリリンカー・オプションを指定して、サイド・ファイルに IMPORT 制御ステートメントを作成する際に使用するプリリンカーの DLL ロード・モジュール名を指示します。あるいは、プリリンカーが、NAME プリリンカ

一制御ステートメントから、またはプリリンク・ステップに対する SYSMOD DD ステートメントの PDS メンバー名から、DLL ロード・モジュール名を入手することができます。

3. この DLL が他の DLL を参照する場合は、このプリリンク・ステップへの入力であるオブジェクト・デックと一緒に、これらの DLL の定義サイド・ファイルを含めてください。これらのサイド・ファイルは、プリリンカーに、現行モジュールの記号参照を他の DLL からエクスポートされた記号に解決するように指示します。

通常のようにリンケージ・エディターまたはバインダーを使用して、プリリンカーによって作成されるオブジェクト・モジュールから DLL ロード・モジュールを作成します。リンケージ・エディターまたはバインダーの RENT オプションを指定してください。

『例: プロシージャー型 DLL アプリケーションのサンプル JCL』

関連タスク

470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』

例: プロシージャー型 DLL アプリケーションのサンプル JCL

以下の例は、DLL サブプログラムを呼び出すメインプログラムから構成されるアプリケーションの作成方法を示しています。最初のステップでは、サブプログラム DemoDLLSubprogram を含む DLL ロード・モジュールを作成します。2 番目のステップでは、プログラム MainProgram を含むメイン・ロード・モジュールを作成します。3 番目のステップでアプリケーションを実行します。

```
//DLLSAMP  JOB ,
//  TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,
//  NOTIFY=&SYSUID,USER=&SYSUID
// SET LEPFX='SYS1'
//*-------------------------------------------------------------------
//* Compile COBOL subprogram, bind to form a DLL.
//*-------------------------------------------------------------------
//STEP1 EXEC IGYWCL,REGION=80M,GOPGM=DEMODLL,
//       PARM.COBOL='RENT,PGMN(LM),DLL,EXPORTALL',
//       PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN DD *
       Identification division.
       Program-id. "DemoDLLSubprogram".
       Procedure division.
           Display "Hello from DemoDLLSubprogram!".
       End program "DemoDLLSubprogram".
/*
//LKED.SYSDEFSD DD DSN=&&SIDEDECK,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(TRK,(1,1))
//LKED.SYSLMOD   DD DSN=&&GOSET(&GOPGM),DSNTYPE=LIBRARY,DISP=(MOD,PASS)
//LKED.SYSIN     DD DUMMY
//*-------------------------------------------------------------------
//* Compile and bind COBOL main program
//*-------------------------------------------------------------------
//STEP2 EXEC IGYWCL,REGION=80M,GOPGM=MAINPGM,
//       PARM.COBOL='RENT,PGMNAME(LM),DLL',
//       PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN DD *
       Identification division.
       Program-id. "MainProgram".
       Procedure division.
```

```
           Call "DemoDLLSubprogram"
           Stop Run.
       End program "MainProgram".
/*
//LKED.SYSIN    DD DSN=&&SIDEDECK,DISP=(OLD,DELETE)
//*-------------------------------------------------------------------
//* Execute the main program, calling the subprogram DLL.
//*-------------------------------------------------------------------
//STEP3 EXEC PGM=MAINPGM,REGION=80M
//STEPLIB DD DSN=&&GOSET,DISP=(OLD,DELETE)
//        DD DSN=&LEPFX..SCEERUN,DISP=SHR
//SYSOUT  DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
```

DLL での CALL ID の使用

DLL オプションを使用してコンパイルされた COBOL プログラムでは、CALL *identifier* ステートメントおよび CALL *literal* ステートメントを使用して DLL への呼び出しを行うことができます。ただし、CALL *identifier* の場合には、追加の考慮事項があります。

identifier の内容または CALL ステートメントのリテラル には、次のいずれかのプログラムの名前を使用してください。

- CALL *identifier* を含むプログラムから呼び出されるのに適格な、同じコンパイル単位内のネストされたプログラム。
- 別個のバインドされた DLL モジュール内のプログラム。ターゲット・プログラム名は DLL からエクスポートされていなければなりません。また、DLL モジュール名は、ターゲット・プログラムのエクスポートされた名前と一致していなければなりません。

ネストされていない場合、ランタイム環境は、CALL ステートメントを含むプログラムの PGMNAME コンパイラー・オプションの設定に従って、*identifier* 内のプログラム名を解釈します。また、ターゲット DLL からエクスポートされたプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。

階層ファイル・システム (HFS) におけるターゲット DLL の探索には、大文字小文字の区別があります。ターゲット DLL が PDS または PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDS または PDSE メンバーとして DLL を探索するために、ランタイム環境は名前を自動的に大文字に変換します。

ランタイム環境が上記のいずれかの CALL ステートメントを解決できない場合、制御権は CALL ステートメントの ON EXCEPTION または ON OVERFLOW 句に移動します。この状況で CALL ステートメントにこれらの句の 1 つが指定されていないと、言語環境プログラムは重大度 3 の条件を発生させます。

関連タスク

475 ページの『DLL リンケージと動的呼び出しの併用』
470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』

関連参照
324 ページの『DLL』
340 ページの『PGMNAME』
CALL ステートメント (*Enterprise COBOL 言語解説書*)
『HFS での DLL の探索順序』

HFS での DLL の探索順序

階層ファイル・システム (HFS) を使用する場合は、CALL ステートメントの DLL 参照を解決するための探索順序は、言語環境プログラムの POSIX ランタイム・オプションの設定によって異なります。

POSIX ランタイム・オプションが ON に設定されている場合の探索順序は、次のとおりです。

1. ランタイム環境は、HFS に DLL がないかどうか探します。LIBPATH 環境変数が設定されていると、リストされたそれぞれのディレクトリーが探索されます。これが設定されていない場合は、現行ディレクトリーだけが探索されます。HFS での DLL の探索には、大文字小文字の区別があります。
2. HFS で DLL が見つからないと、ランタイム環境は、呼び出し側の MVS ロード・ライブラリー探索順序から DLL をロードすることを試みます。この場合、DLL 名は 8 文字以下でなければなりません。この探索では、ランタイム環境は自動的に DLL 名を大文字に変換します。

POSIX ランタイム・オプションが OFF に設定されていると、探索順序は逆になります。

1. ランタイム環境は、呼び出し側のロード・ライブラリー用の探索順序から DLL をロードすることを試みます。
2. このロード・ライブラリーから DLL をロードできない場合、ランタイム環境は HFS から DLL をロードすることを試みます。

関連タスク
474 ページの『DLL での CALL ID の使用』

関連参照
POSIX (*言語環境プログラム プログラミング・リファレンス*)

DLL リンケージと動的呼び出しの併用

複数の別個にバインドされたモジュールとして構築されるアプリケーション (つまり、言語環境プログラム・エンクレーブ) の場合は、モジュール相互間で 1 つのリンケージ形式 (つまり、動的呼び出しリンケージまたは DLL リンケージのいずれか) だけを使用しなければなりません。*DLL リンケージ* とは、DLL および NODYNAM オプションを使用してコンパイルされたプログラム内の呼び出しを指します (呼び出しは、別個のモジュール内のエクスポートされた名前に解決されます)。DLL リンケージが、別個のモジュール内に定義されているメソッドの呼び出しを指すこともあります。

しかし、アプリケーションによってはもっと多くの柔軟性が必要になります。このような場合は、プログラムが次のようにコンパイルされていれば、言語環境プログ

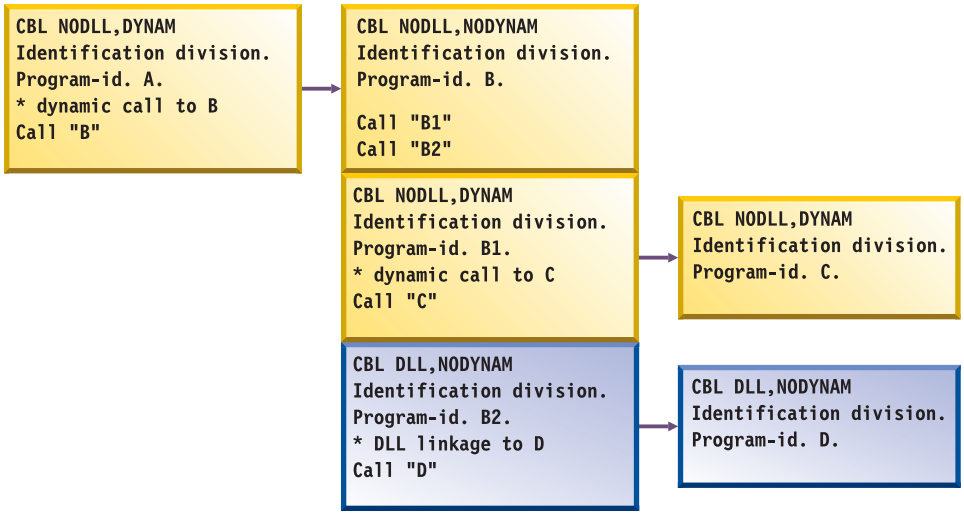
ラム・エンクレーブ内で DLL リンケージと COBOL 動的呼び出しリンケージの両方を使用することができます。

プログラム A	プログラム B	コンパイル・オプション
動的呼び出しを含む	動的呼び出しのターゲット	NODLL
DLL リンケージを使用する	ターゲット・プログラムまたはメソッドを含む	DLL

重要: このサポートは、言語環境プログラム V2R9 およびそれ以降のリリースでのみ使用可能です。

プログラムに、別個にコンパイルされたプログラムに対する CALL ステートメントが含まれている場合、一方のプログラムを DLL コンパイラー・オプションでコンパイルし、他方を NODLL でコンパイルした場合は、CALL がサポートされるのは、2 つのプログラムを一緒に同じモジュールでバインドした場合だけです。

次の図では、別個にバインドされたモジュールをいくつか示します。これらのモジュールには、動的呼び出しと DLL リンケージが混在しています。



DLL リンケージを使用して呼び出されるプログラムを取り消すことはできません。

DLL アプリケーションのコンポーネントはすべて同じ AMODE でなければなりません。COBOL 動的呼び出しによって通常提供される自動 AMODE 切り替えは、DLL リンケージでは利用できません。

DLL でのプロシージャー・ポインターまたは関数ポインターの使用

DLL と非 DLL が混在する実行単位では、プロシージャー・ポインターおよび関数ポインターは注意して使用してください。NODLL オプションを使用してコンパイルされたプログラム内で SET *procedure-pointer-1* TO ENTRY *entryname* ステートメント、または SET *function-pointer-1* TO ENTRY *entryname* ステートメントを使用するときは、DLL オプションを使用してコンパイルされたプログラムへのポインタ

ーを渡してはなりません。しかし、このステートメントを DLL オプションでコンパイルされたプログラムで使用するときは、別個にバインドされた DLL モジュールに含まれているプログラムへのポインターを渡すことができます。

NODYNAM および DLL オプションでコンパイルするとき、*entryname* が ID である場合、その ID 値は、DLL モジュールからエクスポートされる入り口点名を参照しなければなりません。DLL モジュール名は、エクスポートされる入り口点の名前と一致していなければなりません。この場合、以下のことも考慮する必要があります。

- ID に含まれているプログラム名は、CALL ステートメントを含むプログラムの PGMNAME(COMPAT|LONGUPPER|LONGMIXED) コンパイラー・オプションの設定に従って解釈されます。
- ターゲット DLL からエクスポートされるプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。
- HFS におけるターゲット DLL の探索には、大文字小文字の区別があります。
- ターゲット DLL が PDS または PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDS または PDSE メンバーとして DLL を探索するために、名前は自動的に大文字に変換されます。

非 DLL からの DLL の呼び出し

NODLL オプションを使用してコンパイルされた COBOL プログラムから DLL を呼び出すことができますが、制限があります。以下の方法を使用して、DLL リンケージに確実に従うようにすることができます。

- COBOL の非 DLL プログラムから呼び出したい COBOL DLL プログラムを、メインプログラムを含むロード・モジュールに入れます。COBOL の非 DLL プログラムから静的呼び出しを使用して、COBOL DLL プログラムを呼び出します。

  メインプログラムを含むロード・プログラムに含まれる COBOL DLL プログラムからは、他の DLL 内の COBOL DLL プログラムを呼び出せます。
- COBOL DLL プログラムを DLL に入れ、それらを COBOL 非 DLL プログラムから CALL *function-pointer* を使用して呼び出します (*function-pointer* は、呼び出されるプログラムの関数記述子に設定されます)。DLL 内におけるプログラムの関数記述子のアドレスを入手するには、dllload および dllqueryfn を使用する C ルーチンを呼び出します。

478 ページの『例: 非 DLL からの DLL の呼び出し』

関連概念

469 ページの『ダイナミック・リンク・ライブラリー (DLL)』

関連タスク

474 ページの『DLL での CALL ID の使用』
452 ページの『プロシージャー・ポインターと関数ポインターの使用』
470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』

関連参照

324 ページの『DLL』
327 ページの『EXPORTALL』

例: 非 DLL からの DLL の呼び出し

以下のサンプル・コードでは、DLL に含まれていない COBOL プログラム (COBOL1) から、DLL 内の COBOL プログラム (DLL OOC05R の中のプログラム ooc05R) を呼び出す方法が示されています。

```
CBL NODYNAM
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'COBOL1'.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.
       01 DLL-INFO.
           03 DLL-LOADMOD-NAME PIC X(12).
           03 DLL-PROGRAM-NAME PIC X(160).
           03 DLL-PROGRAM-HANDLE FUNCTION-POINTER.
       77 DLL-RC               PIC S9(9) BINARY.
       77 DLL-STATUS           PIC X(1) VALUE 'N'.
           88 DLL-LOADED       VALUE 'Y'.
           88 DLL-NOT-LOADED   VALUE 'N'.

       PROCEDURE DIVISION.

           IF DLL-NOT-LOADED
           THEN
      *      Move the names in. They must be null terminated.
             MOVE Z'OOC05R' TO DLL-LOADMOD-NAME
             MOVE Z'ooc05r' TO DLL-PROGRAM-NAME

      *      Call the C routine to load the DLL and to get the
      *      function descriptor address.
             CALL 'A1CCDLGT' USING BY REFERENCE DLL-INFO
                                   BY REFERENCE DLL-RC
             IF DLL-RC = 0
             THEN
               SET DLL-LOADED TO TRUE
             ELSE
               DISPLAY 'A1CCLDGT failed with rc = '
                 DLL-RC
               MOVE 16 TO RETURN-CODE
               STOP RUN
             END-IF
           END-IF

      *    Use the function pointer on the call statement to call the
      *    program in the DLL.
      *    Call the program in the DLL.
           CALL DLL-PROGRAM-HANDLE

           GOBACK.


#include <stdio.h>
#include <dll.h>
#pragma linkage (A1CCDLGT,COBOL)

typedef struct dll_lm {
  char         dll_loadmod_name[(12]);
  char         dll_func_name[(160]);
  void         (*fptr) (void); /* function pointer */
  } dll_lm;

void A1CCDLGT (dll_lm *dll, int *rc)
```

```
{
  dllhandle *handle;
  void (*fptr1)(void);
  *rc = 0;
   /* Load the DLL                                    */
  handle = dllload(dll->dll_loadmod_name);
  if (handle == NULL) {
     perror("A1CCDLGT failed on call to load DLL./n");
     *rc = 1;
     return;
  }

  /* Get the address of the function              */
  fptr1 = (void (*)(void))
          dllqueryfn(handle,dll->dll_func_name);
  if (fptr1 == NULL) {
     perror("A1CCDLGT failed on retrieving function./n");
     *rc = 2;
     return;
  }
  /* Return the function pointer                  */
  dll->fptr = fptr1;
  return;
}
```

C/C++ プログラムでの COBOL DLL の使用

DLL の COBOL サポートは、COBOL EXTERNAL データを除いて、z/OS C/C++ 製品の DLL サポートと相互協調処理します。EXTERNAL 属性で宣言された COBOL データ項目は、DLL サポートとは無関係です。これらのデータ項目は、プログラムが DLL 内にあるかどうかに関係なく、それらを宣言する実行単位内の任意の COBOL プログラムから名前によってアクセスすることができます。

具体的には、COBOL アプリケーションは、C/C++ DLL からエクスポートされた関数を呼び出すことができます。同様に、C/C++ アプリケーションは、COBOL DLL からエクスポートされた COBOL プログラムを呼び出すことができます。

COBOL オプション DLL、RENT、および EXPORTALL は、C/C++ における DLL、RENT、および EXPORTALL オプションとほとんど同じように機能します。しかし、C/C++ コンパイラーは、デフォルトで DLL 可能コードを作成します。DLL オプションは C にしか適用されません。

C/C++ DLL 関数ポインターを COBOL に渡し、C/C++ 関数ポインターを COBOL 関数ポインター・データ項目として受け取り、それを COBOL 内で使用することができます。

次の例は、サービスへの関数ポインターを戻す C 関数への COBOL 呼び出しと、そのサービスへの COBOL 呼び出しを示しています。

```
Identification Division.
Program-id. Demo.
Data Division.
Working-Storage section.
01  fp usage function-pointer.
Procedure Division.
   Call "c-function" returning fp.
   Call fp.
```

関連タスク
470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』

関連参照
324 ページの『DLL』
327 ページの『EXPORTALL』
343 ページの『RENT』
EXTERNAL (*Enterprise COBOL 言語解説書*)

OO COBOL アプリケーションでの DLL の使用

DLL、THREAD、RENT、および DBCS コンパイラー・オプションを使用して、各クラス定義をコンパイルしてから、RENT バインダー・オプションを使用して、それを別個の DLL モジュールにリンク・エディットしなければなりません。

関連タスク
470 ページの『DLL を作成するためのプログラムのコンパイル』
471 ページの『DLL のリンク』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

関連参照
324 ページの『DLL』
352 ページの『THREAD』
343 ページの『RENT』
323 ページの『DBCS』

第 27 章 マルチスレッド化のための COBOL プログラムの準備

z/OS 環境では、バッチ、TSO、IMS、または UNIX 環境のプロセス内で、COBOL プログラムを複数のスレッドで実行することができます。マルチスレッド化を実行するための明示的な COBOL 言語は存在しません。したがって、THREAD コンパイラー・オプションでコンパイルします。

COBOL はプログラム・スレッドの管理を直接サポートしません。ただし、マルチスレッド化アプリケーション・サーバー、C/C++ ドライバー・プログラムを使用してスレッドを作成するアプリケーション、Java と相互協調処理して Java スレッドを使用するプログラム、および PL/I タスキングを使用するアプリケーションにおいて、THREAD コンパイラー・オプションでコンパイルする COBOL プログラムを実行することができます。すなわち、他のプログラムは COBOL プログラムを呼び出して、COBOL プログラムをプロセス内の複数のスレッドで、またはスレッド内の複数のプログラム起動インスタンスとして、実行させることができます。スレッド化したアプリケーションは、単一の言語環境プログラムのエンクレーブ内で実行しなければなりません。

LOCAL-STORAGE または WORKING-STORAGE の選択: マルチスレッド化プログラムは再帰的プログラムとしてコーディングしなければならないので、データの永続性は、再帰的プログラムの永続性となります。

- LOCAL-STORAGE SECTION 内のデータ項目は、プログラム起動のインスタンスごとに自動的に割り振られます。あるプログラムが複数のスレッドにおいて同時に実行されると、起動ごとに個別の LOCAL-STORAGE データのコピーを使用します。
- WORKING-STORAGE SECTION 内のデータ項目は、プログラムごとに一度割り振られるため、最後に使われた状態がプログラムのすべての起動において使用可能です。

個々のプログラム起動インスタンスに分離するデータの場合、そのデータを LOCAL-STORAGE SECTION で定義します。一般的に、この選択はスレッド化プログラム内の作業データに適しています。データを WORKING-STORAGE に宣言し、プログラムがそのデータの内容を変更する場合には、以下のアクションのどちらかを実行する必要があります。

- WORKING-STORAGE 内のデータに複数のスレッドから同時にアクセスしないようにアプリケーションを構成します。
- データに別々のスレッドから同時にアクセスする場合には、適切な直列化コードを書き込みます。

関連概念

482 ページの『マルチスレッド化』

関連タスク

483 ページの『マルチスレッド化サポートのための THREAD の選択』
484 ページの『マルチスレッド化による制御権移動』
485 ページの『マルチスレッド化によるファイルの処理』
487 ページの『マルチスレッド化による COBOL 制限の処理』

© Copyright IBM Corp. 1991, 2001

関連参照
352 ページの『THREAD』
PROGRAM-ID 段落 (*Enterprise COBOL 言語解説書*)

マルチスレッド化

マルチスレッド化の COBOL サポートを使用するには、プロセス、スレッド、実行単位、プログラム起動インスタンスの相互関係を理解する必要があります。

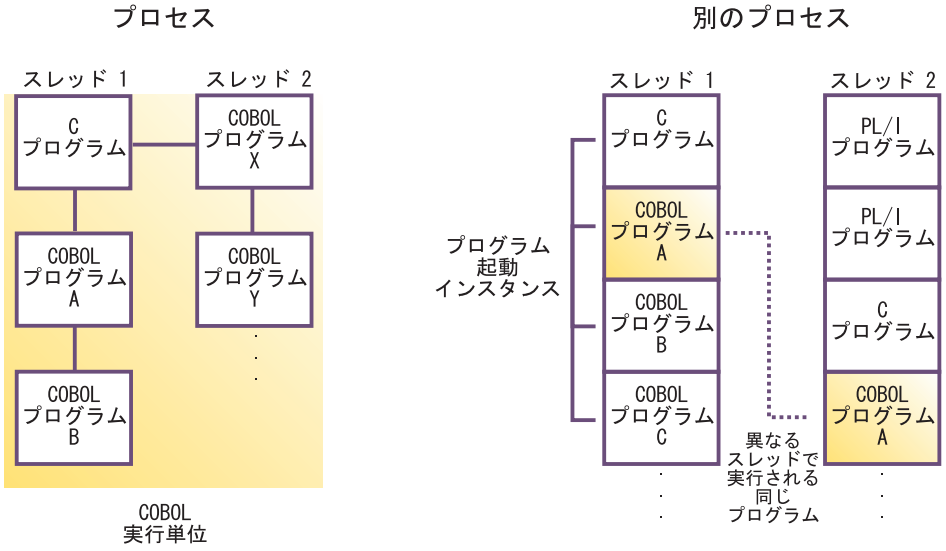
オペレーティング・システムおよびマルチスレッド化アプリケーションは、*プロセス* 内の実行フローを処理することができます。実行フローとは、プログラムのすべて、または一部が実行時に発生する一連のイベントです。1 つのプロセス内で実行されるプログラムはリソースを共用することができます。プロセスは操作することができます。たとえば、システムがプロセスの実行に使う時間において、プロセスに高い、または低い優先順位を持たせることができます。

プロセス内で、アプリケーションは 1 つまたは複数の*スレッド* を開始することができます。スレッドはそれぞれ、そのスレッドを制御するコンピューター命令のストリームです。マルチスレッド化プロセスは、1 つの命令ストリーム (スレッド) で始まり、タスクを実行する他の命令ストリームを後で作成することができます。これらの複数のスレッドは並行して実行することができます。スレッド内で、制御は実行プログラムの間で転送されます。

マルチスレッド化環境で、COBOL *実行単位* は、実行中のアクティブな COBOL プログラムが含まれているスレッドを含むプロセスの部分です。COBOL 実行単位は、スレッドのいずれかの実行スタックでアクティブな COBOL プログラムがなくなるまで継続します。たとえば、呼び出される COBOL プログラムには GOBACK ステートメントが含まれていて、C プログラムに制御を戻します。実行単位内では、COBOL プログラムは非 COBOL プログラムを呼び出すことができ、逆に、非 COBOL プログラムは COBOL プログラムを呼び出すことができます。

スレッド内で、制御は別々の COBOL および 非 COBOL プログラムの間で転送されます。たとえば、COBOL プログラムは別の COBOL プログラムまたは C プログラムを呼び出すことができます。別々に呼び出されたプログラムはそれぞれ、*プログラム起動インスタンス* です。特定プログラムのプログラム起動インスタンスは、指定したプロセス内で複数のスレッドに存在することができます。

次の図に、プロセス、スレッド、実行単位、およびプログラム起動インスタンスの間の関係を示します。



関連概念
プログラム管理モデル (*言語環境プログラム プログラミング・ガイド*)
スレッド (*言語環境プログラム プログラミング・ガイド*)

関連タスク
『マルチスレッド化サポートのための THREAD の選択』
484 ページの『マルチスレッド化による制御権移動』
485 ページの『マルチスレッド化によるファイルの処理』
487 ページの『マルチスレッド化による COBOL 制限の処理』

関連参照
352 ページの『THREAD』

マルチスレッド化サポートのための THREAD の選択

マルチスレッド化をサポートするために、THREAD コンパイラー・オプションを選択します。プログラムが 1 つのアプリケーションによって単一プロセスの複数のスレッドで呼び出される場合には、THREAD を選択します。THREAD でコンパイルすると、スレッド化サポートのために COBOL プログラムが作成されます。ただし、THREAD でコンパイルすると、シリアライゼーション・ロジックが自動的に生成されるため、プログラムのパフォーマンスが低下する可能性があります。

COBOL プログラムを複数のスレッドで実行するには、THREAD オプションを使用して、実行単位内の COBOL プログラムのすべてをコンパイルしなければなりません。また、RENT オプションを使用して COBOL プログラムをコンパイルし、それらをバインダーまたはリンケージ・エディターの RENT オプションとリンクする必要があります。

言語制限: THREAD オプションを使用する場合には、THREAD オプションの説明資料などに記述されているように、ある特定の言語エレメントを使用することができません。

再帰: THREAD コンパイラー・オプションを使用してプログラムをコンパイルするには、PROGRAM-ID 段落に RECURSIVE 句を指定する必要があります。指定しない場合には、エラーが発生します。

関連タスク
20 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』

関連参照
352 ページの『THREAD』

マルチスレッド化による制御権移動

マルチスレッド化環境用に COBOL プログラムを書くときは、適切なプログラム・リンケージおよび終了ステートメントを選択します。

スレッド化プログラムでの取り消しの使用

単一スレッド環境の場合のように、実行単位内で最初に呼び出されるとき、および呼び出されるプログラムに対する CANCEL 後に最初に呼び出されるときは、呼び出されるプログラムは初期状態にあります。CANCEL ステートメントで名前を付けるプログラムがどのスレッドにおいてもアクティブでないことを確認してください。アクティブ・プログラムを取り消そうとした場合には、重大度 3 の言語環境プログラム条件が発生します。

プログラムの終了

プログラムの呼び出し側に戻るには、GOBACK を使用します。あるスレッド内の最初のプログラムから GOBACK を使用するとき、そのスレッドは終了します。そのスレッドがあるエンクレーブ内の初期スレッドである場合には、そのエンクレーブは終了します。

メインプログラムからの場合を除き、GOBACK と同様に、EXIT PROGRAM を使用します。メインプログラムでは EXIT PROGRAM は無効です。

言語環境プログラムのエンクレーブ全体を終了し、メインプログラム (オペレーティング・システムであると考えられます) の呼び出し側に制御を戻すには、STOP RUN を使用します。エンクレーブ内で実行中のすべてのスレッドが終了します。

COBOL 環境の事前初期設定

スレッド化アプリケーションに事前初期設定が必要である場合には、言語環境プログラム・サービス (CEEPIPI インターフェース) を使用します。事前初期設定のための COBOL 固有のインターフェース (ランタイム・オプション RTEREUS と、関数 IGZERRE および ILBOSTP0) を使用して、THREAD オプションでコンパイルしたプログラムから再使用可能環境を設定することはできません。

関連概念
エンクレーブ終了 (*言語環境プログラム プログラミング・ガイド*)

関連タスク
439 ページの『メインプログラムまたはサブプログラムの終了と再入』

マルチスレッド化によるファイルの処理

スレッド化アプリケーションでは、QSAM、VSAM、および行順次ファイルでの入出力のために COBOL ステートメントをコーディングすることができます。各ファイル定義 (FD) には暗黙のシリアライゼーション・ロックがあります。このロックは、以下のステートメントの実行に関連付けられた入力または出力操作の間に、自動シリアライゼーション・ロジックとともに使用されます。

- OPEN
- CLOSE
- READ
- WRITE
- REWRITE
- START
- DELETE

また、自動シリアライゼーションは、以下のステートメントに関連付けられた暗黙の MOVE の場合にも発生します。

```
WRITE recordname FROM identifier
READ file-name INTO identifier
```

自動シリアライゼーションは、以下の条件付き句内で指定されるステートメントには適用されません。

- AT END
- NOT AT END
- INVALID KEY
- NOT INVALID KEY
- AT END-OF-PAGE
- NOT AT END-OF-PAGE

ファイル定義ストレージ

すべてのプログラム起動の際に、ファイル定義に関連付けられたストレージ (FD レコードや SAME RECORD AREA 文節に関連付けられたレコード域) は割り振られ、最後に使われた状態で使用可能です。実行のすべてのスレッドはこのストレージを共用します。OPEN、CLOSE、READ、WRITE、REWRITE、START、および DELETE ステートメントの実行中は、このストレージの自動シリアライゼーションに依存することができますが、これらのステートメントの使用の間は依存することはできません。

ファイル・アクセスの場合の推奨使用法

スレッド化プログラムで使用する場合には、以下のファイル編成をお勧めします。

- 順次編成
- 行順次編成
- 順次アクセスによる相対編成
- 順次アクセスによる索引編成

自動シリアライゼーションの利点を最大に活用し、固有のシリアライゼーション・ロジックの明示的書き込みを避けるには、スレッド化プログラム内のファイルにアクセスするときに、推奨されるファイル編成のいずれか 1 つを使用し、以下の使用パターンを使用します。

推奨される使用パターン (入力用)

```
  OPEN INPUT fn
  . . .
  READ fn INTO local-storage item
  . . .
* Process the record from the local-storage item
  . . .
  CLOSE fn
```

推奨される使用パターン (出力用)

```
  OPEN OUTPUT fn
  . . .
* Construct output record in local-storage item
  . . .
  WRITE rec FROM local-storage item
  . . .
  CLOSE fn
```

他の使用パターンの場合には、以下のアクションのいずれかを実行します。

- アプリケーション・ロジックの安全を検証します。プログラムの 2 つのインスタンスが別々のスレッドで決して同時にアクティブにならないことを確認します。
- POSIX サービスに対する呼び出しを使用して、明示的シリアライゼーション・ロジックをコーディングします。

複数のスレッドからファイルにアクセスする際にシリアライゼーション問題の発生を避けるには、LOCAL-STORAGE SECTION で、ファイルに関連付けられたデータ項目 (ファイル状況データ項目やキー引き数) を定義します。

『例: マルチスレッド化によるファイル入出力の使用パターン』

関連タスク

148 ページの『QSAM ファイルのクローズ』
183 ページの『VSAM ファイルのクローズ』
253 ページの『ERROR 宣言のコーディング』

例: マルチスレッド化によるファイル入出力の使用パターン

以下の例では、マルチスレッド化アプリケーションにおいてファイル入出力の推奨使用パターンから逸脱した際の明示的シリアライゼーション・ロジックの必要性について説明します。また、これらの例では、シリアライゼーションを適切に処理できない結果として発生する予期しない動作についても説明します。

それぞれの例において、サンプル操作を含む 1 つのプログラムの 2 つのインスタンスが 1 つの実行単位内で、2 つの異なるスレッド上で実行されています。

例 1

```
READ F1
. . .
REWRITE R1
```

2 番目のスレッドは、最初のスレッド上で READ ステートメントが実行された後に (ただし、最初のスレッド上で REWRITE ステートメントが実行される前に) READ ステートメントを実行すると考えられます。REWRITE ステートメントは、意図したレコードの更新は行わない可能性があります。目的の結果が確実に出るようにするには、明示的シリアライゼーション・ロジックを書き込みます。

例 2

```
 READ F1
  . . .
* Process the data in the FD record description entry for F1
  . . .
```

2 番目のスレッドは、最初のスレッドが FD レコード記述項目内のレコードをまだ処理中に READ ステートメントを実行すると考えられます。2 番目の READ ステートメントは、最初のスレッドがまだ処理中のレコードをオーバーレイする可能性があります。この問題を回避するには、推奨の手法である READ F1 INTO *local-storage-item* を使用します。

その他のケース: 後に READ NEXT が続く START や、後に DELETE が続く READ など、一連の関連した入出力操作を必要とするその他の使用パターンについても同様の考慮を払わなければなりません。適切な手順を実行して、ファイル入出力の正しい処理を確実に行います。

マルチスレッド化による COBOL 制限の処理

一部の COBOL アプリケーションは、サブシステムまたは他のアプリケーションに依存します。マルチスレッド化環境において、これらの依存性などに起因して、COBOL プログラムに対するいくつかの制限が発生します。

一般的に、実行単位内のアプリケーションに可視であるリソースへのアクセスを同期化する必要があります。この要件の例外には、DISPLAY および ACCEPT があります。これらは、複数のスレッドから、および推奨使用パターンを持つ、サポートされている COBOL ファイル入出力ステートメントから使用できます。すべての同期は、ランタイム環境によって提供されます。

CICS: マルチスレッド化アプリケーションは CICS 環境において実行することはできません。CICS 環境では、THREAD オプションを使用してコンパイルし、複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部である COBOL プログラムを実行できます。

再帰的: マルチスレッド化アプリケーション内のプログラムは再帰的プログラムとしてコーディングする必要があります。したがって、ネストされたプログラムをコーディングしないなど、再帰的プログラムに適用される制限やプログラミング上の考慮事項を忠実に守る必要があります。

再入可能性: マルチスレッド化プログラムは、RENT コンパイラー・オプションを使用してコンパイルし、それらをバインダーまたはリンケージ・エディターの RENT オプションとリンクしなければなりません。

POSIX および PL/I: マルチスレッド化アプリケーションで POSIX スレッドを使用する場合には、言語環境プログラム・ランタイム・オプション POSIX(ON) を指定し

なければなりません。アプリケーションが PL/I タスキングを使用する場合には、POSIX(OFF) を指定しなければなりません。POSIX スレッドと PL/I タスクを同一のアプリケーションに混在させることはできません。

AMODE: マルチスレッド化アプリケーションを AMODE 31 で実行しなければなりません。AMODE 24 を複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部として、THREAD オプションでコンパイルした COBOL プログラムを実行することができます。

非同期シグナル: スレッド化アプリケーションにおいて、COBOL プログラムは、非同期シグナルまたは割り込みによって割り込まれる場合があります。プログラムにそのような割り込みを容認できないロジックが含まれている場合には、そのロジックが継続する間はそうした割り込みを使用不可にする必要があります。C/C++ 関数を呼び出して、シグナル・マスクを適切に設定します。

古い COBOL プログラム: マルチスレッド化アプリケーションの複数のスレッド上で COBOL プログラムを実行するには、それらを Enterprise COBOL でコンパイルし、THREAD オプションを使用する必要があります。古いコンパイラーでコンパイルしたプログラムを実行する場合には、以下の制限を守る必要があります。

- OS/VS COBOL プログラムを初期スレッド (IPT) 上にのみ含むアプリケーションを実行すること。
- 他の古いコンパイラーによって、1 つのスレッド上のみでコンパイルしたプログラムを含むアプリケーションを実行すること (ただし、スレッドは初期スレッド以外のスレッドでも構いません)。

IGZBRDGE、IGZETUN、および IGZEOPT: 静的呼び出しを動的呼び出しに変換するためのマクロである IGZBRDGE を、THREAD オプションでコンパイルしたプログラムと併用してはなりません。このマクロはサポートされません。THREAD オプションでメインプログラムをコンパイルした、アプリケーションに対して、モジュール IGZETUN (ストレージ・チューニング用) または IGZEOPT (ランタイム・オプション用) を使用してはなりません。これらの CSECT は無視されます。

UPSI スイッチ: アプリケーション内のすべてのプログラムおよびすべてのスレッドは UPSI スイッチの単一コピーを共用します。スレッド化アプリケーションでスイッチを変更する場合には、適切なシリアライゼーション・ロジックをコーディングする必要があります。

関連タスク

451 ページの『再帰呼び出しの作成』
485 ページの『マルチスレッド化によるファイルの処理』

第 5 部 オブジェクト指向プログラムの開発

第 28 章 オブジェクト指向プログラムの作成 . . 491
例: 口座 492
サブクラス 493
クラスの定義 495
クラスを定義する CLASS-ID 段落 496
クラスを定義する REPOSITORY 段落 497
例: 外部クラス名と Java パッケージ . . . 498
クラス・インスタンス・データを定義する WORKING-STORAGE SECTION 498
例: クラスの定義 499
クラス・インスタンス・メソッドの定義 500
クラス・インスタンス・メソッドを定義する METHOD-ID 段落 501
クラス・インスタンス・メソッドを定義する INPUT-OUTPUT SECTION 501
クラス・インスタンス・メソッドを定義する DATA DIVISION 502
クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION 503
渡された引き数を取得するための USING 句 503
値を戻すための RETURNING 句 504
インスタンス・メソッドのオーバーライド. . . 504
インスタンス・メソッドの多重定義 505
属性 (get および set) メソッドのコーディング 506
例: get メソッドのコーディング 507
例: メソッドの定義 507
Account クラス. 507
Check クラス 508
クライアントの定義 509
クライアントを定義する REPOSITORY 段落 510
クライアントを定義する DATA DIVISION . . 511
LOCAL-STORAGE または WORKING-STORAGE の選択 512
オブジェクト・リファレンスの比較と設定. . . 513
メソッドの呼び出し (INVOKE) 514
引き数を渡すための USING 句 515
戻り値を取得するための RETURNING 句 515
オーバーライドしたスーパークラス・メソッドの呼び出し 516
クラスのインスタンスの生成と初期化 516
Java クラスのインスタンスの生成 517
COBOL クラスのインスタンスの生成 . . . 517
クラスのインスタンスの解放 518
例: クライアントの定義 519
サブクラスの定義 520
サブクラスを定義する CLASS-ID 段落 521
サブクラスを定義する REPOSITORY 段落 . . 521
サブクラス・インスタンス・データを定義する WORKING-STORAGE SECTION 522
サブクラス・インスタンス・メソッドの定義 . . 522
例: サブクラスの定義 (メソッドを含む) . . . 523
CheckingAccount クラス (Account のサブクラス) 523
ファクトリー・セクションの定義. 524
ファクトリー・データを定義する WORKING-STORAGE SECTION 525
ファクトリー・メソッドの定義 525
ファクトリー・メソッドまたは静的メソッドの隠蔽. 527
ファクトリーまたは静的メソッドの呼び出し 527
プロシージャー指向 COBOL プログラムのラッピング. 528
例: ファクトリーの定義 (メソッドを含む). . . 528
Account クラス. 529
CheckingAccount クラス (Account のサブクラス) 531
Check クラス 532
TestAccounts クライアント・プログラム . . 533
TestAccounts クライアント・プログラムによって生成される出力 534

第 29 章 Java メソッドとの通信 535
JNI サービスへのアクセス 535
Java 例外の取り扱い 537
COBOL プログラムからの例外のスロー . . 537
Java 例外のキャッチ 537
例: Java 例外の取り扱い. 537
ローカルおよびグローバル参照の管理 538
ローカル参照の削除、保管、解放. 538
JNI サービス 539
Java アクセス制御. 539
Java とのデータ共用 540
COBOL および Java での相互運用可能データ型のコーディング. 540
Java の配列とストリングの宣言 541
Java 配列の取り扱い 542
例: Java int 配列の処理 543
Java ストリングの取り扱い. 543
Unicode サービス 543
EBCDIC サービス. 544
UTF-8 サービス 546
例: COBOL で書き込む J2EE クライアント . . . 546

© Copyright IBM Corp. 1991, 2001

第 28 章 オブジェクト指向プログラムの作成

オブジェクト指向 (OO) プログラムを作成するときには、必要なクラスおよびクラスが作業を行うために必要なメソッドやデータを決定しなければなりません。

OO プログラムは、オブジェクト (状態と動作をカプセル化するエンティティー) とそのクラス、メソッド、およびデータに基づいています。 クラス とは、オブジェクトの状態および機能を定義するテンプレートです。 通常、プログラムは、クラスの複数のオブジェクト・インスタンス (または、単に、インスタンス)、すなわち、そのクラスのメンバーであるオブジェクトを複数作成して、それらを操作します。 各インスタンスの状態は、インスタンス・データ とも呼ばれるデータに保管されます。 また、各インスタンスの機能はインスタンス・メソッド と呼ばれます。 クラスは、クラスのすべてのインスタンスが共用するデータ (ファクトリー または*静的* データとも呼ばれる)、およびオブジェクト・インスタンスとは別個にサポートされるメソッド (ファクトリー または*静的* メソッドとも呼ばれる) を定義することができます。

Enterprise COBOL を使用して、以下のことを行うことができます。

- メソッドとデータを COBOL でインプリメントした状態で、クラスを定義する。
- Java および COBOL クラスのインスタンスを作成する。
- Java および COBOL オブジェクトにメソッドを呼び出す。
- Java クラスまたは他の COBOL クラスから継承するクラスを書き込む。
- 多重定義メソッドを定義して呼び出す。

Enterprise COBOL プログラムで、Java Native Interface (JNI) が提供するサービスを呼び出して、COBOL 言語で直接使用可能な基本 OO 機能に加えて、Java 指向機能を取得できます。

Enterprise COBOL クラスでは、CALL ステートメントをコーディングして、プロシージャー型 COBOL プログラムとインターフェースを取ることができます。したがって、COBOL クラス定義構文は、プロシージャー型 COBOL ロジックのラッパー・クラスを書き込むために特に役立ち、Java から既存の COBOL コードにアクセスすることを可能にします。

Java コードは、COBOL クラスのインスタンスを作成したり、これらのクラスのメソッドを呼び出したり、COBOL クラスを拡張したりすることができます。

OO COBOL プログラムや Java プログラムの開発や実行は、UNIX システム・サービス環境で行います。

492 ページの『例: 口座』

関連タスク

495 ページの『クラスの定義』
500 ページの『クラス・インスタンス・メソッドの定義』
509 ページの『クライアントの定義』

© Copyright IBM Corp. 1991, 2001

520 ページの『サブクラスの定義』
524 ページの『ファクトリー・セクションの定義』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
IBM COBOL ソース・プログラムのアップグレード (*Enterprise COBOL コンパイラーおよび実行時プログラム 移行ガイド*)

関連参照
Java 言語仕様

例: 口座

銀行にカスタマーが口座を開設し、その口座で預金や引き出しを行うときの例を見てみましょう。口座は、Account という名前の汎用クラスで表します。多数のカスタマーが存在します。したがって、Account クラスの複数インスタンスが同時に存在すると考えることができます。

必要とするクラスを判別したら、次のステップでは、それらのクラスがそれぞれの作業を実行するために必要なメソッドを判別します。たとえば、Account クラスは以下のサービスを提供する必要があります。

- 口座を開設する。
- 現在の収支を取る。
- 口座に預金する。
- 口座から預金を引き出す。
- 口座状況をレポートする。

Account クラスの以下のメソッドは、上記の要件を満たします。

init 口座を開設し、それに口座番号を割り当てます。

getBalance
口座の現在の収支を戻します。

貸方 指定の金額を口座に預金します。

借方 指定の金額を口座から引き出します。

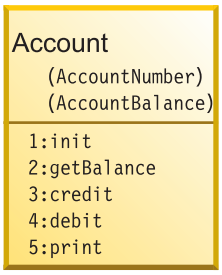
印刷 口座番号と勘定残高を表示します。

Account クラスとそのメソッドを設計すると、クラスはいくつかのインスタンス・データを保持する必要があることがわかります。一般に、Account オブジェクトには次のようなインスタンス・データが必要です。

- 口座番号
- 勘定残高
- カスタマー情報: 名前、住所、自宅の電話番号、勤務先の電話番号、社会保障番号など

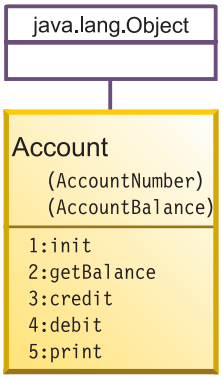
ただし、上記の例を単純化するため、口座番号と勘定残高は、Account クラスが必要とする唯一のインスタンス・データであると想定します。

クラスやメソッドを設計するとき、ダイアグラムは役に立ちます。次のダイアグラムで、Account クラスの設計における最初の試みを示します。



ダイアグラムにおいて、括弧内のワードはインスタンス・データの名前です。番号とコロンに続くワードは、インスタンス・メソッドの名前です。

次の構成図はクラスの相関図で、*継承の階層* とも呼ばれます。Account クラスはクラス java.lang.Object から直接継承します。



サブクラス

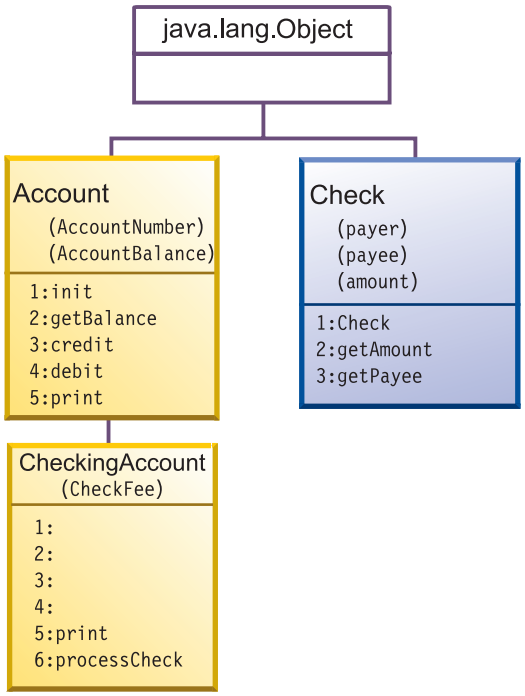
上記の口座の例において、Account は汎用クラスです。ただし、銀行は、当座預金、普通預金、モーゲージ・ローンなど、多くの種類の口座を用意することができます。これらの口座のすべてには、口座の一般的特性がある一方で、すべての種類の口座に必ずしも共有されない追加特性が含まれている場合があります。

たとえば、CheckingAccount クラスには、すべての口座が持つ口座番号や勘定残高に加えて、口座に書き込まれる各当座に適用される当座手数料がある場合があります。また、CheckingAccount クラスには、当座を処理する (すなわち、金額を読み取る、支払人の借方に記入する、受取人の貸方に記入するなど) メソッドも必要です。したがって、CheckingAccount を Account のサブクラスとして定義し、そのサブクラスが必要とする追加のインスタンス・データやインスタンス・メソッドをそのサブクラスで定義することは理にかなっています。

CheckingAccount クラスを設定すると、当座をモデル化するクラスの必要性があることに気が付きます。クラス Check のインスタンスは、少なくても、支払人、受取人、および当座の金額のインスタンス・データを必要とします。

実際の OO アカウント・システムでは多くの追加クラス (ならびにデータベースおよびトランザクション処理ロジック) を設計する必要があるでしょうが、ここでは例を単純化するために省略されています。

継承更新ダイアグラムを以下に示します。



番号とコロンの後にメソッド名が記されていないものは、その番号のメソッドがスーパークラスから継承されていることを示します。

多重継承: OO COBOL アプリケーションでは*多重継承* を使用することはできません。定義するすべてのクラスは、厳密に 1 つの親を持つ必要があります。java.lang.Object は、すべての継承の階層のルートになければなりません。したがって、OO COBOL アプリケーション内で定義されるオブジェクト指向システムのクラス構造はツリー状です。

507 ページの『例: メソッドの定義』

関連タスク

495 ページの『クラスの定義』
500 ページの『クラス・インスタンス・メソッドの定義』
520 ページの『サブクラスの定義』

クラスの定義

COBOL クラス定義は、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION から構成され、その後にファクトリー定義 (オプショナル) とオブジェクト定義 (オプショナル) が続き、最後に、END CLASS マーカーが続きます。

セクション	目的	構文
IDENTIFICATION DIVISION (必須)	クラスの名前。継承情報を提供する。	496 ページの『クラスを定義する CLASS-ID 段落』 (必須) AUTHOR 段落 (オプショナル) INSTALLATION 段落 (オプショナル) DATE-WRITTEN 段落 (オプショナル) DATE-COMPILED 段落 (オプショナル)
ENVIRONMENT DIVISION (必須)	コンピューター環境を記述する。クラス定義内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 497 ページの『クラスを定義する REPOSITORY 段落』 (必須) SOURCE-COMPUTER 段落 (オプショナル) OBJECT-COMPUTER 段落 (オプショナル) SPECIAL-NAMES 段落 (オプショナル)

ファクトリー定義 (オプショナル)	クラスのすべてのインスタンスが共有するデータとオブジェクト・インスタンスとは別々にサポートされるメソッドを定義する。	`IDENTIFICATION DIVISION.`
		`FACTORY.`
		`  DATA DIVISION.`
		`  WORKING-STORAGE SECTION.`
		`*   (Factory data here)`
		`  PROCEDURE DIVISION.`
		`*   (Factory methods here)`
		`END FACTORY.`

オブジェクト定義 (オプショナル)	インスタンス・データとインスタンス・メソッドを定義する。	`IDENTIFICATION DIVISION.`
		`OBJECT.`
		`  DATA DIVISION.`
		`  WORKING-STORAGE SECTION.`
		`*   (Instance data here)`
		`  PROCEDURE DIVISION.`
		`*   (Instance methods here)`
		`END OBJECT.`

SOURCE-COMPUTER、OBJECT-COMPUTER、または SPECIAL-NAMES 段落をクラス CONFIGURATION SECTION に指定すると、それらの段落は、そのクラスが導入するすべてのメソッドを含む、クラス定義全体に適用されます。

クラス CONFIGURATION SECTION は、プログラム CONFIGURATION SECTION と同じ記入項目で構成されています。ただし、クラス CONFIGURATION SECTION には INPUT-OUTPUT SECTION を含めることはできません。INPUT-OUTPUT SECTION の定義は、クラス・レベルでその定義を行うのではなく、それを必要とする個々のメソッドにおいてのみ行います。

上記で説明したように、インスタンス・データとメソッドの定義は、そのクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION において、行います。個別のオブジェクト・インスタンスにではなく、クラス自体に関連付けるデータとメソッドを必要とするクラスに、クラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

各 COBOL クラス定義は、別々のソース・ファイルになければなりません。

499 ページの『例: クラスの定義』

関連タスク

498 ページの『クラス・インスタンス・データを定義する WORKING-STORAGE SECTION』
500 ページの『クラス・インスタンス・メソッドの定義』
520 ページの『サブクラスの定義』
524 ページの『ファクトリー・セクションの定義』
9 ページの『コンピューター環境の記述』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

関連参照

COBOL クラス定義構造 (*Enterprise COBOL 言語解説書*)

クラスを定義する CLASS-ID 段落

IDENTIFICATION DIVISION 内の CLASS-ID 段落を使用して、クラスに名前を付け、それに関する継承情報を入力します。たとえば、次のようになります。

```
Identification Division.                    Required
Class-id. Account inherits Base.  Required
```

CLASS-ID 段落を使用して、以下のクラスを識別します。

- 定義中のクラス (上記の例の Account)。
- 定義中のクラスがその特性を継承する元の即時スーパークラス。スーパークラスは、Java または COBOL でインプリメントされます。

  上記の例において、inherits Base では、Account クラスは、クラス定義内で Base として認識されているクラスからメソッドとデータを継承することを示します。オブジェクト指向 COBOL プログラムにおいて、名前 Base は java.lang.Object を参照するために使用することをお勧めします。

クラス名は、1 バイト文字を使用し、COBOL ユーザー定義語の標準形式化規則に準拠しなければなりません。

ENVIRONMENT DIVISION の CONFIGURATION SECTION で REPOSITORY 段落を使用して、スーパークラス名 (上記の例の Base) を外部に判明しているスーパークラス名 (Base の場合の java.lang.Object) に関連付けます。また、オプションとして、定義中のクラスの名前 (上記の例の Account) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

すべてのクラスを java.lang.Object クラスから直接的または間接的に入手しなければなりません。

関連タスク

497 ページの『クラスを定義する REPOSITORY 段落』

関連参照

CLASS-ID 段落 (*Enterprise COBOL 言語解説書*)
ユーザー定義語 (*Enterprise COBOL 言語解説書*)

クラスを定義する REPOSITORY 段落

REPOSITORY 段落を使用して、クラス定義内で指定のワードを用いるときにそれらのワードがクラス名であることをコンパイラーに宣言し、また、オプションとして、そのクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関連付けます。外部クラス名は大文字小文字を区別します。したがって、Java 形式化規則に準拠しなければなりません。たとえば、Account クラス定義において、以下のようにコーディングします。

```
Environment Division.                  Required
Configuration Section.                 Required
Repository.                            Required
    Class Base is "java.lang.Object"   Required
    Class Account is "Account".        Optional
```

REPOSITORY 段落記入項目では、クラス定義内で Base および Account として参照されるクラスの外部クラス名は、それぞれ、java.lang.Object および Account であることが示されます。

REPOSITORY 段落では、クラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。たとえば、次のようになります。

- ベース
- 定義中のクラスが継承する元のスーパークラス
- クラス定義内のメソッドで参照するクラス

REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。

Java パッケージ の一部である参照クラスごとに外部クラス名を指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。たとえば、Object クラスは java.lang パッケージの一部です。したがって、上記に示したように、その外部名を java.lang.Object として指定します。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形式化規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名は以下の方法でクラス名から作成されます。

- クラス名は大文字に変換されます。
- 各ハイフンはゼロに変更されます。
- 数字の場合、最初の文字は以下のように変更されます。
  - 1 ～ 9 は A ～ I に変更されます。
  - 0 は J に変更されます。

外部名は英大文字小文字混合で名前付けされます。したがって、上記の例で、クラス Account は外部的には Account (英大文字小文字混合) として認識されます。

オプションとして、定義中のクラス (上記の例の Account) の記入項目を REPOSITORY 段落に組み込むことができます。定義中のクラスの記入項目を組み込ん

で、非 COBOL 文字を含む外部クラス名を指定するか、または、クラスが Java パッケージの一部となる場合には、完全パッケージ修飾クラス名を指定する必要があります。

『例: 外部クラス名と Java パッケージ』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)
ID (*Java 言語仕様*)
パッケージ (*Java 言語仕様*)

例: 外部クラス名と Java パッケージ

次の例では、REPOSITORY 段落内の記入項目から外部クラス名を決定する方法を説明します。

```
Environment division.
Configuration section.
Repository.
    Class Employee is "com.acme.Employee"
    Class JavaException is "java.lang.Exception"
    Class Orders.
```

次の表には、ローカル・クラス名 (クラス定義内で使用されるクラス名)、そのクラスを含む Java パッケージ、および関連付けられた外部クラス名が記述されています。

ローカル・クラス名	Java パッケージ	外部クラス名
Employee	com.acme	com.acme.Employee
JavaException	java.lang	java.lang.Exception
Orders	(名前付けなし)	ORDERS

外部クラス名 (REPOSITORY 段落記入項目内のクラス名の後の名前とオプションの IS) は、パッケージ (ある場合) の完全修飾名から構成され、後にピリオドが付き、続いてその後にクラスの単純名が付きます。

関連タスク
497 ページの『クラスを定義する REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・データを定義する WORKING-STORAGE SECTION

OBJECT 段落の DATA DIVISION 内の WORKING-STORAGE SECTION を使用して、COBOL クラスが必要とするインスタンス・データ (すなわち、クラスのそれぞれのインスタンスごとに割り振られるデータ) を記述します。IDENTIFICATION DIVISION 宣言の直前に入れる必要がある OBJECT キーワードは、クラスのインスタンス・データおよびインスタンス・メソッドの定義の開始を示します。たとえば、Account クラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.
Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
  . . .
End Object.
```

インスタンス・データは、オブジェクト・インスタンスが作成されるときに割り振られ、Java ランタイムによるインスタンスのガーベッジ・コレクションが行われるまで存在します。

上記に示すように、単純インスタンス・データの初期化は、VALUE 文節を使って行うことができます。より複雑なインスタンス・データの初期化は、カスタマイズしたメソッドをコーディングし、クラスのインスタンスを作成して初期化して行うことができます。

COBOL インスタンス・データは、Java private 非静的メンバー・データと同じです。他のクラスまたはサブクラス (同じクラス内のファクトリー・メソッドがあればそのメソッドも) は、COBOL インスタンス・データを直接参照することはできません。インスタンス・データは、OBJECT 段落で定義するすべてのインスタンス・メソッドにグローバルです。OBJECT 段落の外側からインスタンス・データにアクセス可能にしたい場合には、アクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義します。

インスタンス・データ宣言のための WORKING-STORAGE SECTION の構文は、プログラムにおける場合と一般的に同じですが、次のような例外があります。

- EXTERNAL 属性を使用することはできません。
- GLOBAL 属性を使用できますが、効力はありません。

関連タスク

516 ページの『クラスのインスタンスの生成と初期化』
518 ページの『クラスのインスタンスの解放』
525 ページの『ファクトリー・メソッドの定義』
506 ページの『属性 (get および set) メソッドのコーディング』

例: クラスの定義

この例では、Account クラスの定義での最初の試みを示します。ただし、メソッド定義は除きます。

```
 cbl dll,thread,pgmname(longmixed)
 Identification Division.
 Class-id. Account inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base    is "java.lang.Object"
     Class Account is "Account".
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
```

```
*
  Procedure Division.
*
*   (Instance method definitions here)
*
 End Object.
*
 End class Account.
```

関連タスク
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
509 ページの『クライアントの定義』

クラス・インスタンス・メソッドの定義

クラス定義の OBJECT 段落の PROCEDURE DIVISION 内の COBOL *インスタンス・メソッド* を定義します。インスタンス・メソッドは、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされる操作を定義します。

COBOL インスタンス・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

除算	目的	構文
IDENTIFICATION (必須)	メソッドの名前を指定する。	501 ページの『クラス・インスタンス・メソッドを定義する METHOD-ID 段落』 (必須) AUTHOR 段落 (オプショナル) INSTALLATION 段落 (オプショナル) DATE-WRITTEN 段落 (オプショナル) DATE-COMPILED 段落 (オプショナル)
ENVIRONMENT (オプショナル)	メソッド内で使用されるファイル名を、オペレーティング・システムに認識された、対応するファイル名に関連付ける。	501 ページの『クラス・インスタンス・メソッドを定義する INPUT-OUTPUT SECTION』 (オプショナル)
DATA (オプショナル)	外部ファイルを定義する。データのコピーを割り振る。	502 ページの『クラス・インスタンス・メソッドを定義する DATA DIVISION』 (オプショナル)
PROCEDURE (オプショナル)	メソッドによって提供されるサービスを完了するために実行可能ステートメントをコーディングする。	503 ページの『クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION』 (オプショナル)

定義: メソッドのシグニチャー は、メソッドの名前とその仮パラメーターの番号とタイプから構成されます。(COBOL メソッドの仮パラメーターの定義は、そのメソッドの PROCEDURE DIVISION ヘッダーの USING 句で行います。)

クラス定義内では、各メソッド名を固有にする必要はありませんが、各メソッドに固有のシグニチャーを与える必要があります。(メソッドに同じ名前を与えても、別々のシグニチャーを与えると、メソッドを多重定義することになります。)

COBOL インスタンス・メソッドは Java public 非静的メソッドと同じです。

507 ページの『例: メソッドの定義』

関連タスク

503 ページの『クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION』
505 ページの『インスタンス・メソッドの多重定義』
504 ページの『インスタンス・メソッドのオーバーライド』
514 ページの『メソッドの呼び出し (INVOKE)』
522 ページの『サブクラス・インスタンス・メソッドの定義』
525 ページの『ファクトリー・メソッドの定義』

クラス・インスタンス・メソッドを定義する METHOD-ID 段落

METHOD-ID 段落は、インスタンス・メソッドに名前を付けるために使用します。IDENTIFICATION DIVISION 宣言とともに、METHOD-ID 段落の直前に置いて、メソッド定義の開始を表します。 たとえば、Account クラス内の credit メソッドの定義は、以下のように開始します。

```
Identification Division.
Method-id. "credit".
```

メソッド名を英数字または各国語リテラルとしてコーディングします。メソッド名は、大文字小文字の区別で処理されますので、Java メソッド名の形式化規則に準拠する必要があります。

他の Java または COBOL メソッドもしくはプログラム (すなわち、クライアント) は、メソッド名を使用してメソッドを呼び出します。

上記の例では、credit がメソッド名です。

関連タスク

514 ページの『メソッドの呼び出し (INVOKE)』
116 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

メソッド名の意味 (*Java 言語仕様*)
ID (*Java 言語仕様*)
METHOD-ID 段落 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッドを定義する INPUT-OUTPUT SECTION

インスタンス・メソッドの ENVIRONMENT DIVISION には、INPUT-OUTPUT SECTION という 1 つのセクションだけがあります。たとえば、情報をファイルから読み取ったメソッドを Account クラスが定義した場合には、その Account クラスは、以下のような INPUT-OUTPUT SECTION を持つと考えられます。

```
Environment Division.
Input-Output Section.
File-Control.
    Select account-file Assign AcctFile.
```

INPUT-OUTPUT SECTION は、メソッド定義で使用したファイル名をオペレーティング・システムに認識された、対応するファイル名に関連付けます。メソッドの INPUT-OUTPUT SECTION の構文は、プログラムの INPUT-OUTPUT SECTION の構文と同じです。

関連タスク
9 ページの『コンピューター環境の記述』

関連参照
INPUT-OUTPUT セクション (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッドを定義する DATA DIVISION

インスタンス・メソッドの DATA DIVISION は、以下の 4 つのセクションのいずれかで構成されます。

FILE SECTION
メソッド FILE SECTION では EXTERNAL ファイルしか定義できないことを除けば、プログラム FILE SECTION と同じです。

LOCAL-STORAGE SECTION
メソッドの呼び出しごとに、LOCAL-STORAGE データの別個のコピーを割り振り、そのメソッドからの戻り時に解放します。

データ項目上の VALUE 文節を指定すると、メソッドの呼び出しのたびに、項目はその値に初期化されます。

メソッド LOCAL-STORAGE SECTION は、プログラム LOCAL-STORAGE SECTION に類似しています。

WORKING-STORAGE SECTION
WORKING-STORAGE データの単一コピーが割り振られます。データは、実行単位が終了するまで、最後に使われた状態で持続します。メソッドの呼び出しのたびに、呼び出しているオブジェクトまたはスレッドに関係なく、データの同じ単一コピーを使用します。

データ項目上の VALUE 文節を指定すると、メソッドの最初の呼び出しのときに、項目はその値に初期化されます。データ項目に対する EXTERNAL 文節を指定することができます。

メソッド WORKING-STORAGE SECTION は、プログラム WORKING-STORAGE SECTION に類似しています。

LINKAGE SECTION
プログラム LINKAGE SECTION と同じです。

インスタンス・メソッドの DATA DIVISION および OBJECT 段落の DATA DIVISION の両方において、データ項目を同じ名前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

関連タスク
15 ページの『データの記述』
464 ページの『EXTERNAL 文節によるデータの共用』

関連参照
データ部の概要 (*Enterprise COBOL 言語解説書*)

クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION

実行可能ステートメントをコーディングして、インスタンス・メソッドによって、インスタンス・メソッドの PROCEDURE DIVISION において提供する予定のサービスをインプリメントします。

プログラムの PROCEDURE DIVISION でコーディングできるメソッドの PROCEDURE DIVISION において、ほとんどの COBOL ステートメントをコーディングすることができます。ただし、メソッドで以下のステートメントをコーディングすることはできません。

- ENTRY
- EXIT PROGRAM
- COBOL 85 の以下のエレメントは廃止されました。
  - ALTER
  - プロシージャー名が指定されていない GOTO
  - SEGMENT-LIMIT
  - USE FOR DEBUGGING

さらに、すべての COBOL クラス定義を THREAD コンパイラー・オプションを使ってコンパイルする必要があるため、SORT または MERGE ステートメントは COBOL メソッドで使用できません。

インスタンス・メソッドで EXIT METHOD または GOBACK ステートメントをコーディングして、呼び出し側のクライアントに制御を戻すことができます。両方のステートメントに同じ効果があります。メソッドが呼び出されるときに RETURNING 句が指定されていると、EXIT METHOD または GOBACK ステートメントは、呼び出し側のクライアントにデータの値を戻します。

各メソッドの PROCEDURE DIVISION では、暗黙の EXIT METHOD が最後のステートメントとして生成されます。

メソッドで STOP RUN を指定することができます。この指定を行うと、実行単位内で実行しているすべてのスレッドを含め、実行単位全体が終了します。

メソッド定義の終了は、END METHOD マーカーを使って行う必要があります。 たとえば、次のステートメントは credit メソッドの終わりを示します。

```
End method "credit".
```

渡された引き数を取得するための USING 句

メソッドの PROCEDURE DIVISION ヘッダーの USING 句において、必要に応じて、仮パラメーターをメソッドに指定します。引き数が BY VALUE で渡される指定をしなければなりません。各パラメーターは、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。各パラメーターのデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

値を戻すための RETURNING 句

メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句において、(ある場合) データ項目がメソッドの結果として戻されるように指定します。データ項目は、メソッドの LINKAGE SECTION で、レベル 01 または レベル 77 項目として定義します。戻り値のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

関連タスク

540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
『インスタンス・メソッドのオーバーライド』
505 ページの『インスタンス・メソッドの多重定義』
513 ページの『オブジェクト・リファレンスの比較と設定』
514 ページの『メソッドの呼び出し (INVOKE)』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

関連参照

352 ページの『THREAD』
手続き部のヘッダー (*Enterprise COBOL 言語解説書*)

インスタンス・メソッドのオーバーライド

サブクラスで定義されたインスタンス・メソッドは、継承されたインスタンス・メソッドをオーバーライドする と言います。オーバーライドしないと、継承されたインスタンス・メソッドは、2 つのメソッドのシグニチャーが同じ場合に、サブクラスでアクセス可能となります。

スーパークラス・インスタンス・メソッド m1 を COBOL サブクラスでオーバーライドするには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでインスタンス・メソッド m1 を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラスのインスタンスで m1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

たとえば、Account クラスは、その LINKAGE SECTION および PROCEDURE DIVISION ヘッダーが以下のような、メソッド debit を定義します。

```
Linkage section.
01 inDebit    pic S9(9) binary.
Procedure Division using by value inDebit.
```

CheckingAccount サブクラスを定義し、Account スーパークラスで定義された debit メソッドをオーバーライドする debit メソッドをそれに持たす場合には、pic S9(9) binary として指定された入力パラメーターを必ず 1 つ 持つサブクラス・メソッドを定義します。クライアントが、CheckingAccount インスタンスへのオブジェクト・リファレンスを使用して、debit を呼び出すと、CheckingAccount debit メソッド (Account スーパークラス内の debit メソッドではなく) が呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・インスタンス・メソッドとオーバーライドしたスーパークラス・インスタンス・メソッドにおいて同一でなければなりません。

インスタンス・メソッドは、COBOL スーパークラスのファクトリー・メソッドをオーバーライドしてはならないし、Java スーパークラスの静的メソッドをオーバーライドすることもできません。

507 ページの『例: メソッドの定義』

関連タスク

503 ページの『クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
514 ページの『メソッドの呼び出し (INVOKE)』
516 ページの『オーバーライドしたスーパークラス・メソッドの呼び出し』
520 ページの『サブクラスの定義』
527 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』

関連参照

継承、オーバーライド、および隠蔽 (*Java 言語仕様*)

インスタンス・メソッドの多重定義

クラスでサポートされる 2 つのメソッド (クラスに定義されているか、またはスーパークラスから継承された) は、それらの名前が同じで、シグニチャーが異なる場合には、*多重定義されている* と言われます。

たとえば、別々の一組のパラメーターを使ってデータを初期化するために、クライアントが別々の版のメソッドを呼び出せるようにするときは、メソッドを多重定義します。

メソッドを多重定義するには、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数や型が、同じクラスでサポートされる同一の名前のメソッドと異なるメソッドを定義します。

たとえば、Account クラスは、必ず 1 つの仮パラメーターを持つインスタンス・メソッド init を定義します。init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.
01 inAccountNumber pic S9(9) binary.
Procedure Division using by value inAccountNumber.
```

クライアントはこのメソッドを呼び出し、inAccountNumber のデータ型と一致する引き数を必ず 1 つ渡して、指定の口座番号 (およびデフォルトの勘定残高ゼロ) で Account インスタンスを初期化します。

ただし、Account クラスでは、たとえば、開始の勘定残高も指定できる別の仮パラメーターを持つ、2 番目のインスタンス・メソッド init を定義することができます。この init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.
01 inAccountNumber pic S9(9) binary.
01 inBalance       pic S9(9) binary.
Procedure Division using by value inAccountNumber
                                  inBalance.
```

クライアントは、目的のメソッドのシグニチャーと一致する引き数を渡して、いずれかの init メソッドを呼び出すことができます。

メソッド戻り値の有無は、多重定義したメソッドと共通している必要はありません。また、PROCEDURE DIVISION RETURNING 句 (ある場合) で指定する戻り値のデータ型は、多重定義したメソッドと同一である必要はありません。

ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。

多重定義メソッドの定義および多重定義メソッドの呼び出しの解決に関する規則は、対応する Java 規則に基づきます。

関連タスク
514 ページの『メソッドの呼び出し (INVOKE)』
525 ページの『ファクトリー・メソッドの定義』

関連参照
多重定義 (*Java 言語仕様*)

属性 (get および set) メソッドのコーディング

X のためのアクセサー (get) またはミューテーター (set) メソッドをコーディングして X を定義するクラスの外側から、インスタンス変数 X へのアクセスを提供することができます。

COBOL のインスタンス変数は *private* です。インスタンス変数を定義するクラスは、インスタンス変数を完全にカプセル化します。したがって、直接アクセスできるのは、同じ OBJECT 段落で定義するインスタンス・メソッドだけです。通常、うまく設計されたオブジェクト指向アプリケーションは、クラスの外側からインスタンス変数にアクセスする必要はありません。

public インスタンス変数の概念は、Java および他のオブジェクト指向言語で定義されており、クラス属性の概念は CORBA で定義されていますが、どちらの概念も COBOL には直接サポートされていません。(CORBA *属性* は、変数が読み取り専用でない場合に、変数の値にアクセスするための自動生成 get メソッドと変数の値を変更するための自動生成 set メソッドを持つ、インスタンス変数です。)

507 ページの『例: get メソッドのコーディング』

関連タスク
498 ページの『クラス・インスタンス・データを定義する WORKING-STORAGE SECTION』
21 ページの『データの処理』

例: get メソッドのコーディング

Account クラスでは、インスタンス・メソッド getBalance を定義して、インスタンス変数 AccountBalance の値をクライアントに戻すことができます。Account クラスの OBJECT 段落で、getBalance および AccountBalance を以下のように定義することができます。

```
 Identification Division.
 Class-id. Account inherits Base.
*  (ENVIRONMENT DIVISION not shown)
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountBalance pic S9(9) value zero.
*   (Other instance data not shown)
*
  Procedure Division.
*
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
*
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*  (Other instance methods not shown)
 End Object.
*
 End class Account.
```

例: メソッドの定義

この例は、直前の例 (499 ページの『例: クラスの定義』) に、Account クラスのインスタンス・メソッド定義を追加し、Java Check クラスの定義を示します。

Account クラス

```
 cbl dll,thread,pgmname(longmixed)
 Identification Division.
 Class-id. Account inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base    is "java.lang.Object"
     Class Account is "Account".
*
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
  Procedure Division.
*
*    init method to initialize the account:
   Identification Division.
```

```
   Method-id. "init".
   Data division.
   Linkage section.
   01 inAccountNumber pic S9(9) binary.
   Procedure Division using by value inAccountNumber.
     Move inAccountNumber to AccountNumber.
   End method "init".
*
*    getBalance method to return the account balance:
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*    credit method to deposit to the account:
   Identification Division.
   Method-id. "credit".
   Data division.
   Linkage section.
   01 inCredit   pic S9(9) binary.
   Procedure Division using by value inCredit.
     Add inCredit to AccountBalance.
   End method "credit".
*
*    debit method to withdraw from the account:
   Identification Division.
   Method-id. "debit".
   Data division.
   Linkage section.
   01 inDebit    pic S9(9) binary.
   Procedure Division using by value inDebit.
     Subtract inDebit from AccountBalance.
   End method "debit".
*
*    print method to display formatted account number and balance:
   Identification Division.
   Method-id. "print".
   Data division.
        Local-storage section.
   01 PrintableAccountNumber  pic ZZZZZZ999999.
   01 PrintableAccountBalance pic $$$$,$$$,$$9CR.
   Procedure Division.
     Move AccountNumber  to PrintableAccountNumber
     Move AccountBalance to PrintableAccountBalance
     Display " Account: " PrintableAccountNumber
     Display " Balance: " PrintableAccountBalance.
   End method "print".
*
 End Object.
*
 End class Account.
```

Check クラス

```
/**
 * A Java class for check information
 */
public class Check {
  private CheckingAccount payer;
  private Account         payee;
  private int             amount;

  public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
```

```
    payer=inPayer;
    payee=inPayee;
    amount=inAmount;
  }

  public int getAmount() {
    return amount;
  }

  public Account getPayee() {
    return payee;
  }
}
```

関連タスク

307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

クライアントの定義

クラス内の 1 つの以上のメソッドからサービスを要求するプログラムまたはメソッドは、そのクラスのクライアント と呼ばれます。

COBOL クライアントまたは Java クライアントで、以下のことを行うことができます。

- Java および COBOL クラスのオブジェクト・インスタンスを作成する。
- Java および COBOL オブジェクトにインスタンス・メソッドを呼び出す。
- COBOL ファクトリー・メソッドおよび Java 静的メソッドを呼び出す。

COBOL クライアントでは、Java Native Interface (JNI) が提供するサービスを呼び出すこともできます。

COBOL クライアント・プログラムは、次のような通常の 4 つの部分で成り立っています。

除算	目的	構文
IDENTIFICATION (必須)	クライアントの名前。	通常のようにコーディング。ただし、クライアント・プログラムは以下のとおりにする必要があります。 • 再帰的 (PROGRAM-ID 段落内の RECURSIVE 宣言) • スレッド対応 (THREAD オプションでコンパイル、スレッド化アプリケーション用コーディング・ガイドラインに準拠)
ENVIRONMENT (必須)	コンピューター環境を記述する。クライアント内で使用されるクラス名をコンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 510 ページの『クライアントを定義する REPOSITORY 段落』 (必須)
DATA (オプショナル)	クライアントが必要とするデータを記述する。	511 ページの『クライアントを定義する DATA DIVISION』 (オプショナル)

除算	目的	構文
PROCEDURE (オプショナル)	クラスのインスタンスの作成、オブジェクト・リファレンス・データ項目の操作、メソッドの呼び出し。	INVOKE、IF、および SET ステートメントを使ってコーディング

メソッドはサービスを別のメソッドから要求することがあります。したがって、COBOL メソッドがクライアントとして働き、ここで解説されているステートメントを使用することができます。

THREAD コンパイラー・オプションを使って、オブジェクト指向構文を含む、または Java と相互協調処理する、すべての COBOL プログラムをコンパイルする必要があるため、以下の言語エレメントは COBOL クライアントで使用できません。

- SORT または MERGE ステートメント
- ネストされたプログラム

THREAD コンパイラー・オプションを使用してコンパイルするプログラムは、再帰的である必要があります。各 OO COBOL クライアント・プログラムの PROGRAM-ID 段落で、RECURSIVE 文節を指定しなければなりません。

519 ページの『例: クライアントの定義』

関連タスク

481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』
535 ページの『第 29 章 Java メソッドとの通信』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
535 ページの『JNI サービスへのアクセス』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
516 ページの『クラスのインスタンスの生成と初期化』
513 ページの『オブジェクト・リファレンスの比較と設定』
514 ページの『メソッドの呼び出し (INVOKE)』
527 ページの『ファクトリーまたは静的メソッドの呼び出し』

関連参照

352 ページの『THREAD』

クライアントを定義する REPOSITORY 段落

ENVIRONMENT DIVISION の CONFIGURATION SECTION 内の REPOSITORY 段落を使用して、COBOL クライアントで指定のワードを用いるときにそれらのワードがクラス名であることをコンパイラーに宣言し、また、オプションとして、そのクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関連付けます。外部クラス名は大文字小文字を区別します。したがって、Java 形式化規則に準拠しなければなりません。たとえば、Account および Check クラスを使用するクライアント・プログラムにおいて、以下のようにコーディングすることがあります。

```
Environment division.      Required
Configuration section.     Required
  Source-Computer.  IBM-390.
  Object-Computer.  IBM-390.
Repository.                Required
    Class Account is "Account"
    Class Check   is "Check".
```

REPOSITORY 段落記入項目では、クライアント内で Account および Check として参照されるクラスの外部クラス名は、それぞれ、Account および Check であることが示されます。

REPOSITORY 段落では、クライアントにおいて明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。

REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定する必要があります。

Java パッケージの一部である参照クラスごとに外部クラス名を指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形式化規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名の作成は、クラス定義で外部クラス名が REPOSITORY 段落記入項目に組み込まれていない場合と同じ方法で、クラス名から行われます。外部名は英大文字小文字混合で名前付けされます。したがって、上記の例で、クラス Account およびクラス Check は、それぞれ、外部的には Account および Check (英大文字小文字混合) として認識されます。

CONFIGURATION SECTION の SOURCE-COMPUTER、OBJECT-COMPUTER、および SPECIAL-NAMES 段落はオプショナルです。

関連タスク
497 ページの『クラスを定義する REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)
ID (*Java 言語仕様*)
パッケージ (*Java 言語仕様*)

クライアントを定義する DATA DIVISION

DATA DIVISION のいずれかのセクションを使用して、クライアントが必要とするデータを記述することができます。たとえば、次のようになります。

```
Data Division.
Local-storage section.
01  anAccount        usage object reference Account.
01  aCheckingAccount usage object reference CheckingAccount.
01  aCheck           usage object reference Check.
01  payee            usage object reference Account.
. . .
```

クライアントはクラスを参照するため、オブジェクト・リファレンス と呼ばれる 1 つまたは複数の特殊なデータ項目、すなわち、クラスのインスタンスへの参照が必要です。インスタンス・メソッドへのすべての要求は、メソッドがサポートされる (すなわち、継承によって定義されているか、または使用可能である) クラスのインスタンスへのオブジェクト・リファレンスが必要です。オブジェクト・リファレンスをコーディングして COBOL クラスのインスタンスを参照する場合と同じ構文を使って、オブジェクト・リファレンスをコーディングして Java クラスのインスタンスを参照します。 上記の例の `usage object reference` 句は、オブジェクト・リファレンス・データ項目を示します。

上記の 4 つの オブジェクト・リファレンスはすべて、クラス名が `OBJECT REFERENCE` 句の後に現れるために、*型式化* オブジェクト・リファレンスと呼ばれます。型式化オブジェクト・リファレンスの参照先は、`OBJECT REFERENCE` 句で名前付けしたクラスのインスタンス、またはそのサブクラスのいずれか 1 つに限られます。 したがって、anAccount は、Account クラスのインスタンス、またはそのサブクラスのいずれかを参照できますが、他のクラスのインスタンスを参照することはできません。同様に、aCheck の参照先は、Check クラスのインスタンス、またはそのサブクラスのインスタンスに限られます。

上記に示されていない、別のタイプのオブジェクト・リファレンスは、`OBJECT REFERENCE` 句の後にクラス名がありません。そのような参照を*汎用* オブジェクト・リファレンスと呼びます。すべてのクラスのインスタンスを参照できるという意味です。一般的に、汎用オブジェクト・リファレンスのコーディングは避けてください。汎用オブジェクト・リファレンスは、非常に限定された環境においてのみ Java と相互運用可能であるからです (`INVOKE` *`class-name`* `NEW . . .` ステートメントの `RETURNING` 句で使用されたとき)。

`OBJECT REFERENCE` 句で使用するクラス名は、`CONFIGURATION SECTION` の `REPOSITORY` 段落で定義しなければなりません。

LOCAL-STORAGE または WORKING-STORAGE の選択

一般的に、`WORKING-STORAGE SECTION` を使用して、クライアント・プログラムが必要とする作業データを定義することができます。ただし、プログラムが複数のスレッド上で同時に実行できる場合には、上記に示したように、データの定義を `LOCAL-STORAGE SECTION` で行おうとする場合があります。各スレッドは、`LOCAL-STORAGE` データの別々のコピーへのアクセス権を持っていますが、`WORKING-STORAGE` データの単一のコピーへのアクセス権は共用します。`WORKING-STORAGE SECTION` でデータを定義する場合には、データへのアクセスを同期化するか、または、2 つのスレッドが同時にそのデータへアクセスしないようにする必要があります。

関連タスク

481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
514 ページの『メソッドの呼び出し (INVOKE)』
510 ページの『クライアントを定義する REPOSITORY 段落』

関連参照

RETURNING 句 (*Enterprise COBOL 言語解説書*)

オブジェクト・リファレンスの比較と設定

オブジェクト・リファレンスは、条件ステートメントを使用して比較することができます。たとえば、次の IF ステートメントのいずれかをコーディングして、オブジェクト・リファレンス anAccount がオブジェクト・インスタンスをまったく参照していないことをチェックします。

```
If anAccount = Null . . .
If anAccount = Nulls . . .
```

JNI サービス IsSameObject への呼び出しをコーディングして、2 つのオブジェクト・リファレンス (object1 および object2) が同一のオブジェクト・インスタンスを参照するのか、各オブジェクト・リファレンスがオブジェクト・インスタンスをまったく参照しないのかをチェックすることができます。引き数および戻り値が Java と相互運用可能であることを確認し、呼び出し可能サービスへのアドレス可能度を確立するには、IsSameObject への呼び出しに先立って、これらのデータ定義およびステートメントをコーディングします。

```
Local-storage Section.
. . .
01  is-same Pic X.
    88 is-same-false Value X'00'.
    88 is-same-true  Value X'01' Through X'FF'.
Linkage Section.
    Copy JNI.
Procedure Division.
    Set Address Of JNIEnv To JNIEnvPtr
    Set Address Of JNINativeInterface To JNIEnv
    . . .
    Call IsSameObject Using By Value JNIEnvPtr object1 object2
                      Returning is-same
    If is-same-true . . .
```

メソッド内では、オブジェクト・リファレンスと SELF を比較する IsSameObject への呼び出しをコーディングして、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト・リファレンスが参照するかをチェックすることができます。

上記の代わりに、Java equals メソッド (java.lang.Object からの継承) を呼び出して、2 つのオブジェクト・リファレンスが同一のオブジェクト・インスタンスを参照するかを決定することができます。

SET ステートメントを使用して、オブジェクト・リファレンスがオブジェクト・インスタンスを参照しないように設定することができます。たとえば、次のように設定します。

```
Set anAccount To Null.
```

また、SET ステートメントを使用して、1 つのオブジェクト・リファレンスが別のオブジェクト・リファレンスと同じインスタンスを参照するように設定することもできます。たとえば、次のようになります。

```
Set anotherAccount To anAccount.
```

この SET ステートメントによって、anotherAccount は anAccount と同じオブジェクト・インスタンスを参照します。受け取り側 (anotherAccount) が汎用オブジェクト・リファレンスである場合、送り出し側 (anAccount) は、汎用オブジェクト・リファレンスか、または型式化オブジェクト・リファレンスのどちらかになります。受け取り側が型式化オブジェクト・リファレンスである場合は、送り出し側も、受

け取り側と同じクラス、または、そのサブクラスのいずれか 1 つにバインドされた、型式化オブジェクト・リファレンスでなければなりません。

メソッド内では、オブジェクト・リファレンスを SELF に設定して、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト・リファレンスが参照するように設定することができます。たとえば、次のように設定します。

```
Set anAccount To Self.
```

関連タスク

540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
535 ページの『JNI サービスへのアクセス』

関連参照

IsSameObject (*Java Native Interface*)

メソッドの呼び出し (INVOKE)

Java クライアントでは、COBOL でインプリメントされたクラスのオブジェクト・インスタンスを作成して、それらのオブジェクトにメソッドを呼び出すことができます。上記を実行するには、標準の Java 構文を使用します。

COBOL クライアントでは、INVOKE ステートメントをコーディングして、Java または COBOL クラスで定義したメソッドを呼び出すことができます。 たとえば、次のようにして呼び出します。

```
Invoke Account "createAccount"
  using by value 123456
  returning anAccount
Invoke anAccount "credit" using by value 500.
```

最初のステートメントは、クラス名 Account を使用して、createAccount という名前のメソッドを呼び出します。このメソッドは、Account クラスでサポートされる (すなわち、定義または継承された) Java 静的メソッドまたは COBOL ファクトリー・メソッドのいずれかでなければなりません。using by value 123456 は、123456 が、値によって渡された、メソッドへの入力引き数であることを示します。123456 および anAccount は、それぞれ、(おそらく多重定義された) createAccount メソッドの仮パラメーターおよび戻りの型の定義に準拠しなければなりません。

2 番目の INVOKE ステートメントは、戻されたオブジェクト・リファレンス anAccount を使用して、Account クラスで定義したインスタンス・メソッド credit を呼び出します。入力引き数 500 は、(おそらく多重定義された) credit メソッドの仮パラメーターの定義に準拠しなければなりません。

実行時におけるその値がターゲット・メソッドのシグニチャー内のメソッド名と一致するリテラルとして、または ID として呼び出すメソッドの名前をコーディングします。メソッド名は、英数字または各国語リテラル、あるいはデータ項目でなければならず、大文字小文字を区別して解釈されます。

たとえば、オブジェクト・リファレンス (上記の 2 番目のステートメントの場合のように) を使用して INVOKE ステートメントをコーディングするとき、ステートメントは以下の 2 つの形式のどちらかで開始します。

```
Invoke objRef "literal-name" . . .
Invoke objRef identifier-name . . .
```

メソッド名が ID である場合には、オブジェクト・リファレンス (objRef) を、指定する型のない USAGE OBJECT REFERENCE として、すなわち、汎用オブジェクト・リファレンスとして、定義しなければなりません。

オブジェクト・リファレンスの参照先のクラスで、呼び出したメソッドがサポートされない場合、INVOKE ステートメントの ON EXCEPTION 句をコーディングしていないときには、重大度 3 の言語環境プログラム条件が実行時に発生します。

オプションの範囲終了符号 END-INVOKE を INVOKE ステートメントで使用することができます。

INVOKE ステートメントは、RETURN-CODE 特殊レジスターを設定しません。

引き数を渡すための USING 句

INVOKE ステートメントの USING 句で、引き数をメソッドに (ある場合) 指定します。引き数のデータ型を、目的のターゲット・メソッドのシグニチャーと一致するように、コーディングします。ターゲット・メソッドが多重定義されている場合、引き数のデータ型は、同じ名前を持つメソッドの中から選択するために使用されます。

引き数が BY VALUE で渡されるように指定しなければなりません。これは、呼び出されたメソッド内の対応する仮パラメーターが変更されても、引き数は影響を受けないという意味です。各引き数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

戻り値を取得するための RETURNING 句

INVOKE ステートメントの RETURNING 句において、データ項目がメソッドの結果として (ある場合) 戻されるように指定します。戻される項目は、クライアントの DATA DIVISION で定義します。

INVOKE ステートメントの RETURNING 句で指定する項目は、以下のターゲット・メソッドが戻すタイプに準拠する必要があります。

- ターゲット・メソッドが COBOL でインプリメントされ、戻される項目がオブジェクト・リファレンスでない場合には、ターゲット・メソッドにおける RETURNING 項目の定義とまったく同様に、戻される項目の DATA DIVISION 定義をコーディングします。
- ターゲット・メソッドが Java でインプリメントされた場合には、戻される Java データ項目と相互運用可能になるように戻される項目の DATA DIVISION 定義をコーディングします。
- 戻される項目がオブジェクト・リファレンスである場合には、ターゲット・メソッドが戻すオブジェクト・リファレンスと同じクラスに型式化したオブジェクト・リファレンスとして、戻される項目の DATA DIVISION 定義をコーディングします。

すべての場合において、戻される値のデータ型は、Java と相互運用可能になる型のうちのいずれかでなければなりません。

関連タスク

505 ページの『インスタンス・メソッドの多重定義』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』

503 ページの『クラス・インスタンス・メソッドを定義する PROCEDURE DIVISION』
『オーバーライドしたスーパークラス・メソッドの呼び出し』
527 ページの『ファクトリーまたは静的メソッドの呼び出し』
116 ページの『COBOL での国別データ (Unicode) の使用』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

オーバーライドしたスーパークラス・メソッドの呼び出し

クラス内で、現行クラスに定義されている同一シグニチャーを有するメソッドを呼び出すのではなく、オーバーライドしたスーパークラス・メソッドを呼び出す必要がある場合があります。

たとえば、CheckingAccount クラスが、その即時スーパークラス Account に定義されている `debit` インスタンス・メソッドをオーバーライドすると想定します。続いて、次のステートメントをコーディングして、CheckingAccount クラスのメソッド内で Account の `debit` メソッドを呼び出すことができます。

```
Invoke Super "debit" Using By Value amount.
```

`debit` メソッドのシグニチャーに一致するように、`amount` を `PIC S9(9) BINARY` として定義します。

CheckingAccount クラスは、Account クラスに定義されている `print` メソッドをオーバーライドします。`print` メソッドには仮パラメーターがありません。したがって、CheckingAccount クラス内のメソッドは、次のステートメントでスーパークラス `print` メソッドを呼び出すことができます。

```
Invoke Super "print".
```

キーワード `SUPER` が示しているのは、呼び出す対象は、現行クラスのメソッドではなく、スーパークラス・メソッドであることです。(`SUPER` は、現在実行中のメソッドの起動に使用されているオブジェクトへの暗黙の参照です。)

492 ページの『例: 口座』

関連タスク
504 ページの『インスタンス・メソッドのオーバーライド』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

クラスのインスタンスの生成と初期化

Java または COBOL クラスに定義されているインスタンス・メソッドを使用するためには、まず、そのクラスのインスタンスを生成しなければなりません。クラス *class-name* の新しいインスタンスを生成するには、また、作成したオブジェクトへの参照 *object-reference* を取得するには、次の形式のステートメントをコーディングします。

```
INVOKE class-name NEW . . . RETURNING object-reference
```

オブジェクト・リファレンス は、クライアントの DATA DIVISION で定義しなければなりません。

メソッド内で INVOKE . . . NEW ステートメントをコーディングするとき、また、戻されたオブジェクト・リファレンスの使用がメソッド起動の期間に限定されていないときは (たとえば、オブジェクト・リファレンスをメソッド戻り値として使用する場合)、次のように JNI サービス NewGlobalRef を呼び出して、戻されたオブジェクト・リファレンスをグローバル参照に変換しなければなりません。

```
Call NewGlobalRef using by value JNIEnvPtr object-reference
                  returning object-reference
```

NewGlobalRef を呼び出さない場合には、戻されたオブジェクト・リファレンスはあくまでもローカル参照にすぎないため、メソッドが戻った後で自動的に解放されます。

NewGlobalRef サービスに対するアドレス可能度を取得するには、メソッドの LINKAGE SECTION に COPY JNI ステートメントをコーディングし、NewGlobalRef を呼び出す前に、メソッドの PROCEDURE DIVISION に、以下の 2 つのステートメントをコーディングします。

```
Set address of JNIEnv to JNIEnvPtr
Set address of JNINativeInterface to JNIEnv
```

Java クラスのインスタンスの生成

Java クラスのインスタンスを生成するには、INVOKE . . . NEW ステートメントの USING 句をコーディングし、RETURNING 句の直前で、コンストラクターのシグニチャーに一致する引き数の数および型 BY VALUE を渡して、クラスがサポートする、任意のパラメーター化コンストラクターを呼び出すことができます。各引き数のデータ型は、Java と相互運用可能な型のいずれか 1 つでなければなりません。デフォルトの (パラメーターなし) コンストラクターを呼び出すには、USING 句を省略します。

たとえば、Check クラスのインスタンスを生成し、そのインスタンス・データを初期化し、生成した Check インスタンスへの参照 aCheck を取得するには、次のステートメントを COBOL クライアントにコーディングすることができます。

```
Invoke Check New
  using by value aCheckingAccount, payee, 125
  returning aCheck
```

COBOL クラスのインスタンスの生成

COBOL クラスのインスタンスを生成するには、INVOKE . . . NEW ステートメントの RETURNING 句に、型式化オブジェクト・リファレンスまたは汎用オブジェクト・リファレンスのいずれかを指定することができます。ただし、USING 句をコーディングすることはできません。インスタンス・データは、クラス定義の VALUE 文節に指定されたとおりに初期化されます。したがって、INVOKE . . . NEW ステートメントは、単一のインスタンス・データのみを有する COBOL クラスのインスタンスを生成するときに役立ちます。

たとえば、次のステートメントは、Account クラスのインスタンスを作成し、Account クラス定義の OBJECT 段落の WORKING-STORAGE SECTION の VALUE 文節に指定されたとおりに、インスタンス・データを初期化し、新しいインスタンスへの参照 outAccount を提供します。

```
Invoke Account New returning outAccount
```

VALUE 文節だけを使用して初期化することができない COBOL クラス・データの初期化を可能にするには、COBOL クラスを設計する際に、FACTORY 段落にパラメーター化生成メソッドを定義し、OBJECT 段落にパラメーター化初期化メソッドを定義する必要があります。

1. パラメーター化ファクトリー生成メソッドで、以下の手順を実行します。
   a. INVOKE *class-name* NEW RETURNING *objectRef* をコーディングして、*class-name* のインスタンスを作成し、VALUE 文節を有するインスタンス・データ項目に初期値を与えます。
   b. JNI サービス NewGlobalRef を呼び出して、*objectRef* をローカル参照からグローバル参照に変換します。
   c. パラメーター化した初期化メソッドをインスタンス (*objectRef*) 上で呼び出し、指定された引き数 BY VALUE をファクトリー・メソッドに渡します。
2. 初期化メソッドで、ロジックをコーディングし、仮パラメーターを介して指定された値を使用して、インスタンス・データ初期化を完了します。

続いて、COBOL クラスのインスタンスを作成して、それを適切に初期化するには、クライアントは、パラメーター化ファクトリー・メソッドを呼び出し、目的の引き数 BY VALUE を渡します。

528 ページの『例: ファクトリーの定義 (メソッドを含む)』

関連タスク

535 ページの『JNI サービスへのアクセス』
538 ページの『ローカルおよびグローバル参照の管理』
511 ページの『クライアントを定義する DATA DIVISION』
514 ページの『メソッドの呼び出し (INVOKE)』
524 ページの『ファクトリー・セクションの定義』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』

関連参照

VALUE 文節 (*Enterprise COBOL 言語解説書*)
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

クラスのインスタンスの解放

どのクラスの個々のオブジェクト・インスタンスでも、オブジェクト・インスタンスを解放する処理を行う必要はありません。したがって、オブジェクト・インスタンスを解放するために有効な構文はありません。

Java ランタイム・システムは、*ガーベッジ・コレクション* を自動的に行います。すなわち、使用しなくなったオブジェクトをメモリーから削除します。

ただし、参照済みオブジェクトのガーベッジ・コレクションを許可するために、ネイティブ COBOL クライアント内のオブジェクトへのローカル参照またはグローバル参照を明示的に解放する必要がある場合があります。

関連タスク

538 ページの『ローカルおよびグローバル参照の管理』

例: クライアントの定義

次の例では、Account クラスの小さいクライアント・プログラムを示します (507 ページの『例: メソッドの定義』)。プログラムは以下を実行します。

1. ファクトリー・メソッド createAccount を呼び出して、デフォルト収支ゼロの Account インスタンスを作成します。
2. インスタンス・メソッド credit を呼び出して、$500 をこの新規の口座に預金します。
3. インスタンス・メソッド print を呼び出して、口座の状況を表示します。

```
 cbl dll,thread,pgmname(longmixed)
 Identification division.
 Program-id. "TestAccounts" recursive.
 Environment division.
 Configuration section.
 Repository.
     Class Account is "Account".
 Data Division.
*  Working data is declared in LOCAL-STORAGE instead of
*  WORKING-STORAGE so that each thread has its own copy:
 Local-storage section.
 01  anAccount usage object reference Account.
*
 Procedure division.
 Test-Account-section.
     Display "Test Account class"
*  Create account 123456 with 0 balance:
     Invoke Account "createAccount"
       using by value 123456
       returning anAccount
*  Deposit 500 to the account:
     Invoke anAccount "credit" using by value 500
     Invoke anAccount "print"
     Display space
*
     Stop Run.
 End program "TestAccounts".
```

528 ページの『例: ファクトリーの定義 (メソッドを含む)』

関連タスク

525 ページの『ファクトリー・メソッドの定義』
527 ページの『ファクトリーまたは静的メソッドの呼び出し』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

サブクラスの定義

クラス (サブクラス、派生クラス、または子クラスと呼ばれる) を、別のクラスを特殊化したもの (スーパークラス、基本クラス、または親クラスと呼ばれる) にすることができます。サブクラスは、そのスーパークラスのメソッドおよびインスタンス・データを継承し、*is-a* 関係によって、そのスーパークラスに関連付けられています。たとえば、サブクラス P がスーパークラス Q から継承し、サブクラス Q がスーパークラス S から継承した場合、P のインスタンスは Q のインスタンスであり、また (推移性によって) S のインスタンスでもあります。したがって、P のインスタンスは、Q と S のメソッドおよびデータを継承します。

制約事項: COBOL プログラムでは*多重継承* を使用することはできません。定義する各 COBOL クラスには、Java または COBOL でインプリメントされた即時スーパークラスは必ず 1 つだけでなければなりません。また、それぞれのクラスは、直接的または間接的に java.lang.Object から派生したものでなければなりません。継承のセマンティクスは、Java によって定義されます。

サブクラスを使用することの利点:

- コードの再利用。継承を通じて、サブクラスは、スーパークラスにすでに存在するメソッドを再利用することができます。
- 特殊化。サブクラスでは、スーパークラスが処理しないケースを処理するために新規のメソッドを追加することができます。また、スーパークラスが必要としない新規のデータ項目を追加することもできます。
- アクションの変更。サブクラスは、スーパークラスにあるシグニチャーと同じシグニチャーのメソッドを定義して、スーパークラスから継承するメソッドをオーバーライドすることができます。メソッドをオーバーライドするときは、メソッドの実行内容について、いくつかの小さい変更を行うだけの場合、または全面的な変更を行う場合があります。

サブクラスの構造および構文は、クラス定義の構造および構文と同一です。サブクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION に、インスタンス・データとメソッドを定義します。個別のオブジェクト・インスタンスにではなく、サブクラス自体に関連付けるデータとメソッドを必要とするサブクラスに、サブクラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

COBOL インスタンス・データは private です。サブクラスが COBOL スーパークラスのインスタンス・データにアクセスすることができるのは、スーパークラスがそのアクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義する場合に限られます。

492 ページの『例: 口座』
523 ページの『例: サブクラスの定義 (メソッドを含む)』

関連タスク

495 ページの『クラスの定義』
504 ページの『インスタンス・メソッドのオーバーライド』
506 ページの『属性 (get および set) メソッドのコーディング』

522 ページの『サブクラス・インスタンス・メソッドの定義』
524 ページの『ファクトリー・セクションの定義』

関連参照
継承、オーバーライド、および隠蔽 (*Java 言語仕様*)
COBOL クラス定義構造 (*Enterprise COBOL 言語解説書*)

サブクラスを定義する CLASS-ID 段落

CLASS-ID 段落を使用して、サブクラスに名前を付け、そのサブクラスがどの即時 Java または COBOL スーパークラスからその特性を継承するのかを示します。たとえば、次のようになります。

```
Identification Division.                    Required
Class-id. CheckingAccount inherits Account. Required
```

上記の例で、定義されるサブクラスは CheckingAccount です。CheckingAccount は、サブクラス定義において Account として認識されているクラスのすべてのメソッドを継承します。CheckingAccount メソッドが Account インスタンス・データにアクセスできるのは、Account クラスが、そのアクセスを可能にするために、属性 (get または set) メソッドを指定する場合に限られます。

ENVIRONMENT DIVISION の CONFIGURATION SECTION の REPOSITORY 段落に、即時スーパークラスの名前を指定しなければなりません。オプションとして、外部で認識されているクラスの名前にスーパークラス名を関連付けることができます。また、定義中のサブクラスの名前 (上記の例の CheckingAccount) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

関連タスク
496 ページの『クラスを定義する CLASS-ID 段落』
506 ページの『属性 (get および set) メソッドのコーディング』
『サブクラスを定義する REPOSITORY 段落』

サブクラスを定義する REPOSITORY 段落

REPOSITORY 段落を使用して、サブクラス定義内で指定のワードを用いるときにそれらのワードがクラス名であることをコンパイラーに宣言し、また、オプションとして、そのクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関連付けます。 たとえば、CheckingAccount サブクラス定義において、以下のようにコーディングします。

```
Environment Division.                              Required
Configuration Section.                             Required
Repository.                                        Required
    Class CheckingAccount is "CheckingAccount" Optional
    Class Check           is "Check"           Required
    Class Account         is "Account".        Required
```

REPOSITORY 段落記入項目では、サブクラス定義内で CheckingAccount、Check、および Account として参照されるクラスの外部クラス名は、それぞれ、CheckingAccount、Check、および Account であることが示されます。

REPOSITORY 段落では、サブクラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下にその例を示します。

- 定義中のサブクラスが継承する元のユーザー定義スーパークラス
- サブクラス定義内のメソッドで参照するクラス

サブクラス内の REPOSITORY 段落記入項目をコーディングする場合の規則は、クラス内の REPOSITORY 段落記入項目をコーディングする場合の規則と同一です。

関連タスク
497 ページの『クラスを定義する REPOSITORY 段落』

関連参照
REPOSITORY 段落 (*Enterprise COBOL 言語解説書*)

サブクラス・インスタンス・データを定義する WORKING-STORAGE SECTION

サブクラス OBJECT 段落の DATA DIVISION 内の WORKING-STORAGE SECTION を使用して、サブクラスのスーパークラスに定義したインスタンス・データのほかに、サブクラスが必要とするインスタンス・データを記述します。クラスにインスタンス・データを定義するときに使用する構文と同じ構文を使用します。

たとえば、Account クラスの CheckingAccount サブクラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.
Object.
 Data division.
 Working-storage section.
 01 CheckFee pic S9(9) value 1.
 . . .
End Object.
```

関連タスク
498 ページの『クラス・インスタンス・データを定義する WORKING-STORAGE SECTION』

サブクラス・インスタンス・メソッドの定義

サブクラスは、そのスーパークラスからメソッドを継承します。サブクラス定義において、継承したメソッドと同じシグニチャーを有するインスタンス・メソッドを定義して、サブクラスが継承するインスタンス・メソッドをオーバーライドすることができます。また、サブクラス定義において、サブクラスが必要とする新規メソッドを定義することもできます。

サブクラス・インスタンス・メソッドの構造と構文は、クラス・インスタンス・メソッドの構造と構文と同一です。サブクラス定義の OBJECT 段落の PROCEDURE DIVISION にサブクラス・インスタンス・メソッドを定義します。

523 ページの『例: サブクラスの定義 (メソッドを含む)』

関連タスク
500 ページの『クラス・インスタンス・メソッドの定義』
504 ページの『インスタンス・メソッドのオーバーライド』
505 ページの『インスタンス・メソッドの多重定義』

例: サブクラスの定義 (メソッドを含む)

次の例では、Account クラスの CheckingAccount サブクラスのインスタンス・メソッド定義を示します (507 ページの『例: メソッドの定義』を参照)。

- processCheck メソッドは、Check クラスの Java インスタンス・メソッド getAmount および getPayee を呼び出して、チェック・データを取得します。Account クラスから継承した credit および debit インスタンス・メソッドを呼び出して、当座の受取人を貸し方に記入し、支払人を借方に記入します。
- print メソッドは、Account クラスに定義されている print インスタンス・メソッドをオーバーライドします。オーバーライドした print メソッドを呼び出して、口座状況を表示し、また当座手数料も表示します。CheckFee は、サブクラスに定義するインスタンス・データ項目です。

CheckingAccount クラス (Account のサブクラス)

```
 cbl dll,thread,pgmname(longmixed)
 Identification Division.
 Class-id. CheckingAccount inherits Account.
 Environment Division.
 Configuration section.
 Repository.
     Class CheckingAccount is "CheckingAccount"
     Class Check           is "Check"
     Class Account         is "Account".
*
*  (FACTORY paragraph not shown)
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 CheckFee pic S9(9) value 1.
  Procedure Division.
*
*    processCheck method to get the check amount and payee,
*    add the check fee, and invoke inherited methods debit
*    to debit the payer and credit to credit the payee:
   Identification Division.
   Method-id. "processCheck".
   Data division.
   Local-storage section.
   01 amount pic S9(9) binary.
   01 payee usage object reference Account.
   Linkage section.
   01 aCheck usage object reference Check.
*
   Procedure Division using by value aCheck.
     Invoke aCheck "getAmount" returning amount
     Invoke aCheck "getPayee" returning payee
     Invoke payee  "credit" using by value amount
     Add checkFee to amount
     Invoke self   "debit"  using by value amount.
   End method "processCheck".
*
*    print method override to display account status:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 printableFee pic $$,$$$,$$9.
   Procedure Division.
     Invoke super "print"
     Move CheckFee to printableFee
```

```
     Display " Check fee: " printableFee.
   End method "print".
*
 End Object.
*
 End class CheckingAccount.
```

関連タスク

307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
514 ページの『メソッドの呼び出し (INVOKE)』
504 ページの『インスタンス・メソッドのオーバーライド』
516 ページの『オーバーライドしたスーパークラス・メソッドの呼び出し』

ファクトリー・セクションの定義

クラス定義の FACTORY 段落を使用して、個別のオブジェクト・インスタンスにではなく、クラス自体に関連付けるデータとメソッドを定義します。

- COBOL ファクトリー・データ は、Java private 静的データと同じです。データの単一コピーは、そのクラス用にインスタンス生成され、クラスのすべてのオブジェクト・インスタンスに共用されます。

  クラスのすべてのインスタンスからデータを収集するときは、ごく一般的にファクトリー・データを使用します。たとえば、ファクトリー・データ項目を定義して、作成するクラスのインスタンス数の現在高を集計できます。
- COBOL ファクトリー・メソッド は Java public 静的メソッドと同じです。これらのメソッドは、どのオブジェクト・インスタンスとも無関係に、クラスによってサポートされます。

  VALUE 文節を使用しただけではインスタンス・データを初期化できないときは、ごく一般的に、ファクトリー・メソッドを使用して、オブジェクトの生成をカスタマイズします。

対照的に、クラスのそれぞれのオブジェクト・インスタンスごとに作成されるデータを定義したり、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされるメソッドを定義したりする場合には、クラス定義の OBJECT 段落を使用します。

ファクトリー定義は、以下の 3 つの部分から構成され、その後に END FACTORY ステートメントが続きます。

除算	目的	構文
IDENTIFICATION (必須)	ファクトリー定義の開始を示す。	IDENTIFICATION DIVISION. FACTORY.
DATA (オプショナル)	このクラス用に一度割り振られたデータを記述する (クラスのそれぞれのインスタンスごとに割り振られたデータとは正反対)。	525 ページの『ファクトリー・データを定義する WORKING-STORAGE SECTION』 (オプショナル)
PROCEDURE (オプショナル)	ファクトリー・メソッドを定義する。	ファクトリー・メソッドの定義: 525 ページの『ファクトリー・メソッドの定義』

528 ページの『例: ファクトリーの定義 (メソッドを含む)』

関連タスク

495 ページの『クラスの定義』
516 ページの『クラスのインスタンスの生成と初期化』
『ファクトリー・メソッドの定義』
528 ページの『プロシージャー指向 COBOL プログラムのラッピング』

ファクトリー・データを定義する WORKING-STORAGE SECTION

FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION を使用して、COBOL クラスが必要とするファクトリー・データ、すなわちクラスのすべてのオブジェクト・インスタンスが共有する、静的に割り振られたデータを記述します。
IDENTIFICATION DIVISION 宣言の直前に入れる必要がある FACTORY キーワードは、クラスのファクトリー・データおよびファクトリー・メソッドの定義の開始を示します。たとえば、Account クラスのファクトリー・データの定義は、以下のようになります。

```
Identification division.
Factory.
 Data division.
 Working-storage section.
 01 NumberOfAccounts pic 9(6) value zero.
 . . .
End Factory.
```

上記に示すように、単純ファクトリー・データの初期化は、VALUE 文節を使って行うことができます。

COBOL ファクトリー・データは、Java private 静的データと同じです。他のクラスまたはサブクラス (必要に応じて、同じクラス内のインスタンス・メソッドも) は、COBOL ファクトリー・データを直接参照することはできません。ファクトリー・データは、FACTORY 段落で定義するすべてのファクトリー・メソッドにグローバルです。FACTORY 段落の外側からファクトリー・データにアクセス可能にする場合には、アクセスを可能にするためにファクトリー属性 (get または set) メソッドを定義します。

関連タスク

506 ページの『属性 (get および set) メソッドのコーディング』
516 ページの『クラスのインスタンスの生成と初期化』

ファクトリー・メソッドの定義

クラス定義の FACTORY 段落の PROCEDURE DIVISION に COBOL ファクトリー・メソッド を定義します。ファクトリー・メソッドは、クラスのどのオブジェクト・インスタンスとは無関係に、クラスによってサポートされる操作を定義します。

一般的には、そのインスタンスが複雑な初期化を必要とするクラスについて、すなわち、VALUE 文節だけの使用では割り当てることができない値に対して、ファクトリー・メソッドを定義します。ファクトリー・メソッド内でインスタンス・メソッドを呼び出して、インスタンス・データを初期化することができます。ファクトリー・メソッドは、インスタンス・データに直接アクセスすることはできません。

また、ファクトリー属性 (get および set) メソッドをコーディングして、FACTORY 段落の外側からファクトリー・データにアクセス可能にすることもできます。たとえば、同じクラスのインスタンス・メソッドから、またはクライアント・プログラムからファクトリー・データにアクセス可能にすることができます。たとえば、Account クラスはファクトリー・メソッド getNumberOfAccounts を定義して、口座数の現在の集計を戻すことができます。

また、ファクトリー・メソッドを使用して、Java プログラムからアクセス可能になるようにプロシージャー指向の COBOL プログラムをラップすることもできます。

ファクトリー・メソッドの定義では、インスタンス・メソッドを定義するときに用いる構文と同じ構文を使用します。COBOL ファクトリー・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

除算	目的	構文
IDENTIFICATION (必須)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ (必須)
ENVIRONMENT (オプショナル)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
DATA (オプショナル)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
PROCEDURE (オプショナル)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ

クラス定義内では、各ファクトリー・メソッド名を固有にする必要はありませんが、各ファクトリー・メソッドに固有のシグニチャーを与える必要があります。ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。たとえば、CheckingAccount サブクラスは、2 つの版のファクトリー・メソッド createCheckingAccount、すなわち、口座を初期化してデフォルトの収支ゼロを設定する版と、開始残高を渡せるようにする版を提供します。クライアントは、目的のメソッドのシグニチャーと一致する引き数を渡して、createCheckingAccount メソッドのいずれかを呼び出すことができます。

ファクトリー・メソッドの DATA DIVISION および FACTORY 段落の DATA DIVISION の両方において、データ項目を同じ名前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

COBOL ファクトリー・メソッドは Java public 静的メソッドと同じです。

528 ページの『例: ファクトリーの定義 (メソッドを含む)』

関連タスク

500 ページの『クラス・インスタンス・メソッドの定義』
506 ページの『属性 (get および set) メソッドのコーディング』
505 ページの『インスタンス・メソッドの多重定義』
516 ページの『クラスのインスタンスの生成と初期化』
527 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』

『ファクトリーまたは静的メソッドの呼び出し』
528 ページの『プロシージャー指向 COBOL プログラムのラッピング』

ファクトリー・メソッドまたは静的メソッドの隠蔽

サブクラスで定義されたファクトリー・メソッドは継承された COBOL または Java メソッドを*隠す* と言います。そのようにしないと、COBOL または Java メソッドが、2 つのメソッドのシグニチャーが同じ場合に、サブクラス内でアクセス可能となります。

スーパークラス・ファクトリー・メソッド f1 を COBOL サブクラスで隠すには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでファクトリー・メソッド f1 を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラス名を使用して f1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・ファクトリー・メソッドと隠されたスーパークラス・メソッドにおいて同一でなければなりません。

ファクトリー・メソッドは、Java または COBOL スーパークラスに、インスタンス・メソッドを隠してなりません。

528 ページの『例: ファクトリーの定義 (メソッドを含む)』

関連タスク

540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
504 ページの『インスタンス・メソッドのオーバーライド』
514 ページの『メソッドの呼び出し (INVOKE)』

関連参照

継承、オーバーライド、および隠蔽 (*Java 言語仕様*)
手続き部のヘッダー (*Enterprise COBOL 言語解説書*)

ファクトリーまたは静的メソッドの呼び出し

COBOL ファクトリー・メソッドまたは Java 静的メソッドを COBOL メソッドまたはクライアント・プログラムに呼び出すには、クラス名を INVOKE ステートメントの第 1 オペランドとしてコーディングします。

たとえば、クライアント・プログラムは次のステートメントをコーディングして、createCheckingAccount という名前の多重定義 CheckingAccount ファクトリー・メソッドの 1 つを呼び出して、口座番号 777777 および開始残高 $300 の当座預金を作成することができます。

```
Invoke CheckingAccount "createCheckingAccount"
  using by value 777777 300
  returning aCheckingAccount
```

ファクトリー・メソッドを定義する同じクラス内からファクトリー・メソッドを呼び出す場合にも、クラス名を INVOKE ステートメントの第 1 オペランドとして使用します。

実行時におけるその値がメソッド名であるリテラルとして、または ID として呼び出すメソッドの名前をコーディングします。メソッド名は、英数字または各国語リテラル、あるいはデータ項目でなければならず、大文字小文字を区別して解釈されます。

INVOKE ステートメントで名前を付けるクラスで、呼び出したメソッドがサポートされない場合、INVOKE ステートメントの ON EXCEPTION 句をコーディングしないときには、重大度 3 の言語環境プログラム条件が実行時に発生します。

USING 句で COBOL ファクトリー・メソッドまたは Java 静的メソッドに引き数を渡すときの適合要件と、RETURNING 句で戻り値を受けるときの適合要件は、インスタンス・メソッドを呼び出す場合と同じです。

『例: ファクトリーの定義 (メソッドを含む)』

関連タスク
514 ページの『メソッドの呼び出し (INVOKE)』
116 ページの『COBOL での国別データ (Unicode) の使用』
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』

関連参照
INVOKE ステートメント (*Enterprise COBOL 言語解説書*)

プロシージャー指向 COBOL プログラムのラッピング

ラッパー は、オブジェクト指向コードとプロシージャー指向コード間のインターフェースを指定するクラスです。

ファクトリー・メソッドは、既存のプロシージャー型 COBOL コード用にラッパーを書き込み、Java プログラムからアクセス可能にするときの便利な方法を提供します。

COBOL コードをラップするするためには、以下の手順を実行します。

1. FACTORY 段落を含む単純 COBOL クラスを作成する。
2. FACTORY 段落で、CALL ステートメントを使用してプロシージャー型プログラムを呼び出すファクトリー・メソッドをコーディングする。

Java プログラムは、静的メソッドの呼び出しの式を使用して、すなわち、COBOL プロシージャー型プログラムを呼び出して、ファクトリー・メソッドを呼び出すことができます。

例: ファクトリーの定義 (メソッドを含む)

次の例では、直前の例 (507 ページの『例: メソッドの定義』、 519 ページの『例: クライアントの定義』、および 523 ページの『例: サブクラスの定義 (メソッドを含む)』) を更新します。

- Account クラスは、ファクトリー・データとパラメーター化ファクトリー・メソッド createAccount を追加します。こうすることで、渡される口座番号を使用して Account インスタンスの作成が可能になります。
- CheckingAccount サブクラスは、ファクトリー・データおよび多重定義のパラメーター化ファクトリー・メソッド createCheckingAccount を追加します。createCheckingAccount の 1 つのインプリメンテーションでデフォルトの収支ゼロの口座を初期化し、もう 1 つのインプリメンテーションで開始残高を渡せるようにします。クライアントは、目的のメソッドのシグニチャーと一致する引き数を渡して、メソッドを呼び出すこともできます。
- TestAccounts クライアントは、Account および CheckingAccount クラスのファクトリー・メソッドによって提供されるサービスを呼び出して、Java Check クラスのインスタンスを生成します。
- TestAccounts クライアント・プログラムからの出力を表示します。

また、上記の例の完全なソース・コードが HFS の cobol/demo/oosample サブディレクトリーに入っています。一般的には、このソースへの絶対パスは、/usr/lpp/cobol/demo/oosample です。そこで MAKE ファイルを作成して、コードのコンパイルとリンクを行うことができます。

Account クラス

```
 cbl dll,thread,pgmname(longmixed),lib
 Identification Division.
 Class-id. Account inherits Base.
 Environment Division.
 Configuration section.
 Repository.
     Class Base    is "java.lang.Object"
     Class Account is "Account".
*
 Identification division.
 Factory.
  Data division.
  Working-storage section.
  01 NumberOfAccounts pic 9(6) value zero.
*
  Procedure Division.
*
*    createAccount method to create a new Account
*    instance, then invoke the OBJECT paragraph's init
*    method on the instance to initialize its instance data:
   Identification Division.
   Method-id. "createAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 outAccount object reference Account.
*      Facilitate access to JNI services:
     Copy JNI.
   Procedure Division using by value inAccountNumber
       returning outAccount.
*      Establish addressability to JNI environment structure:
     Set address of JNIEnv to JNIEnvPtr
     Set address of JNINativeInterface to JNIEnv
     Invoke Account New returning outAccount
*      Convert outAccount to a global reference so that
*      it remains valid after return from this method:
     Call NewGlobalRef using by value JNIEnvPtr outAccount
       returning outAccount
     Invoke outAccount "init" using by value inAccountNumber
```

```
     Add 1 to NumberOfAccounts.
   End method "createAccount".
*
 End Factory.
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 AccountNumber  pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
  Procedure Division.
*
*    init method to initialize the account:
   Identification Division.
   Method-id. "init".
   Data division.
   Linkage section.
   01 inAccountNumber pic S9(9) binary.
   Procedure Division using by value inAccountNumber.
     Move inAccountNumber to AccountNumber.
   End method "init".
*
*    getBalance method to return the account balance:
   Identification Division.
   Method-id. "getBalance".
   Data division.
   Linkage section.
   01 outBalance pic S9(9) binary.
   Procedure Division returning outBalance.
     Move AccountBalance to outBalance.
   End method "getBalance".
*
*    credit method to deposit to the account:
   Identification Division.
   Method-id. "credit".
   Data division.
   Linkage section.
   01 inCredit   pic S9(9) binary.
   Procedure Division using by value inCredit.
     Add inCredit to AccountBalance.
   End method "credit".
*
*    debit method to withdraw from the account:
   Identification Division.
   Method-id. "debit".
   Data division.
   Linkage section.
   01 inDebit    pic S9(9) binary.
   Procedure Division using by value inDebit.
     Subtract inDebit from AccountBalance.
   End method "debit".
*
*    print method to display formatted account number and balance:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 PrintableAccountNumber  pic ZZZZZZ999999.
   01 PrintableAccountBalance pic $$$$,$$$,$$9CR.
   Procedure Division.
     Move AccountNumber  to PrintableAccountNumber
     Move AccountBalance to PrintableAccountBalance
     Display " Account: " PrintableAccountNumber
     Display " Balance: " PrintableAccountBalance.
   End method "print".
```

```
*
 End Object.
*
 End class Account.
```

CheckingAccount クラス (Account のサブクラス)

```
 cbl dll,thread,pgmname(longmixed),lib
 Identification Division.
 Class-id. CheckingAccount inherits Account.
 Environment Division.
 Configuration section.
 Repository.
     Class CheckingAccount is "CheckingAccount"
     Class Check           is "Check"
     Class Account         is "Account".
*
 Identification division.
 Factory.
  Data division.
  Working-storage section.
  01 NumberOfCheckingAccounts pic 9(6) value zero.
*
  Procedure Division.
*
*    createCheckingAccount overloaded method to create a new
*    CheckingAccount instance with a default balance, invoke
*    inherited instance method init to initialize the account
*    number, and increment factory data tally of checking accounts:
   Identification Division.
   Method-id. "createCheckingAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 outCheckingAccount object reference CheckingAccount.
*      Facilitate access to JNI services:
     Copy JNI.
   Procedure Division using by value inAccountNumber
       returning outCheckingAccount.
*      Establish addressability to JNI environment structure:
     Set address of JNIEnv to JNIEnvPtr
     Set address of JNINativeInterface to JNIEnv
     Invoke CheckingAccount New returning outCheckingAccount
*      Convert outCheckingAccount to a global reference so
*      that it remains valid after return from this method:
     Call NewGlobalRef using by value JNIEnvPtr outCheckingAccount
                       returning outCheckingAccount
     Invoke outCheckingAccount "init"
       using by value inAccountNumber
     Add 1 to NumberOfCheckingAccounts.
   End method "createCheckingAccount".
*
*    createCheckingAccount overloaded method to create a new
*    CheckingAccount instance, invoke inherited instance methods
*    init to initialize the account number and credit to set the
*    balance, and increment factory data tally of checking accounts:
   Identification Division.
   Method-id. "createCheckingAccount".
   Data division.
   Linkage section.
   01 inAccountNumber  pic S9(6) binary.
   01 inInitialBalance pic S9(9) binary.
   01 outCheckingAccount object reference CheckingAccount.
     Copy JNI.
   Procedure Division using by value inAccountNumber
                                     inInitialBalance
       returning outCheckingAccount.
```

```
      Set address of JNIEnv to JNIEnvPtr
      Set address of JNINativeInterface to JNIEnv
      Invoke CheckingAccount New returning outCheckingAccount
      Call NewGlobalRef using by value JNIEnvPtr outCheckingAccount
                        returning outCheckingAccount
      Invoke outCheckingAccount "init"
        using by value inAccountNumber
      Invoke outCheckingAccount "credit"
        using by value inInitialBalance
      Add 1 to NumberOfCheckingAccounts.
    End method "createCheckingAccount".
*
 End Factory.
*
 Identification division.
 Object.
  Data division.
  Working-storage section.
  01 CheckFee pic S9(9) value 1.
  Procedure Division.
*
*    processCheck method to get the check amount and payee,
*    add the check fee, and invoke inherited methods debit
*    to debit the payer and credit to credit the payee:
   Identification Division.
   Method-id. "processCheck".
   Data division.
   Local-storage section.
   01 amount pic S9(9) binary.
   01 payee usage object reference Account.
   Linkage section.
   01 aCheck usage object reference Check.
   Procedure Division using by value aCheck.
     Invoke aCheck "getAmount" returning amount
     Invoke aCheck "getPayee" returning payee
     Invoke payee  "credit" using by value amount
     Add checkFee to amount
     Invoke self   "debit"  using by value amount.
   End method "processCheck".
*
*    print method override to display account status:
   Identification Division.
   Method-id. "print".
   Data division.
   Local-storage section.
   01 printableFee pic $$,$$$,$$9.
   Procedure Division.
     Invoke super "print"
     Move CheckFee to printableFee
     Display " Check fee: " printableFee.
   End method "print".
*
 End Object.
*
 End class CheckingAccount.
```

Check クラス

```
/**
 * A Java class for check information
 */
public class Check {
  private CheckingAccount payer;
  private Account         payee;
  private int             amount;

  public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
```

```
    payer=inPayer;
    payee=inPayee;
    amount=inAmount;
  }

  public int getAmount() {
    return amount;
  }

  public Account getPayee() {
    return payee;
  }
}
```

TestAccounts クライアント・プログラム

```
 cbl dll,thread,pgmname(longmixed)
 Identification division.
 Program-id. "TestAccounts" recursive.
 Environment division.
 Configuration section.
 Repository.
     Class Account         is "Account"
     Class CheckingAccount is "CheckingAccount"
     Class Check           is "Check".
 Data Division.
*  Working data is declared in LOCAL-STORAGE instead of
*  WORKING-STORAGE so that each thread has its own copy:
 Local-storage section.
 01  anAccount        usage object reference Account.
 01  aCheckingAccount usage object reference CheckingAccount.
 01  aCheck           usage object reference Check.
 01  payee            usage object reference Account.
*
 Procedure division.
 Test-Account-section.
     Display "Test Account class"
*  Create account 123456 with 0 balance:
     Invoke Account "createAccount"
       using by value 123456
       returning anAccount
*  Deposit 500 to the account:
     Invoke anAccount "credit" using by value 500
     Invoke anAccount "print"
     Display space
*
     Display "Test CheckingAccount class"
*  Create checking account 777777 with balance of 300:
     Invoke CheckingAccount "createCheckingAccount"
       using by value 777777 300
       returning aCheckingAccount
*  Set account 123456 as the payee:
     Set payee to anAccount
*  Initialize check for 125 to be paid by account 777777 to payee:
     Invoke Check New
       using by value aCheckingAccount, payee, 125
       returning aCheck
*  Debit the payer, and credit the payee:
     Invoke aCheckingAccount "processCheck"
       using by value aCheck
     Invoke aCheckingAccount "print"
     Invoke anAccount "print"
*
     Stop Run.
 End program "TestAccounts".
```

TestAccounts クライアント・プログラムによって生成される出力

```
Test Account class
 Account:       123456
 Balance:         $500

Test CheckingAccount class
 Account:       777777
 Balance:         $174
 Check fee:         $1
 Account:       123456
 Balance:         $625
```

関連タスク

516 ページの『クラスのインスタンスの生成と初期化』
525 ページの『ファクトリー・メソッドの定義』
527 ページの『ファクトリーまたは静的メソッドの呼び出し』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』

第 29 章 Java メソッドとの通信

Java との言語間インターオペラビリティーを確保するには、以下の項目に関する特定の規則およびガイドラインに従う必要があります。

- Java Native Interface (JNI) のサービスの使用
- データ型のコーディング
- COBOL プログラムのコンパイル

Java で書き込まれたメソッドを COBOL プログラムから呼び出したり、COBOL で書き込まれたメソッドを Java プログラムから呼び出したりすることができます。Java の基本オブジェクト機能に対応するには、COBOL オブジェクト指向言語をコーディングする必要があります。追加の Java 機能に対応するには、JNI サービスを呼び出すことができます。

Java プログラムはマルチスレッド化され、非同期シグナルを用いる場合があります。したがって、THREAD オプションを使用して COBOL プログラムをコンパイルしてください。

546 ページの『例: COBOL で書き込む J2EE クライアント』

関連タスク

116 ページの『COBOL での国別データ (Unicode) の使用』
『JNI サービスへのアクセス』
540 ページの『Java とのデータ共用』
491 ページの『第 28 章 オブジェクト指向プログラムの作成』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
481 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照

Java 2 Enterprise Edition Developer's Guide

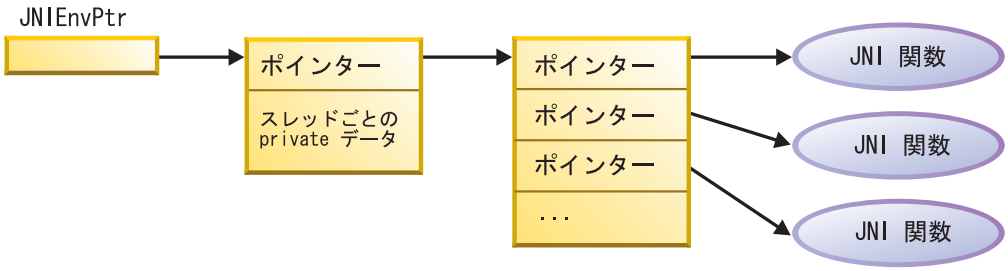
JNI サービスへのアクセス

Java Native Interface (JNI) は、COBOL と Java を併用するアプリケーションを開発する際に使用できる多くの呼び出し可能サービスを提供します。このサービスへのアクセスを円滑に行うには、COBOL プログラムの LINKAGE SECTION に JNI.cpy をコピーします。このコピーブックには、以下が含まれています。

- Java JNI タイプに対応する COBOL データ定義
- 呼び出し可能サービス関数にアクセスするための関数ポインターが含まれている JNI 環境構造 JNINativeInterface

次の図に示すように、JNI 環境ポインターからの 2 つのレベルの間接化技法によって JNI 環境構造を取得します。

© Copyright IBM Corp. 1991, 2001



特殊レジスター JNIEnvPtr を使用して、JNI 環境ポインターを参照し、JNI 環境構造のアドレスを取得します。JNIEnvPtr は、USAGE POINTER を使って暗黙的に定義されます。それを受取データ項目として使用することはできません。

したがって、JNI 環境構造の内容を参照する前に、以下のステートメントをコーディングして、そのアドレス可能度を確立する必要があります。

```
Linkage section.
COPY JNI
. . .
Procedure division.
. . .
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
. . .
```

このコードは、以下の項目のアドレスを設定します。

- JNIEnv は JNI.cpy が提供するポインター・データ項目です。JNIEnvPtr は、環境ポインターが含まれている COBOL 特殊レジスターです。
- JNINativeInterface は JNI.cpy に含まれている COBOL グループ構造です。この構造には、JNI 呼び出し可能サービスの関数ポインターの配列が含まれている JNI 環境構造がマップされています。

上記のステートメントをコーディングした後に、関数ポインターを参照する CALL ステートメントを使って、JNI 呼び出し可能サービスにアクセスすることができます。次の例に示すように、環境ポインターを必要とするサービスに、最初の引き数として JNIEnvPtr 特殊レジスターを渡すことができます。

```
01 InputArrayObj usage object reference jlongArray.
01 ArrayLen pic S9(9) comp-5.
. . .
    Call GetArrayLength using by value JNIEnvPtr InputArrayObj
      returning ArrayLen
```

すべての引き数を値によって JNI 呼び出し可能サービスに渡します。

一部の JNI 呼び出し可能サービスは、Java クラス・オブジェクト・リファレンスを引き数として必要とします。クラスに関連付けられたクラス・オブジェクトへの参照を取得するには、以下の JNI 呼び出し可能サービスのどちらかを使用します。

- GetObjectClass
- FindClass

制約事項: JNI 環境ポインターはスレッド固有です。したがって、それをスレッドからスレッドへ渡すことはできません。

関連タスク
538 ページの『ローカルおよびグローバル参照の管理』
『Java 例外の取り扱い』
509 ページの『クライアントの定義』

関連参照
673 ページの『付録 F. JNI.cpy』
Java Native Interface

Java 例外の取り扱い

JNI サービスを使用して、Java 例外をスローしたり、キャッチしたりします。

COBOL プログラムからの例外のスロー

COBOL メソッドから Java 例外をスローするには、以下のサービスのどちらかを使用します。

- Throw
- ThrowNew

スローしたオブジェクトは、java.lang.Throwable のサブクラスのインスタンスにする必要があります。

Java 仮想マシン (JVM) は、呼び出しが含まれているメソッドが完了して JVM に戻されるまで、スローした例外の認識も処理も行いません。

Java 例外のキャッチ

Java 例外をスローした可能性のあるメソッドを呼び出した後に、以下の JNI サービスを使用して、例外が実際に発生したかをテストしたり、例外を処理したり、例外をクリアしたり (クリアが適切な場合) します。

- ExceptionOccurred
- ExceptionCheck
- ExceptionDescribe
- ExceptionClear

エラー分析を行うには、戻された例外オブジェクトによってサポートされるメソッドを使用します。このオブジェクトは、java.lang.Throwable のインスタンスです。

『例: Java 例外の取り扱い』

例: Java 例外の取り扱い

次の例では、Java からの例外をキャッチし、エラー分析を行うために java.lang.Throwable の PrintStackTrace メソッドを用いるための JNI サービスの使用を示します。

```
Repository.
    Class JavaException is "java.lang.Exception".
. . .
Local-storage section.
01 ex usage object reference JavaException.
Linkage section.
COPY "JNI.cpy".
. . .
```

```
Procedure division.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    . . .
    Invoke anObj "someMethod"
    Perform ErrorCheck
. . .
ErrorCheck.
    Call ExceptionOccurred
        using by value JNIEnvPtr
        returning ex
    If ex not = null then
        Call ExceptionClear using by value JNIEnvPtr
        Display "Caught an unexpected exception"
        Invoke ex "printStackTrace"
        Stop run
    End-if
```

ローカルおよびグローバル参照の管理

Java 仮想マシンは、COBOL メソッドなど、ネイティブ・メソッドで使用するオブジェクト・リファレンスを追跡します。この追跡によって、ガーベッジ・コレクションの際、まだ使用中のオブジェクトが解放されないようにします。オブジェクト・リファレンスには、以下の 2 つのクラスがあります。

ローカル参照
ローカル参照は、呼び出したメソッドが実行している間だけ有効であり、ネイティブ・メソッドが戻った後、自動的に解放されます。

グローバル参照
グローバル参照は、明示的に削除するまで有効です。グローバル参照は、JNI サービス NewGlobalRef を使用して、ローカル参照から作成することができます。

以下のオブジェクト・リファレンスは常にローカルです。

- メソッド・パラメーターとして受け取られるオブジェクト・リファレンス
- メソッド内からの JNI 関数への呼び出しによって戻されるオブジェクト・リファレンス
- INVOKE . . . NEW ステートメントを使用して、メソッド内に作成するオブジェクト・リファレンス

ローカル参照またはグローバル参照のいずれかをオブジェクト・リファレンス引き数として JNI サービスに渡すことができます。

メソッドをコーディングして、グローバル参照を RETURNING 値として戻すことができます。ローカル参照は、メソッドに作成したオブジェクトに戻すことはできません。戻すと、無効になります。まず、参照をグローバル参照に変換してから、それをグローバル参照に戻します。

ローカル参照は、それが作成されたスレッド内でのみ有効です。スレッドを別のスレッドに渡すことはできません。

ローカル参照の削除、保管、解放

ローカル参照は、メソッド内で、随時に手動で削除できます。

ローカル参照は、メソッドの LOCAL-STORAGE SECTION に定義したオブジェクト・リファレンス内にのみ保管します。SET ステートメントを使って、ローカル参照をオブジェクト・インスタンス変数、ファクトリー変数、またはメソッドの WORKING-STORAGE SECTION 内の変数に保管する場合には、まず、ローカル参照をグローバル参照に変換します。そのようにしない場合には、エラーが発生します。これらのストレージ域はメソッドが戻るときもそのまま持続しますが、ローカル参照は無効になるからです。

ほとんどのケースにおいて、メソッドが戻るときに発生するローカル参照の自動解放に依存することができます。ただし、一部のケースにおいては、JNI サービス DeleteLocalRef を使用して、ローカル参照を明示的に解放する必要があります。以下に、明示的解放が適切な 2 つの状態を示します。

- メソッドにおいて、ラージ・オブジェクトにアクセスすることで、オブジェクトへのローカル参照を作成します。膨大な計算を行った後で、メソッドが戻ります。このラージ・オブジェクトを別の計算に必要としない場合には、このオブジェクトを解放してください。このローカル参照は、ガーベッジ・コレクションの間に オブジェクトを解放する妨げになるからです。
- メソッドに多数のローカル参照を作成しますが、それらのすべてのローカル参照を同時には使用しません。Java 仮想マシンは、各ローカル参照を追跡するためのスペースが必要です。したがって、システムがメモリー不足にならないように、不要になったローカル参照を解放してください。

  たとえば、COBOL メソッドにおいて、大規模な配列のオブジェクトをループし、エレメントをローカル参照として検索し、それぞれの反復ごとに 1 つのエレメントを操作します。それぞれの反復後に、配列エレメントへのローカル参照を解放することができます。

JNI サービス

以下の呼び出し可能サービスを使用して、ローカル参照およびグローバル参照を管理します。

サービス	入力引き数	戻り値	目的
NewGlobalRef	• JNI 環境ポインター • ローカルまたはグローバル・オブジェクト・リファレンス	グローバル参照、またはシステムがメモリー不足のときは NULL	入力オブジェクト・リファレンスが参照するオブジェクトに新規グローバル参照を作成する
DeleteGlobalRef	• JNI 環境ポインター • グローバル・オブジェクト・リファレンス	なし	入力オブジェクト・リファレンスが参照するオブジェクトへのグローバル参照を削除する
DeleteLocalRef	• JNI 環境ポインター • ローカル・オブジェクト・リファレンス	なし	入力オブジェクト・リファレンスが参照するオブジェクトへのローカル参照を削除する

Java アクセス制御

Java アクセス修飾子 protected および private は、Java Native Interface を使用すると、強制されません。したがって、たとえば、COBOL プログラムは、Java クラ

イアントからは呼び出し不可能であった、protected または private Java メソッドを呼び出すことができます。この使用法はお勧めできません。

Java とのデータ共用

COBOL データ型には、Java データ型と同じものがありますが、同じでないものもあります。Java データ型と同じデータ型を、INVOKE ステートメントの USING 句に引き数として渡したり、INVOKE ステートメントの RETURNING 値として受け取ったりすることができます。同様に、これらのデータ型を Java メソッドから USING 句のパラメーターとして受け取ったり、COBOL メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句の値として戻したりすることができます。

配列およびストリングを渡したり受け取ったりするには、以下のように、それらをオブジェクト・リファレンスとして宣言します。

- 提供されている特殊配列クラスのうちの 1 つのインスタンスが含まれているオブジェクト・リファレンスとして、配列を宣言します。
- jstring クラスのインスタンスが含まれているオブジェクト・リファレンスとして、ストリングを宣言します。

グループ・データ構造を渡したり、または受け取ったりするには、Java レコード・クラスを作成して、構造を表したり、クラスのオブジェクト・インスタンスを渡したり、または受け取ったりします。「IBM VisualAge for Java エンタープライズ版」の「エンタープライズ・アクセス・ビルダー」を使用して、レコード・クラスを作成できます。

関連タスク

『COBOL および Java での相互運用可能データ型のコーディング』
541 ページの『Java の配列とストリングの宣言』
542 ページの『Java 配列の取り扱い』
543 ページの『Java ストリングの取り扱い』
514 ページの『メソッドの呼び出し (INVOKE)』
455 ページの『第 25 章 データの共用』

COBOL および Java での相互運用可能データ型のコーディング

Java と通信するときは、この表に記述されている COBOL データ型だけを使用してください。

Java の基本データ型	対応する COBOL データ型
boolean[1]	PIC X の後に以下の形式とまったく同じ 2 つの条件名を記述する。 *level-number data-name* PIC X. 88 *data-name*-false value X'00'. 88 *data-name*-true value X'01' through X'FF'.
byte[1]	1 バイト英数字: PIC X または PIC A
short	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 文節付き。ここで、1<=*n*<=4
int	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 文節付き。ここで、5<=*n*<=9

Java の基本データ型	対応する COBOL データ型
long	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(*n*) の PICTURE 文節。ここで、10<=*n*<=18
float[2]	USAGE COMP-1
double[2]	USAGE COMP-2
char	1 国別文字: PIC N USAGE NATIONAL
クラス型 (オブジェクト・リファレンス)	USAGE OBJECT REFERENCE *class-name*

1. boolean と byte は、両方とも PIC X に対応するため、区別する必要があります。PIC X は、この表に記述されている 2 つの条件名で引き数またはパラメーターを定義したときに限って、boolean として解釈されます。それ以外の場合、PIC X データ項目は、Java byte 型として解釈されます。
2. Java 浮動小数点データは IEEE 浮動小数点で表します。Enterprise COBOL は、16 進数浮動小数点表記を使用します。INVOKE ステートメントを使用して浮動小数点引き数を渡すとき、または、浮動小数点データを Java メソッドから受け取るとき、引き数およびデータは必要に応じて、自動的に変換されます。

Java の配列とストリングの宣言

Java と通信するときは、次の表に記述されている COBOL データ型をコーディングして、配列とストリングの宣言を、それぞれ、特殊配列クラスと jstring で行います。

Java データ型	対応する COBOL データ型
boolean[ ]	オブジェクト・リファレンス jbooleanArray
byte[ ]	オブジェクト・リファレンス jbyteArray
short[ ]	オブジェクト・リファレンス jshortArray
int[ ]	オブジェクト・リファレンス jintArray
long[ ]	オブジェクト・リファレンス jlongArray
char[ ]	オブジェクト・リファレンス jcharArray
String[ ]	オブジェクト・リファレンス jstring

Java との配列およびストリングのインターオペラビリティーを確保するために、これらの特殊クラスを使用するときは、次の例に示すように、REPOSITORY 段落で使用するクラスの記入項目をコーディングしなければなりません。

```
Configuration section.
Repository.
    Class jbooleanArray is "jbooleanArray".
```

以下の Java 配列型は現在、COBOL プログラムとの相互協調処理にはサポートされていません。

Java データ型	対応する COBOL データ型
float[ ]	オブジェクト・リファレンス jfloatArray
double[ ]	オブジェクト・リファレンス jdoubleArray

Java データ型	対応する COBOL データ型
Object[ ]	オブジェクト・リファレンス jobjectArray

関連タスク
497 ページの『クラスを定義する REPOSITORY 段落』

Java 配列の取り扱い

COBOL プログラムの配列を表すには、その配列の Java の型に対応するデータ型の単一基本項目が含まれているグループ構造をコーディングします。その配列に適した OCCURS または OCCURS DEPENDING ON 文節を指定します。

たとえば、次のコードは、jlongArray オブジェクトから 500 以下の整数値を受け取る構造を指定します。

```
01 longArray.
    02 X pic S9(9) comp-5 occurs 1 to 500 times depending on N.
```

特殊な Java 配列クラスのオブジェクトを操作するには、JNI が提供するサービスを呼び出します。サービスを使用して、配列の個々のエレメントにアクセスして設定し、呼び出したサービスを使って、以下を行います。

目的	JNI サービス
Java 配列オブジェクト内のエレメント数を取得する	GetArrayLength
COBOL グループとして表されている配列から新しい Java 配列オブジェクトを作成する	NewBooleanArray NewByteArray NewCharArray NewShortArray NewIntArray NewLongArray
配列エレメントを Java 配列からストレージ・バッファーに抽出する。ポインターをストレージ・バッファーに戻すことができます。ポインターは、LINKAGE SECTION に定義される COBOL グループ構造のアドレスとして使用することができます。	GetBooleanArrayElements GetByteArrayElements GetCharArrayElements GetShortArrayElements GetIntArrayElements GetLongArrayElements
Java 配列から抽出したエレメントが含まれているストレージ・バッファーを解放する。更新した配列値を配列オブジェクトに条件付でマップし直すことができます。	ReleaseBooleanArrayElements() ReleaseByteArrayElements() ReleaseCharArrayElements() ReleaseShortArrayElements() ReleaseIntArrayElements() ReleaseLongArrayElements()

543 ページの『例: Java int 配列の処理』

関連タスク
540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
541 ページの『Java の配列とストリングの宣言』
535 ページの『JNI サービスへのアクセス』

例: Java int 配列の処理

この例では、Java 配列クラスと JNI サービスを使用した、COBOL での Java 配列の処理を説明します。

```
cbl lib,thread,dll
Identification division.
Class-id. OOARRAY inherits Base.
Environment division.
Configuration section.
Repository.
    Class Base is "java.lang.Object"
    Class jintArray is "jintArray".
Identification division.
Object.
Procedure division.
  Identification division.
  Method-id. "ProcessArray".
  Data Division.
  Local-storage section.
  01 intArrayPtr pointer.
  01 intArrayLen pic S9(9) comp-5.
  Linkage section.
     COPY JNI.
  01 inIntArrayObj usage object reference jintArray.
  01 intArrayGroup.
     02 X pic S9(9) comp-5
         occurs 1 to 1000 times depending on intArrayLen.
  Procedure division using by value inIntArrayObj.
      Set address of JNIEnv to JNIEnvPtr
      Set address of JNINativeInterface to JNIEnv

      Call GetArrayLength
        using by value JNIEnvPtr inIntArrayObj
        returning intArrayLen
      Call GetIntArrayElements
        using by value JNIEnvPtr inIntArrayObj 0
        returning IntArrayPtr
      Set address of intArrayGroup to intArrayPtr

*  . . . process the array elements X(I) . . .

      Call ReleaseIntArrayElements
        using by value JNIEnvPtr inIntArrayObj intArrayPtr 0.
  End method "ProcessArray".
End Object.
End class OOARRAY.
```

Java ストリングの取り扱い

Java ストリング・データは、Unicode で表します。COBOL プログラムでストリングを表すには、jstring クラスのオブジェクト・リファレンスとして、それを宣言します。続いて、JNI サービスの 2 つのセットのいずれか 1 つを使用して、オブジェクトから COBOL 英数字または国別 (Unicode) データを抽出します。

Unicode サービス

以下の標準サービスを使用して、jstring オブジェクト・リファレンスと COBOL 国別データ項目 (PIC N(n)) 間の変換を行います。

サービス	入力引き数	戻り値
NewString	• JNI 環境ポインター • COBOL 国別データ項目などの、Unicode ストリングへのポインター • ストリングの文字数。2 進数フルワード	jstring オブジェクト・リファレンス。
GetStringLength	• JNI 環境ポインター • jstring オブジェクト・リファレンス	jstring オブジェクト・リファレンスの Unicode 文字数。2 進数フルワード。
GetStringChars	• JNI 環境ポインター • jstring オブジェクト・リファレンス • ブール・データ項目へのポインター	• jstring オブジェクトから抽出した Unicode 文字の配列へのポインター。 • ブール・データ項目へのポインターが NULL でないとき、ブール値は、ストリングのコピーが作成される場合には true に、コピーが作成されない場合には false に設定されます。戻されたポインターは、ReleaseStringChars が呼び出されるまで有効です。
ReleaseStringChars	• JNI 環境ポインター • jstring オブジェクト・リファレンス • GetStringChars から戻された Unicode 文字の配列へのポインター	なし。配列のストレージは解放されます。

上記のサービスへのアクセスは、JNI 環境構造 JNINativeInterface の関数ポインターを使用して行います。

EBCDIC サービス

JNI の拡張機能である、以下の z/OS サービスを使用して、jstring オブジェクト・リファレンスと COBOL 英数字データ (PIC X(n)) 間の変換を行います。

サービス	入力引き数	戻り値
NewStringPlatform	• JNI 環境ポインター • jstring オブジェクトに変換するヌル終了 EBCDIC 文字ストリングへのポインター • 結果のための jstring オブジェクト・リファレンスへのポインター • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング[1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。jstring オブジェクト・リファレンス・ポインターはヌルに設定されます。

サービス	入力引き数	戻り値
GetStringPlatformLength	• JNI 環境ポインター • 長さのための Jstring オブジェクト・リファレンス • 結果のための 2 進数フルワード値へのポインター • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング[1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。jstring オブジェクト・リファレンス・ポインターはヌルに設定されます。 3 番目の引き数に、2 番目の引き数が参照するヌル終了バイトなど、変換した Java ストリングを保持するために必要な出力バッファーの長さ (バイト単位) を戻します。
GetStringPlatform	• JNI 環境ポインター • ヌル終了ストリングに変換する Jstring オブジェクト・リファレンス • 変換済みストリングのための出力バッファーへのポインター • 2 進数フルワード整数としての出力バッファーの長さ • ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング[1] として表す	2 進数フルワード整数としての戻りコード: **0** 正常。 **-1** 誤った形式の入力データまたは正しくない入力文字。 **-2** 非サポート・エンコード。出力ストリングはヌル・ストリングに設定されます。 **-3** 変換バッファーがいっぱいです。

1. ポインターがヌルの場合、Java file.encoding プロパティーからのエンコードが使用されます。

EBCDIC サービスは、「IBM Developer Kit for OS/390, Java 2 Technology Edition」に添付された DLL としてパッケージされています。CALL *literal* ステートメントを使用して、これらのサービスを呼び出します。呼び出しは、オブジェクト指向言語を使用する COBOL プログラムのリンク手順に含める必要がある、libjvm.x DLL サイド・デックを介して解決されます。

たとえば、次のコードは、EBCDIC ストリング 'MyConverter' から Java ストリング・オブジェクトを作成します。(このコード・フラグメントは、次のセクションで詳細に説明する、J2EE クライアント・プログラムからのものです。)

```
Move z"MyConverter" to stringBuf
Call "NewStringPlatform"
  using by value JNIEnvPtr
                 address of stringBuf
                 address of jstring1
                 0
  returning rc
```

EBCDIC サービスが、COBOL プログラムから呼び出す唯一の JNI サービスである場合には、JNI.cpy コピーブックをコピーする必要はないし、JNI 環境ポインターとのアドレス可能度を確立する必要もありません。

上記のサービスの詳細については、「IBM Developer Kit for OS/390, Java 2 Technology Edition」に添付された jni_convert.h を参照してください。

上記のサービスへのアクセスは、JNI 環境構造 JNINativeInterface の関数ポインターを使用して行います。

UTF-8 サービス

Java Native Interface は、jstring オブジェクト・リファレンスと UTF-8 ストリングの間の変換用のサービスも提供します。これらのサービスは、COBOL からの使用にはお勧めしません。

関連タスク

540 ページの『COBOL および Java での相互運用可能データ型のコーディング』
541 ページの『Java の配列とストリングの宣言』
535 ページの『JNI サービスへのアクセス』
307 ページの『第 17 章 オブジェクト指向プログラムのコンパイル、リンク、および実行』
116 ページの『COBOL での国別データ (Unicode) の使用』

例: COBOL で書き込む J2EE クライアント

次の例では、J2EE 準拠 EJB サーバー上で実行している Enterprise Java Beans にアクセスできる COBOL クライアント・プログラムを示します。このプログラムは、*Java 2 Enterprise Edition Developer's Guide* の章『Getting Started』に記述されている J2EE クライアント・プログラムと同等です。

インプリメンテーションの内容を比較する上でのユーザーの便宜のために、COBOL インプリメンテーションの後に、「Guide」から抜粋したサンプル・クライアントの Java 版を記載します。Enterprise Java Bean は、サンプル・ガイドに記載されている簡単な通貨コンバーター EJB の Java インプリメンテーションであると想定されています。したがって、ここには記述されていません。

COBOL クライアント、ConverterClient.cbl:

```
 Process pgmname(longmixed),lib,dll,thread
*****************************************************************
* Demo J2EE client written in COBOL.                            *
*                                                               *
* Based on the sample J2EE client written in Java, that is      *
* given in the "Getting Started" chapter of "The Java(tm) 2     *
* Enterprise Edition Developer's Guide"                         *
*                                                               *
* The client:                                                   *
*   - locates the home interface of a session Enterprise Java   *
*     Bean  (a simple currency converter Bean)                  *
*   - creates an Enterprise Bean instance                       *
*   - invokes a Business Method (currency conversion)           *
*****************************************************************
 Identification division.
 Program-id. "ConverterClient" is recursive.
 Environment Division.
 Configuration section.
 Repository.
     Class InitialContext is "javax.naming.InitialContext"
     Class PortableRemoteObject
                          is "javax.rmi.PortableRemoteObject"
     Class JavaObject     is "java.lang.Object"
```

```
     Class JavaClass      is "java.lang.Class"
     Class JavaException  is "java.lang.Exception"
     Class jstring        is "jstring"
     Class Converter      is "Converter"
     Class ConverterHome  is "ConverterHome".
   Data division.
 Working-storage section.
 01 initialCtx        object reference InitialContext.
 01 obj               object reference JavaObject.
 01 classObj          object reference JavaClass.
 01 ex                object reference JavaException.
 01 currencyConverter object reference Converter.
 01 home              object reference ConverterHome.
 01 homeObject redefines home object reference JavaObject.
 01 jstring1          object reference jstring.
 01 stringBuf         pic X(500) usage display.
 01 len               pic s9(9) comp-5.
 01 rc                pic s9(9) comp-5.
 01 amount            comp-2.
 Linkage section.
     Copy JNI.
 Procedure division.
     Set address of JNIenv to JNIEnvPtr
     Set address of JNINativeInterface to JNIenv

*****************************************************************
* Create JNDI naming context.                                   *
*****************************************************************
     Invoke InitialContext New returning initialCtx
     Perform JavaExceptionCheck

*****************************************************************
* Create a jstring object for the string "MyConverter" for use  *
* as argument to the lookup method.                             *
*****************************************************************
     Move z"MyConverter" to stringBuf
     Call "NewStringPlatform"
       using by value JNIEnvPtr
                      address of stringBuf
                      address of jstring1
                      0
       returning rc
     If rc not = zero then
        Display "Error occurred creating jstring object"
        Stop run
     End-if

*****************************************************************
* Use the lookup method to obtain a reference to the home       *
* object bound to the name "MyConverter".  (This is the JNDI    *
* name specified when deploying the J2EE application.)          *
*****************************************************************
     Invoke initialCtx "lookup" using by value jstring1
       returning obj
     Perform JavaExceptionCheck

*****************************************************************
* Narrow the home object to be of type ConverterHome.           *
* First obtain class object for the ConverterHome class, by     *
* passing the null-terminated ASCII string "ConverterHome" to   *
* the FindClass API.  Then use this class object as the         *
* argument to the static method "narrow".                       *
*****************************************************************
     Move z"ConverterHome" to stringBuf
     Call "__etoa"
       using by value address of stringBuf
       returning len
```

```
     If len = -1 then
        Display "Error occurred on ASCII conversion"
        Stop run
     End-if
     Call FindClass
       using by value JNIEnvPtr
                      address of stringBuf
       returning classObj
     If classObj = null
        Display "Error occurred locating ConverterHome class"
        Stop run
     End-if
     Invoke PortableRemoteObject "narrow"
       using by value obj
                      classObj
       returning homeObject
     Perform JavaExceptionCheck

*****************************************************************
* Create the ConverterEJB instance and obtain local object      *
* reference for its remote interface                            *
*****************************************************************
     Invoke home "create" returning currencyConverter
     Perform JavaExceptionCheck

*****************************************************************
* Invoke business methods                                       *
*****************************************************************
     Invoke currencyConverter "dollarToYen"
       using by value +100.00E+0
       returning amount
     Perform JavaExceptionCheck

     Display amount

     Invoke currencyConverter "yenToEuro"
       using by value +100.00E+0
       returning amount
     Perform JavaExceptionCheck

     Display amount

*****************************************************************
* Remove the object and return.                                 *
*****************************************************************
     Invoke currencyConverter "remove"
     Perform JavaExceptionCheck

     Goback
     .

*****************************************************************
* Check for thrown Java exceptions                              *
*****************************************************************
 JavaExceptionCheck.
     Call ExceptionOccurred using by value JNIEnvPtr
       returning ex
     If ex not = null then
        Call ExceptionClear using by value JNIEnvPtr
        Display "Caught an unexpected exception"
        Invoke ex "PrintStackTrace"
        Stop run
     End-if
     .
 End program "ConverterClient".
```

同等の Java クライアント、ConverterClient.java:

```
/*
 *
 * Copyright 2000 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 *
 */

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import Converter;
import ConverterHome;

public class ConverterClient {

   public static void main(String[] args) {
       try {
           Context initial = new InitialContext();
           Object objref = initial.lookup("MyConverter");

           ConverterHome home =
               (ConverterHome)PortableRemoteObject.narrow(objref,
                                           ConverterHome.class);

           Converter currencyConverter = home.create();

           double amount = currencyConverter.dollarToYen(100.00);
           System.out.println(String.valueOf(amount));
           amount = currencyConverter.yenToEuro(100.00);
           System.out.println(String.valueOf(amount));

           currencyConverter.remove();

       } catch (Exception ex) {
           System.err.println("Caught an unexpected exception!");
           ex.printStackTrace();
       }
   }
}
```

第 6 部 特殊処理

第 30 章 割り込みとチェックポイント・リスタート 553
チェックポイントの設定. 553
チェックポイントの設計. 554
チェックポイントの正常実行に関するテスト . . 555
チェックポイント・データ・セットを定義する DD ステートメント 555
例: チェックポイント・データ・セットの定義 556
チェックポイント時に生成されるメッセージ . . 556
プログラムの再始動 557
自動再始動の要求 557
据え置き再始動の要求 558
据え置き再始動を要求する形式 559
例: 据え置き再始動の要求 559
再始動のためのジョブの再実行依頼 560
例: 特定のチェックポイント・ステップでジョブを再始動する 560
例: ステップ再始動を要求する 560
例: ステップ再始動のためにジョブを再実行依頼する 561
例: チェックポイント・リスタートのためにジョブを再実行依頼する 561

第 31 章 2 桁年号の日付の処理 563
2000 年言語拡張 (MLE) 564
言語拡張の原則および目標 564
日付関連ロジック問題の解決 566
世紀ウィンドウの使用 567
例: 世紀ウィンドウ 567
内部ブリッジングの使用. 568
例: 内部ブリッジング 569
全フィールド拡張への移行 569
例: ファイルを拡張日付形式に変換する . . 570
年先頭型、年単独型、および年末尾型日付フィールドの使用 571
互換性のある日付 572
例: 年先頭型日付フィールドの比較 573
他の日付形式の使用 573
例: 年部分の分離 574
日付としてのリテラルの操作 574
仮定による世紀ウィンドウ 575
非日付の取り扱い 576
トリガーおよび限界の設定 577
例: 限界の使用 578
符号条件の使用. 579
日付によるソートおよびマージ 579
例: 日付および時刻によるソート 580
日付フィールドへの算術の実行 581
ウィンドウ化日付フィールドからのオーバーフローを考慮する 581
評価の順序の指定 582
日付処理の明示的制御 583
DATEVAL の使用 583
UNDATE の使用 584
例: DATEVAL 584
例: UNDATE 584
日付関連診断メッセージの分析および回避. . . . 585
日付の処理における問題の回避 587
パック 10 進数フィールドでの問題の回避. . . 587
拡張日付フィールドからウィンドウ化日付フィールドへの移動 587

© Copyright IBM Corp. 1991, 2001

第 30 章 割り込みとチェックポイント・リスタート

時間を延長してプログラムを実行していると、ジョブ終了になる前に、割り込みによって処理が停止することがあります。z/OS のチェックポイント・リスタート機能を使用すれば、割り込みを受けたプログラムを、ジョブ・ステップの先頭からまたは設定したチェックポイントから再始動することができます。

チェックポイント・リスタート機能は、多くの余分な処理を引き起こすので、マシン誤動作、入出力エラー、またはオペレーターの意図的な介入が原因の割り込みであると予想される場合にだけ使用するようにしてください。

チェックポイント・ルーチンは、プログラムが入っている COBOL ロード・モジュールから開始されます。プログラムの実行中に、チェックポイント・ルーチンは、COBOL RERUN 文節を使用して指定されたポイントでレコードを作成します。チェックポイント・レコードには、プログラムがそのチェックポイントに達したときにレジスターおよび主記憶域の中に保管されていた情報のスナップショットが入れられます。

再始動ルーチンは、割り込みを受けたプログラムを再始動します。プログラムが割り込みを受けた後、いつでも再始動することができます。すなわち、即時に再始動すること (自動再始動) も、後で再始動すること (据え置き再始動) もできます。

関連タスク
『チェックポイントの設定』
557 ページの『プログラムの再始動』
560 ページの『再始動のためのジョブの再実行依頼』
z/OS DFSMS: Checkpoint/Restart

関連参照
555 ページの『チェックポイント・データ・セットを定義する DD ステートメント』
556 ページの『チェックポイント時に生成されるメッセージ』
559 ページの『据え置き再始動を要求する形式』

チェックポイントの設定

チェックポイントを設定するには、ジョブ制御ステートメントを使用し、ENVIRONMENT DIVISION で RERUN 文節を使用してください。それぞれの RERUN 文節を特定の COBOL ファイルと関連付ける必要があります。

RERUN 文節は、COBOL ファイルの指定された数のレコードが処理されるたび、または END OF VOLUME に達したときに、チェックポイント・レコードをチェックポイント・データ・セットに書き込むことを示します。RERUN 文節は、EXTERNAL 属性で定義されたファイルには使用できません。

複数の COBOL ファイルからのチェックポイント・レコードを 1 つのチェックポイント・データ・セットに書きこむことができますが、チェックポイント・レコー

© Copyright IBM Corp. 1991, 2001

ド専用の別個のデータ・セットを使用する必要があります。チェックポイント・レコードをプログラム・データ・セットの 1 つに組み込むことはできません。

チェックポイント・データ・セットは順次編成でなければなりません。チェックポイントを、VSAM データ・セット、または拡張形式 QSAM データ・セットに割り振られたデータ・セットに書き込むことはできません。また、実行単位内のいずれかのプログラムが拡張形式 QSAM データ・セットをオープンしている場合にも、チェックポイントを取ることはできません。

チェックポイント・レコードは、DD ステートメントによって定義されたチェックポイント・データ・セットに書き込まれます。DD ステートメントでは、次のチェックポイント方式も選択します。

単一 (単一のチェックポイントを保管します)
ある時点では、チェックポイント・レコードは 1 つだけ存在します。最初のチェックポイント・レコードが作成されると、それ以降のチェックポイント・レコードは前のレコードにオーバーレイします。

この方式は、ほとんどのプログラムで受け入れられます。チェックポイント・データ・セット上のスペースを節約し、最新のチェックポイントでプログラムを再始動することができます。

複数 (複数の連続するチェックポイントを保管します)
チェックポイントは順次記録され、番号が付けられます。それぞれのチェックポイントが保管されます。

最後に取られたチェックポイント以外のチェックポイントでプログラムを再始動したい場合に、この方式を使用します。

COBOL 85 標準に完全に準拠するには、複数チェックポイント方式を使用する必要があります。

ソート操作時のチェックポイントに関しては、次の要件があります。

- ソート操作中にチェックポイントを設定する場合は、実行用のジョブ制御プロシージャーの中に SORTCKPT の DD ステートメントを追加します。
- ASCII で照合されるソートでチェックポイント・レコードを設定することができますが、チェックポイント・データ・セットを示す *system-name* に ASCII ファイルを指定してはなりません。

チェックポイントの設計

データを容易に再構成できるように、プログラムの中の重要点でチェックポイントを設計しなければなりません。チェックポイントから再始動までの間に、ファイルの内容を変更してはなりません。

ディスク・ファイルを使用するプログラムでは、前に処理したレコードを識別できるようにプログラムを設計してください。たとえば、利子に関連して周期的に更新されるローン・レコードが入っているディスク・ファイルを考えてみます。チェックポイントが取られ、レコードが更新され、その後プログラムに対して割り込みが行われる場合、最後のチェックポイント以降に更新されたレコードが、プログラムの再始動時に再度更新されないことをテストすることができます。これを行うため

に、各レコードに日付フィールドをセットアップし、レコードが処理されるたびに日付を更新します。次に再始動された後、日付フィールドをテストし、レコードが前に処理されたかどうかを判別します。

印刷ファイルを効率よく位置変更するためには、ページの最後の行を印刷した後にファイル上でチェックポイントを取ってください。

チェックポイントの正常実行に関するテスト

チェックポイントを出す入力ステートメントまたは出力ステートメントが実行されるたびに、RETURN-CODE 特殊レジスターがチェックポイント・ルーチンからの戻りコードで更新されます。したがって、チェックポイントが成功したかどうかをテストして、再始動に関する条件が適切かどうかを判別できます。戻りコードが 4 より大きい場合は、チェックポイントに関するエラーが発生しています。戻りコードを調べ、誤った出力が作成される原因となるような再始動を回避してください。

関連タスク

223 ページの『DFSORT によるチェックポイント・リスタートの使用』

関連参照

『チェックポイント・データ・セットを定義する DD ステートメント』

z/OS DFSMS: Checkpoint/Restart の『Return codes』の項

チェックポイント・データ・セットを定義する DD ステートメント

チェックポイント・データ・セットを定義するには、次の DD ステートメントを使用します。

テープの場合:

```
//ddname  DD DSNAME=data-set-name,
//           [VOLUME=SER=volser,]UNIT=device-type,
//           DISP=({NEW|MOD},PASS)
```

直接アクセス装置の場合:

```
//ddname  DD DSNAME=data-set-name,
//           [VOLUME=(PRIVATE,RETAIN,SER=volser),]
//           UNIT=device-type,SPACE=(subparms),
//           DISP=({NEW|MOD},PASS,KEEP)
```

ddname

DD ステートメントへのリンクを提供します。COBOL RERUN 文節で使用される *assignment-name* の ddname 部分と同じ。

data-set-name

再始動プロシージャーに対してチェックポイント・データ・セットを識別します。チェックポイント・レコードを記録するために使用される、データ・セットに与えられる名前。

volser

通し番号でボリュームを識別します。

device-type
装置を識別します。

subparms
データ・セットに必要なトラック・スペースの大きさを指定します。

MOD 複数連続チェックポイント・メソッドを指定します。

NEW 単一チェックポイント・メソッドを指定します。

PASS ジョブの最後のジョブ・ステップを除き、ジョブ・ステップの正常完了時に、データ・セットの削除を妨げます。最後のステップであれば、データ・セットは削除されます。

KEEP ジョブ・ステップが異常終了した場合に、データ・セットを保管します。

『例: チェックポイント・データ・セットの定義』

例: チェックポイント・データ・セットの定義

次の例は、チェックポイント・データ・セットを定義するために使用できるジョブ制御言語および COBOL のコーディングを示しています。

単一のチェックポイント・レコードの書き込み (テープを使用):

```
//CHECKPT  DD DSNAME=CHECK1,VOLUME=SER=ND0003,
//            UNIT=TAPE,DISP=(NEW,KEEP),LABEL=(,NL)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHECKPT EVERY
            5000 RECORDS OF ACCT-FILE.
```

単一のチェックポイント・レコードの書き込み (ディスクを使用):

```
//CHEK     DD DSNAME=CHECK2,
//            VOLUME=(PRIVATE,RETAIN,SER=DB0030),
//            UNIT=3380,DISP=(NEW,KEEP),SPACE=(CYL,5)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHEK EVERY
            20000 RECORDS OF PAYCODE.
            RERUN ON CHEK EVERY
            30000 RECORDS OF IN-FILE.
```

複数の連続したチェックポイント・レコードの書き込み (テープを使用):

```
//CHEKPT   DD DSNAME=CHECK3,VOLUME=SER=111111,
//            UNIT=TAPE,DISP=(MOD,PASS),LABEL=(,NL)
            . . .
       ENVIRONMENT DIVISION.
            . . .
            RERUN ON CHEKPT EVERY
            10000 RECORDS OF PAY-FILE.
```

チェックポイント時に生成されるメッセージ

システム・チェックポイント・ルーチンは、コンソールに通知メッセージを表示することによって、取られたチェックポイントの状況をオペレーターに知らせます。

チェックポイントが正常に完了するたびに、メッセージが表示され、ジョブ名 (*ddname*、*unit*、*volser*) と *checkid* が関連付けられます。

checkid は取られたチェックポイントの識別名です。制御プログラムは、*checkid* を 8 文字の文字ストリングとして割り当てます。先頭文字は文字 *C* で、その後にチェックポイントを示す 10 進数が続きます。たとえば、次のメッセージは、ジョブ・ステップで設定された 4 番目のチェックポイントを示します。

```
checkid C0000004
```

プログラムの再始動

システム再始動ルーチンは、以下のことを行います。

- チェックポイント・レコードに記録された情報を検索する
- 主記憶域およびすべてのレジスターの内容を復元する
- プログラムを再始動する

再始動ルーチンは、次のいずれかの方法で開始することができます。

- 割り込みによってプログラムが停止した時に、自動的に
- 後で据え置き再始動として

ジョブ制御言語の RD パラメーターによって、再始動のタイプが決まります。RD パラメーターは、JOB または EXEC ステートメントのいずれかに指定することができます。JOB ステートメントにコーディングされた場合、このパラメーターは、EXEC ステートメントの RD パラメーターをオーバーライドします。

再始動とチェックポイント書き込みの両方を抑止する場合は、RD=NC をコーディングします。

制約事項: SORT または MERGE 操作時に COBOL プログラムによって取られたチェックポイントで再始動しようとすると、エラー・メッセージが出され、再始動は取り消されます。DFSORT によって取られたチェックポイントだけが有効です。

データ・セットの DD ステートメントに SYSOUT パラメーターが指定されていると、そのデータ・セットは、再始動のタイプに応じてさまざまな方法で処理されます。

チェックポイント・データ・セットがマルチボリュームの場合は、チェックポイント項目が書き込まれたボリュームのシーケンス番号を、VOLUME パラメーターに含めなければなりません。チェックポイント・データ・セットが、標準外ラベル付きまたはラベルなしの 7 トラック・テープに置かれている場合は、SYSCHK DD ステートメントに DCB=(TRTCH=C,. . .) が含まれなくてはなりません。

自動再始動の要求

自動再始動は、最後に取られたチェックポイントでだけ行われます。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。

自動再始動が行われる時は、システムはユニット・レコード装置を除くすべての装置の位置を変更します。

自動再始動を行いたい場合は、RD=R または RD=RNC を次のようにコーディングします。

- RD=R は、最新のチェックポイントで再始動を行うことを示します。チェックポイントを記録するためには、プログラム中の少なくとも 1 つのデータ・セットに対して RERUN 文節をコーディングしてください。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。
- RD=RNC は、チェックポイントを書き込まないこと、およびどの再始動もジョブ・ステップの先頭から行うことを示します。この場合、RERUN 文節は不要です。もしあっても、無視されます。

RD パラメーターを省略すると、CHKPT マクロ命令がアクティブのままとなり、処理中にチェックポイントが取られることがあります。最初のチェックポイントの後で割り込みが行われる場合、自動再始動が行われます。

自動的に再始動するためには、プログラムが次の条件を満たしていなければなりません。

- プログラムでは、RD パラメーターを使用するか、またはチェックポイントを取ることによって、再始動を要求しなければなりません。
- ジョブを中止させた異常終了は、再始動を可能にするコードを戻さなければなりません。
- オペレーターが、再始動を許可しなければなりません。

560 ページの『例: ステップ再始動を要求する』

据え置き再始動の要求

据え置き再始動は、必ずしも最後に取られたチェックポイントとは限らず、任意のチェックポイントから行うことができます。ジョブ・ステップの先頭以外のチェックポイントからプログラムを再始動することができます。

据え置き再始動が正常に完了すると、システムは、ジョブが再始動したことを示すメッセージをコンソールに表示します。それから、制御権がプログラムに与えられます。

据え置き再始動を行いたい場合は、RD パラメーターを RD=NR としてコーディングします。この形式のパラメーターは自動再始動を抑止しますが、RERUN 文節がコーディングされていれば、チェックポイント・レコードの書き込みを許可します。

据え置き再始動は、JOB カードで RESTART パラメーターを指定し、チェックポイント・データ・セットを識別する SYSCHK DD ステートメントを使用して、要求してください。SYSCHK DD ステートメントがジョブの中にあっても、JOB ステートメントに RESTART パラメーターが含まれていないと、その SYSCHK DD ステートメントは無視されます。CHECKID サブパラメーターの指定なしの RESTART パラメーターをジョブに含める場合は、ジョブの最初の EXEC ステートメントの前に SYSCHK DD ステートメントがあってはなりません。

560 ページの『例: 特定のチェックポイント・ステップでジョブを再始動する』

関連タスク

223 ページの『DFSORT によるチェックポイント・リスタートの使用』

関連参照
『据え置き再始動を要求する形式』

据え置き再始動を要求する形式

JOB ステートメントおよび SYSCHK DD ステートメントの RESTART パラメーターの形式は、次のとおりです。

```
//jobname  JOB MSGLEVEL=1,RESTART=(request[,checkid])
//SYSCHK   DD  DSNAME=data-set-name,
//             DISP=OLD[,UNIT=device-type,
//             VOLUME=SER=volser]
```

MSGLEVEL=1 (または MSGLEVEL=(1,*y*))
MSGLEVEL は必須です。

RESTART=(*request*,[*checkid*])
再始動が行われる特定のチェックポイントを識別します。

request
次のいずれかの形式を取ります。

***** ジョブの先頭から再始動することを示します。

stepname
ジョブ・ステップの先頭から再始動することを示します。

stepname.procstep
ジョブ・ステップ内のプロシージャー・ステップから再始動することを示します。

checkid
再始動が行われるチェックポイントを指示します。

SYSCHK
チェックポイント・データ・セットを制御プログラムに識別するために使用される DD 名。SYSCHK DD ステートメントは、再実行依頼されたジョブの最初の EXEC ステートメントの直前で、かつ JOBLIB ステートメントの後になければなりません。

data-set-name
チェックポイント・データ・セットを識別します。チェックポイントが取られた時に使用された名前と同じでなければなりません。

device-type* および *volser
チェックポイント・データ・セットが入っているボリュームの装置タイプおよび通し番号を識別します。

『例: 据え置き再始動の要求』

例: 据え置き再始動の要求

次のように、C0000003 のチェックポイント ID (CHECKID) で、IGYWCLG プロシージャーの GO ステップの再始動が示される場合があります。

```
//jobname  JOB MSGLEVEL=1,RESTART=(stepname.GO,C0000003)
//SYSCHK   DD DSNAME=CHEKPT,
//            DISP=OLD[,UNIT=3380,VOLUME=SER=111111]
           . . .
```

再始動のためのジョブの再実行依頼

再始動のためにジョブを再実行依頼するときは、再始動されるジョブ・ステップの実行に影響を与える可能性のある DD ステートメントに注意してください。再始動ルーチンは、再入力されたジョブの DD ステートメントの情報を使用して、再始動後に使用するファイルをリセットします。次の点に注意してください。

- ジョブ・ステップ終了時にデータ・セットを削除したい場合は、実行時に (DELETE ではなく) PASS または KEEP の条件付き後処理を指定してください。この後処理を指定すると、割り込みによって強制的に再始動が実行される場合にそのデータ・セットが使えるようになります。

  ステップの先頭からジョブを再始動したい場合は、前回の実行で作成したデータ・セット (DD ステートメントで NEW と定義された) をすべて廃棄するか、またはデータ・セットに OLD のマークを付けるように DD ステートメントを変更しなければなりません。
- 再始動時に、カード上の入力データ・セットを、チェックポイント時にそれらがあった場所と同じ場所に置いてください。テープまたはディスク上の入力データ・セットは、システムによって自動的に再度位置決めされます。

561 ページの『例: ステップ再始動のためにジョブを再実行依頼する』
561 ページの『例: チェックポイント・リスタートのためにジョブを再実行依頼する』

例: 特定のチェックポイント・ステップでジョブを再始動する

この例は、特定のステップからジョブを再始動するための一連のジョブ制御ステートメントを示しています。

```
//PAYROLL  JOB  MSGLEVEL=1,REGION=80K,
//              RESTART=(STEP1,CHECKPT4)
//JOBLIB   DD   DSNAME=PRIV.LIB3,DISP=OLD
//SYSCHK   DD   DSNAME=CHKPTLIB,
//              [UNIT=TAPE,VOL=SER=456789,]
//              DISP=(OLD,KEEP)
//STEP1    EXEC PGM=PROG4,TIME=5
```

例: ステップ再始動を要求する

この例は、RD パラメーターの使用を示しています。この RD パラメーターは、異常終了したジョブ・ステップのステップ再始動を要求します。DD ステートメントの DDCKPNT で、チェックポイント・データ・セットが定義されます。このステップの場合、RERUN 文節の実行後、CHKPT 取り消しが出されなければ、自動チェックポイント・リスタートを行うことができます。

```
//J1234    JOB  386,SMITH,MSGLEVEL=1,RD=R
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,
//              [VOLUME=SER=91468,]
//              LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=T91468,DISP=(,,KEEP),
```

```
//              UNIT=SYSDA,SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT  DD   UNIT=TAPE,DISP=(MOD,PASS,CATLG),
//              DSNAME=C91468,LABEL=(,NL)
```

例: ステップ再始動のためにジョブを再実行依頼する

次の例は、ステップ再始動のためにジョブを再実行依頼する前に、ステートメントを制御するために行うことができる変更を示しています。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、最初のジョブ・ステップから再始動を開始するように指示します。
- WORK DD ステートメントには、このデータ・セットに対して KEEP という条件付き後処理が最初に割り当てられていました。
  - 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは削除されているので、このステートメントに変更を加える必要はありません。
  - ステップが異常終了した場合、データ・セットは保管されています。その場合は、新しいデータ・セット (以下に示すように T91468 ではなく S91468) を定義するか、またはジョブを再実行依頼する前にデータ・セットの状況を OLD に変更しなければなりません。
- 新しいデータ・セット (C91468 ではなく R91468) も、チェックポイント・データ・セットとして定義されました。

```
//J3412    JOB  386,SMITH,MSGLEVEL=1,RD=R,RESTART=*
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,
//              [VOLUME=SER=91468,]LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=S91468,
//              DISP=(,,KEEP),UNIT=SYSDA,
//              SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT DD   UNIT=TAPE,DISP=(MOD,PASS,CATLG),
//              DSNAME=R91468,LABEL=(,NL)
```

560 ページの『例: ステップ再始動を要求する』

例: チェックポイント・リスタートのためにジョブを再実行依頼する

次の例は、チェックポイント・リスタートのためにジョブを再実行依頼する前に、ステートメントを制御するために行うことができる変更を示しています。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、C0000002 という名前のチェックポイント項目で最初のステップから再始動を開始するように指示します。
- DD ステートメントの DDCKPNT では、チェックポイント・データ・セットに CATLG という条件付き後処理が最初に割り当てられました。

– 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは保管されています。その場合、SYSCHK DD ステートメントには、チェックポイント・データ・セットの検索に必要なすべての情報が含まれていなければなりません。
– ジョブが異常終了した場合、データ・セットはカタログされています。その場合、以下に示されているように、SYSCHK DD ステートメントに必要なパラメーターは DSNAME と DISP だけです。

V=R が指定された場合は、実行されているジョブでチェックポイントが取られても、十分なページング不能な動的ストレージが使用可能になるまで、そのジョブは再始動することはできません。

```
//J3412    JOB  386,SMITH,MSGLEVEL=1,RD=R,
//              RESTART=(*,C0000002)
//SYSCHK   DD   DSNAME=C91468,DISP=OLD
//S1       EXEC PGM=MYPROG
//INDATA   DD   DSNAME=INVENT,UNIT=TAPE,DISP=OLD,
//              VOLUME=SER=91468,LABEL=RETPD=14
//REPORT   DD   SYSOUT=A
//WORK     DD   DSNAME=T91468,DISP=(,,KEEP),
//              UNIT=SYSDA,SPACE=(3000,(5000,500)),
//              VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT  DD   UNIT=TAPE,DISP=(MOD,KEEP,CATLG),
//              DSNAME=C91468,LABEL=(,NL)
```

第 31 章 2 桁年号の日付の処理

2000 年言語拡張では、COBOL プログラムに簡単な変更を行うだけで、日付フィールドを定義することができます。コンパイラーは、世紀ウィンドウを使用して整合性を保証したうえで、これらの日付を認識し、作用します。COBOL プログラムで自動日付認識をインプリメントするには、以下のステップを使用してください。

1. プログラム内のデータ項目のうち、日付を含むデータ項目のデータ記述記入項目に DATE FORMAT 文節を追加します。比較で使用されないものを含め、すべての日付を DATE FORMAT 文節を使用して識別しなければなりません。
2. 日付を拡張する場合は、MOVE または COMPUTE ステートメントを使用して、ウィンドウ化日付フィールドの内容を拡張日付フィールドにコピーします。
3. 必要であれば、DATEVAL および UNDATE 組み込み関数を使用して、日付フィールドと非日付の間で交換を行います。
4. YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウとして設定します。
5. DATEPROC(FLAG) コンパイラー・オプションを使用してプログラムをコンパイルし、診断メッセージを検討して、日付処理によって予期しない副作用が起こっていないかどうかを調べます。
6. コンパイルの結果、通知レベルの診断メッセージだけしかなければ、DATEPROC(NOFLAG) コンパイラー・オプションを使用してプログラムを再コンパイルして、すっきりしたリストを作成することができます。

特定のプログラミング手法を使用して、日付処理を利用したり、日付フィールド使用の効果を制御することができます (たとえば、日付の比較、日付によるソートとマージ、および日付を伴う算術演算の実行など)。2000 年言語拡張は、比較、移動と保管、増減など、日付フィールドに関してよく使われる操作について、年先頭型、年単独型、年末尾型の日付フィールドをサポートします。

関連概念

564 ページの『2000 年言語拡張 (MLE)』

関連タスク

566 ページの『日付関連ロジック問題の解決』
571 ページの『年先頭型、年単独型、および年末尾型日付フィールドの使用』
574 ページの『日付としてのリテラルの操作』
577 ページの『トリガーおよび限界の設定』
579 ページの『日付によるソートおよびマージ』
581 ページの『日付フィールドへの算術の実行』
583 ページの『日付処理の明示的制御』
585 ページの『日付関連診断メッセージの分析および回避』
587 ページの『日付の処理における問題の回避』

© Copyright IBM Corp. 1991, 2001

関連参照
DATE FORMAT 文節 (*Enterprise COBOL 言語解説書*)
322 ページの『DATEPROC』
358 ページの『YEARWINDOW』

2000 年言語拡張 (MLE)

2000 年言語拡張 という用語は、21 世紀の日付に関連する論理問題を補助するために、DATEPROC コンパイラー・オプションが活動化する Enterprise COBOL の機能を指します。

使用可能にされた場合、言語拡張には以下のものが含まれます。

- DATE FORMAT 文節。この文節は、日付フィールドを識別し、日付における年部分の位置を指定するために、DATA DIVISION 内の項目に追加されます。
- 以下の組み込み関数について、関数戻り値を日付フィールドとして再解釈すること。

```
DATE-OF-INTEGER
DATE-TO-YYYYMMDD
DAY-OF-INTEGER
DAY-TO-YYYYDDD
YEAR-TO-YYYY
```

- 次の形式の ACCEPT ステートメントにおける概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDD を日付フィールドとして再解釈すること。

```
ACCEPT identifier FROM DATE
ACCEPT identifier FROM DATE YYYYMMDD
ACCEPT identifier FROM DAY
ACCEPT identifier FROM DAY YYYYDDD
```

- 組み込み関数 UNDATE と DATEVAL。これらは、日付フィールドおよび非日付の選択的再解釈に使用されます。
- 組み込み関数 YEARWINDOW。これは、YEARWINDOW コンパイラー・オプションによって設定された世紀ウィンドウの開始年を検索します。

DATEPROC コンパイラー・オプションは、識別された日付フィールドの特殊な日付中心処理を使用可能にします。YEARWINDOW コンパイラー・オプションは、2 桁のウィンドウ化西暦年を解釈するために使用する 100 年ウィンドウ (世紀ウィンドウ) を指定します。

関連概念
『言語拡張の原則および目標』

関連参照
322 ページの『DATEPROC』
358 ページの『YEARWINDOW』

言語拡張の原則および目標

2000 年言語拡張から最大限の利点を得るためには、これが COBOL 言語に導入された理由を理解する必要があります。

2000 年言語拡張が焦点を当てているのは、次の原則だけです。

- 日付のセマンティクスを使用して再コンパイルされるプログラムは、十分にテストされ、企業にとって価値のある資産です。関連する制限事項は、プログラムの 2 桁の年号が 1900-1999 の範囲に制限されることだけです。
- 日付の年号以外の部分には特別な処理は行われません。このため、サポートされる日付形式の年号以外の部分は X で表されます。それ以外の表記を使用すると、既存のプログラムの意味が変わってしまうことがあります。提供されている唯一の日付依存型セマンティクスは、プログラムの世紀ウィンドウに関して、日付の 2 桁の年部分を自動的に拡張 (および短縮) するということです。
- 4 桁の年号部分を持つ日付が重要になるのは、通常、ウィンドウ化日付と組み合わせて使用される場合です。それ以外の場合、4 桁年号日付と非日付の間に違いはほとんどありません。

これらの原則に基づき、2000 年言語拡張は多数の目標を満たすように設計されています。日付処理問題を解決するために満足しなければならない目標を評価し、それらを 2000 年言語拡張の目標と比較して、アプリケーションがそこからどのように利益を得ることができるかを判別しなければなりません。新規アプリケーション、または既存アプリケーションへの拡張では、もっと後になるまで拡張できない古いデータをアプリケーションで使用している場合を除いて、拡張部分の使用を考慮してはなりません。

2000 年言語拡張の目標は、次のとおりです。

- 現在指定されているアプリケーション・プログラムの実用的な存続期間を拡張します。
- ソース変更を最小限にとどめ、可能であれば、DATA DIVISION の日付フィールドの宣言の増加だけに限定します。世紀ウィンドウ・ソリューションをインプリメントするためには、PROCEDURE DIVISION のプログラム・ロジックを変更する必要がないようにします。
- 日付フィールドを追加するときは、プログラムの既存のセマンティクスを保持しなければなりません。たとえば、次のステートメントの場合のように、日付がリテラルとして表される場合、そのリテラルは、比較対象の日付フィールドと互換性があると見なされます (ウィンドウ操作または拡張されます)。

  ```
  If Expiry-Date Greater Than 980101 . . .
  ```

  既存のプログラムは、リテラルとして表された 2 桁年号の日付が 1900-1999 の範囲内にあると想定するので、拡張でこの想定が変更されることはありません。
- ウィンドウ操作機能は長期間の使用を目的としたものではありません。後で実施できる長期間解決策が採用されるまで、アプリケーションの実用的な存続期間をとりあえず 2000 年まで拡張することを意図しています。
- 拡張日付フィールド機能は、ファイルおよびデータベースの日付フィールドの拡張を支援するものとして、長期間の使用を意図しています。

拡張部分は、完全指定または完全な日付中心のデータ型に、認識できるセマンティクス (たとえばグレゴリオ暦の月と日の部分) を与えません。拡張部分は、日付の年号部分に特別なセマンティクスを与えるだけです。

日付関連ロジック問題の解決

日付処理問題を解決するための助けとして、以下のいずれかのアプローチを採用することができます。

世紀ウィンドウ

世紀ウィンドウを定義し、ウィンドウ化日付を含むフィールドを指定します。コンパイラーは、これらのデータ・フィールドの 2 桁の年号を世紀ウィンドウに従って解釈します。

内部ブリッジング

ファイルおよびデータベースはまだ 4 桁年の日付に変更されていないが、プログラムでは 4 桁に拡張した年のロジックを使用したい場合、そのような日付を 4 桁年の日付として処理するための内部ブリッジング手法を使用することができます。

全フィールド拡張

このソリューションは、2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用する方法です。これは、すべてのアプリケーションについて信頼できる日付処理がなされる唯一の方法です。

ソリューションを達成するためのそれぞれのアプローチで 2000 年言語拡張を使用できますが、次に示すように、それぞれ利点と欠点があります。

	ソリューション別の利点および欠点		
局面	**世紀ウィンドウ**	**内部ブリッジング**	**全フィールド拡張**
インプリメンテーション	速くて容易ですが、すべてのアプリケーションに合うわけではありません。	データ破壊のリスクがあります。	データベース、コピーブック、およびプログラムに対する変更をすべて同期させる必要があります。
テスト	プログラム・ロジックの変更はないので、テストはほとんど必要ありません。	プログラム・ロジックに行われる変更が直接的なものなので、テストは楽です。	
修正期間	プログラムは 2000 年を過ぎても機能しますが、長期的なソリューションとはなり得ません。	プログラムは 2000 年を過ぎても機能しますが、永続的なソリューションとはなり得ません。	永続的なソリューションです。
パフォーマンス	パフォーマンスが低下する可能性があります。	良好なパフォーマンスが得られます。	最適なパフォーマンスが得られます。
保守			保守が容易になります。

『例: 世紀ウィンドウ』
569 ページの『例: 内部ブリッジング』
570 ページの『例: ファイルを拡張日付形式に変換する』

関連タスク
『世紀ウィンドウの使用』
568 ページの『内部ブリッジングの使用』
569 ページの『全フィールド拡張への移行』

世紀ウィンドウの使用

世紀ウィンドウとは、1950-2049 のような 100 年の間隔で、その中では 2 桁の年号がすべて固有となるものです。ウィンドウ化日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウ開始日を指定することができます。DATEPROC オプションが有効であると、コンパイラーは、プログラムの 2 桁の日付フィールドにこのウィンドウを適用します。たとえば、1930-2029 の世紀ウィンドウの場合、COBOL は 2 桁の年号を次のように解釈します。

00 ～ 29 の年の値は、2000-2029 として解釈されます。
30 ～ 99 の年の値は、1930-1999 として解釈されます。

この世紀ウィンドウをインプリメントするためには、DATE FORMAT 文節を使用してプログラム内の日付フィールドを識別し、YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウのいずれかとして定義してください。

- 固定ウィンドウの場合は、YEARWINDOW オプションの値として 1900 ～ 1999 の 4 桁の年号を指定します。たとえば、YEARWINDOW(1950) は、1950-2049 の固定ウィンドウを定義します。
- スライディング・ウィンドウの場合は、YEARWINDOW オプションの値として -1 ～ -99 の負の整数を指定します。たとえば、YEARWINDOW(-48) は、プログラムが稼働される年の 48 年前に始まるスライディング・ウィンドウを定義します。したがって、プログラムが 2001 年に稼働される場合、世紀ウィンドウは 1953-2052 となり、2002 年に稼働される場合は、自動的に 1954-2053 となる、といった具合です。

コンパイラーは、世紀ウィンドウを、識別された日付フィールドの操作に自動的に適用します。ウィンドウ操作をインプリメントするための余分のプログラム・ロジックは必要ありません。

『例: 世紀ウィンドウ』

例: 世紀ウィンドウ

次の例では、自動日付ウィンドウ操作機能を使用するためにプログラムをどのように変更するかを (太字で) 示しています。このプログラムは、ビデオ・テープが期限内に戻されたかどうかを検査します。

```
CBL  LIB,QUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-60)
    . . .
    01  Loan-Record.
        05  Member-Number   Pic X(8).
        05  Tape-ID         Pic X(8).
```

```
       05  Date-Due-Back   Pic X(6) Date Format yyxxxx.
       05  Date-Returned   Pic X(6) Date Format yyxxxx.
   . . .
       If Date-Returned > Date-Due-Back Then
          Perform Fine-Member.
```

この例では、PROCEDURE DIVISION には変更を加えていません。2 つの日付フィールドに DATE FORMAT 文節を追加した意味は、コンパイラーがそれらをウィンドウ化日付フィールドとして認識し、したがって、IF ステートメントを処理するときには世紀ウィンドウを適用するということです。たとえば、Date-Due-Back (返却予定日) が 000102 (2000 年 1 月 2 日) で、Date-Returned (返却日) が 991231 (1999 年 12 月 31 日) である場合は、Date-Returned が Date-Due-Back より小さい (前である) ので、プログラムは Fine-Member 段落を実行しません。

内部ブリッジングの使用

内部ブリッジングを行う場合は、次のようにプログラムを構造化することができます。

1. 2 桁年号の日付を持つ入力ファイルを読み取る。
2. これらの 2 桁の日付をウィンドウ化日付フィールドとして宣言し、コンパイラーがこれらの日付を自動的に 4 桁年号の日付に拡張できるように、これらを拡張日付フィールドに移動する。
3. プログラムの本体では、すべての日付処理に 4 桁年号の日付を使用する。
4. ウィンドウ操作により日付を 2 桁の年号に戻す。
5. 2 桁年号の日付を出力ファイルに書き込む。

このプロセスは、完全拡張日付ソリューションへの便利な移行パスになり、ウィンドウ化日付を使用するよりもパフォーマンス上の利点もあるかもしれません。

この手法を使うと、プログラム・ロジックへの変更は最小で済みます。単に、日付を拡張および短縮するステートメントを追加し、また日付を参照するステートメントを、レコードの中の 2 桁年号のフィールドではなく WORKING-STORAGE の中の 4 桁年号の日付フィールドを使用するように変更するだけです。

出力のために日付を変換して 2 桁の年に戻しているので、年が世紀ウィンドウの範囲外になる可能性を考慮しておいてください。たとえば、日付フィールドの値が 2020 年で、世紀ウィンドウが 1920-2019 であると、その日付はウィンドウの範囲外となり、単に 2 桁の年号に移動すると間違いになります。これを保護するためには、ON SIZE ERROR 句付きの COMPUTE ステートメントを使用して日付を保管することにより、その日付が世紀ウィンドウの範囲内にあるかどうかを調べることができます。

569 ページの『例: 内部ブリッジング』

関連タスク

567 ページの『世紀ウィンドウの使用』
581 ページの『日付フィールドへの算術の実行』
569 ページの『全フィールド拡張への移行』

例: 内部ブリッジング

次の例では、内部ブリッジングをインプリメントするためにプログラムをどのように変更できるかを (太字で) 示しています。

```
CBL  DATEPROC(FLAG),YEARWINDOW(-60)
    . . .
    File Section.
    FD  Customer-File.
    01  Cust-Record.
        05  Cust-Number     Pic 9(9) Binary.
        . . .
        05  Cust-Date       Pic 9(6) Date Format yyxxxx.
    Working-Storage Section.
    77  Exp-Cust-Date       Pic 9(8) Date Format yyyyxxxx.
    . . .
    Procedure Division.
        Open I-O Customer-File.
        Read Customer-File.
        Move Cust-Date to Exp-Cust-Date.
        . . .
   *=====================================================*
   * Use expanded date in the rest of the program logic  *
   *=====================================================*
        . . .
        Compute Cust-Date = Exp-Cust-Date
                On Size Error Display "Exp-Cust-Date outside
                                       century window"
                End-Compute
        Rewrite Cust-Record.
```

全フィールド拡張への移行

2000 年言語拡張を使用すると、日付フィールドを完全に拡張するソリューションに向けて徐々に移行することができます。

1. 世紀ウィンドウ・ソリューションを適用し、より永続的なソリューションをインプリメントするためのリソースが用意されるまで、このソリューションを使う。
2. 内部ブリッジング・ソリューションを適用する。これは、ファイルでは 2 桁年号の形式で日付を保持した状態のまま、プログラムの中では拡張日付を使用する方法です。プログラムの本体の中ではロジックにそれ以上の変更を行わないので、全フィールド拡張ソリューションにより容易に進むことができます。
3. ファイル設計およびデータベース定義を、4 桁年号の日付を使用するように変更する。
4. COBOL コピーブックを、4 桁年号の日付フィールドを反映するように変更する。
5. ユーティリティー・プログラム (または特殊な目的の COBOL プログラム) を実行して、古い形式のファイルから新しい形式にコピーする。
6. プログラムを再コンパイルし、レグレッション・テストおよび日付テストを行う。

最初の 2 つのステップを完了したら、それ以降のステップは何回でも繰り返すことができます。すべてのファイルのすべての日付フィールドを同時に変更する必要はありません。この方法では、段階的に変換するファイルを、業務上の必要性または他のアプリケーションとのインターフェースなどを基準にして選択することができます。

このメソッドを使用するときは、ファイルを拡張日付形式に変換するための特殊目的プログラムを書く必要があります。

『例: ファイルを拡張日付形式に変換する』

例: ファイルを拡張日付形式に変換する

次の例は、日付フィールドを拡張すると同時にあるファイルから別のファイルへコピーする単純プログラムを示しています。日付が拡張されるため、出力ファイルのレコード長は入力ファイルのレコード長より長くなっています。

```
CBL  LIB,QUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-80)
      ************************************************
      ** CONVERT - Read a file, convert the date    **
      **           fields to expanded form, write   **
      **           the expanded records to a new    **
      **           file.                            **
      ************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CONVERT.

 ENVIRONMENT DIVISION.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE
                  ASSIGN TO INFILE
                  FILE STATUS IS INPUT-FILE-STATUS.

           SELECT OUTPUT-FILE
                  ASSIGN TO OUTFILE
                  FILE STATUS IS OUTPUT-FILE-STATUS.

       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE
           RECORDING MODE IS F.
       01  INPUT-RECORD.
           03  CUST-NAME.
               05  FIRST-NAME  PIC X(10).
               05  LAST-NAME   PIC X(15).
           03  ACCOUNT-NUM     PIC 9(8).
           03  DUE-DATE        PIC X(6) DATE FORMAT YYXXXX.      (1)
           03  REMINDER-DATE   PIC X(6) DATE FORMAT YYXXXX.
           03  DUE-AMOUNT      PIC S9(5)V99 COMP-3.

       FD  OUTPUT-FILE
           RECORDING MODE IS F.
       01  OUTPUT-RECORD.
           03  CUST-NAME.
               05  FIRST-NAME  PIC X(10).
               05  LAST-NAME   PIC X(15).
           03  ACCOUNT-NUM     PIC 9(8).
           03  DUE-DATE        PIC X(8) DATE FORMAT YYYYXXXX.    (2)
           03  REMINDER-DATE   PIC X(8) DATE FORMAT YYYYXXXX.
           03  DUE-AMOUNT      PIC S9(5)V99 COMP-3.

 WORKING-STORAGE SECTION.

       01  INPUT-FILE-STATUS   PIC 99.
       01  OUTPUT-FILE-STATUS  PIC 99.

       PROCEDURE DIVISION.

           OPEN INPUT INPUT-FILE.
           OPEN OUTPUT OUTPUT-FILE.
```

```
READ-RECORD.
    READ INPUT-FILE
         AT END GO TO CLOSE-FILES.
    MOVE CORRESPONDING INPUT-RECORD TO OUTPUT-RECORD.     (3)
    WRITE OUTPUT-RECORD.

    GO TO READ-RECORD.

CLOSE-FILES.
    CLOSE INPUT-FILE.
    CLOSE OUTPUT-FILE.

    EXIT PROGRAM.

END PROGRAM CONVERT.
```

注:

(1) 入力レコード内のフィールド DUE-DATE と REMINDER-DATE はともに、2 桁年号の要素を持つグレゴリオ暦日付です。これらのフィールドは、このプログラムでは DATE FORMAT 文節で定義されており、したがって、コンパイラーはこれらをウィンドウ化日付フィールドと認識します。

(2) 出力レコードには、同じ 2 つのフィールドが拡張日付形式で入れられます。これらは DATE FORMAT 文節で定義されているので、コンパイラーはこれらを 4 桁年号の日付フィールドとして扱います。

(3) MOVE CORRESPONDING ステートメントは、INPUT-RECORD 内の各項目を OUTPUT-RECORD 内の対応する項目に個々に移動します。2 つのウィンドウ化日付フィールドが対応する拡張日付フィールドに移動されると、コンパイラーは現行の世紀ウィンドウを使用して年号値を拡張します。

年先頭型、年単独型、および年末尾型日付フィールドの使用

年先頭型 日付フィールドは、DATE FORMAT 指定が YY または YYYY と、その後の 1 つまたは複数の X から構成される日付フィールドです。*年単独型* 日付フィールドの日付形式は YY または YYYY だけです。これらのどちらかのタイプの 2 つの日付フィールドを比較するとき、その 2 つの日付には互換性がなければなりません。すなわち、その 2 つは、年以外の文字の数は同じでなければなりません。年部分の桁数は同じである必要はありません。

年末尾型 日付フィールドは、DATE FORMAT 文節で YY または YYYY の前に 1 つまたは複数の X が指定されている日付フィールドです。そのような日付形式は日付の表示によく使われますが、コンピューターにとってはあまり便利ではありません。それは、日付の中で最も比重の大きい部分である年が日付表現における比重の小さい部分になるからです。

使用している DFSORT (または等価の製品) のバージョンに適切な機能が備わっている場合、年末尾型日付は、SORT または MERGE ステートメントではウィンドウ化キーとしてサポートされます。ソートおよびマージの操作以外で年末尾型日付フィールドの機能がサポートされるのは、等しいか等しくないかの比較とある種の代入操作だけです。オペランドは、日付形式が同一 (年末尾型) の日付であるか、または日付と非日付でなければなりません。コンパイラー自体には、年末尾型日付の操作

のための自動ウィンドウ操作機能は含まれていません。年末尾型日付の算術演算など、サポートされていない方法で使用されると、コンパイラーはエラー・レベル・メッセージを提供します。

年末尾型日付のためのより一般的な日付処理が必要な場合は、日付の年部分を分離して処理する必要があります。

573 ページの『例: 年先頭型日付フィールドの比較』

関連概念
『互換性のある日付』

関連タスク
579 ページの『日付によるソートおよびマージ』
573 ページの『他の日付形式の使用』

互換性のある日付

互換性のある日付 という用語の意味は、次のように、それがどの COBOL 部で使用されるかによって異なります。

- DATA DIVISION での使用は、日付フィールドの宣言と、従属データ項目や REDEFINES 文節などの COBOL 言語エレメントを管理する規則に関連しています。次の例では、Review-Year は Review-Date への従属データ項目として宣言することができるので、Review-Date と Review-Year には互換性があります。

  ```
  01 Review-Record.
      03  Review-Date            Date Format yyxxxx.
          05  Review-Year Pic XX Date Format yy.
          05  Review-M-D  Pic XXXX.
  ```

- PROCEDURE DIVISION での使用は、比較、移動、算術式などの演算で日付フィールドを一緒に使用する方法に関連しています。年先頭型および年単独型日付フィールドが互換性のあるものであると見なすためには、日付フィールドの年以外の文字の文字数が同じでなければなりません。たとえば、DATE FORMAT YYXXXX というフィールドは、同じ日付形式を持つ別のフィールドおよび YYYYXXXX フィールドと互換性がありますが、YYXXX フィールドとは互換性はありません。

  年末尾型日付フィールドの場合は、DATE FORMAT 文節が同じである必要があります。特に、ウィンドウ操作日付フィールドと拡張年末尾型日付フィールドとの相互演算は許されません。たとえば、XXXXYY という日付形式の日付フィールドは、XXXXYY という別の日付フィールドに移動できますが、XXXXYYYY という形式の日付フィールドには移動できません。

演算の中の日付フィールドに互換性があれば、日付フィールドに対して、または日付フィールドと非日付が組み合わされたものに対して演算を実行できます。たとえば、次のように定義されているとします。

```
01  Date-Gregorian-Win  Pic 9(6) Packed-Decimal Date Format yyxxxx.
01  Date-Julian-Win     Pic 9(5) Packed-Decimal Date Format yyxxx.
01  Date-Gregorian-Exp  Pic 9(8) Packed-Decimal Date Format yyyyxxxx.
```

2 つのフィールドの年以外の数字の数が異なるので、次のステートメントは成立しません。

```
If Date-Gregorian-Win Less than Date-Julian-Win . . .
```

2 つのフィールドの年以外の数字の数が同じなので、次のステートメントは受け入れられます。

```
If Date-Gregorian-Win Less than Date-Gregorian-Exp . . .
```

この場合は、比較が確実に意味のあるものになるようにするために、世紀ウィンドウがウィンドウ化日付フィールド (Date-Gregorian-Win) に適用されます。

非日付を日付フィールドと一緒に使用した場合、非日付は日付フィールドと互換性があると見なされるか、または単純な数値として扱われます。

例: 年先頭型日付フィールドの比較

この例では、ウィンドウ化日付フィールドが拡張日付フィールドと比較されます。したがって、Date-Due-Back には世紀ウィンドウが適用されます。

```
77  Todays-Date         Pic X(8) Date Format yyyyxxxx.
01  Loan-Record.
    05  Date-Due-Back   Pic X(6) Date Format yyxxxx.
. . .
    If Date-Due-Back > Todays-Date Then . . .
```

この場合は、Todays-Date に DATE FORMAT 文節を指定して、それを拡張日付フィールドとして定義しなければなりません。そうしないと、そのフィールドは非日付フィールドとして扱われ、その結果、そのフィールドは年の桁数が Date-Due-Back と同じであるものと見なされます。コンパイラーは 1900-1999 の仮定による世紀ウィンドウを適用するため、矛盾した比較が作成されます。

他の日付形式の使用

日付フィールドが自動ウィンドウ操作に適格となるためには、そのフィールドの最初の部分または唯一の部分として 2 桁の年を含んでいなければなりません。フィールドの残り部分がある場合、残りの部分は 1 ～ 4 文字でなければならず、その内容は重要ではありません。たとえば、3 桁の年間通算日や、企業に固有のイベントの 2 文字の ID を使用できます。

アプリケーションの中に、このような基準に合わない日付フィールドがある場合は、DATE FORMAT 文節によって日付の年部分だけを日付フィールドとして定義するためのコード変更が必要かもしれません。このようなタイプの日付形式の例として、次のようなものがあります。

- 2 桁の年、月の省略形を表す 3 文字、および日を表す 2 桁の数で構成される 7 文字のフィールド。日付フィールドには 1 ～ 4 文字の年以外の文字しか使えないため、この形式はサポートされません。
- 形式 DDMMYY のグレゴリオ暦日付。年の部分が日付の最初の部分ではないので、自動ウィンドウ操作は適用されません。このような年末尾型日付は、SORT または MERGE ステートメントではウィンドウ化キーとして全面的にサポートされており、また一部の COBOL 操作でもサポートされています。

このような場合に日付のウィンドウ操作を使用する必要があるなら、日付の年部分を分離するために何らかのコードを追加する必要があります。

574 ページの『例: 年部分の分離』

例: 年部分の分離

次の例では、2 つの日付フィールドには DDMMYY 形式の日付が含まれています。

```
03  Last-Review-Date Pic 9(6).
03  Next-Review-Date Pic 9(6).
. . .
Add 1 to Last-Review-Date Giving Next-Review-Date.
```

この例で、Last-Review-Date が 230197 (1997 年 1 月 23 日) なら、ADD ステートメントの実行後、Next-Review-Date は 230198 (1998 年 1 月 23 日) になります。これは、次の年次検査 (review) 日付を設定するための簡単な方法です。しかし、Last-Review-Date が 230199 の場合に 1 を加算すると 230200 になり、意図した結果にはなりません。

これらの日付フィールドは年が日付フィールドの最初の部分ではないので、年の部分を分離するための何らかのコードを追加しないと、DATE FORMAT 文節を適用することはできません。次の例では、2 つの日付フィールドの年部分が分離されており、COBOL は世紀ウィンドウを適用して矛盾のない結果を維持できます。

```
03  Last-Review-Date Date Format xxxxyy.
    05  Last-R-DDMM  Pic 9(4).
    05  Last-R-YY    Pic 99 Date Format yy.
03  Next-Review-Date Date Format xxxxyy.
    05  Next-R-DDMM  Pic 9(4).
    05  Next-R-YY    Pic 99 Date Format yy.
. . .
Move Last-R-DDMM to Next-R-DDMM.
Add 1 to Last-R-YY Giving Next-R-YY.
```

日付としてのリテラルの操作

ウィンドウ化日付フィールドが、関連する 88 レベルの条件名を持つ場合、VALUE 文節のリテラルは、1900-1999 の仮定による世紀ウィンドウではなく、コンパイル単位の世紀ウィンドウに対してウィンドウ操作されます。

たとえば、次のようなデータ定義があるとします。

```
05  Date-Due        Pic 9(6) Date Format yyxxxx.
88  Date-Target              Value 051220.
```

世紀ウィンドウが 1950-2049 で、Date-Due の内容が 051220 (2005 年 12 月 20 日) であれば、以下の最初の条件は真に評価され、2 番目の条件は偽に評価されます。

```
If Date-Target
If Date-Due = 051220
```

リテラル 051220 は非日付として扱われるため、1900-1999 の仮定による世紀ウィンドウに対してウィンドウ操作され、1905 年 12 月 20 日を表すことになります。しかし、88 レベルの条件名の VALUE 文節に同じリテラルが指定された場合には、そのリテラルは、それが付加されるデータ項目の一部になります。このデータ項目はウィンドウ化日付フィールドなので、それが参照されるときには、必ず世紀ウィンドウが適用されます。

比較式の中で DATEVAL 組み込み関数を使用してリテラルを日付フィールドに変換することも可能です。その場合、その組み込み関数からの出力は、ウィンドウ化日付

フィールドまたは拡張日付フィールドとして扱われるので、比較に矛盾は生じません。たとえば、上記の定義を使用すれば、以下の 2 つの条件はどちらも真に評価されます。

```
If Date-Due = Function DATEVAL (051220 "YYXXXX")
If Date-Due = Function DATEVAL (20051220 "YYYYXXXX")
```

レベル 88 条件名については、VALUE 文節に THRU オプションを指定できますが、YEARWINDOW コンパイラー・オプションでは、スライディング・ウィンドウではなく固定世紀ウィンドウを指定しなければなりません。たとえば、次のようになります。

```
05  Year-Field      Pic 99 Date Format yy.
88  In-Range               Value 98 Thru 06.
```

この形式の場合、2 番目の項目のウィンドウ操作値は、1 つ目の項目のウィンドウ操作値より大きくなければなりません。ただし、コンパイラーがこの差を検査できるのは、YEARWINDOW コンパイラー・オプションで固定世紀ウィンドウが指定されている場合だけです (たとえば、YEARWINDOW(-60) ではなく YEARWINDOW(1940) が指定されている場合です)。

ウィンドウ操作の順序に関する要件は、年末尾型日付フィールドには適用されません。年末尾型日付フィールドに対して THROUGH 句を含む条件名 VALUE 文節を指定する場合、2 つのリテラルは通常の COBOL 規則に従っていなければなりません。つまり、最初のリテラルは 2 番目のリテラルより小さくなければなりません。

関連概念
『仮定による世紀ウィンドウ』
576 ページの『非日付の取り扱い』

関連タスク
583 ページの『日付処理の明示的制御』

仮定による世紀ウィンドウ

プログラムがウィンドウ化日付フィールドに対して演算を実行する場合、コンパイラーは、そのコンパイル単位の世紀ウィンドウ (すなわち、YEARWINDOW コンパイラー・オプションによって定義されたもの) を適用します。ウィンドウ化日付フィールドを非日付と一緒に使用する場合で、コンテキスト上、非日付もウィンドウ化日付として扱う必要がある場合、コンパイラーは仮定による世紀ウィンドウを使って非日付フィールドを解決します。

仮定による世紀ウィンドウは 1900-1999 であり、これは、一般にはコンパイル単位の世紀ウィンドウと同じではありません。

多くの場合、特にリテラルの非日付の場合、この仮定による世紀ウィンドウは正しい選択です。次の構成では、リテラルは、世紀ウィンドウがたとえば 1975 ～ 2074 の場合でも、1972 年 1 月 1 日という元の意味を保たなければならず、2072 に変わるべきではありません。

```
01  Manufacturing-Record.
    03  Makers-Date Pic X(6) Date Format yyxxxx.
. . .
    If Makers-Date Greater than "720101" . . .
```

この仮定が正しくても、DATEVAL 組み込み関数を使うことによって年を明確に指定し、警告レベルの診断メッセージ (仮定による世紀ウィンドウを適用することから生じるメッセージ) が出ないようにするのが適切です。

```
If Makers-Date Greater than
              Function Dateval("19720101" "YYYYXXXX") . . .
```

時には、仮定が正しくない場合があります。次の例の場合、Project-Controls が COPY メンバーの中にあり、このメンバーが、西暦 2000 年処理用にまだアップグレードされていない他のアプリケーションによって使用され、このため Date-Target では DATE FORMAT 文節を指定することができないものとします。

```
01  Project-Controls.
    03  Date-Target     Pic 9(6).
. . .
01  Progress-Record.
    03  Date-Complete   Pic 9(6) Date Format yyxxxx.
. . .
    If Date-Complete Less than Date-Target . . .
```

この例で、Date-Complete が Date-Target より前の日付 (より小) になるようにするには、次の 3 つの条件が真でなければなりません。

- 世紀ウィンドウが 1910-2009 である。
- Date-Complete が 991202 (グレゴリオ日付 1999 年 12 月 2 日) である。
- Date-Target が 000115 (グレゴリオ日付 2000 年 1 月 15 日) である。

ただし、Date-Target は、DATE FORMAT 文節を持っていないので、非日付です。したがって、これに適用される世紀ウィンドウは、仮定による世紀ウィンドウ 1900-1999 であり、1900 年 1 月 15 日として処理されることになります。この結果、Date-Complete は Date-Target より大きいことになり、これは意図した結果ではありません。

この場合には、DATEVAL 組み込み関数を使って、この比較のために Date-Target を日付フィールドに変換すべきです。たとえば、次のようになります。

```
If Date-Complete Less than
                Function Dateval (Date-Target "YYXXXX") . . .
```

関連タスク

583 ページの『日付処理の明示的制御』

非日付の取り扱い

最も単純な種類の非日付は、リテラル値です。以下の項目も非日付です。

- データ記述に DATE FORMAT 文節が含まれていないデータ項目。
- 一部の算術式の結果 (中間または最終的)。たとえば、2 つの日付フィールドの差は非日付ですが、日付フィールドと非日付の和は日付フィールドです。
- UNDATE 組み込み関数からの出力。

非日付を日付フィールドと一緒に使用する場合、コンパイラーは非日付を形式が日付フィールドと互換性のある日付、または単純な数値と見なします。この解釈は、次に示すように、日付フィールドおよび非日付が使われたコンテキストによって異なります。

- 比較

日付フィールドを非日付と比較する場合、非日付は、年と年以外の文字の文字数の点で日付フィールドと互換性があると見なされます。次の例では、971231 という非日付リテラルをウィンドウ化日付フィールドと比較します。

```
01  Date-1              Pic 9(6) Date Format yyxxxx.
. . .
    If Date-1 Greater than 971231 . . .
```

非日付リテラル 971231 は、DATE FORMAT が Date-1 と同じであるかのように扱われますが、基本年は 1900 年です。

- 算術演算

  サポートされるすべての算術演算において、非日付フィールドは単純な数値として扱われます。次の例では、Date-2 のグレゴリオ日付に数値 10000 を加算しています。つまり、日付に 1 年が加えられます。

  ```
  01  Date-2              Pic 9(6) Date Format yyxxxx.
  . . .
      Add 10000 to Date-2.
  ```

- MOVE ステートメント

  日付フィールドを非日付に移動することはサポートされません。しかし、UNDATE 組み込み関数を使ってこれを行うことができます。

  非日付を日付フィールドに移動する場合、送り出しフィールドは、年と年以外の文字の文字数の点で、受け取りフィールドと互換性があると見なされます。たとえば、非日付をウィンドウ化日付フィールドに移動する場合、非日付フィールドには 2 桁年と互換性のある日付が入っているものと想定されます。

トリガーおよび限界の設定

トリガーと限界は、その値が非数値であるため、またはその値の中の年以外の部分が実際の日付であり得ないような値であるため、有効な日付には決して一致しない特殊な値です。トリガーと限界は、日付フィールドにおいて認識されますが、日付フィールドと組み合わせて使われる非日付データにおいても認識されます。

フィールドのタイプ	特殊値
英数字のウィンドウ化日付または年フィールド	HIGH-VALUE、LOW-VALUE、および SPACE
DATE FORMAT 文節で少なくとも 1 つの X が指定された英数字および数字のウィンドウ化日付フィールド (すなわち、年号だけではない日付フィールド)	すべて 9 またはすべて 0

トリガーと限界の相違は値にあるのではなく、その使用方法にあります。トリガーまたは限界としては、任意の特殊値を使用できます。

特殊値をトリガーとして使うと、「日付が未初期化」または「会計が決済日超過」などの個々の条件を示すことができます。特殊値を限界として使うと、それは有効などの日付よりも前または後の日付として使われることになります。LOW-VALUE、SPACE、および 0 は下限であり、HIGH-VALUE および全桁 9 は上限です。

トリガーと限界のサポートを有効にするには、DATEPROC コンパイラー・オプションの TRIG サブオプションを指定します。DATEPROC(TRIG) コンパイラー・オプション

が有効であると、ウィンドウ化日付フィールドの自動拡張において (比較や算術などでオペランドとして使われる前)、それらの特殊値が考慮されます。

DATEPROC(TRIG) オプションを使用すると、ウィンドウ化日付の比較処理が遅くなります。DATEPROC(NOTRIG) オプションは、すべてのウィンドウ化日付フィールドの有効な日付値を想定するパフォーマンス・オプションです。

実ウィンドウ化日付フィールドまたは仮定ウィンドウ化日付フィールドの内容がトリガーである場合は、通常のウィンドウ操作の場合のように世紀の値が 19 または 20 と推測されるのではなく、日付の拡張結果の世紀部分にトリガー値が波及するかのようにして拡張されます。それによって、アプリケーションで特殊値をテストしたり、上限または下限として特殊値を使用したりできます。また、DATEPROC(TRIG) を指定すると、SORT および MERGE ステートメントによる DFSORT 特殊標識のサポートも使用可能になります。これらは、トリガーと限界に対応します。

『例: 限界の使用』

関連タスク
579 ページの『符号条件の使用』

例: 限界の使用

たとえば、予約購読の期限が切れたかどうかをアプリケーションで検査する際に、一部の予約購読は無期限扱いにしたいという場合を考えてみましょう。期限日付フィールドは、通常の期限日付を入れたり、「無期限」予約購読として使う高い限界値を入れたりするのに使うことができます。たとえば、次のようなコードを考えてみます。

```
Process DateProc(Flag,Trig). . .
. . .
  01 SubscriptionRecord.
. . .
     03 ExpirationDate  PIC 9(6) Date Format yyxxxx.
. . .
  77 TodaysDate  Pic 9(6) Date Format yyxxxx.
. . .
     If TodaysDate >= ExpirationDate
        Perform SubscriptionExpired
```

次のような設定値になっているとします。

- 今日は 2000 年 1 月 4 日で、TodaysDate では 000104 として表されます。
- ある予約購読レコードの通常の期限日付が 1999 年 12 月 31 日で、991232 として表されます。
- 特別な予約購読期限日付は 999999 とコーディングされています。

どちらの日付もウィンドウ化されているので、最初の予約購読は 20000104 を 19991231 に対して比較するものとしてテストされます。そのためテストは正常に実施されます。しかし、コンパイラーが特殊値を検出すると、ウィンドウ操作ではなく拡張が引き起こされます。このため、20000104 が 99999999 と比較されるものとしてテストが行われるので、テストの結果は偽となり、これは常に偽となります。

符号条件の使用

アプリケーションによっては、日付フィールドのトリガーとして、つまり何らかの特別な処理が必要であることを示すために、0 などの特殊な値を使います。たとえば、Orders ファイルで Order-Date に 0 の値を使うと、レコードはオーダー・レコードではなく顧客合計レコードであることを意味するという場合です。プログラムは、次のように日付を 0 と比較します。

```
01  Order-Record.
    05  Order-Date      Pic S9(5) Comp-3 Date Format yyxxx.
. . .
    If Order-Date Equal Zero Then . . .
```

しかし、DATEPROC コンパイラー・オプションの NOTRIG サブオプションを指定してコンパイルしている場合、この比較は有効ではありません。リテラル値の Zero は非日付であるため、仮定による世紀ウィンドウに対してウィンドウ操作されて値は 1900000 になります。

別の方法として、あるいはワークステーションでコンパイルする場合は、次のように、リテラル比較ではなく符号条件を使用することができます。

```
If Order-Date Is Zero Then . . .
```

符号条件を使うと、Order-Date は非日付として扱われ、世紀ウィンドウは考慮に入れられません。

このアプローチが適用されるのは、符号条件内のオペランドが算術式ではなく単純な ID である場合だけです。式を指定した場合は、適宜世紀ウィンドウが適用されてその式が最初に評価されます。そのあと、その式の結果に対して符号条件が比較されます。

DATEPROC コンパイラー・オプションの TRIG サブオプションの代わりに UNDATE 組み込み関数を使用しても、同じ結果が得られます。

関連概念

576 ページの『非日付の取り扱い』

関連タスク

577 ページの『トリガーおよび限界の設定』
583 ページの『日付処理の明示的制御』

日付によるソートおよびマージ

DFSORT は、ソートおよびマージを行う IBM ライセンス・プログラムです。DFSORT に言及している場所では、任意の同等のソートまたはマージ製品を使用することができます。

ご使用のソート製品が Y2PAST オプションとウィンドウ化西暦年 ID (Y2B、Y2C、Y2D、Y2S、および Y2Z) をサポートしている場合は、ウィンドウ化日付フィールドをソート・キーとして使用してソートおよびマージ操作を実行することができます。実際のところ DATE FORMAT 文節を使って指定できるすべての日付フィールドがサポートされますが、それには 2 進年フィールドと年末尾型日付フィールドも含まれます。フィールドは、YEARWINDOW コンパイラー・オプションで指定された世紀ウィンドウに従って、ウィンドウ化西暦年シーケンスでソートされます。

ご使用のソート製品が Y2T、Y2U、Y2W、Y2X、および Y2Y の日付フィールド ID もサポートする場合は、DATEPROC コンパイラー・オプションの TRIG サブオプションを使用することができます。(APAR PQ19684 を適用した DFSORT では、これらの日付フィールド ID のためのサポートが追加されています。)

DFSORT で認識される特殊標識は、COBOL でサポートされるものとまったく同じです。つまり、英数字の日付または年フィールドの場合の LOW-VALUE、HIGH-VALUE、および SPACE と、少なくとも 1 つの非年桁を含む数字および英数字の日付フィールドでの全桁 0 および全桁 9 です。

『例: 日付および時刻によるソート』

関連タスク
OPTION 制御ステートメント (Y2PAST) (*DFSORT アプリケーション・プログラミングの手引き*)

関連参照
322 ページの『DATEPROC』
358 ページの『YEARWINDOW』

例: 日付および時刻によるソート

以下の例はトランザクション・ファイルのソートを示したものであり、トランザクション・レコードは顧客番号内の日付および時刻によってソートされます。フィールド Trans-Date は、ウィンドウ操作される年間通算日の日付フィールドです。

```
SD  Transaction-File
    Record Contains 29 Characters
    Data Record is Transaction-Record

01  Transaction-Record.
    05  Trans-Account  PIC 9(8).
    05  Trans-Type     PIC X.
    05  Trans-Date     PIC 9(5) Date Format yyxxx.
    05  Trans-Time     PIC 9(6).
    05  Trans-Amount   PIC 9(7)V99.
. . .
    Sort Transaction-File
         On Ascending Key  Trans-Account
                           Trans-Date
                           Trans-Time
         Using Input-File
         Giving Sorted-File.
```

COBOL は DFSORT に関連情報を渡し、DFSORT がソート操作を正しく行えるようにします。COBOL が常に DFSORT に渡す情報のほか、COBOL は以下の情報も渡します。

- Y2PAST ソート・オプションとしての世紀ウィンドウ
- ウィンドウ化西暦年フィールドおよび Trans-Date の日付形式

DFSORT はこの情報を使用してソートを実行します。

日付フィールドへの算術の実行

数値日付フィールドには、数値データ項目と同じ方法で算術演算を実行することができ、該当するところでは計算で世紀ウィンドウが使用されます。しかし、算術式や算術ステートメントのどこで日付フィールドを使用できるかについては制限があります。

日付フィールドが含まれる算術演算は、次のものに限定されます。

- 非日付を日付フィールドに加算すること
- 非日付を日付フィールドから減算すること
- 互換性のある日付フィールドから日付フィールドを減算し、非日付の結果を与えること

以下の算術演算は使用できません。

- 互換性のない日付フィールド間の演算
- 2 つの日付フィールドを加算すること
- 日付フィールドを非日付から減算すること
- 日付フィールドに適用される単項減算
- 日付フィールドの関係した乗算、除算、またはべき乗計算
- 年末尾型日付フィールドを指定した算術式
- 年末尾型日付フィールドを指定する算術式 (送り出しフィールドが非日付の場合に、受け取りデータ項目として指定する場合を除く)

日付フィールドの年号部分には日付のセマンティクスが用意されています (年号以外の部分にはありません)。たとえば、値 980831 を含むグレゴリオ暦のウィンドウ操作日付フィールドに 1 を加算すると、980901 ではなく 980832 の結果になります。

ウィンドウ化日付フィールドからのオーバーフローを考慮する

(年末尾型でない) ウィンドウ化日付フィールドが算術演算に関与するときは、世紀ウィンドウに応じて、まずフィールドの年号要素の値が 1900 または 2000 だけ増分されたかのように処理されます。たとえば、次のようになります。

```
01 Review-Record.
    03 Last-Review-Year Pic 99 Date Format yy.
    03 Next-Review-Year Pic 99 Date Format yy.
. . .
    Add 10 to Last-Review-Year Giving Next-Review-Year.
```

世紀ウィンドウが 1910-2009 で、Last-Review-Year の値が 98 である場合は、1998 を与えるため、まず Last-Review-Year が 1900 だけ増分されたかのようにして計算が進められます。次に、ADD 演算が実行され、2008 の結果が与えられます。この結果は、08 として Next-Review-Year に保管されます。

しかし、次のステートメントを使用すると、2018 という結果になります。

```
Add 20 to Last-Review-Year Giving Next-Review-Year.
```

この結果は、世紀ウィンドウの範囲外になります。結果が Next-Review-Year に保管されると、それ以降に Next-Review-Year が参照された場合に 1918 として解釈

されるので、誤りになります。この場合、演算の結果は、ON SIZE ERROR 句が ADD ステートメントに指定されているかどうかによって異なります。

- SIZE ERROR が指定されている場合、受け取りフィールドは変更されず、SIZE ERROR 命令ステートメントが実行されます。
- SIZE ERROR が指定されていない場合、結果は、左端から桁が切り捨てられて受け取りフィールドに保管されます。

この考慮事項は、内部ブリッジングを使用するときには大切です。出力ファイルに書き出すために 4 桁年号の日付フィールドを 2 桁に短縮するときは、日付が世紀ウィンドウの範囲内になるようにする必要があります。そうすると、2 桁年号の日付がフィールドで正しく表されるようになります。

適切な計算が行われるようにするには、短縮を行うための COMPUTE ステートメントに、ウィンドウ外条件を処理するための SIZE ERROR 句を指定しなければなりません。たとえば、次のようになります。

```
Compute Output-Date-YY = Work-Date-YYYY
On Size Error Perform CenturyWindowOverflow.
```

ウィンドウ化日付受け取り側についての SIZE ERROR 処理は、世紀ウィンドウの範囲外となるすべての年号値を認識します。すなわち、世紀ウィンドウの開始年より小さい年号値も、世紀ウィンドウの終了年より大きい年号値の場合と同様に、SIZE ERROR 条件を引き起こします。

また、DATEPROC(TRIG) コンパイラー・オプションが有効なら、結果に 0 または 9 のトリガー値が含まれている場合、その結果の年部分 (それぞれ、00 または 99) が世紀ウィンドウの範囲内であっても SIZE ERROR 条件が発生します。

評価の順序の指定

算術式の中の日付フィールドに関する制限のために、以前は正常にコンパイルされたプログラムが、今では一部のデータ項目が日付フィールドに変更された結果、診断メッセージが出るようになることがあるかもしれません。

次の例を考えてみてください。

```
01 Dates-Record.
    03 Start-Year-1 Pic 99 Date Format yy.
    03 End-Year-1 Pic 99 Date Format yy.
    03 Start-Year-2 Pic 99 Date Format yy.
    03 End-Year-2 Pic 99 Date Format yy.
. . .
    Compute End-Year-2 = Start-Year-2 + End-Year-1 - Start-Year-1.
```

この場合、評価される最初の算術式は次のものです。

```
Start-Year-2 + End-Year-1
```

しかし、2 つの日付フィールドを加算することは許可されません。これらの日付フィールドを解決するためには、括弧を使用して、許可される算術式の部分を分離しなければなりません。たとえば、次のようになります。

```
Compute End-Year-2 = Start-Year-2 + (End-Year-1 - Start-Year-1).
```

この場合、評価される最初の算術式は次のものです。

```
End-Year-1 - Start-Year-1
```

ある日付フィールドから別の日付フィールドを減算することは許可され、非日付の結果が与えられます。次に、この非日付の結果が日付フィールド End-Year-1 に加算され、日付フィールドの結果が与えられ、これが End-Year-2 に保管されます。

関連タスク
568 ページの『内部ブリッジングの使用』

日付処理の明示的制御

COBOL データ項目を、特定の条件のときのみ、あるいはプログラムの特定の部分でのみ、日付フィールドとして扱いたい場合があるかもしれません。あるいは、アプリケーションに含まれる 2 桁年の日付フィールドを、他のソフトウェア・プロダクトとの対話があるためにウィンドウ化日付フィールドとして宣言できないことがあるかもしれません。たとえば、日付フィールドが、特に解釈のための操作をすることなく純粋な 2 進数の内容によってのみ認識されるコンテキストでは、そのフィールドの日付をウィンドウ操作することはできません。このような日付フィールドには、以下のものがあります。

- VSAM ファイルのキー
- DB2 のようなデータベース・システムの検索フィールド
- CICS コマンドのキー・フィールド

逆に日付フィールドを、プログラムの特定の部分では非日付として扱いたいときがあるかもしれません。

COBOL は、このような条件を扱うために 2 つの組み込み関数を提供します。

DATEVAL
非日付を日付フィールドに変換します。

UNDATE 日付フィールドを非日付に変換します。

DATEVAL の使用

DATEVAL 組み込み関数を使用すると、非日付を日付フィールドに変換することができ、COBOL は関連する日付処理をそのフィールドに適用するようになります。この関数の最初の引き数には変換対象の非日付を指定し、2 番目の引き数には日付形式を指定します。2 番目の引き数は、DATE FORMAT 文節の日付パターンの指定に類似した指定を持つリテラル・ストリングです。

ほとんどの場合コンパイラーは、非日付の解釈について正しい仮定をしますが、その仮定に伴って警告レベルの診断メッセージを出します。一般にこのメッセージは、ウィンドウ化日付がリテラルと比較された場合に発生します。

```
03  When-Made        Pic x(6) Date Format yyxxxx.
. . .
If When-Made = "850701" Perform Warranty-Check.
```

リテラルは、互換性のあるウィンドウ化日付と見なされますが、1900 ～ 1999 の世紀ウィンドウが使われているので、これは 1985 年 7 月 15 日を表しています。DATEVAL 組み込み関数を使ってリテラルの日付の年を明示して、警告メッセージを出さないようにすることができます。

```
If When-Made = Function Dateval("19850701" "YYYYXXXX")
    Perform Warranty-Check.
```

『例: DATEVAL』

UNDATE の使用

UNDATE 組み込み関数は、日付フィールドを非日付に変換することにより、日付処理なしで参照できるようにします。

重要: UNDATE を使うのは最後の手段としてのみにし、できる限り使わないようにしてください。プログラムでの日付フィールドのフローをコンパイラーが見失ってしまうからです。もし見失った場合、日付の比較が正しくウィンドウ操作されないことがあります。MOVE および COMPUTE では、関数 UNDATE の代わりに DATE FORMAT 文節を使うようにしてください。

『例: UNDATE』

例: DATEVAL

例として、プログラムにフィールド Date-Copied があって、このフィールドがプログラムの中で何回も参照されるが、その参照のほとんどが単にレコード間でそのフィールドを移動させるだけ、または印刷用に形式設定し直すだけであるものとします。1 つの参照箇所においてのみ、他の日付との比較の目的でその内容を日付であると見なしています。

この場合、このフィールドは非日付としておいたまま、比較ステートメントの中で DATEVAL 組み込み関数を使用するほうがよいでしょう。たとえば、次のようになります。

```
03  Date-Distributed Pic 9(6) Date Format yyxxxx.
03  Date-Copied      Pic 9(6).
. . .
If FUNCTION DATEVAL(Date-Copied "YYXXXX") Less than
                    Date-Distributed . . .
```

この例では、DATEVAL 組み込み関数は Date-Copied を日付フィールドに変換し、比較を意味のあるものにしています。

関連参照

DATEVAL (*Enterprise COBOL 言語解説書*)

例: UNDATE

以下の例の Invoice-Record のフィールド Invoice-Date は、ウィンドウ操作による年間通算日です。一部のレコードでは、このフィールドの値が 00999 であり、これは「本当の」送り状 (invoice) レコードではなくて、ファイル制御情報を内容とするレコードであることを示すものです。

Invoice-Date には DATE FORMAT 文節が与えられています。プログラム内でのこのフィールドへの参照のほとんどが日付固有であるからです。しかし、制御レコードの有無が検査される場合には、年部分の値が 00 になっていると、混乱を招きかねません。Invoice-Date の 00 の年は、世紀ウィンドウに応じて、1900 または 2000

のいずれかの年を表します。これは、非日付 (例の中ではリテラル 00999) と比較されますが、それは、常に仮定による世紀ウィンドウに対してウィンドウ操作されるため、常に 1900 年を表します。

比較が矛盾した結果にならないようにするためには、UNDATE 組み込み関数を使用して Invoice-Date を非日付に変換しなければなりません。したがって、IF ステートメントがどの日付フィールドも比較しない場合には、ウィンドウ操作を適用させる必要はありません。たとえば、次のようになります。

```
01  Invoice-Record.
    03  Invoice-Date    Pic x(5) Date Format yyxxx.
. . .
    If FUNCTION UNDATE(Invoice-Date) Equal "00999" . . .
```

関連参照

UNDATE (*Enterprise COBOL 言語解説書*)

日付関連診断メッセージの分析および回避

DATEPROC(FLAG) コンパイラー・オプションが有効であると、コンパイラーは、日付フィールドを定義または参照するそれぞれの文ごとに診断メッセージを作成します。コンパイラー生成の他のすべてのメッセージと同じく、日付関連の各メッセージも以下の重大度レベルの 1 つを持っています。

- 通知レベル。これは、日付フィールドの定義または使用に注意を喚起するためのものです。
- 警告レベル。プログラムにコーディングされている情報が不十分なために、日付フィールドまたは非日付に関してコンパイラーがなんらかの仮定を設定したことを示すため、または正しいことを手作業で検査しなければならない日付ロジックの場所を示すためのものです。コンパイルは続行し、仮定は適用されます。
- エラー・レベル。日付フィールドの使い方が誤っていることを表します。コンパイルは続行しますが、実行時結果は予測不能です。
- 重大レベル。日付フィールドの使い方が誤っていることを表します。このエラーが生成される原因となったステートメントは、コンパイルから廃棄されます。

MLE メッセージを最も簡単に使用するには、FLAG オプション設定を使ってコンパイルします。これによって、ソース・リストの中でメッセージの参照する行の直後にメッセージが出力されるようになります。すべての MLE メッセージを表示したり、重大度によって選択したりできます。

すべての MLE メッセージを表示するには、FLAG(I,I) および DATEPROC(FLAG) コンパイラー・オプションを指定します。自分のプログラム内の日付フィールドが MLE でどのように処理されるかを理解するため、最初はすべてのメッセージを表示することをお勧めします。たとえば、コンパイル・リストを使ってプログラム内のデータ使用の静的分析をしたい場合は、FLAG (I,I) を使用してください。

しかし、MLE 固有のコンパイルでは FLAG(W,W) を指定することをお勧めします。重大レベル (S レベル) エラー・メッセージとエラー・レベル (E レベル) メッセージはすべて訂正する必要があります。警告レベル (W レベル) のメッセージについ

ては、各メッセージを調べ、以下の指針に従って、メッセージを除去するか、または回避不能なメッセージの場合はコンパイラーが正しい仮定を行うようにしなければなりません。

- 診断メッセージが、DATE FORMAT 文節を持っていなければならない日付データ項目を示すことがあります。これらの項目に DATE FORMAT 文節を追加するか、またはこれらの項目への参照で DATEVAL 組み込み関数を使用してください。
- 日付フィールドを対象にする比較条件の中、または日付フィールドを含む算術式の中でリテラルを使用する場合には、特別な注意が必要です。リテラル (および非日付データ項目) に DATEVAL 関数を使って、使用する DATE FORMAT パターンを指定することができます。UNDATE 関数は、最後の手段として、日付中心の動作を望まないコンテキストで使用される日付フィールドを使用可能にするために使うことができます。
- REDEFINES および RENAMES 文節が指定されている場合、日付フィールドと非日付が同じ保管場所を占有していると、コンパイラーは警告レベルの診断メッセージを出すことがあります。このような場合には注意深く調べて、種々の別名の付いたデータ項目のすべての使い方が正しいこと、および非日付と認識された再定義が実際にどれも日付でないこと、またはプログラム内の日付ロジックに悪影響を与えていないことを確認してください。

W レベル・メッセージが出ても気にしなければいいのですが、コードを変更して戻りコード = 0 のコンパイルを達成するのも良い方法です。

警告レベルの診断メッセージを回避するには、以下の簡単な指針に従ってください。

- 比較で使っていなくても、日付データが含まれているすべてのデータ項目に DATE FORMAT 文節を追加する。
- 日付フィールドが意味をなさないコンテキスト、たとえば FILE STATUS、PASSWORD、ASSIGN USING、LABEL RECORD、または LINAGE 項目などでは、日付フィールドを指定しない。指定すると、警告レベルのメッセージが出されて、日付フィールドが非日付として扱われます。
- 日付フィールドの暗黙の別名または明示された別名が、日付フィールドだけからなるグループ項目などの中で互換性があることを確認する。
- 日付フィールドが VALUE 文節を定義されている場合、その値が日付フィールド定義と互換性があることを確認する。
- 非日付を日付フィールドとして扱いたい場合 (たとえば、非日付を日付フィールドに移動する場合や、ウィンドウ化日付を非日付と比較する場合など) で、ウィンドウ化日付比較を行う場合は、DATEVAL 組み込み関数を使用する。DATEVAL を使わないと、コンパイラーは非日付の使用に関してある仮定を行い、警告レベルの診断メッセージを作成します。その仮定が正しくても、DATEVAL を使用すればメッセージを排除できます。
- 日付フィールドを非日付として扱いたい場合 (たとえば、日付フィールドを非日付に移動する場合や、非日付をウィンドウ化日付と比較する場合など) で、ウィンドウ化日付比較を行わない場合は、UNDATE 組み込み関数を使用する。

関連タスク

583 ページの『日付処理の明示的制御』

日付関連診断メッセージの分析(*COBOL 2000 年言語拡張の手引き*)

日付の処理における問題の回避

2000 年言語拡張を使用するために COBOL プログラムを変更した場合、プログラムの部分によっては、予測できない動作上の変更を解決するための特別な注意が必要になることがあります。ここでは、考慮に入れておく必要のあるいくつかの分野について説明します。

パック 10 進数フィールドでの問題の回避

COMPUTATIONAL-3 フィールド (パック 10 進数形式) は、しばしば、桁数が奇数であると定義されます。このフィールドがそのような桁数として使われない場合でもそう定義されます。それは、内部表現ではパック 10 進数の桁数が常に奇数になっているからです。

たとえば、6 桁のグレゴリオ暦日付を入れるフィールドを PIC S9(6) COMP-3 と宣言すると、4 バイトのストレージが予約されることになります。しかしプログラマーは、PIC S9(7) を指定しても予約バイトは常に 4 バイトであることを考慮して、フィールドを PIC S9(7) として宣言し、最高位桁を常に 0 にするようにしているかもしれません。

このフィールドに DATE FORMAT YYXXXX という文節を追加した場合、コンパイラーから診断メッセージが出ます。PICTURE 文節内の桁数が日付形式指定のサイズに一致しないからです。その場合、フィールド使用を 1 つずつ慎重に調べる必要があります。高位桁を使わない場合は、単にフィールド定義を PIC S9(6) に変更することができます。使う場合 (たとえば同じフィールドに日付以外の値も入れることがある場合)、他のなんらかのアクションが必要になります。

- REDEFINES 文節を使って、日付および非日付の両方としてフィールドを定義する (この場合も、警告レベルの診断メッセージが出ます)。
- 日付を入れる別の WORKING-STORAGE フィールドを定義し、数値フィールドを新規フィールドに移動する。
- データ項目に DATE FORMAT 文節を追加せず、それを日付フィールドとして参照するところで DATEVAL 組み込み関数を使用する。

拡張日付フィールドからウィンドウ化日付フィールドへの移動

拡張英数字日付フィールドをウィンドウ化日付フィールドに移動する場合には、英数字移動についての通常の COBOL 規則に従いません。送り出しフィールドと受け取りフィールドのどちらも日付フィールドである場合、移動は通常の左寄せではなく右寄せになります。拡張からウィンドウ操作への (縮小) 移動の場合、年の先頭の 2 桁が切り捨てられることになります。

送り出しフィールドの内容によっては、このような移動は結果的に誤りを生じることがあります。たとえば、次のようになります。

```
77  Year-Of-Birth-Exp    Pic x(4) Date Format yyyy.
77  Year-Of-Birth-Win    Pic xx   Date Format yy.
. . .
    Move Year-Of-Birth-Exp to Year-Of-Birth-Win.
```

Year-Of-Birth-Exp が '1925' を含んでいる場合、Year-Of-Birth-Win は '25' が含まれます。しかし、世紀ウィンドウが 1930-2029 の場合、Year-Of-Birth-Win のあとの参照は 25 と見なされます。しかし、それは誤っています。

第 7 部 パフォーマンスおよび生産性の向上

第 32 章 プログラムの調整 591
最適プログラミング・スタイルの使用 592
構造化プログラミングの使用 592
一括表示表現 592
シンボリック定数の使用. 593
定数計算のグループ化 593
重複計算のグループ化 593
効率的なデータ型の選択. 594
計算データ項目. 594
一貫性のあるデータ型 595
算術式. 595
べき乗計算 595
テーブルの効率的な処理. 595
テーブル参照の最適化 597
定数および変数項目の最適化 597
重複項目の最適化 598
可変長項目の最適化 598
直接指標付けと相対指標付けの比較 598
コードの最適化. 599
最適化. 599
含まれているプログラムのプロシージャー統合 600
PERFORM プロシージャー統合 600
例: PERFORM プロシージャー統合 601
パフォーマンスを向上させるためのコンパイラー機能の選択 602
パフォーマンスに関連するコンパイラー・オプション 603
パフォーマンスの評価 606
CICS、IMS、または VSAM での効率的な実行 . . 607
CICS 607
IMS 607
VSAM. 607

第 33 章 コーディングの簡略化 609
反復コーディングの除去. 609
例: COPY ステートメントの使用. 610
言語環境プログラム呼び出し可能サービスの使用 611
言語環境プログラム呼び出し可能サービスのサンプル・リスト 612
言語環境プログラム・サービスの呼び出し. . . 613
例: 言語環境プログラム呼び出し可能サービス 614

© Copyright IBM Corp. 1991, 2001

第 32 章 プログラムの調整

プログラムが分かりやすいものであってこそ、パフォーマンスの評価を行うことができます。制御フローが混乱したプログラムは、理解したり維持するのが困難です。また、制御フローが混乱していると、コードの最適化も禁止されます。このため、パフォーマンスの向上を直接試みる前に、いくつかの局面を評価する必要があります。

1. プログラムの基礎アルゴリズムを調べる。最高のパフォーマンスを得るためには、適切なアルゴリズムが不可欠です。たとえば、百万個の品目を分類するような洗練されたアルゴリズムは、単純なアルゴリズムよりも何百万倍も高速になります。
2. データ構造を調べる。データ構造はアルゴリズムに適したものにする必要があります。プログラムが頻繁にデータにアクセスする場合は、可能であれば、データにアクセスするために必要なステップの数を減らします。
3. アルゴリズムとデータ構造を改善したら、パフォーマンスに影響を与える COBOL ソース・コードのその他の詳細を調べる。

より優れたコード・シーケンスを生成し、システム・サービスをより活用するようなプログラムを作成することができます。プログラムのパフォーマンスに影響を与えるのは、次の 5 つの分野です。

- コーディング技法。これには、最適化プログラムを援助するプログラミング・スタイルの使用、効率的なデータ型の選択、およびテーブルの効率的な処理が含まれます。
- 最適化。`OPTIMIZE` コンパイラー・オプションを使用してコードを最適化することができます。
- コンパイラー・オプションおよび `USE FOR DEBUGGING ON ALL PROCEDURES`。特定のコンパイラー・オプションおよび言語は、プログラムの効率に影響を与えます。
- ランタイム環境。コンパイル済みプログラムの実行方法を制御するランタイム・オプションおよびその他の実行時考慮事項は、十分注意して選択しなければなりません。
- CICS、IMS、または使用している VSAM のもとでの実行。これらのプログラムを効率的に実行するのに役立つさまざまなヒントがあります。

関連概念

599 ページの『最適化』

関連タスク

592 ページの『最適プログラミング・スタイルの使用』
594 ページの『効率的なデータ型の選択』
595 ページの『テーブルの効率的な処理』
599 ページの『コードの最適化』
602 ページの『パフォーマンスを向上させるためのコンパイラー機能の選択』
ランタイム・オプションの指定 (*言語環境プログラム プログラミング・ガイド*)
607 ページの『CICS、IMS、または VSAM での効率的な実行』

© Copyright IBM Corp. 1991, 2001

関連参照
603 ページの『パフォーマンスに関連するコンパイラー・オプション』
ストレージ・パフォーマンスに関する考慮事項 (*言語環境プログラム プログラミング・ガイド*)

最適プログラミング・スタイルの使用

特定の状況によっては、使用するコーディング・スタイルが、最適化プログラムによるコードの処理方法に影響を与える可能性があります。

構造化プログラミングの使用

構造化プログラミング・ステートメント (EVALUATE やインライン PERFORM など) を使用すると、プログラムがより理解しやすくなり、線形の制御フローが生成されます。すると、最適化プログラムはプログラムのより多くの領域に作用することができるため、より効率のよいコードが与えられます。

次の構成は使用しないでください。

- ALTER ステートメント
- 逆方向分岐（PERFORM が適さないループに必要とされる場合を除く）
- 変則的な制御フローを伴う PERFORM プロシージャー。たとえば、プロシージャーの終わりに制御が渡されないために、PERFORM ステートメントに戻れない PERFORM プロシージャー。

トップダウン・プログラミングの構成を使用してください。ライン外の PERFORM ステートメントは、トップダウン・プログラミングを行う本来の手段です。最適化プログラムを使用すると、ライン外の PERFORM ステートメントがインラインの PERFORM ステートメントと同じくらい効率的になることがよくあります。これは、最適化プログラムがリンケージ・コードを簡略化または除去するためです。

一括表示表現

一括表示は、多数の計算を節約することができます。たとえば、次のようにコーディングします。

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I)
END-PERFORM
COMPUTE TOTAL = TOTAL * DISCOUNT
```

以下のコーディングより効率的になります。

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I) * DISCOUNT
END-PERFORM
```

最適化プログラムは、一括表示を行いません。

シンボリック定数の使用

最適化プログラムに、プログラム中でデータ項目を定数として認識させたい場合には、その項目を VALUE 文節を使用して初期化し、プログラム内のどの場所でもその項目を変更しないようにしてください。

データ項目を BY REFERENCE によってサブプログラムに渡すと、最適化プログラムはその項目を外部データ項目と見なして、サブプログラムを呼び出すたびに、その項目が変更されるものと想定します。

リテラルをデータ項目に移動すると、最適化プログラムはそれを定数と見なしますが、それは、MOVE ステートメントに続くプログラムの限られた領域内においてのみです。

定数計算のグループ化

式のいくつかの項目が定数の場合は、最適化プログラムがそれらを最適化できるようにしてください。式を評価する場合、コンパイラーは COBOL の左から右への評価規則に従います。したがって、すべての定数を式の左側に移動させるか、または括弧で囲んでグループ化します。

たとえば、V1、V2、および V3 が変数で、C1、C2、および C3 が定数の場合、定数計算を含む式の方が、定数計算を含まない式より望ましいものになります。

効率がよい

```
V1 * V2 * V3 * (C1 * C2 * C3)
C1 + C2 + C3 + V1 + V2 + V3
```

効率がよくない

```
V1 * V2 * V3 * C1 * C2 * C3
V1 + C1 + V2 + C2 + V3 + C3
```

実動プログラミングではしばしば、式の右側に定数演算項目を置く傾向があります。しかしその結果、最適化が行われないために、効率のよくないコードが生成される可能性があります。

重複計算のグループ化

異なる式のいくつかのコンポーネントが重複している場合は、コンパイラーがそれらを最適化できるようにしてください。算術式の評価の場合、コンパイラーは、左から右への COBOL の評価規則に従います。したがって、すべての重複を式の左側に移動させるか、または括弧で囲んでグループ化します。

たとえば、V1 ～ V5 が変数の場合、V2 * V3 * V4 の計算は、次の 2 つのステートメントの間で重複します (共通の副次式と呼ばれます)。

```
COMPUTE A = V1 * (V2 * V3 * V4)
COMPUTE B = V2 * V3 * V4 * V5
```

次の例では、共通の副次式は V2 + V3 です。

```
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V2 + V3 + V4
```

次の例には、共通の副次式はありません。

```
COMPUTE A = V1 * V2 * V3 * V4
COMPUTE B = V2 * V3 * V4 * V5
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V4 + V2 + V3
```

最適化プログラムは重複計算を除去することができるので、人工的な一時計算を取り入れる必要はありません。不要な処理が入らなければ、プログラムはわかりやすいものになります。

効率的なデータ型の選択

適切なデータ型と PICTURE 文節を選択すると、計算で頻繁に使用される区域において USAGE DISPLAY データ項目を回避できるので、より効率的なコードが作成される可能性があります。一貫性のあるデータ型を使用すると、データ項目に演算を実行する際に変換を行う必要性を減らすことができます。また、固定小数点データ型と浮動小数点データ型をいつ使用するかを慎重に判別することで、プログラム・パフォーマンスを向上させることができます。

計算データ項目

データ項目が主に算術計算のために、または添え字として使用される場合は、その項目のデータ記述記入項目に USAGE BINARY をコーディングします。2 進データを処理する操作は、10 進データを処理する操作よりも高速で行われます。

しかし、固定小数点算術ステートメントが大きな精度 (有効数字) の中間結果を持つ場合、コンパイラーは、オペランドをパック 10 進数形式に変換したうえで、10 進数演算を使用します。固定小数点算術ステートメントについては、コンパイラーは通常、精度が 8 桁以下にとどまる場合には、2 進オペランドを使用した簡単な計算に 2 進数算術演算を使用します。18 桁を超えると、コンパイラーは常に 10 進数算術演算を使用します。9 ～ 18 桁の精度では、コンパイラーはいずれの形式でも使用できます。

BINARY データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 文節の S)。
- 8 桁以下。

8 桁を超えるデータ項目、または DISPLAY データ項目で使用されるデータ項目については、PACKED-DECIMAL を使用してください。PACKED-DECIMAL データ項目で生成されるコードは、場合によっては (特にステートメントが複雑であるか、丸めを指定している場合は) BINARY データ項目で生成されるコードと同じ速さになることがあります。

PACKED-DECIMAL データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 文節の S)。
- ハーフバイトを残さずに正確なバイト数を占めるように、奇数の桁数 (PICTURE 文節の 9 の数)。
- 乗算および除算にライブラリー・ルーチンを使用しないようにするには、PICTURE 指定を 15 桁以下にする。

一貫性のあるデータ型

異なる型のオペランドに演算を実行する場合、一方のオペランドを他方の型と同じ型に変換しなければなりません。各変換ごとにいくつかの命令が必要になります。たとえば、いずれかのオペランドは小数点以下の桁数が適切な数になるように位取りを指定する必要があるかもしれません。

両方のオペランドに同じ使用法を与え、さらに適切な PICTURE 指定を与えることで、一貫性のあるデータ型を使用することによって、変換を大部分は回避できます。つまり、比較、加算、または減算を行う 2 つの数値は、同じ使用法を持つだけでなく、さらに小数部の桁数（PICTURE 文節の V の後の 9 の数）も同じでなければなりません。

算術式

浮動小数点モードで値が求められる算術式の計算が最も効率的になるのは、オペランドに必要な変換が最小限で済む場合です。COMP-1 または COMP-2 であるオペランドを使用すると、最も効率のよいコードが作成されます。

浮動小数点データに高速変換を行うには、整数項目を BINARY または PACKED-DECIMAL (9 桁以下) として宣言します。さらに、COMP-1 または COMP-2 の項目から、9 桁以下の固定小数点整数への変換は (SIZE ERROR が有効ではない場合)、COMP-1 または COMP-2 項目の値が 1,000,000,000 未満の場合に効率がよくなります。

べき乗計算

高速の計算およびより正確な結果を得るためには、大きな指数のべき乗計算には浮動小数点を使用してください。たとえば、以下に示す最初のステートメントは、2 番目のステートメントより速くかつより正確に計算されます。

```
COMPUTE fixed-point1 = fixed-point2 ** 100000.E+00

COMPUTE fixed-point1 = fixed-point2 ** 100000
```

これは、浮動小数点の指数があるため、べき乗計算の計算に浮動小数点が使用されるからです。

関連概念
46 ページの『数値データの形式』

テーブルの効率的な処理

テーブル処理の操作には、特にテーブル処理がアプリケーションの主要な部分を構成している場合には、十分な注意を払ってください。テーブル処理の操作の効率を向上させる技法はいくつかあります。これらはまた、最適化プログラムの効率にも影響を与えます。努力の成果が十分報われる可能性があります。

次の 2 つの指針は、テーブル・エレメントの参照方法の選択に影響を与えます。

- 添え字付けではなく指標付けを使用する。

  コンパイラーは重複する指標や添え字を除去できますが、たとえ添え字が BINARY であっても、テーブル・エレメントへの元の参照は、添え字よりも指標を使用す

る方がより効率的です。これは、指標の値がエレメント・サイズを因子分解したものであるのに対して、添え字を使用する場合は、添え字の値にエレメント・サイズを乗算しなければならないためです。指標にはすでにテーブルの先頭からの変位が含まれており、実行時にこの値を計算する必要はありません。ただし、添え字の方が理解しやすく維持するのが簡単かもしれません。

- 相対指標付けを使用する。

  相対指標参照 (つまり、符号なし数値リテラルが指標名に加えられるか、または指標名から引かれる参照) は、直接指標参照と同じ位の速さで、時にはより高速で実行されます。オフセットを因子分解した代替索引を保管してもメリットはありません。

指標または添え字のいずれを使用する場合でも、以下の指針は、コーディングという面でよりよいパフォーマンスを得る助けとなります。

- 定数および重複する指標または添え字を左側に置く。

  定数および重複する指標や添え字を左側に置くことによって、実行時の計算を削減または除去することができます。すべての指標または添え字が変数である場合でも、テーブルを使用するときは、プログラムで互いに隣接して出てくる参照に関しては、右端の添え字が最も頻繁に変更されるように試みてください。こうすると、ページングだけでなくストレージ参照のパターンも向上します。指標または添え字がすべて重複していると、指標または添え字の計算全体が共通副次式となります。

- 関連するテーブルの長さと一致するようなエレメントの長さを指定する。

  異なるテーブルに添え字または指標を付けるときは、すべてのテーブルのエレメント長が同じ場合に、最も効率がよくなります。等しいエレメント長の場合、テーブルの最後の次元のストライドは等しくなります。すると、最適化プログラムは、1 つのテーブルで計算された右端の指標または添え字を再利用できるようになります。エレメントの長さおよび各次元での出現回数が同じである場合、最後の次元以外は、次元のストライドもまた等しくなり、その結果、添え字計算相互間の共通性はより大きくなります。最適化プログラムは、右端以外の添え字または指標を再使用することができます。

- 指標および添え字検査をプログラムにコーディングすることによって、参照エラーを回避する。

  指標および添え字を妥当性検査する必要がある場合は、SSRANGE コンパイラー・オプションを使用するよりも、COBOL プログラム内に独自の検査をコーディングする方が速い場合があります。

以下の指針で扱われている状況では、テーブルの効率をさらに向上させることができます。

- すべての添え字に 2 進数データ項目を使用する。

  添え字を使用してテーブルをアドレッシングする場合は、8 桁以下の BINARY 符号付きデータ項目を使用してください。さらに場合によっては、データ項目の桁数を 4 桁以下にすると、処理時間を短縮できます。

- 可変長テーブル項目に 2 進数データ項目を使用する。

可変長項目を持つテーブルの場合は、OCCURS DEPENDING ON (ODO) のコードを改善することができます。可変長項目が参照されるたびに不要な変換が起こらないようにするためには、BINARY を OCCURS . . . DEPENDING ON のオブジェクトに指定してください。

- 可能であれば固定長データ項目を使用する。

  可変長データ項目を使用する場合、それらの使用頻度が高くなる前に、固定長データ項目にコピーすると、オーバーヘッドを緩和することができます。

- 使用する探索メソッドのタイプに従ってテーブルを編成する。

  テーブルが順次に探索される場合には、探索基準を満たす可能性が最も高いデータ値をテーブルの始まりに置くようにします。テーブルが二分探索アルゴリズムを使用して探索される場合は、検索キー・フィールドに基づいてアルファベット順にソートされたテーブルに、データ値を入れてください。

関連概念
『テーブル参照の最適化』

関連タスク
68 ページの『テーブルでの項目の参照』
594 ページの『効率的なデータ型の選択』

関連参照
348 ページの『SSRANGE』

テーブル参照の最適化

ELEMENT(S1 S2 S3) というテーブル・エレメント参照で、S1、S2、および S3 が添え字であれば、コンパイラーは次のように式を計算します。

```
comp_s1 * d1 + comp_s2 * d2 + comp_s3 * d3 + base_address
```

ここで、comp_s1 は、2 進数に変換した後の S1 の値、comp-s2 は、2 進数に変換した後の S2 の値、といった具合です。それぞれの次元のストライドは d1、d2、および d3 です。ある次元のストライドとは、その次元でのオカレンス番号が 1 だけ異なり、その他のオカレンス番号が等しいという、テーブル・エレメント間の距離 (バイト単位) のことです。たとえば、上記の例の 2 次元のストライド d2 は、ELEMENT(S1 1 S3) と ELEMENT(S1 2 S3) との間の距離 (バイト単位) です。

指標計算は添え字計算に類似していますが、指標値ではそれらの中にストライドを因数分解しているので、乗算をする必要がないという点が異なります。指標計算には、指標をレジスターにロードすることも含まれます。これらのデータ転送は、個々の添え字計算の項を最適化する場合と、ほぼ同様に最適化することができます。

コンパイラーは式を左から右へと計算していくので、定数または重複する添え字が左端にあると、最適化プログラムが計算を除去する可能性が最も高くなります。

定数および変数項目の最適化

C1、C2、. . . は、定数データ項目であり、V1、V2、. . . は変数データ項目です。したがって、テーブル・エレメント参照 ELEMENT(V1 C1 C2) の場合、コンパイラーは個々の項 comp_c1 * d2 および comp_c2 * d3 だけしか、式から定数として除去できません。

```
comp_v1 * d1 + comp_c1 * d2 + comp_c2 * d3 + base_address
```

しかし、テーブル・エレメント参照 ELEMENT(C1 C2 V1) の場合、コンパイラーは、副次式 comp_c1 * d1 + comp_c2 * d2 全体を、式から定数として除去することができます。

```
comp_c1 * d1 + comp_c2 * d2 + comp_v1 * d3 + base_address
```

テーブル・エレメント参照 ELEMENT(C1 C2 C3) では、添え字はすべて定数なので、実行時に添え字計算は行われません。次の式の場合、

```
comp_c1 * d1 + comp_c2 * d2 + comp_c3 * d3 + base_address
```

最適化プログラムを使用すると、この参照は、スカラー (テーブルでない) 項目への参照と同じくらい効率的になります。

重複項目の最適化

テーブル・エレメント参照 ELEMENT(V1 V3 V4) および ELEMENT(V2 V3 V4) では、個々の項 comp_v3 * d2 および comp_v4 * d3 だけが、テーブル・エレメントの参照に必要な式における共通副次式です。

```
comp_v1 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address
comp_v2 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address
```

しかし、2 つのテーブル・エレメント参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V4) の場合は、副次式 comp_v1 * d1 + comp_v2 * d2 全体が、テーブル・エレメントの参照に必要な 2 つの式間で共通となります。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
comp_v1 * d1 + comp_v2 * d2 + comp_v4 * d3 + base_address
```

2 つの参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V3) では、式は同じです。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
```

最適化プログラムを使用すると、同じエレメントへの 2 度目 (およびそれ以降)の参照は、スカラー (テーブルでない) 項目への参照と同じ効率になります。

可変長項目の最適化

従属 OCCURS DEPENDING ON データ項目の入っているグループ項目は可変長です。プログラムは、可変長データ項目が参照されるたびに特殊コードを実行しなければなりません。

このコードはライン外のものなので、最適化の妨げになる可能性があります。さらに、可変長データ項目を処理するためのコードは、固定サイズ・データ項目を処理するコードよりかなり効率が下がり、処理時間も大幅に増加することがあります。たとえば、可変長データ項目を比較したり移動したりするためのコードには、ライブラリー・ルーチンの呼び出しが必要なため、固定長データ項目の場合の同じコードよりかなり低速になります。

直接指標付けと相対指標付けの比較

相対指標参照は、直接指標参照と同じくらい迅速に実行されます。

ELEMENT (I5, J3, K2) の直接指標付けには、次のプリプロセスが必要になります。

```
SET I5 TO I
SET I5 UP BY 5
SET J3 TO J
SET J3 DOWN BY 3
SET K2 TO K
SET K2 UP BY 2
```

この処理のため、直接指標付けは、ELEMENT (I + 5, J - 3, K + 2) の相対指標付けよりも効率が悪くなります。

関連概念
『最適化』

関連タスク
595 ページの『テーブルの効率的な処理』

コードの最適化

プログラムが最終テストの段階にあるときは、OPTIMIZE を指定して、テストされたコードと実動コードが同一になるようにします。

プログラムが再コンパイルされずに頻繁に使用される場合は、開発中にこのコンパイラー・オプションを使用することもできます。しかし、アセンブラー言語の拡張部分 (LIST コンパイラー・オプション) を使用してプログラムの微調整を行う場合を除き、再コンパイルを頻繁に行うと、OPTIMIZE のオーバーヘッドが利点を上回ることがあります。

プログラムのユニット・テストについては、最適化されていないコードをデバッグする方が簡単です。

最適化プログラムがプログラム上でどのように機能するかを見るためには、OPTIMIZE オプションを指定した場合としない場合でのコンパイルを行い、そのうえで生成されたコードを比較します。(アセンブラー言語で生成コードをリストするよう要求するには、LIST コンパイラー・オプションを使用します。)

関連概念
『最適化』

関連参照
333 ページの『LIST』
339 ページの『OPTIMIZE』

最適化

生成されるコードの効率を向上させるためには、OPTIMIZE コンパイラー・オプションを使用して、COBOL 最適化プログラムが以下のことを行うようにすることができます。

- 不必要な制御転送および非効率的な分岐を除去します。ソース・プログラムを見るだけではわからない、コンパイラーが生成する分岐も含みます。
- 含まれている (ネストされた) プログラムに対する PERFORM ステートメントおよび CALL ステートメントの両方のコンパイル済みコードを単純化します。可能であれば、最適化プログラムはステートメントをインラインに設定し、リンケー

ジ・コードがなくて済むようにします。この最適化は、*プロシージャー統合* と呼ばれます。プロシージャー統合を行えない場合、最適化プログラムは、できるだけ単純なリンケージ (2 つ程度の命令) を使用して、呼び出されるプログラムとの間を往復します。

- プログラムの結果に何の影響も与えない重複計算 (添え字計算や繰り返しのステートメントなど) を除去します。
- プログラムのコンパイル時に定数計算を実行することによって、定数計算を除去します。
- 定数条件式を除去します。
- 連続した項目 (MOVE CORRESPONDING の使用によって頻繁に発生するような) の移動を集約して、単一の移動にします。移動を集約するには、移動元も移動先も連続していなければなりません。
- 実行できないコードをプログラムから削除して、警告メッセージで識別します (到達不能コードの排除)。
- 参照されないデータ項目を DATA DIVISION から廃棄し、これらのデータ項目をその VALUE 文節に初期化するコードの生成を抑止します。(最適化プログラムがこのアクションを取るのは、FULL サブオプションが指定された場合だけです。)

含まれているプログラムのプロシージャー統合

含まれているプログラムのプロシージャー統合では、含まれているプログラムのコードが、含まれているプログラムへの CALL に置き換わります。結果として生じるプログラムは、CALL リンケージのオーバーヘッドもなく、より線形の制御フローとなり、より速く実行されます。

プログラム・サイズ: 含まれているプログラムを複数の CALL ステートメントが呼び出している場合、それぞれのプログラムがこのような各ステートメントと置き換わるのであれば、含まれているプログラムが相当大きくなる可能性があります。最適化プログラムはこの増加を 50 % 以内に制限し、それ以降はプログラムを統合しません。これにより最適化プログラムはその CALL ステートメントに、次善の最適化を選択します。すなわち、リンケージのオーバーヘッドを命令 2 つほどにする方法です。

到達不能コード: この統合の結果として、含まれている 1 つのプログラムが何回も繰り返されることがあります。その後の最適化はプログラムの各コピーで進められるため、コードのコピー先のコンテキストによっては、到達不能な部分が検出されることがあります。

PERFORM プロシージャー統合

PERFORM プロシージャー統合とは、PERFORM ステートメントがその実行済みプロシージャーによって置き換えられるプロセスのことです。この利点は、結果として生じるプログラムが、PERFORM リンケージのオーバーヘッドがなく、より線形になった制御フローで、より速く実行される点です。

プログラム・サイズ: 実行されたプロシージャーが複数のステートメントによって呼び出され、それぞれが PERFORM ステートメントを置き換えると、プログラムが大きくなる可能性があります。最適化プログラムはこの増加を 50 % 以内に制限し、それ以降はプロシージャーを統合しません。プログラム・サイズが問題である場合

は、セクション名の優先順位番号を使用することによって、特定のインスタンスのプロシージャー統合を避けることができます。

ある PERFORM ステートメントを、それによって実行されるプロシージャーで置換したくない場合は、PERFORM ステートメントをあるセクションに置き、実行済みプロシージャーを、異なる優先順位番号を持つ別のセクションに置きます。これにより最適化プログラムはその PERFORM ステートメントに、次善の最適化を選択します。すなわち、リンケージのオーバーヘッドを命令 2 つほどにする方法です。

到達不能コード: プロシージャー統合の結果、1 つの PERFORM プロシージャーが何回も繰り返されることがあります。その後の最適化はプロシージャーの各コピーで進められるため、コードのコピー先のコンテキストによっては、到達不能な部分が検出されることがあります。

『例: PERFORM プロシージャー統合』

関連概念

597 ページの『テーブル参照の最適化』

関連参照

339 ページの『OPTIMIZE』

例: PERFORM プロシージャー統合

次のプログラムのすべての PERFORM ステートメントは、プロシージャー統合によって変換されます。

```
1   SECTION 5.
11. PERFORM 12
    STOP RUN.
12. PERFORM 21
    PERFORM 21.
2   SECTION 5.
21. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
```

このプログラムは、最初から次のように作成されていたかのようにコンパイルされます。

```
1   SECTION 5.
11.
12. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
    IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
    STOP RUN.
```

それに対して、次のプログラムでは、最初の PERFORM ステートメントである PERFORM 12 だけが、プロシージャー統合によって最適化されます。

```
1   SECTION.
11. PERFORM 12
    STOP RUN.
12. PERFORM 21
```

```
    PERFORM 21.
2   SECTION 5.
21. IF A < 5 THEN
      ADD 1 TO A
      DISPLAY A
    END-IF.
```

関連概念
597 ページの『テーブル参照の最適化』

関連タスク
599 ページの『コードの最適化』
591 ページの『第 32 章 プログラムの調整』

パフォーマンスを向上させるためのコンパイラー機能の選択

パフォーマンス関連コンパイラー・オプションの選択および USE FOR DEBUGGING ON ALL PROCEDURES ステートメントの使用は、プログラムが最適化される方法に影響を与える可能性があります。

カスタマイズ済みシステムには、最適なパフォーマンスを得るために特定のオプションが必要なものもあります。以下の手順を実行します。

1. システム・デフォルトの内容を調べるには、プログラムの短縮リストを入手し、リストされたオプション設定値を検討する。
2. システム・プログラマーと一緒に、どのオプションがインストール先のオーバーライド不能オプションとして固定されているかを調べる。
3. インストール先で固定されていないオプションについては、プログラムをコンパイルするためのパフォーマンス関連オプションを選択する。

   重要: COBOL プログラムの調整方法については、システム・プログラマーと相談してください。そのようにして、選択するオプションを、インストール・システムで開発中のプログラムに適合するものにします。

コンパイラー・オプションのほかに、考慮する必要のあるもう 1 つのコンパイラー機能として、USE FOR DEBUGGING ON ALL PROCEDURES ステートメントがあります。これは、コンパイラーの最適化プログラムに大きな影響を与える可能性があります。ON ALL PROCEDURES オプションは、プロシージャー名への転送ごとに、余分のコードを生成します。これはデバッグには大変便利ですが、プログラムは非常に大型になり、実質上最適化を抑制する可能性があります。

COBOL ではセグメント化言語を使用できますが、オーバーレイは実行されないので、ストレージ割り振りを改善することはできません。

関連概念
599 ページの『最適化』

関連タスク
599 ページの『コードの最適化』
377 ページの『リストの入手』

関連参照
603 ページの『パフォーマンスに関連するコンパイラー・オプション』

パフォーマンスに関連するコンパイラー・オプション

次の表は、各オプションの目的の簡単な説明、パフォーマンス上の長所と短所、および該当する場合は使用上の注意を示しています。

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の短所	使用上の注意
316 ページの『AWO』	バッファーおよび装置スペースの最適使用を入手する	このオプションを使用すると、パフォーマンス上の節約になります。これは、入出力を処理するためにデータ管理サービスを呼び出す回数が減るからです。	一般には、ありません。	AWO を使用すると、APPLY WRITE-ONLY 文節は、V モード・レコードを持つ物理順次のプログラム内において、すべてのファイルに適用されます。
DATA(31) (321 ページの『DATA』を参照)	DFSMS に、16MB 境界より上の QSAM バッファーを割り振らせる (RENT および DATA(31) コンパイラー・オプションを使用して)	拡張形式 QSAM データ・セットは大量のバッファーを必要とする可能性があるので、制限のないストレージでバッファーを割り振ると、仮想記憶域の制約問題を回避することができます。	一般には、ありません。	DFSMS が備わった z/OS システムでは、アプリケーションがストライプ拡張形式 QSAM データ・セットを使用する場合には、RENT および DATA(31) コンパイラー・オプションを使用して、16MB 境界より上のストレージから、QSAM ファイル用の入出力バッファーを割り振らせるようにしてください。
326 ページの『DYNAM』	サブプログラム (CALL ステートメントによって呼び出された) が実行時に動的にロードされるようにする	サブプログラムが変更されても、アプリケーションをリンク・エディットする必要がないので、サブプログラムの保守が容易になります。	呼び出しは言語環境プログラム・ルーチンを介して行う必要があるため、パフォーマンスがわずかに低下します。	不要になった仮想記憶域を解放するには、CANCEL ステートメントを出してください。
327 ページの『FASTSRT』	IBM DFSORT プロダクト (または同等の製品) がすべての入出力を処理することを指定する	各レコードの処理後に Enterprise COBOL に戻るというオーバーヘッドを排除します。	なし。	直接作業ファイルをソート作業ファイルとして使用する場合は、FASTSRT の使用をお勧めします。すべてのソートがこのオプションに適格とは限りません。

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の短所	使用上の注意
NUMPROC(PFD) (336 ページの『NUMPROC』を参照)	数値演算の無効な符号処理をバイパスさせる	数値比較に関してはかなり効率のよいコードを生成します。	COMP-3 および DISPLAY 数値データ項目へのほとんどの参照では、NUMPROC(PFD) を使用すると、符号を「調整」するための余分のコードの生成が禁止されます。この余分のコードは、他のタイプの最適化も妨げる可能性があります。NUMPROC(MIG) および NUMPROC(NOPFD) を使用した場合は、余分のコードが生成されます。	NUMPROC(PFD) を使用すると、コンパイラーは、データが正しい符号を持っていると見なし、符号「調整」プロセスをバイパスします。すべての外部データ・ファイルに COMP-3 または DISPLAY 符号付き数値データの適切な符号が入っているとは限らないので、NUMPROC(PFD) が不適切なプログラムもあります。パフォーマンスを重視するアプリケーションについては、NUMPROC(PFD) の使用をお勧めします。
339 ページの『OPTIMIZE』	パフォーマンスがよくなるように、生成されるコードを最適化する	一般には、ランタイム・コードの効率がよくなります。	コンパイル時間が長くなること。OPTIMIZE は、NOOPTIMIZE に比べ、コンパイルにかかる処理時間が長くなります。	NOOPTIMIZE は通常、頻繁なコンパイルが必要となるプログラム開発の段階で使用され、デバッグも容易にします。実稼働用には、OPTIMIZE の使用をお勧めします。
343 ページの『RENT』	再入可能プログラムを生成する	プログラムを共用ストレージ (LPA/ELPA) に入れ、実行をより高速化することができます。	プログラムを再入可能にするために追加のコードが生成されます。	
RMODE(ANY) (344 ページの『RMODE』を参照)	プログラムをどこにでもロードできるようにする	RMODE(ANY) と NORENT を使用すると、プログラムとその WORKING-STORAGE を 16MB 境界より上に置き、16MB 境界より下のストレージを解放することができます。	一般には、ありません。	
NOSSRANGE (348 ページの『SSRANGE』を参照)	すべてのテーブル参照および参照変更式が適切な範囲内にあるかどうかを検査するためのコードを除去する	SSRANGE は、テーブル参照を検査するための追加コードを生成します。NOSSRANGE を使用すると、コードは生成されません。	なし。	一般に、テーブル参照のたびに検査する必要はなく、数度の検査だけで済む場合には、独自の検査をコーディングする方が、SSRANGE を使用するより速くなります。SSRANGE は、CHECK(OFF) ランタイム・オプションを使用すれば、実行時にオフにすることができます。パフォーマンスを重視するアプリケーションについては、NOSSRANGE の使用をお勧めします。

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の短所	使用上の注意
TEST(NONE) または NOTEST (349 ページの『TEST』を参照)	デバッグ・ツールをフルに活用するために作成される追加のオブジェクト・コードを回避するには、TEST(NONE) または NOTEST を使用する。また、TEST(NONE) を使用する場合は、TEST オプションの SEPARATE サブオプションを使用して、オブジェクト・コードのサイズをさらに縮小することができる。	TEST コンパイラー・オプションに、NONE 以外のフック位置サブオプション (すなわち、ALL、STMT、PATH、BLOCK) を指定して実行すると、追加のコードが生成されるので、実動環境で使用すると、パフォーマンスが大きく低下する可能性があります。コンパイルされたフックを指定すればするほど、より多くの追加コードが生成され、パフォーマンス低下も大きくなります。	なし。	TEST をフック位置サブオプションを指定せずに、または NONE 以外のサブオプションを指定して使用すると、NOOPTIMIZE コンパイラー・オプションが強制されます。実稼働の場合は、NOTEST または TEST(NONE,SYM) を使用することをお勧めします。その場合、SEPARATE サブオプションは付けても付けなくても構いません。この結果、コンパイルされたフックではなく、オーバーレイ・フックとなります。 実稼働時、プログラムが異常終了したときに変数のシンボリック・ダンプを定様式ダンプで入手したい場合には、TEST(NONE,SYM) でコンパイルします。その場合、SEPARATE サブオプションは付けても付けなくても構いません。
TRUNC(OPT) (353 ページの『TRUNC』を参照)	算術演算の受け取りフィールドを切り捨てるためのコードの生成を回避する	余分のコードを生成しないので、一般にパフォーマンスは向上します。	TRUNC(BIN) と TRUNC(STD) はともに、BINARY データ項目が変更されるたびに、余分のコードを生成します。TRUNC(BIN) は、そのパフォーマンスについては COBOL (OS/390 および VM 版) V2 R2 で改善されたとはいえ、上記のオプション中では最も低速のオプションです。	TRUNC(STD) は COBOL 85 標準に従いますが、TRUNC(BIN) および TRUNC(OPT) は従いません。TRUNC(OPT) を使用すると、コンパイラーは、データが PICTUREおよび USAGE の仕様に従っていると見なします。可能な場合は、TRUNC(OPT) を使用することをお勧めします。

関連概念

599 ページの『最適化』
38 ページの『ストレージとそのアドレス可能度』

関連タスク

294 ページの『コンパイル・エラー・メッセージのリストの生成』
606 ページの『パフォーマンスの評価』
14 ページの『バッファーおよび装置スペースの最適化』

602 ページの『パフォーマンスを向上させるためのコンパイラー機能の選択』
217 ページの『FASTSRT を使ってのソートのパフォーマンスの向上』
156 ページの『ストライプ拡張形式 QSAM データ・セットの使用』
595 ページの『テーブルの効率的な処理』

関連参照

52 ページの『符号表現および処理』
157 ページの『QSAM ファイル用のバッファーの割り振り』

パフォーマンスの評価

このワークシートを使用すると、プログラムのパフォーマンスを評価するのに役立ちます。各質問に「はい」と答える場合は、おそらくパフォーマンスは向上しています。

パフォーマンスのトレードオフを比較検討する際には、各オプションの機能およびパフォーマンスの利点と欠点を十分に理解するようにしてください。パフォーマンスの向上よりも、機能を重視する場合も多くあります。

コンパイラー・オプション	考慮事項	はい?
AWO	可能な場合は AWO オプションを使用していますか。	
DATA	QSAM ストライプ・データ・セットを使用するときは、DATA(31) オプションを使用していますか。	
DYNAM	NODYNAM を使用していますか。パフォーマンスのトレードオフを比較検討してください。	
FASTSRT	直接作業ファイルをソート作業ファイルとして使用するとき、FASTSRT オプションを選択しましたか。	
NUMPROC	可能な場合、NUMPROC(PFD) を使用していますか。	
OPTIMIZE	実稼働に OPTIMIZE を使用していますか。 OPTIMIZE(FULL) を使用できますか。	
RENT	NORENT を使用していますか。パフォーマンスのトレードオフを比較検討してください。	
RMODE(ANY)	NORENT プログラムで RMODE(ANY) を使用していますか。パフォーマンスとストレージ使用とを比較検討してください。	
SSRANGE	NOSSRANGE を実稼働に使用していますか。	
TEST	NOTEST、TEST(NONE,SYM)、または TEST(NONE,SYM,SEPARATE) を実稼働に使用していますか。	
TRUNC	可能な場合、TRUNC(OPT) を使用していますか。	

関連タスク

602 ページの『パフォーマンスを向上させるためのコンパイラー機能の選択』

関連参照

603 ページの『パフォーマンスに関連するコンパイラー・オプション』

CICS、IMS、または VSAM での効率的な実行

CICS または IMS のもとで実行されるオンライン・プログラム、または VSAM を使用するプログラムに関しては、以下のヒントに従うことによって、パフォーマンスを向上させることができます。

CICS

アプリケーションが CICS のもとで実行される場合は、EXEC CICS LINK コマンドを COBOL CALL ステートメントに変換すると、トランザクションの応答時間が短縮されます。

IMS

アプリケーションが IMS のもとで実行される場合は、アプリケーション・プログラムおよびライブラリー・ルーチンをプリロードしておくと、ロードと探索のオーバーヘッドを削減できます。入出力活動も減少する可能性があります。

システムのパフォーマンスを向上させるためには、RENT コンパイラー・オプションを使用して、可能な限りアプリケーションとライブラリー・ルーチンをプリロードします。

また、ライブラリー・ルーチン保存 (LRR) 機能を使用すると、IMS/TM 領域でのパフォーマンスを向上することができます。

VSAM

VSAM ファイルを使用するときは、順次アクセスの場合はデータ・バッファーの数、ランダム・アクセスの場合は索引バッファーの数が増えます。また、アプリケーションに適した制御インターバル・サイズ (CISZ) を選択しなければなりません。CISZ を小さくすると、挿入のコストはかかりますが、ランダム処理の検索が速くなります。一方、CISZ を大きくすると、順次処理の効率が上がります。

パフォーマンスを向上させるためには、レコードを順次アクセスし、複数の代替索引の使用を可能な限り避けます。代替索引を使用する場合は、アクセス方式サービス・プログラムの方が、AIXBLD ランタイム・オプションよりも、代替索引をより効率的に作成します。

関連タスク

408 ページの『CICS のもとで実行する COBOL プログラムのコーディング』
425 ページの『第 22 章 IMS のもとでの COBOL プログラムの実行』
193 ページの『VSAM パフォーマンスの向上』
言語環境プログラム カスタマイズ

関連参照

ランタイム・オプションの指定 (*言語環境プログラム プログラミング・ガイド*)

第 33 章 コーディングの簡略化

この資料には、生産性を向上させるための手法が用意されています。COPY ステートメント、COBOL 組み込み関数、および言語環境プログラムの呼び出し可能サービスを使用することによって、反復コーディングや、多数の算術計算または他の複雑なタスクをコーディングする必要性を回避することができます。

プログラムに頻繁に使用されるコード・シーケンス (共通データ項目のブロック、入出力ルーチン、エラー・ルーチン、または COBOL プログラム全体) が含まれている場合は、それらのコード・シーケンスを 1 度作成し、それらを COBOL コピー・ライブラリーに入れてください。その後は、COPY ステートメントを使用してライブラリーからこれらのコード・シーケンスを取り出し、コンパイル時にプログラムに含めることができます。この結果、反復コーディングが除去されます。

COBOL は、ストリングおよび数値を扱うためのさまざまな機能を提供します。これらの機能がコーディングの単純化に役立ちます。

言語環境プログラムの日付・時刻呼び出し可能サービスは、日付をフルワード 2 進整数として、タイム・スタンプを長精度 (64 ビット) 浮動小数点値として保管します。これらの形式を使用することにより、算術計算を日時の値に基づいて単純かつ効率的に行うことができます。これらの計算を実行するために、アプリケーションの言語ライブラリーの外側でサービスを使用する特別なサブルーチンを書く必要はありません。

関連概念
55 ページの『数字組み込み関数』
56 ページの『数学および日付の言語環境プログラム・サービス』

関連タスク
『反復コーディングの除去』
105 ページの『データ項目の変換 (組み込み関数)』
107 ページの『データ項目の評価 (組み込み関数)』
611 ページの『言語環境プログラム呼び出し可能サービスの使用』

反復コーディングの除去

COPY ステートメントを使用すると、保管されたソース・ステートメントをプログラムの任意の部分に組み込むことができます。ソース・ステートメントは、プログラムのどの部に入れてもよく、どのコード・シーケンス・レベルでもコーディングすることができます。COPY ステートメントは任意の深さにネストできます。

複数のコピー・ライブラリーを指定するには、複数システム定義または複数定義と IN/OF 句 (IN/OF *library-name*) の組み合わせのいずれかを使用してください。

z/OS バッチ
JCL を使用して、SYSLIB DD ステートメント上のデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントの IN/OF 句を使用します。

© Copyright IBM Corp. 1991, 2001

TSO ALLOCATE コマンドを使用して、SYSLIB のデータ・セットを連結します。あるいは、複数の ALLOCATE ステートメントを出し、COPY ステートメントの IN/OF 句を使用します。

UNIX SYSLIB 環境変数を使用して、コピーブックへの複数のパスを定義します。あるいは、複数の環境変数を使用し、COPY ステートメントの IN/OF 句を使用します。

たとえば、次のようになります。

```
COPY MEMBER1 OF COPYLIB
```

この修飾句を省略した場合、デフォルトは SYSLIB です。

COPY とデバッグ行: コピーされたテキストをデバッグ行として (たとえば、7 桁目に D が挿入されているかのように) 扱わせるには、COPY ステートメントの最初の行に D を入れてください。COPY ステートメント自体をデバッグ行にすることはできません。これに D が入っていても、WITH DEBUGGING モードが指定されていなければ、COPY ステートメントが処理されることはありません。

『例: COPY ステートメントの使用』

関連参照
360 ページの『コンパイラー指示ステートメント』

例: COPY ステートメントの使用

たとえば、ライブラリー記入項目 CFILEA が次の FD 記入項目から構成されているとします。

```
    BLOCK CONTAINS 20 RECORDS
    RECORD CONTAINS 120 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS FILE-OUT.
01  FILE-OUT       PIC X(120).
```

次のように、ソース・プログラム・コードの COPY ステートメントを使用すれば、メンバー CFILEA を取り出すことができます。

```
FD FILEA
         COPY CFILEA.
```

このライブラリー記入項目はプログラムにコピーされ、その結果生じるプログラム・リストは次のようになります。

```
FD FILEA
         COPY CFILEA.
C    BLOCK CONTAINS 20 RECORDS
C    RECORD CONTAINS 120 CHARACTERS
C    LABEL RECORDS ARE STANDARD
C    DATA RECORD IS FILE-OUT.
C    01  FILE-OUT   PIC X(120).
```

コンパイラー・ソース・リストで、COPY ステートメントは別個の行に印刷され、コピーされた行の前には C が付けられます。

DOWORK という名前のメンバーが次のステートメントで保管されたとします。

```
    COMPUTE QTY-ON-HAND = TOTAL-USED-NUMBER-ON-HAND
    MOVE QTY-ON-HAND to PRINT-AREA
```

保管されたメンバー DOWORK を検索するには、次のように書きます。

```
paragraph-name.
    COPY DOWORK.
```

DOWORK プロシージャーに入っているステートメントが、*paragraph-name* の後に置かれます。

EXIT コンパイラー・オプションを使用して LIBEXIT モジュールを指定すると、結果がこの章で示されるものと異なることがあります。

関連タスク

609 ページの『反復コーディングの除去』

関連参照

360 ページの『コンパイラー指示ステートメント』

言語環境プログラム呼び出し可能サービスの使用

言語環境プログラム呼び出し可能サービスは、さまざまなプログラミング・タスクを容易にしています。言語環境プログラム・サービスは、標準の COBOL CALL ステートメントを使用して呼び出した場合、以下のタスクに役立てることができます。

- 条件の処理

  言語環境プログラムの条件処理機能を使用すると、COBOL アプリケーションが予期しないエラーに反応できるようになります。言語構造体またはランタイム・オプションを使用すると、各条件を処理したいレベルを選択することができます。たとえば、COBOL プログラム内の特定のエラーを処理することを決定し、それを言語環境プログラムに処理させるか、またはオペレーティング・システムに処理させることができます。

  言語環境プログラムの条件処理のサポートでは、COBOL はプロシージャー・ポインター・データ項目を提供します。

- 動的ストレージの管理

  これらのサービスを使用すると、ストレージの獲得、解放、および再割り振りを行うことができます。それに加えて、独自のストレージ・プールを作成することもできます。

- 日時の計算

  日付・時刻サービスを使用すると、現在のローカル時刻および日付をいくつかの形式で入手し、日付・時刻の変換を実行することができます。2 つの呼び出し可能サービス CEEQCEN と CEESCEN は、2 桁の年号 (たとえば、1991 を表す 91、2002 を表す 02 など) を扱う予測可能な方法を提供します。

- 数学計算

  数学呼び出し可能サービスを使うと実行しやすい計算には、対数、指数、三角、平方根、および整数関数などがあります。

  COBOL では組み込み関数もサポートされます。この組み込み関数の中には、呼び出し可能サービスによって提供される数学関数および日付関数と同じ働きをする関数もいくつかあります。言語環境プログラム呼び出し可能サービスと組み込

み関数は、多少の例外はありますが、同じ結果を戻します。これらの相違点を理解したうえで、どちらを使用するかを決定してください。

- メッセージの処理

  メッセージ処理サービスには、メッセージの入手、ディスパッチング、およびフォーマット設定が含まれます。非 CICS アプリケーションのメッセージはファイルまたはプリンターに送ることができます。CICS メッセージは CICS 一時データ・キューに送られます。言語環境プログラムは、メッセージを分割して、宛先のレコード長を収容できるようにし、日本語や英語のような正しい各国語でメッセージを表示します。

- 各国語のサポート

  これらのサービスを使用すれば、アプリケーションは、アプリケーション・ユーザーに必要な言語を容易にサポートすることができます。言語と国を設定すると、デフォルトの日付、時刻、番号、および通貨形式を入手することができます。たとえば、日付を 23 June 00 または 6,23,00 と表示することができます。

- デバッグ・ツールを開始し、言語環境プログラムの定様式ダンプを獲得するような一般的なサービス

  デバッグ・ツールは、COBOL-CICS プログラムのバッチ式デバッグおよび対話式デバッグの両方をはじめ、COBOL アプリケーションに高度なデバッグ機能を提供します。デバッグ・ツールを使用すると、COBOL アプリケーションをホストからデバッグしたり、VisualAge COBOL バージョン 3.0 (またはそれ以降) と一緒にワークステーションからデバッグしたりすることができます。

  選択したオプションに応じて、言語環境プログラムの定様式ダンプには、変数の名前と値のほか、プログラム・トレースバック、制御ブロック、ストレージ、およびファイルに関する情報が入れられます。言語環境プログラム・ダンプはすべて、共通する、適切にラベル付けされた、読み取りやすい形式を持っています。

614 ページの『例: 言語環境プログラム呼び出し可能サービス』

関連概念

『言語環境プログラム呼び出し可能サービスのサンプル・リスト』
55 ページの『数字組み込み関数』
56 ページの『数学および日付の言語環境プログラム・サービス』

関連タスク

613 ページの『言語環境プログラム・サービスの呼び出し』
452 ページの『プロシージャー・ポインターと関数ポインターの使用』

言語環境プログラム呼び出し可能サービスのサンプル・リスト

以下に、言語環境プログラムで使用できる呼び出し可能サービスの例をいくつか示しています。リストされているサービスよりもっと多くのサービスを使用できます。

関数型	サービス	目的
条件処理サービス	CEEHDLR	ユーザー条件処理ルーチンを登録する
	CEESGL	条件を発生させる、または通知する
	CEEMRCR	条件処理ルーチンが完了した後でプログラムが実行を再開する場所を指示する

関数型	サービス	目的
動的ストレージ・サービス	CEEGTST	ストレージを獲得する
	CEECZST	これまでに割り振られたストレージ・ブロックのサイズを変更する
	CEEFRST	ストレージを解放する
日付および時刻	CEECBLDY	日付を表すストリングを COBOL 整数日付形式に変換する (日付を 1600 年 12 月 31 日以降の日数として表す)
	CEEQCEN、CEESCEN	言語環境プログラムの世紀ウィンドウ (年を表すのに 2 桁を使用するプログラムの場合に有用) を照会および設定する
	CEEGMTO	ローカル・システム時間とグリニッジ標準時の差を計算する
	CEELOCT	3 つの形式を選択して現在のローカル時間を入手する
数学サービス	CEESIABS	整数の絶対値を計算する
	CEESSNWN	単精度浮動小数点数に最も近い整数を計算する
	CEESSCOS	角の余弦を計算する
メッセージ処理サービス	CEEMOUT	メッセージをディスパッチする
	CEEMGET	メッセージを検索する
各国語サポート・サービス	CEE3LNG	現在の国語を変更または照会する
	CEE3CTY	現在の国を変更または照会する
	CEE3MCS	特定の国のデフォルトの通貨記号を入手する
一般サービス	CEE3DMP	言語環境プログラム定様式ダンプを入手する
	CEETEST	デバッグ・ツールを開始する

関連参照
言語環境プログラム プログラミング・リファレンス

言語環境プログラム・サービスの呼び出し

言語環境プログラム・サービスを呼び出すには、その特定のサービスを表す正しいパラメーターを指定した CALL ステートメントを使用します。

```
CALL "CEESSSQT" using argument, feedback-code, result
```

CALL ステートメントの変数は、呼び出す関数に必要な定義を指定して、DATA DIVISION に定義しなければなりません。

```
77  argument          comp-1.
77  feedback-code     pic  x(12)  display.
77  result            comp-1.
```

この例では、言語環境プログラム・サービス CEESSSQT は、変数 argument の平方根の値を計算し、この値を変数 result に入れて戻します。

フィードバック・コード・パラメーターを指定するかどうかを選択することができます。フィードバック・コードを指定する場合、feedback-code に戻される値は、サービスが正しく完了されたかどうかを示します。フィードバック・コードではなく OMITTED を指定したときに、サービスが失敗すると、言語環境プログラム条件が自動的に言語環境プログラム条件マネージャーに通知されます。このような条件

は、ユーザー作成条件処理ルーチンに組み込まれたリカバリー・ロジックによって処理するか、または処理されない条件については、デフォルトの言語環境プログラム処理を許可することができます。いずれの場合も、ロジックを作成して呼び出しを終えるたびにフィードバックをチェックする必要はありません。

呼び出し可能サービスの呼び出し時に、フィードバック・コードに OMITTED を指定すると、サービスが正常に完了した場合は、RETURN-CODE 特殊レジスターが 0 に設定されます。サービスが成功しなかった場合、このレジスターは変更されません。フィードバック・コードに OMITTED を指定しない場合は、サービスが成功したかどうかに関係なく、RETURN-CODE 特殊レジスターは常に 0 に設定されます。

『例: 言語環境プログラム呼び出し可能サービス』

関連概念
一般的な呼び出し可能サービス (*言語環境プログラム プログラミング・ガイド*)

関連参照
一般的な呼び出し可能サービス (*言語環境プログラム プログラミング・リファレンス*)
CALL ステートメント (*Enterprise COBOL 言語解説書*)

例: 言語環境プログラム呼び出し可能サービス

多くの呼び出し可能サービスは、旧バージョンの COBOL では大量のコーディングが必要であったような機能を、まったく新しい機能として提供します。この例で示しているサンプル COBOL プログラムは、言語環境プログラム・サービスの CEEDAYS および CEEDATE を使用して、COBOL ACCEPT ステートメントの結果として生じる日付をフォーマットし、表示します。CEEDAYS および CEEDATE を使用すると、言語環境プログラムがないと必要になるようなコーディングが少なくて済みます。

```
 ID DIVISION.
 PROGRAM-ID. HOHOHO.
*************************************************************
* FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *
*            WWWWWWWWW, MMMMMMMM DD, YYYY                  *
*                                                          *
*            For example:  MONDAY, MARCH 13, 2000          *
*                                                          *
*************************************************************
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   CHRDATE.
     05 CHRDATE-LENGTH     PIC S9(4) COMP VALUE 10.
     05 CHRDATE-STRING     PIC X(10).
 01   PICSTR.
     05 PICSTR-LENGTH      PIC  S9(4) COMP.
     05 PICSTR-STRING      PIC  X(80).
*
 77   LILIAN PIC           S9(9) COMP.
 77   FORMATTED-DATE       PIC X(80).
*
 PROCEDURE DIVISION.
*****************************************************************
*    USE  LE DATE/TIME CALLABLE SERVICES TO PRINT OUT          *
*    TODAY'S DATE FROM COBOL ACCEPT STATEMENT.                 *
*****************************************************************
```

```
     ACCEPT CHRDATE-STRING FROM DATE.
*
     MOVE "YYMMDD" TO PICSTR-STRING.
     MOVE 6 TO PICSTR-LENGTH.
     CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , OMITTED.
*
     MOVE " WWWWWWWWWZ, MMMMMMMMMZ DD, YYYY " TO PICSTR-STRING.
     MOVE 50 TO PICSTR-LENGTH.
     CALL "CEEDATE" USING LILIAN , PICSTR , FORMATTED-DATE ,
           OMITTED.
*
     DISPLAY "******************************".
     DISPLAY FORMATTED-DATE.
     DISPLAY "******************************".
*
     STOP RUN.
```

第 8 部 付録

© Copyright IBM Corp. 1991, 2001

付録 A. 中間結果と算術精度

コンパイラーは、算術ステートメントを、演算子優先順位に従って実行される一連の演算として処理し、中間フィールドにそれらの演算の結果が入るように設定します。コンパイラーはいくつかのアルゴリズムを使用して、確保する整数および小数部の桁数を判別します。

中間結果は、次の場合に得ることができます。

- 動詞の直後に複数のオペランドが入った、ADD または SUBTRACT ステートメント。
- 一連の算術演算または複数の結果フィールドを指定する COMPUTE ステートメント。
- 条件ステートメントまたは参照変更指定に含まれている算術式。
- GIVING オプションおよび複数の結果フィールドを使用する、ADD、SUBTRACT、MULTIPLY、または DIVIDE ステートメント。
- 組み込み関数をオペランドとして使用するステートメント。

621 ページの『例: 中間結果の計算』

中間結果の精度は、コンパイルする際にデフォルト・オプション ARITH(COMPAT) (*互換モード* と呼ばれる) を使用するか、ARITH(EXTEND) (*拡張モード* と呼ばれる) を使用するかによって、次のように異なります。

互換モードでの算術演算の計算は、COBOL (OS/390 および VM 版) バージョン 2 リリース 2 より前の IBM COBOL のリリースの場合と変わりません。

- 固定小数点の中間結果の場合は、最大 30 桁が使用されます。
- 浮動小数点組み込み関数は、長精度 (64 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが長精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように計算されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、長精度浮動小数点に変換されてから処理されます。

拡張モードでの算術演算の計算には、次のような特性があります。

- 固定小数点の中間結果の場合は、最大 31 桁が使用されます。
- 浮動小数点組み込み関数は、拡張精度 (128 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが拡張精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように計算されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、拡張精度浮動小数点に変換されてから処理されます。

© Copyright IBM Corp. 1991, 2001

関連概念

46 ページの『数値データの形式』
60 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照

621 ページの『固定小数点データと中間結果』
627 ページの『浮動小数点データと中間結果』
628 ページの『非算術ステートメントでの算術式』
316 ページの『ARITH』

中間結果として使用される用語

コンパイラーが中間結果を入れるために確保する整数および小数部の桁数の説明では、以下の用語が使用されます。

i 中間結果において繰り上げられた整数の桁数。(ROUNDED オプションを使用している場合は、正確さを得るために必要なら、整数がもう 1 桁繰り上げられます。)

d 中間結果において繰り上げられた小数部の桁数。(ROUNDED オプションを使用している場合は、正確さを得るために必要なら、小数部がもう 1 桁繰り上げられます。)

dmax 特定のステートメントで、次のうちの最も大きいもの。

- 最終結果フィールド (1 つまたは複数) に必要な小数部の桁数。
- 除数または指数を除き、オペランドに対して定義された小数部の最大桁数。
- 関数オペランドの *outer-dmax*。

inner-dmax

関数への参照で、次のうちの最も大きいもの。

- その基本引き数に対して定義された小数部の桁数。
- いずれかの算術式の引き数の *dmax*。
- そのいずれかの組み込み関数の *outer-dmax*。

outer-dmax

関数結果がそれ自体の計算の外で演算に寄与する小数部の桁数 (たとえば、その関数が算術式中のオペランドであるか、または別の関数への引き数である場合)。

op1 生成される算術ステートメントの第 1 オペランド (除算では除数)。

op2 生成される算術ステートメントの第 2 オペランド (除算では被除数)。

i1、i2 それぞれ *op1* と *op2* 内の整数の数値。

d1、d2

それぞれ、*op1* および *op2* 内の小数部の桁数。

ir 生成された算術ステートメントまたは演算が実行されたときの中間結果。(中間結果は、レジスターまたは保管場所のいずれかで生成されます。)

ir1*、*ir2

連続する中間結果。(連続する中間結果は同じ保管場所にすることができます。)

例: 中間結果の計算

次の例は、コンパイラーが、必要に応じて中間結果を保管しながら、算術ステートメントを一連の演算として実行する方法を示しています。

```
COMPUTE Y = A + B * C - D / E + F ** G
```

結果は、以下の順序で計算されます。

1.	Exponentiate F	by G	yielding *ir1*.
2.	Multiply B	by C	yielding *ir2*.
3.	Divide E	into D	yielding *ir3*.
4.	Add A	to ir2	yielding *ir4*.
5.	Subtract *ir3*	from *ir4*	yielding *ir5*.
6.	Add *ir5*	to *ir1*	yielding Y.

関連概念

55 ページの『算術式』

関連参照

620 ページの『中間結果として使用される用語』

固定小数点データと中間結果

コンパイラーは、中間結果の整数および小数部の桁数を、以下のセクションで説明しているようにして決定します。

加算、減算、乗算、および除算

次の表は、加算、減算、乗算、または除算の結果として理論上可能な精度を示しています。

演算	整数桁	小数桁
+ または -	(*i1* または *i2*) + 1、どちらか大きい方	*d1* または *d2*、どちらか大きい方
*	*i1* + *i2*	*d1* + *d2*
/	*i2* + *d1*	(*d2* - *d1*) または *dmax*、どちらか大きい方

算術ステートメントのオペランドを十分な小数桁で定義して、最終結果の正確度が希望どおりになるようにしなければなりません。

次の表は、加算、減算、乗算、または除算に関連する算術演算の固定小数点中間結果において、コンパイラーが繰り上げる桁数を示しています。これは、*互換モード*にある場合 (すなわち、デフォルト・コンパイラー・オプション ARITH(COMPAT) を使用してコンパイルする場合) です。

i + *d* の値	*d* の値	*i* + *dmax* の値	*ir* において繰り上げられる桁数
<30 =30	任意の値	任意の値	*i* 整数桁数および *d* 小数桁数
>30	<*dmax* =*dmax*	任意の値	30-*d* 整数桁数および *d* 小数桁数
	>*dmax*	<30 =30	*i* 整数桁数および 30-*i* 小数桁数
		>30	30-*dmax* 整数桁数および *dmax* 小数桁数

次の表は、加算、減算、乗算、または除算に関連する算術演算の固定小数点中間結果において、コンパイラーが繰り上げる桁数を示しています。これは、*拡張モード*にある場合 (すなわち、オプション ARITH(EXTEND) を使用してコンパイルする場合) です。

i + *d* の値	*d* の値	*i* + *dmax* の値	*ir* において繰り上げられる桁数
<31 =31	任意の値	任意の値	*i* 整数桁数および *d* 小数桁数
>31	<*dmax* =*dmax*	任意の値	31-*d* 整数桁数および *d* 小数桁数
	>*dmax*	<31 =31	*i* 整数桁数および 31-*i* 小数桁数
		>31	31-*dmax* 整数桁数および *dmax* 小数桁数

指数

べき乗計算は、式 *op1* ** *op2* で表されます。*op2* の特性に基づいて、コンパイラーは固定小数点数のべき乗計算を次の 3 つのいずれかの方法で処理します。

- *op2* が小数部で表されている場合は、浮動小数点命令が使用されます。
- *op2* が整数リテラルまたは定数である場合、値 *d* は次のように計算されます。

  *d = d1 * |op2|*

  また、値 *i* は、*op1* の特性に基づいて、次のように計算されます。
  - *op1* がデータ名または変数である場合は、次のように計算されます。

    *i = i1 * |op2|*
  - *op1* がリテラルまたは定数である場合、*i* は、*op1* ** |*op2*| の値の整数の桁数に等しく設定されます。

  互換モード (ARITH(COMPAT) を使用してコンパイルする場合) では、*i* と *d* を計算し終わると、コンパイラーは、次の表に示すアクションを実行して、べき乗計算の中間結果 *ir* を処理します。

i + *d* の値	その他の条件	取られるアクション
<30	任意。	*ir* において、*i* の整数桁数および *d* の小数桁数が繰り上げられる。

i + *d* の値	その他の条件	取られるアクション
=30	*op1* が奇数の桁数を持つ。	*ir* において、*i* の整数桁数および *d* の小数桁数が繰り上げられる。
	op1 が偶数の桁数を持つ。	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 30 桁の整数の場合は、*ir* において、*i* の整数桁数および *d* の小数桁数が繰り上げられる。
>30	任意。	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)。

拡張モード (ARITH(EXTEND) を使用してコンパイルする場合) では、*i* と *d* を計算し終わると、コンパイラーは、次の表に示すアクションを取り、べき乗計算の中間結果 *ir* を処理します。

i + *d* の値	その他の条件	取られるアクション
<31	任意。	*ir* において、*i* の整数桁数および *d* の小数桁数が繰り上げられる。
=31 >31	任意。	*op2* が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 31 桁の整数の場合は、*ir* において、*i* の整数桁数および *d* の小数桁数が繰り上げられる。

op2 が負である場合は、1 という値が予備計算によって作成された結果で除算されます。使用される *i* と *d* の値は、上記に示された固定小数点データの除算規則に従って計算されます。

- *op2* が整数データ名または変数である場合は、*dmax* の小数桁数と、30-*dmax* (互換モード) または 31-*dmax* (拡張モード) 整数桁数が使用されます。*op1* はそれ自体によって (|*op2*| - 1) 回、乗算されます (*op2* がゼロ以外の場合)。

  op2 が 0 であると、結果は 1 になります。0 による除算とべき乗計算の SIZE ERROR 条件が適用されます。

10 桁以上の有効数字を持つ固定小数点の指数部は、常に 9 桁に切り捨てられます。指数がリテラルまたは定数である場合、E レベルのコンパイラー診断メッセージが発行されます。それ以外の場合は、実行時に通知メッセージが発行されます。

624 ページの『例: 固定小数点算術演算でのべき乗計算』

関連参照

620 ページの『中間結果として使用される用語』
624 ページの『中間結果の切り捨て』
624 ページの『2 進数データと中間結果』
627 ページの『浮動小数点データと中間結果』
625 ページの『固定小数点算術演算で評価される組み込み関数』
316 ページの『ARITH』
SIZE ERROR 句 (*Enterprise COBOL 言語解説書*)

例: 固定小数点算術演算でのべき乗計算

次の例は、コンパイラーが、必要に応じて中間結果を保管しながら、ゼロ以外の整数累乗へのべき乗計算を一連の乗算として実行する方法を示しています。

```
COMPUTE Y = A ** B
```

B が 4 であれば、結果は次のように計算されます。使用される *i* と *d* の値は、固定小数点データおよび中間結果に関する乗算規則に従って計算されます (以下を参照してください)。

1. Multiply A by A yielding *ir1*.
2. Multiply *ir1* by A yielding *ir2*.
3. Multiply *ir2* by A yielding *ir3*.
4. Move *ir3* to *ir4*.

ir4 は *dmax* の小数桁数を持っています。

B は正であるので、*ir4* は Y に移動されます。しかし B が -4 であった場合は、次のような追加ステップが実行されます。

5. Divide *ir4* into 1 yielding *ir5*.

ir5 は *dmax* の小数桁数を持っており、Y に移動されます。

関連参照
620 ページの『中間結果として使用される用語』
621 ページの『固定小数点データと中間結果』

中間結果の切り捨て

中間結果において桁数が互換モードの 30、または拡張モードの 31 を超えるときは常に、コンパイラーは、以下に参照されている表で示されているように、30 (互換モード) または 31 (拡張モード) 桁までに切り捨て、警告を表示します。切り捨てが実行時に起こった場合は、メッセージが出され、プログラムは実行を続けます。

固定小数点計算で発生する可能性のある中間結果の切り捨てを回避したい場合には、代わりに浮動小数点オペランド (COMP-1 または COMP-2) を使用してください。

関連概念
46 ページの『数値データの形式』

関連参照
621 ページの『固定小数点データと中間結果』
316 ページの『ARITH』

2 進数データと中間結果

2 進数オペランドを含む演算で、18 桁を超える中間結果が必要な場合は、コンパイラーは演算を実行する前に、オペランドを内部 10 進数に変換します。結果フィールドが 2 進数である場合、コンパイラーは、演算の結果を内部 10 進数から 2 進数に変換します。

2 進数オペランドが最も効率的に使用されるのは、中間結果が 9 桁を超えない場合です。

関連参照
621 ページの『固定小数点データと中間結果』
316 ページの『ARITH』

固定小数点算術演算で評価される組み込み関数

コンパイラーは、組み込み関数の *inner-dmax* および *outer-dmax* 値をその関数の特性から決定します。

整数関数

整数組み込み関数は整数を戻します。したがって、この関数の *outer-dmax* は常に 0 です。引き数がすべて整数でなければならない整数関数の場合は、*inner-dmax* も常に 0 になります。

次の表に、*inner-dmax* および関数結果の精度を要約しています。

機能	*Inner-dmax*	関数結果の桁精度
DATE-OF-INTEGER	0	8
DATE-TO-YYYYMMDD	0	8
DAY-OF-INTEGER	0	7
DAY-TO-YYYYDDD	0	7
FACTORIAL	0	30 (互換モード)、31 (拡張モード)
INTEGER-OF-DATE	0	7
INTEGER-OF-DAY	0	7
LENGTH	n/a	9
MOD	0	min(*i1 i2*)
ORD	n/a	3
ORD-MAX		9
ORD-MIN		9
YEAR-TO-YYYY	0	4
INTEGER		固定小数点引き数の場合: 引き数より 1 桁大きくなります。浮動小数点引き数の場合: 30 (互換モード)、31 (拡張モード)
INTEGER-PART		固定小数点引き数の場合: 引き数と同じ桁数になります。浮動小数点引き数の場合: 30 (互換モード)、31 (拡張モード)

混合関数

混合 組み込み関数とは、結果の型がその引き数の型に依存する関数です。引き数がすべて数値で、どの引き数も浮動小数点でない場合、その混合関数は固定小数点です。(混合関数のいずれかの引き数が浮動小数点である場合、その関数は浮動小数点命令によって評価され、浮動小数点結果を戻します。) 混合関数が固定小数点算術

演算で評価されたときは、引き数がすべて整数であれば結果は整数になり、それ以外の場合は結果は固定小数点になります。

混合関数 MAX、MIN、RANGE、REM、および SUM の場合、*outer-dmax* は常に *inner-dmax* に等しくなります (したがって、引き数がすべて整数であれば、両方とも 0 になります)。これらの関数について戻される結果の精度を判別するためには、固定小数点算術演算および中間結果に関する規則 (以下を参照) を、アルゴリズムの各ステップに適用してください。

MAX

1. 最初の引き数を関数結果に割り当てる。
2. 残りの引き数ごとに、以下を行う。
   a. 関数結果の代数値を引き数と比較する。
   b. 2 つの値のうちの大きな方を関数結果に割り当てる。

MIN

1. 最初の引き数を関数結果に割り当てる。
2. 残りの引き数ごとに、以下を行う。
   a. 関数結果の代数値を引き数と比較する。
   b. 2 つの値のうちの小さな方を関数結果に割り当てる。

RANGE

1. MAX 用のステップを使用して、最大の引き数を選択する。
2. MIN 用のステップを使用して、最小の引き数を選択する。
3. 最大の引き数から最小の引き数を引く。
4. その差を関数結果に割り当てる。

REM

1. 引き数 1 を引き数 2 で割る。
2. ステップ 1 の結果からすべての非整数桁を除去する。
3. ステップ 2 の結果に引き数 2 を掛ける。
4. ステップ 3 の結果を引き数 1 から引く。
5. その差を関数結果に割り当てる。

SUM

1. 値 0 を関数結果に割り当てる。
2. 引き数ごとに、以下を行う。
   a. その引き数を関数結果に足す。
   b. その和を関数結果に割り当てる。

関連参照

620 ページの『中間結果として使用される用語』
621 ページの『固定小数点データと中間結果』
627 ページの『浮動小数点データと中間結果』
316 ページの『ARITH』

浮動小数点データと中間結果

式について以下の条件が当てはまる場合は、浮動小数点命令を使用して算術式が計算されます。

- 受け取り側またはオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
- 指数に小数部の桁が含まれている。
- 指数が、べき乗計算または除算演算子を含む式であり、かつ *dmax* が 0 より大きい。
- 組み込み関数が浮動小数点関数である。

算術式の演算が浮動小数点で計算される場合は、すべてのオペランドが浮動小数点に変換され、しかも浮動小数点命令を使用して演算が行われたかのように、式全体が計算されます。

式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、互換モードにおいて、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- これ以外の場合には、長精度が使用されます。

長精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、長精度浮動小数点命令が使用されたかのように計算されます。

拡張モードでは、式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- 受け取り側およびオペランドがすべて COMP-1 または COMP-2 データ項目で、受け取り側またはオペランドの少なくとも 1 つが COMP-2 データ項目であり、かつ式に乗算またはべき乗演算が含まれていない場合には、長精度が使用されます。
- これ以外の場合には、拡張精度が使用されます。

拡張精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、拡張精度浮動小数点命令が使用されたかのように計算されます。

注意: 浮動小数点演算で、指数オーバーフローが発生した中間結果フィールドがあると、ジョブは異常終了します。

浮動小数点算術演算で評価されるべき乗計算

浮動小数点べき乗計算は常に、長精度浮動小数点算術演算を使用して計算されます。拡張モードでは、浮動小数点べき乗計算は常に、拡張精度浮動小数点算術演算を使用して計算されます。

COBOL では、負の数を小数で累乗した値は定義されていません。たとえば、(-2) ** 3 は -8 ですが、(-2) ** (3.000001) は定義されていません。べき乗計算が浮動小数点で行われ、結果が未定義である可能性がある場合には、実行時に指数の値が計

算され、それが実際に整数値を持つかどうか判別されます。整数値を持っていない場合には、診断メッセージが出されます。

浮動小数点算術演算で評価される組み込み関数

互換モードでは、浮動小数点組み込み関数は常に、長精度 (64 ビット) の浮動小数点値を戻します。拡張モードでは、浮動小数点組み込み関数は常に、拡張精度 (128 ビット) の浮動小数点値を戻します。

少なくとも 1 つの浮動小数点引き数を持つ混合関数は、浮動小数点算術演算を使用して計算されます。

関連参照

620 ページの『中間結果として使用される用語』
316 ページの『ARITH』

非算術ステートメントでの算術式

算術式は、算術ステートメント以外のコンテキスト内に現れても構いません。たとえば、IF または EVALUATE ステートメントを持つ算術式を使用することができます。このようなステートメントでは、固定小数点データを持つ中間結果および浮動小数点データを持つ中間結果に関する規則が適用されます。ただし、次のような変更があります。

- 省略された IF ステートメントは、省略されていないかのように処理されます。
- 被比較数の少なくとも 1 つが算術式であるような明示比較条件では、*dmax* は、いずれかの被比較数の任意のオペランド (除数と指数を除く) 用に定義された小数部の最大小数桁数になります。以下のいずれかの条件が真である場合は、浮動小数点算術演算の規則が適用されます。
  - いずれかの被比較数のオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
  - 指数に小数部の桁が含まれている。
  - 指数が、べき乗計算または除算演算子を含む式であり、かつ *dmax* が 0 より大きい。

  たとえば、次のようになります。

  ```
  IF operand-1 = expression-1 THEN . . .
  ```

  operand-1 が COMP-2 と定義されるデータ名である場合は、*expression-1* には、たとえ固定小数点オペランドしか含まれていなくても、浮動小数点算術演算の規則が適用されます。というのは、これは固定小数点オペランドと比較されるためです。
- ある算術式と別のデータ項目または算術式との間の比較で、関係演算子が使用されない場合 (すなわち、明示的な比較条件がない場合) は、その算術式は、被比較数の属性を考慮に入れずに計算されます。たとえば、次のようになります。

  ```
  EVALUATE expression-1
    WHEN expression-2 THRU expression-3
    WHEN expression-4
    . . .
  END-EVALUATE
  ```

上記のステートメントでは、それぞれの算術式は、その特性に応じて、固定小数点または浮動小数点算術で行われます。

関連概念

60 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照

620 ページの『中間結果として使用される用語』
621 ページの『固定小数点データと中間結果』
627 ページの『浮動小数点データと中間結果』
IF ステートメント (*Enterprise COBOL 言語解説書*)
EVALUATE ステートメント (*Enterprise COBOL 言語解説書*)
条件式 (*Enterprise COBOL 言語解説書*)

付録 B. 複合 OCCURS DEPENDING ON

複合 OCCURS DEPENDING ON (ODO) は、COBOL 85 標準への拡張部分としてサポートされます。

コンパイラーによって認められる複合 ODO の基本形式は、以下のとおりです。

- 可変位置項目またはグループ: DEPENDING ON オプションを指定した OCCURS 文節によって記述されたデータ項目の後に、非従属データ項目またはグループが続きます。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 文節によって記述されたデータ項目の後に、OCCURS 文節によって記述された非従属データ項目が続きます。
- 可変長エレメントを持つテーブル: OCCURS 文節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 文節によって記述された従属データ項目が含まれています。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

複合 ODO は、ディスク・スペースの節約には役立ちますが、使用するのが難しく、コードを理解困難なものにする可能性があります。

『例: 複合 ODO』

関連タスク

633 ページの『ODO オブジェクトの値を変更するときの指標エラーの回避』
634 ページの『可変テーブルにエレメントを追加するときのオーバーレイ回避』

関連参照

632 ページの『ODO オブジェクトの値における変更の影響』
OCCURS DEPENDING ON 文節 (*Enterprise COBOL 言語解説書*)

例: 複合 ODO

次の例は、複合 ODO の可能なタイプのオカレンスを示しています。

```
01  FIELD-A.
    02 COUNTER-1                          PIC S99.
    02 COUNTER-2                          PIC S99.
    02 TABLE-1.
       03 RECORD-1 OCCURS 1 TO 5 TIMES
                   DEPENDING ON COUNTER-1    PIC X(3).
    02 EMPLOYEE-NUMBER                    PIC X(5). (1)
    02 TABLE-2 OCCURS 5 TIMES                       (2)(3)
              INDEXED BY INDX.                     (4)
       03 TABLE-ITEM                      PIC 99.   (5)
       03 RECORD-2 OCCURS 1 TO 3 TIMES
                   DEPENDING ON COUNTER-2.
          04 DATA-NUM                     PIC S99.
```

© Copyright IBM Corp. 1991, 2001

定義: この例では、COUNTER-1 は *ODO オブジェクト* です。つまり、RECORD-1 の DEPENDING ON 文節のオブジェクトです。RECORD-1 は *ODO サブジェクト* であると言われます。同様に、COUNTER-2 は、対応する ODO サブジェクトである RECORD-2 の ODO オブジェクトです。

上の例に示されている複合 ODO オカレンスのタイプは、次のとおりです。

(1) 可変位置項目: EMPLOYEE-NUMBER は、同じレベル 01 レコード内の可変長テーブルに続くデータ項目です。可変長テーブルに従属するわけではありません。

(2) 可変位置テーブル: TABLE-2 は、同じレベル 01 レコード内の可変長テーブルに続くテーブルです。可変長テーブルに従属するわけではありません。

(3) 可変長エレメントを持つテーブル: TABLE-2 は、従属データ項目 RECORD-2 を含むテーブルです。オカレンス回数は、その ODO オブジェクトの内容によって変わります。

(4) 可変長エレメントを持つテーブルの指標名、INDX。

(5) 可変長エレメントを持つテーブルのエレメント、TABLE-ITEM。

長さの計算方法

各レコードの可変部分の長さは、その ODO オブジェクトとその ODO サブジェクトの長さとの積です。たとえば、上記に示された複合 ODO 項目の 1 つに参照が行われるたびに、使用される際の実際の長さは、次のように計算されます。

- TABLE-1 の長さは、COUNTER-1 の内容 (RECORD-1 のオカレンスの回数) に 3 (RECORD-1 の長さ) を掛けることによって計算されます。
- TABLE-2 の長さは、COUNTER-2 の内容 (RECORD-2 のオカレンスの回数) に 2 (RECORD-2 の長さ) を掛け、TABLE-ITEM の長さを加算することによって計算されます。
- FIELD-A の長さは、COUNTER-1、COUNTER-2、TABLE-1、EMPLOYEE-NUMBER、および TABLE-2 の長さに 5 を掛けたものを加算することによって計算されます。

ODO オブジェクトの値の設定

グループ内のすべての ODO オブジェクトを設定しなければ、そのグループ内のどの複合 ODO 項目も参照することはできません。たとえば、上記のコードの EMPLOYEE-NUMBER を参照するには、その前に、COUNTER-1 と COUNTER-2 を設定しておかなければなりません。ただし、EMPLOYEE-NUMBER は ODO オブジェクトに直接依存して値を得るわけではありません。

制約事項: ODO オブジェクトは可変的に配置することはできません。

ODO オブジェクトの値における変更の影響

DEPENDING ON オプションを指定した OCCURS 文節によって記述されるデータ項目の後に、同じグループ内で、非従属データ項目 (複合 ODO の形式の) が続く場合、ODO のオブジェクトの値に変更を加えると、レコード内の複合 ODO 項目へのその後の参照が影響を受けます。

- 関連 ODO 文節を含むグループのサイズは、ODO オブジェクトの新しい値を反映します。
- ODO サブジェクトを含むグループへの MOVE は、ODO オブジェクトの新しい値に基づいて行われます。
- ODO 文節を使用して記述された項目の後に非従属項目がある場合、それらの位置は ODO オブジェクトの新しい値の影響を受けます。(非従属項目の内容を保持するためには、ODO オブジェクトの値が変更される前に、非従属項目を作業域に移動しておき、後でそれらを戻してください。)

ODO オブジェクトの値は、データをその ODO オブジェクトに移動するか、その ODO オブジェクトが含まれているグループに移動すると、変更される可能性があります。また、ODO オブジェクトが READ ステートメントのターゲットであるレコードに含まれている場合にも、その値が変更されることがあります。

関連タスク
『ODO オブジェクトの値を変更するときの指標エラーの回避』
634 ページの『可変テーブルにエレメントを追加するときのオーバーレイ回避』

ODO オブジェクトの値を変更するときの指標エラーの回避

複合 ODO 指標名 (つまり、可変長エレメントを含むテーブルの指標名) を、テーブルの従属データ項目の ODO オブジェクトの値を変更した後で参照する場合は、十分に注意しなければなりません。ODO オブジェクトの値を変更すると、テーブルの長さが変わるため、関連する複合 ODO 指標のバイト・オフセットはもう有効ではありません。予防策を講じなければ、次のような指標名への参照をコーディングすると、結果は予測できません。

- テーブルのエレメントへの参照
- SET *integer-data-item* TO *index-name* 形式の SET ステートメント (形式 1)
- SET *index-name* UP|DOWN BY *integer* 形式の SET ステートメント (形式 2)

この種のエラーを回避するためには、次のようにしてください。

1. 指標項目を整数データ項目に保管する。(これを行うと、暗黙の変換が行われます。整数項目は、指標のオフセットに対応するテーブル・エレメント出現番号を受け取ります。)
2. ODO オブジェクトの値を変更する。
3. 直ちに整数データ項目から指標項目を復元する。(これを行うと、暗黙の変換が行われます。指標項目は、整数項目のテーブル・エレメント出現番号に対応するオフセットを受け取ります。オフセットは、その時点で有効なテーブルの長さに従って計算されます。)

次のコードは、ODO オブジェクト COUNTER-2 が変更される場合の指標名の保管方法と復元方法を示しています (631 ページの『例: 複合 ODO』を参照)。

```
 77  INTEGER-DATA-ITEM-1      PIC 99.
 . . .
     SET INDX TO 5.
*         INDX is valid at this point.
     SET INTEGER-DATA-ITEM-1 TO INDX.
*         INTEGER-DATA-ITEM-1 now has the
*         occurrence number corresponding to INDX.
     MOVE NEW-VALUE TO COUNTER-2.
```

```
*         INDX is not valid at this point.
     SET INDX TO INTEGER-DATA-ITEM-1.
*         INDX is now valid, containing the offset
*         corresponding to INTEGER-DATA-ITEM-1, and
*         can be used with the expected results.
```

関連参照

SET ステートメント (*Enterprise COBOL 言語解説書*)

可変テーブルにエレメントを追加するときのオーバーレイ回避

後ろに同じグループの 1 つまたは複数の非従属データ項目が続いている、可変出現テーブルの中のエレメントの数を増やす場合は、十分注意しなければなりません。ODO オブジェクトの値を増分し、テーブルにエレメントを追加すると、テーブルの後に続く可変位置データ項目を誤ってオーバーレイする可能性があります。

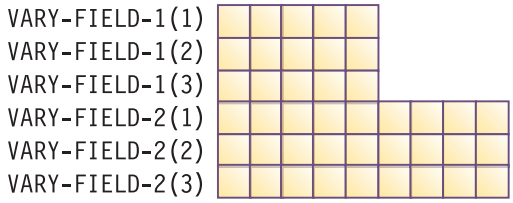
この種のエラーを回避するためには、次のようにしてください。

1. テーブルの後に続く可変位置データ項目を別のデータ域に保管する。
2. ODO オブジェクトの値を増分する。
3. データを新しいテーブル・エレメントに移動する (必要な場合)。
4. 可変位置データ項目を、それらを保管したデータ域から復元する。

次の例では、テーブル VARY-FIELD-1 にエレメントを追加しますが、このテーブルのエレメントの数は ODO オブジェクト CONTROL-1 に左右されます。VARY-FIELD-1 の後には、非従属可変位置データ項目である GROUP-ITEM-1 が続いており、このエレメントがオーバーレイされる可能性があります。

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-1                              PIC X(10).
    05  CONTROL-1                            PIC S99.
    05  CONTROL-2                            PIC S99.
    05  VARY-FIELD-1 OCCURS 1 TO 10 TIMES
          DEPENDING ON CONTROL-1              PIC X(5).
    05  GROUP-ITEM-1.
        10  VARY-FIELD-2
              OCCURS 1 TO 10 TIMES
              DEPENDING ON CONTROL-2          PIC X(9).
01  STORE-VARY-FIELD-2.
    05  GROUP-ITEM-2.
        10  VARY-FLD-2
              OCCURS 1 TO 10 TIMES
              DEPENDING ON CONTROL-2          PIC X(9).
```

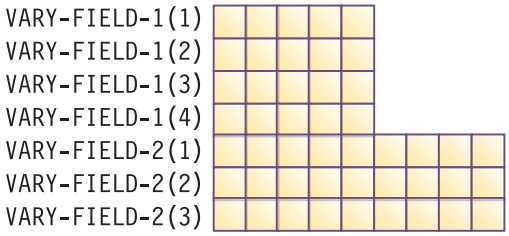
VARY-FIELD-1 の各エレメントは 5 バイトで、VARY-FIELD-2 の各エレメントは 9 バイトです。CONTROL-1 と CONTROL-2 の両方に値 3 が含まれている場合は、VARY-FIELD-1 および VARY-FIELD-2 のストレージは次のように描くことができます。



VARY-FIELD-1 に 4 番目のエレメントを追加する場合は、次のようにコーディングすれば、VARY-FIELD-2 の最初の 5 バイトのオーバーレイを回避することができます。(GROUP-ITEM-2 は、可変位置 GROUP-ITEM-1 の一時記憶域として働きます。)

```
MOVE GROUP-ITEM-1 TO GROUP-ITEM-2.
ADD 1 TO CONTROL-1.
MOVE five-byte-field TO
  VARY-FIELD-1 (CONTROL-1).
MOVE GROUP-ITEM-2 TO GROUP-ITEM-1.
```

VARY-FIELD-1 と VARY-FIELD-2 の更新後のストレージは次のようになります。



VARY-FIELD-1 の 4 番目のエレメントが VARY-FIELD-2 の最初のエレメントをオーバーレイしていないことに注意してください。

付録 C. 2 バイト文字セット (DBCS) データの変換

言語環境プログラム・サービス・ルーチン IGZCA2D および IGZCD2A の使用目的は、STRING、UNSTRING、および参照変更などの操作を正常に行うために、DBCS データが含まれる英数字データ項目と DBCS のみのデータ項目との間での変換を行うことにありました。これらのサービス・ルーチンは互換性を確保するために、継続して提供されてはいますが、この目的のためには、現在ではその代わりに国別データ項目および国別変換操作の利用をお勧めします。

上記のサービス・ルーチンは、コード・ページ引き数をサポートしません。また、CODEPAGE コンパイラー・オプションで指定するコード・ページに対応していません。

DBCS コンパイラー・オプションは、これらのサービス・ルーチンの操作には影響を与えません。

関連タスク
118 ページの『国別データの変換』
123 ページの『DBCS データを含む英数字データ項目の処理』

関連参照
『DBCS 表記』
『英数字から DBCS データへの変換 (IGZCA2D)』
640 ページの『DBCS データから英数字データへの変換 (IGZCD2A)』
318 ページの『CODEPAGE』

DBCS 表記

これらの記号は、DBCS 項目を記述するために、DBCS データ変換の例で使用されています。

記号	意味
< と >	それぞれ、シフトアウト (SO) およびシフトイン (SI)
D0, D1, D2, . . ., Dn	2 バイト EBCDIC 文字を除く任意の DBCS 文字
.A, .B, .C, . . .	任意の 2 バイトの EBCDIC 文字。ピリオド (.) は値 X'42' を表します。
A、B、または s などの単一の文字	任意の 1 バイトの EBCDIC 文字

英数字から DBCS データへの変換 (IGZCA2D)

言語環境プログラムの IGZCA2D サービス・ルーチンは、2 バイト文字を含む英数字データを DBCS のみのデータに変換します。

© Copyright IBM Corp. 1991, 2001

IGZCA2D の構文

IGZCA2D サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1
変換の送り出しフィールドで、英数字データ項目として処理されます。

parameter-2
変換の受け取りフィールドで、DBCS データ項目として処理されます。

parameter-2 で参照変更を使用することはできません。

parameter-3
変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

parameter-4
変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

使用上の注意

- *parameter-1*、*parameter-3*、および *parameter-4* は、BY REFERENCEY または BY CONTENT でルーチンに渡すことができますが、*parameter-2* は BY REFERENCE で渡さなければなりません。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、CALL ステートメントを使って、変換ルーチンに渡すようにしてください。さもないと、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCA2D は以下の変更を行います。
  - DBCS データはそのまま変更しないで、シフト・コードを除去します。
  - 単一バイト (非スペース) EBCDIC 文字 X'nn' を X'42nn' で表す文字に変換します。
  - 単一バイト・スペース (X'40') を、X'4240' ではなく DBCS スペース (X'4040') に変換します。
- IGZCA2D は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- *parameter-3* の内容の有効な範囲は、1 ～ 16,777,215 で、*parameter-4* の内容の有効な範囲は、1 ～ 15,777,214 です。

639 ページの『例: IGZCA2D』

関連参照

639 ページの『IGZCA2D 戻りコード』

IGZCA2D 戻りコード

IGZCA2D は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。それを次の表に示します。

戻りコード	解説
0	parameter-1 が変換され、その結果が parameter-2 に入れられました。
2	parameter-1 が変換され、その結果が parameter-2 に入れられました。 parameter-2 は、右側に DBCS スペースが埋め込まれました。
4	parameter-1 が変換され、その結果が parameter-2 に入れられました。 parameter-2 に入れられた DBCS データは、右側が切り捨てられました。
6	parameter-1 が変換され、その結果が parameter-2 に入れられました。 X'00' ～ X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。
8	parameter-1 が変換され、その結果が parameter-2 に入れられました。 X'00' ～ X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 parameter-2 は、右側に DBCS スペースが埋め込まれました。
10	parameter-1 が変換され、その結果が parameter-2 に入れられました。 X'00' ～ X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 parameter-2 の中の DBCS データは、右側が切り捨てられました。
12	parameter-1 の中の対になったシフト・コードの間で奇数のバイトが検出されました。変換は行われませんでした。
13	parameter-1 の中で、対になっていないかまたはネストされたシフト・コードが検出されました。変換は行われませんでした。
14	parameter-1 と parameter-2 がオーバーラップしました。変換は行われませんでした。
15	parameter-3 または parameter-4 に与えられた値が範囲外でした。変換は行われませんでした。
16	parameter-4 の中で奇数のバイトがコーディングされました。変換は行われませんでした。

例: IGZCA2D

次の CALL ステートメントは、alpha-item 内の英数字データを DBCS データに変換します。変換の結果は dbcs-item に入れられます。

```
CALL "IGZCA2D" USING BY REFERENCE alpha-item dbcs-item
    BY CONTENT LENGTH OF alpha-item LENGTH OF dbcs-item
```

変換前の alpha-item および dbcs-item の内容と長さが次のものであるとします。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = D4D5D6D7D8D9D0

LENGTH OF alpha-item = 12
LENGTH OF dbcs-item  = 14
```

変換後、alpha-item および dbcs-item には次のものが入ります。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = .A.BD1D2D3.C.D
```

RETURN-CODE レジスターの内容は 0 です。

関連参照
637 ページの『DBCS 表記』

DBCS データから英数字データへの変換 (IGZCD2A)

言語環境プログラムの IGZCD2A ルーチンは、純粋な DBCS データを、2 バイト文字を含むことのできる英数字データに変換します。

IGZCD2A の構文

IGZCD2A サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1
変換の送り出しフィールドで、DBCS データ項目として処理されます。

parameter-2
変換の受け取りフィールドで、英数字データ項目として処理されます。

parameter-3
変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

parameter-4
変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

使用上の注意

- *parameter-1*、*parameter-3*、および *parameter-4* は、BY REFERENCEY または BY CONTENT でルーチンに渡すことができますが、*parameter-2* は BY REFERENCE で渡さなければなりません。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、変換ルーチンに渡すようにしてください。さもないと、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCD2A は以下の変更を行います。
  - 2 バイトの EBCDIC 文字ではない DBCS 文字の前後にシフト・コードを挿入します。
  - 2 バイト文字を 1 バイト文字に変換します。
  - DBCS スペース (X'4040') を 単一バイト・スペース (X'40') に変換します。

- IGZCD2A は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- 変換されたデータに 2 バイト文字が含まれている場合、シフト・コードは *parameter-2* の長さに含まれます。
- *parameter-3* の内容の有効な範囲は、1 ～ 16,777,214 で、*parameter-4* の内容の有効な範囲は、1 ～ 15,777,215 (英数字項目の最大サイズ) です。

『例: IGZCD2A』

関連参照
『IGZCD2A 戻りコード』

IGZCD2A 戻りコード

IGZCD2A は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。それを次の表に示します。

戻りコード	解説
0	parameter-1 が変換され、その結果が parameter-2 に入れられました。
2	parameter-1 が変換され、その結果が parameter-2 に入れられました。parameter-2 は、右側に単一バイト・スペースが埋め込まれました。
4	parameter-1 が変換され、その結果が parameter-2 に入れられました。parameter-2 は、右側で切り捨てられました。[1]
14	parameter-1 と parameter-2 がオーバーラップしました。変換は行われませんでした。
15	parameter-3 または parameter-4 の値が範囲外でした。変換は行われませんでした。
16	parameter-3 の中で奇数のバイトがコーディングされました。変換は行われませんでした。

1. 切り捨てが DBCS 文字の中で行われる場合、その切り捨ては偶数バイト境界で行われ、シフトイン (SI) が挿入されます。必要な場合には、シフトインの後で、英数字データに単一バイト・スペースが埋め込まれます。

例: IGZCD2A

次の CALL ステートメントは、dbcs-item 内の DBCS データを 2 バイト文字を含む英数字データに変換します。変換の結果は alpha-item に入れられます。

```
CALL "IGZCD2A" USING BY REFERENCE dbcs-item alpha-item
    BY CONTENT LENGTH OF dbcs-item LENGTH OF alpha-item
```

変換前の dbcs-item および alpha-item の内容と長さが次のものであるとします。

```
dbcs-item  = .A.BD1D2D3.C.D
alpha-item = sssssssssss

LENGTH OF dbcs-item  = 14
LENGTH OF alpha-item = 12
```

変換後、dbcs-item および alpha-item には次のものが入ります。

```
dbcs-item  = .A.BD1D2D3.C.D
alpha-item = AB<D1D2D3>CD
```

RETURN-CODE レジスターの内容は 0 です。

関連参照
637 ページの『DBCS 表記』

付録 D. XML 参照資料

付録 D では、XML パーサーが特殊レジスター XML-CODE で戻る XML 例外コードについて説明します。また、パーサーが検査する、XML 仕様の整形式性制約条件に関する資料も記載します。

関連参照
『続行を許可する XML 例外』
648 ページの『続行を許可しない XML 例外』
652 ページの『XML 適合性』
XML 仕様 (www.w3c.org/xml)

続行を許可する XML 例外

次の表には、XML イベント EXCEPTION に関連した例外コード、および XML データの処理を続行可能なときに XML パーサーが特殊レジスター XML-CODE で戻す例外コードが記述されています。すなわち、コードは、以下の範囲のいずれかが入っています。

- 1-99
- 100,001-165,535
- 200,001-265,535

この表には、例外、および、例外後の続行要求時にパーサーが実行するアクションが記述されています。これらの記述内容において、「XML テキスト」という用語は、構文解析中の XML 文書が英数字データ項目形式なのか、国別データ項目形式なのかによって、それぞれ、XML-TEXT または XML-NTEXT を意味します。

コード	説明	続行されるパーサーのアクション
1	パーサーで、エレメントの内容に含まれない空白文字を走査中に、無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
2	パーサーで、エレメント内容に含まれない、処理命令、エレメント、コメント、または文書タイプ宣言の無効な開始が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

© Copyright IBM Corp. 1991, 2001

コード	説明	続行されるパーサーのアクション
3	パーサーで、重複する属性名が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
4	パーサーで、属性値にマークアップ文字 '<' が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
5	エレメントの開始および終了タグ名が一致しません。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
6	パーサーで、エレメント内容に無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
7	パーサーで、エレメント内容に、エレメント、コメント、処理命令、または CDATA セクションの無効な開始が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
8	パーサーで、エレメント内容に、一致する開始文字シーケンス '<![CDATA[' のない、CDATA 終了文字シーケンス ']]>' が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

コード	説明	続行されるパーサーのアクション
9	パーサーで、コメント内に無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
10	パーサーで、コメント内に、後に '>' が付いていない文字シーケンス '--' (ダッシュダッシュ) が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
11	パーサーで、処理命令データ・セグメント内に無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
12	処理命令ターゲット名が 小文字、大文字、または大小混合の 'xml' でした。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
13	パーサーで、16 進文字参照 (形式 &#xdddd; の) 内に無効な数字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
14	パーサーで、10 進数文字参照 (形式 &#dddd; の) 内に無効な数字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

コード	説明	続行されるパーサーのアクション
15	XML 宣言内のエンコード宣言値が小文字または大文字の A ～ Z で始まっていませんでした。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
16	文字参照が適切な XML 文字を参照しませんでした。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
17	パーサーで、エンティティー参照名に無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
18	パーサーで、属性値に無効文字が見つかりました。	パーサーでは、文書の終わりに到達するまで、あるいは続行不能の原因となるエラーが検出されるまで、エラーの検出を続行します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
50	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書エンコード宣言が認識可能なエンコードを指定しませんでした。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
51	文書は EBCDIC でエンコードされ、文書エンコード宣言はサポートされている EBCDIC エンコードを指定しましたが、パーサーでは、CODEPAGE コンパイラー・オプションによって指定されているコード・ページはサポートされません。	パーサーでは、文書エンコード宣言によって指定したエンコードが使用されます。
52	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書エンコード宣言は ASCII エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。

コード	説明	続行されるパーサーのアクション
53	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書エンコード宣言はサポートされている Unicode エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
54	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書エンコード宣言は、パーサーでサポートされていない Unicode エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
55	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書エンコード宣言は、パーサーでサポートされていないエンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
56	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書エンコード宣言が認識可能なエンコードを指定しませんでした。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
57	文書は ASCII でエンコードされ、文書エンコード宣言はサポートされている ASCII エンコードを指定しましたが、パーサーでは、CODEPAGE コンパイラー・オプションによって指定したコード・ページはサポートされません。	パーサーでは、文書エンコード宣言によって指定したエンコードが使用されます。
58	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書エンコード宣言がサポートされている EBCDIC エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
59	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書エンコード宣言がサポートされている Unicode エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。

コード	説明	続行されるパーサーのアクション
60	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書エンコード宣言は、パーサーでサポートされていない Unicode エンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
61	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書エンコード宣言は、パーサーでサポートされていないエンコードを指定しました。	パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。
100,001 ～ 165,535	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプション、および文書エンコード宣言によって指定したエンコードはいずれも、サポートされている EBCDIC コード・ページですが、同一ではありません。XML-CODE には、エンコード宣言に 100,000 をプラスするための CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。文書エンコード宣言に対して (100,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。
200,001 ～ 265,535	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプション、および文書エンコード宣言によって指定したエンコードはいずれも、サポートされている ASCII コード・ページですが、同一ではありません。XML-CODE には、エンコード宣言に 200,000 をプラスするための CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、CODEPAGE コンパイラー・オプションによって指定したエンコードが使用されます。文書エンコード宣言に対して (200,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。

関連タスク

241 ページの『XML 文書のエラー処理』

続行を許可しない XML 例外

以下の XML 例外では、XML-CODE をゼロに設定し、例外を処理後にパーサーに制御を戻しても、これ以降イベントはパーサーから戻されません。制御は、NOT ON EXCEPTION 句で指定するステートメントに、または NOT ON EXCEPTION 句をコーディングしていない場合には、句ステートメントの末尾に、渡されます。

コード	説明
100	パーサーで、XML 宣言の開始を走査中に、文書の末尾に達しました。
101	パーサーで、XML 宣言の末尾を走査中に、文書の末尾に達しました。
102	パーサーで、ルート・エレメントを走査中に、文書の末尾に達しました。

コード	説明
103	パーサーで、XML 宣言内のバージョン情報を走査中に、文書の末尾に達しました。
104	パーサーで、XML 宣言内のバージョン情報値を走査中に、文書の末尾に達しました。
106	パーサーで、XML 宣言内のエンコード宣言値を走査中に、文書の末尾に達しました。
108	パーサーで、XML 宣言内のスタンドアロン宣言値を走査中に、文書の末尾に達しました。
109	パーサーで、属性名を走査中に、文書の末尾に達しました。
110	パーサーで、属性値を走査中に、文書の末尾に達しました。
111	パーサーで、属性値内の文字参照またはエンティティー参照を走査中に、文書の末尾に達しました。
112	パーサーで、空のエレメント・タグを走査中に、文書の末尾に達しました。
113	パーサーで、ルート・エレメント名を走査中に、文書の末尾に達しました。
114	パーサーで、エレメント名を走査中に、文書の末尾に達しました。
115	パーサーで、エレメント内容の文字データを走査中に、文書の末尾に達しました。
116	パーサーで、エレメント内容の処理命令を走査中に、文書の末尾に達しました。
117	パーサーで、エレメント内容のコメントまたは CDATA セクションを走査中に文書の末尾に達しました。
118	パーサーで、エレメント内容のコメントを走査中に文書の末尾に達しました。
119	パーサーで、エレメント内容の CDATA セクションを走査中に文書の末尾に達しました。
120	パーサーで、エレメント内容の文字参照またはエンティティー参照を走査中に文書の末尾に達しました。
121	パーサーで、ルート・エレメントの末尾を走査中に、文書の末尾に達しました。
122	パーサーで、文書タイプ宣言の無効の可能性のある開始が見つかりました。
123	パーサーで、2 つ目の文書タイプ宣言が見つかりました。
124	ルート・エレメントの先頭文字が文字 '_' または ':' ではありませんでした。
125	エレメントの先頭の属性名の先頭文字が文字 '_' または ':' ではありませんでした。
126	パーサーで、エレメント名内に、またはエレメント名の後のいずれかに無効文字が見つかりました。
127	パーサーで、属性名の後に '=' 以外の文字が見つかりました。
128	パーサーで、無効な属性値区切り文字が見つかりました。
130	属性名の先頭文字が文字 '_' または ':' ではありませんでした。
131	パーサーで、属性名内に、または属性名の後のいずれかに無効文字が見つかりました。
132	空のエレメント・タグが、'/' の後に続く '>' で終了しませんでした。
133	エレメント終了タグ名の先頭文字が文字 '_' または ':' ではありませんでした。

コード	説明
134	エレメント終了タグ名が '>' で終了しませんでした。
135	エレメント名の先頭文字が文字 '_' または ':' ではありませんでした。
136	パーサーで、エレメント内容に、コメントまたは CDATA セクションの無効な開始が見つかりました。
137	パーサーで、コメントの無効な開始が見つかりました。
138	処理命令ターゲット名の先頭文字が文字 '_' または ':' ではありませんでした。
139	パーサーで、処理命令ターゲット名内に、または処理命令ターゲット名の後のいずれかに無効文字が見つかりました。
140	処理命令が終了文字シーケンス '?>' で終了しませんでした。
141	パーサーで、文字参照またはエンティティー参照内の '&' の後に無効文字が見つかりました。
142	バージョン情報が XML 宣言にありませんでした。
143	XML 宣言内の 'version' の後に '=' がありませんでした。
144	XML 宣言内のバージョン宣言値が欠落しているか、または不適切に区切られています。
145	XML 宣言内のバージョン情報値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
146	パーサーで、XML 宣言内のバージョン情報値の終了区切り文字の後に無効文字が見つかりました。
147	パーサーで、XML 宣言にオプションのエンコード宣言ではない、無効な属性が見つかりました。
148	XML 宣言内の 'encoding' の後に '=' がありませんでした。
149	XML 宣言内のエンコード宣言値が欠落しているか、または不適切に区切られています。
150	XML 宣言内のエンコード宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
151	パーサーで、XML 宣言内のエンコード宣言値の終了区切り文字の後に無効文字が見つかりました。
152	パーサーで、XML 宣言にオプションのスタンドアロン宣言ではない、無効な属性が見つかりました。
153	XML 宣言内の 'standalone' の後に '=' がありませんでした。
154	XML 宣言内のスタンドアロン宣言値が欠落しているか、または不適切に区切られています。
155	スタンドアロン宣言値が 'yes' または 'no' 以外の値になっていました。
156	XML 宣言内のスタンドアロン宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
157	パーサーで、XML 宣言内のスタンドアロン宣言値の終了区切り文字の後に無効文字が見つかりました。
158	XML 宣言が正しい文字シーケンス '?>' で終了しなかったか、無効属性が含まれていました。
159	パーサーで、ルート・エレメントの末尾の後に文書タイプ宣言の開始が見つかりました。

コード	説明
160	パーサーで、ルート・エレメントの末尾の後にエレメントの開始が見つかりました。
300	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しました。
301	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションは Unicode を指定しました。
302	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされていない ASCII コード・ページを指定しました。
303	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書エンコード宣言は空であったか、またはサポートされていない英字エンコード別名を含んでいました。
304	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書にはエンコード宣言が含まれていませんでした。
305	文書は EBCDIC でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書エンコード宣言はサポートされている EBCDIC エンコードを指定しませんでした。
306	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しました。
307	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションは Unicode を指定しました。
308	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書にはエンコード宣言が含まれていませんでした。
309	CODEPAGE コンパイラー・オプションはサポートされている ASCII コード・ページを指定しましたが、文書は Unicode でエンコードされました。
310	CODEPAGE コンパイラー・オプションはサポートされている EBCDIC コード・ページを指定しましたが、文書は Unicode でエンコードされました。
311	CODEPAGE コンパイラー・オプションはサポートされていないコード・ページを指定しましたが、文書は Unicode でエンコードされました。
312	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書エンコード宣言は空であったか、またはサポートされていない英字エンコード別名を含んでいました。
313	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書にはエンコード宣言が含まれていませんでした。
314	文書は ASCII でエンコードされましたが、CODEPAGE コンパイラー・オプションはサポートされておらず、文書エンコード宣言はサポートされている ASCII エンコードを指定しませんでした。
315	文書は UTF-16 リトル・エンディアンでエンコードされましたが、このプラットフォームでは、パーサーは UTF-16 リトル・エンディアンをサポートしません。
316	文書は UCS4 でエンコードされましたが、パーサーは UCS4 をサポートしません。
317	パーサーで、文書エンコードを判別できません。文書は破損している可能性があります。

コード	説明
318	文書は UTF-8 でエンコードされましたが、パーサーは UTF-8 をサポートしません。
319	文書は UTF-16 ビッグ・エンディアンでエンコードされましたが、このプラットフォームでは、パーサーは UTF-16 ビッグ・エンディアンをサポートしません。
500-999	内部エラーです。このエラーをサービス担当者に報告してください。

関連タスク

241 ページの『XML 文書のエラー処理』

XML 適合性

Enterprise COBOL に組み込まれている XML パーサーは、XML 仕様に記述されている定義に従った場合、XML 準拠のプロセッサーではありません。XML パーサーは、構文解析する XML 文書の妥当性検査を行いません。XML パーサーは、各種の整形式性エラーの検査は行いますが、妥当性検証をしない XML プロセッサーに必要とされるアクションのすべてを行うわけではありません。

特に、XML パーサーは、内部文書タイプ定義 (DTD 内部サブセット) を処理しません。したがって、XML パーサーは、デフォルト属性値の提供、属性値の正規化、および事前定義エンティティーを除く、内部エンティティーの置換テキストの組み込みを行いません。ただし、XML パーサーは、文書タイプ宣言全体を DOCUMENT-TYPE-DESCRIPTOR XML イベントの XML-TEXT または XML-NTEXT の内容として渡します。したがって、アプリケーションは、必要に応じて、これらのアクションを実行することができます。

オプションとして、パーサーを使用すると、プログラムはエラー後に XML 文書の処理を続行することができます。この目的は、XML 文書と処理プログラムのデバッグを円滑に行えるようにすることです。

XML 仕様での定義を要約すると、以下のような、テキスト・オブジェクトは整形式の XML 文書です。

- 概して言えば、XML 文書の文法に準拠している。
- XML 仕様に規定されている明示的な整形式性制約条件をすべて満足している。
- 文書内で直接的または間接的に参照する、構文解析したエンティティー (テキストのセグメント) がそれぞれ、整形式である。

COBOL XML パーサーでは、文書タイプ宣言を除き、文書が XML 文法に準拠しているかを検査します。文書タイプ宣言は、未検査の状態でそのままアプリケーションに渡されます。

以下の文書は、XML 仕様からの注釈です。オリジナル URL (www.w3.org/TR/REC-xml) に存在しない内容については、W3C は責任を負いません。注釈はすべて、仕様には含まれておらず、*イタリック体* で記述されています。

Copyright (C) 1994-2001 W3C (R) (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), All Rights Reserved. W3C liability, trademark, document use, and software licensing rules apply. (www.w3.org/Consortium/Legal/ipr-notice-20000612)

また、XML 仕様には、12 事項の明示的な整形式性制約が記述されています。COBOL XML パーサーが部分的または全面的に検査する制約事項は**太字**で記述されています。

1. 内部サブセット内のパラメーター・エンティティー (PE):
   「内部 DTD サブセットの中では、パラメーター・エンティティー参照は、マークアップ宣言の内部ではなく、マークアップ宣言が発生できる場所でしか発生できない。(この制約は、外部パラメーター・エンティティーの中で発生する参照や外部サブセットには適用されない。)」

   パーサーで、内部 DTD サブセットは処理されません。したがって、この制約は強制されません。
2. 外部サブセット:
   「外部サブセットが存在している場合は、そのプロダクションを extSubset に一致させなくてはならない。」

   パーサーで、外部サブセットは処理されません。したがって、この制約は強制されません。
3. 宣言と宣言の間のパラメーター・エンティティー:
   「DeclSep のパラメーター・エンティティー参照のテキストの置換はそのプロダクションを extSubsetDecl に一致させなくてはならない。」

   パーサーで、内部 DTD サブセットは処理されません。したがって、この制約は強制されません。
4. **エレメント・タイプの突き合わせ:**
   「エレメントの終了タグの名前は、開始タグのエレメント・タイプと一致させなければならない。」

   パーサーで、この規制は強制されます。
5. **属性指定の一意性:**
   「同じ開始タグまたは空エレメント・タグの中に 2 回以上現れてよい属性名はない。」

   パーサーで、一意性について、指定したエレメント内の最大 10 までの属性名が検査され、この制約は部分的にサポートされます。アプリケーションでは、この制限を超えて、すべての属性名を検査することができます。
6. 外部エンティティー参照の禁止:
   「属性値は、外部エンティティーへの直接的または間接的エンティティー参照を含むことができない。」

   パーサーで、この規制は強制されません。

7. 属性値内の '<' の禁止:
   「属性値の中で直接的または間接的に参照するエンティティーの置換テキストは、'<' を含んではならない。」
   パーサーで、この規制は強制されません。
8. **正しい文字:**
   「文字参照の使用に言及する文字は、そのプロダクションを Char に一致させなくてはならない。」
   パーサーで、この規制は強制されます。
9. エンティティーの宣言:
   「DTD のない文書、パラメーター・エンティティー参照が含まれない内部 DTD サブセットだけしかない文書、または、standalone='yes' の文書においては、外部サブセットまたはパラメーター・エンティティー内で発生しないエンティティー参照において与えられている Name が、外部サブセット、またはパラメーター・エンティティー内で発生しないエンティティー宣言の中のものと一致しなければならない。ただし，整形式文書は、次のエンティティー amp、lt、gt、apos、quot を宣言する必要はない。一般的エンティティーの宣言は、属性リスト宣言内のデフォルト値に現れる、それに対するどの参照よりも先行しなければならない。
   エンティティーが外部サブセットや外部パラメーター・エンティティーの中で宣言されている場合には、妥当性検証をしないプロセッサーは、それらの宣言を読み取って処理することを義務づけられないことに注意すること。そうした文書については、エンティティーは宣言されなくてはならないという規則は、standalone='yes' である場合に限って整形式性制約となる。」
   パーサーで、この規制は強制されません。
10. 解析済みエンティティー:
    「エンティティー参照は、解析されないエンティティーの名前を含んではならない。解析されないエンティティーを参照してもよいのは、ENTITY 型または ENTITIES 型として宣言した属性値の中だけである。」
    パーサーで、この規制は強制されません。
11. 再帰の禁止:
    「解析されるエンティティーは、直接的または間接的を問わず、それ自身への再帰的参照を含めてはならない。」
    パーサーで、この規制は強制されません。
12. DTD 内:
    「パラメーター・エンティティー参照が出現してよいのは、DTD の中だけである。」
    このエラーは発生する可能性がないため、パーサーで、この規制は強制されません。

上記の文書は、XML 仕様からの注釈です。オリジナル URL (www.w3.org/TR/REC-xml) に存在しない内容については、W3C は責任を負いません。上記の注釈はすべて、仕様には含まれていません。本文書は、W3C スタッフや他の関係者による校閲を受け、ディレクターによって W3C 推奨文書として承認されています。本文書は継続的に提供される文書であり、参照資料として使用されたり、別文書で仕様に含まれる資料として引用される可能性があります。仕様の正規版は英語バージョンで、W3C のサイトで確認することができます。翻訳文書については、翻訳上の誤りが含まれる可能性があります。

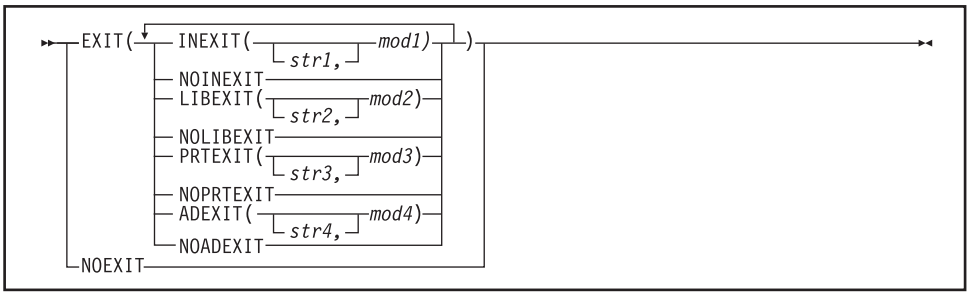
関連概念
225 ページの『COBOL でのXML パーサー』

関連参照
XML 仕様 (www.w3c.org/xml)
2.8 Prolog および 文書タイプ宣言 (*XML 仕様* URL: www.w3.org/TR/REC-xml#sec-prolog-dtd)

付録 E. EXIT コンパイラー・オプション

SYSIN、SYSLIB (またはコピー・ライブラリー)、および SYSPRINT の代わりにユーザー提供のモジュールを受け入れるのをコンパイラーに許可するには、EXIT オプションを使用します。

SYSADATA の場合、ADEXIT サブオプションは、SYSADATA レコードごとに、そのレコードがファイルに書き込まれた直後に呼び出されることになるモジュールを提供します。



デフォルト: NOEXIT

省略形: EX(INX|NOINX、LIBX|NOLIBX、PRTX|NOPRTX、ADX|NOADX)

サブオプションをまったく指定せずに EXIT オプションを指定すると、NOEXIT が実施されます。サブオプションは、コンマまたはスペースで区切って任意の順序で指定することができます。サブオプションの肯定形式と否定形式 (INEXIT|NOINEXT、LIBEXIT|NOLIBEXIT、PRTEXIT|NOPRTEXIT、または ADEXIT|NOADEXIT) の両方を指定した場合は、最後に指定された形式が有効になります。同じサブオプションを複数回指定すると、最後に指定したものが有効になります。

EXIT オプションは、呼び出し時に PARM フィールド (TSO の場合はコマンド引き数) に指定するか、インストール時にしか指定できません。EXIT オプションを PROCESS (CBL) ステートメントに指定してはなりません。

INEXIT([*'str1'*,]*mod1*)
: コンパイラーは、SYSIN ではなく、ユーザー提供のロード・モジュール（*mod1* はモジュール名）からソース・コードを読み取ります。

LIBEXIT([*'str2'*,]*mod2*)
: コンパイラーは、*library-name* または SYSLIB ではなく、ユーザー提供のロード・モジュール (*mod2* はモジュール名) からコピーブックを入手します。COPY ステートメントまたは BASIS ステートメントと一緒に使用するためです。

PRTEXIT([*'str3'*,]*mod3*)
: コンパイラーは、プリンター宛先の出力を、SYSPRINT ではなく、ユーザー提供のロード・モジュール（*mod3* はモジュール名）に渡します。

© Copyright IBM Corp. 1991, 2001

ADEXIT([*'str4'*,]*mod4*)
コンパイラーは、SYSADATA 出力を、ユーザー提供のロード・モジュール（*mod4* はモジュール名）に渡します。

モジュール名 *mod1*、*mod2*、*mod3*、および *mod4* は、同じものを参照することが可能です。

サブオプション *str1*、*str2*、*str3*、および *str4* は、ロード・モジュールに渡される文字ストリングです。これらのストリングはオプショナルです。ストリングは最高 64 文字までの長さにすることができ、アポストロフィで囲む必要があります。任意の文字を使用できますが、組み込みアポストロフィは二重にしなければならず、小文字は大文字に変換されます。

str1、*str2*、*str3*、または *str4* のいずれかが指定された場合、そのストリングは次の形式で、適切なユーザー出口モジュールに渡されます。

LL	ストリング

LL は、そのストリングの長さを含む (ハーフワード境界上の) ハーフワードです。

669 ページの『例: INEXIT ユーザー出口』

関連タスク
『ユーザー出口作業域の使用』
659 ページの『出口モジュールからの呼び出し』
667 ページの『EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用』

関連参照
659 ページの『INEXIT の処理』
661 ページの『LIBEXIT の処理』
664 ページの『PRTEXIT の処理』
665 ページの『ADEXIT の処理』
667 ページの『出口モジュールのエラー処理』

ユーザー出口作業域の使用

出口を使用すると、コンパイラーはユーザー出口作業域を提供します。この作業域は、出口モジュールによって獲得される GETMAIN ストレージのアドレスを保管するために使用できます。この作業域を使用して、モジュールを再入可能にすることができます。

ユーザー出口作業域は、フルワード境界に常駐する 4 フルワードです。このフルワードは、最初の出口ルーチンが呼び出される前に、2 進ゼロに初期化されます。作業域のアドレスは、パラメーター・リストの出口モジュールに渡されます。初期化後、コンパイラーは作業域に参照を行いません。

コンパイル時に複数の出口がアクティブになる場合は、作業域を使用するための独自の規則を確立する必要があります。たとえば、INEXIT モジュールは作業域の最

初のワードを使用し、LIBEXIT モジュールは 2 番目のワードを使用し、PRTEXIT モジュールは 3 番目のワードを使用し、ADEXIT モジュールは 4 番目のワードを使用します。

関連参照
『INEXIT の処理』
661 ページの『LIBEXIT の処理』
664 ページの『PRTEXIT の処理』
665 ページの『ADEXIT の処理』

出口モジュールからの呼び出し

COBOL プログラムまたはライブラリー・ルーチンを呼び出すには、出口モジュール内で COBOL 標準リンケージを使用します。呼び出しチェーンを正しくトレースするためには、これらの規則を知る必要があります。

プログラムまたはルーチンに呼び出しを行うと、レジスターは次のようにセットアップされます。

R1 呼び出されるプログラムまたはライブラリー・ルーチンに渡されるパラメーター・リストを指す

R13 呼び出しプログラムまたはルーチンが提供するレジスター保管域を指す

R14 呼び出しプログラムまたはルーチンの戻りアドレスを保持する

R15 呼び出されるプログラムまたはルーチンのアドレスを保持する

出口モジュールは、24 の RMODE 属性と ANY の AMODE 属性を持っていなければなりません。

INEXIT の処理

この出口モジュールは、SYSIN ではなく、ユーザー提供のロード・モジュールからソース・コードを読み取るために使用されます。

INEXIT の処理は、次のとおりです。

コンパイラーによるアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod1*) をロードします。	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します。	処理するためのソースを準備します。OPEN 要求の状況をコンパイラーに戻します。
ソース・ステートメントが必要になったときは、GET 命令コードを使用してこの出口モジュールを呼び出します。	次のステートメントのアドレスと長さ、または (ソース・ステートメントがそれ以上存在しない場合は) データ終わり標識のいずれかを戻します。
データ終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します。	その出力に関連のあるすべてのリソースを解放します。

INEXIT のパラメーター・リスト

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。次の表に、パラメーター・リストの内容を説明します。

オフセット	次のものの アドレスが入る	項目の説明
00	ユーザー出口タイプ	実行するユーザー出口を識別するハーフワード。 1=INEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 2=GET
08	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の成功を示します。 0=操作は成功しました。 4=データ終わり。 12=操作は失敗しました。
12	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域。
16	データ長	出口モジュールが設定するフルワードで、GET 操作によって戻されるレコードの長さを指定します (80 でなければなりません)。
20	データ または *str1*	出口モジュールが設定するフルワードであり、GET 操作の際、ユーザー所有のバッファー内のレコードのアドレスが入ります。 *str1* は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

669 ページの『例: INEXIT ユーザー出口』

関連タスク

667 ページの『EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用』

関連参照

661 ページの『LIBEXIT の処理』

LIBEXIT の処理

この出口モジュールは、SYSLIB または *library-name* データ・セットに代わって使用されます。コンパイラーは、COPY または BASIS ステートメントが検出されるたびに、このモジュールを呼び出してコピーブックを入手します。

LIBEXIT を指定する場合は、LIB コンパイラー・オプションが有効でなければなりません。

LIBEXIT の処理は、次のとおりです。

コンパイラーによるアクション	出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod2*) をロードします。	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します。	指定された *library-name* を処理のために準備します。 OPEN 要求の状況をコンパイラーに渡します。
library-name が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します。	指定された *library-name* 内の要求された *text-name* (または *basis-name*) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
GET 命令コードを使用してこの出口モジュールを呼び出します。	アクティブ・コピーブックからコピーされるレコードの長さとアドレス、またはデータ終わり標識のいずれかをコンパイラーに渡します。
データ終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します。	その入力に関連のあるすべてのリソースを解放します。

ネストされた COPY ステートメントによる LIBEXIT の処理

アクティブ・コピーブックからのレコードに、COPY ステートメントを含めることができます。(ただし、ネストされた COPY ステートメントに REPLACING 句を含めたり、REPLACING 句を持つ COPY ステートメントに、ネストされた COPY ステートメントを含めたりすることはできません。)

コンパイラーは、*text-name* への再帰呼び出しを許可しません。つまり、コピーブックは、そのコピーブックのデータの終わりに達するまでの間、ネストされた一連の COPY ステートメントの中で一度しか指定できません。

次の表は、1 つまたは複数の有効な COPY ステートメントがあるときに、LIBEXIT の処理がどのように変わるかを示しています。

コンパイラーによるアクション		出口モジュールによる結果としてのアクション
初期化時に出口モジュール (*mod2*) をロードします。		
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します。		指定された *library-name* を処理のために準備します。 OPEN 要求の状況をコンパイラーに渡します。
library-name が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します。		指定された *library-name* 内の要求された *text-name* (または *basis-name*) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。 位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
コンパイラーが有効なネストされた COPY ステートメントを検出したとき	ネストされた *COPY* ステートメントからの要求された library-name が前にオープンされていなければ、OPEN 命令コードを使用してこの出口モジュールを呼び出します。	アクティブ・コピーブックに関する制御情報をスタックに押し入れます。要求されたアクション (OPEN または FIND) を完了させます。新しく要求された *text-name* (または *basis-name*) がアクティブ・コピーブックになります。
	要求された新しい *text-name* のために FIND 命令コードを使用してこの出口モジュールを呼び出します。	
	GET 命令コードを使用してこの出口モジュールを呼び出します。	アクティブ・コピーブックからコピーされるレコードの長さとアドレス、またはデータ終わり標識のいずれかをコンパイラーに渡します。データの終わりに、スタックからその制御情報をポップします。
library-name が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します。		前のアクティブ・コピーブックのところに、位置決めを設定し直します。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
GET 命令コードを使用してこの出口モジュールを呼び出します。 同じレコードが渡されたかどうかを検査します。		このコピーブックから前に渡されたものと同じレコードをコンパイラーに渡します。検査後に、アクティブ・コピーブックからコピーされるレコードの長さとアドレス、またはデータ終わり標識のいずれかをコンパイラーに渡します。
データ終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します。		その入力に関連のあるすべてのリソースを解放します。

LIBEXIT のパラメーター・リスト

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。次の表に、LIBEXIT で使用されるパラメーター・リストの内容を説明します。

オフセット	次のものの アドレスが入る	項目の説明
00	ユーザー出口タイプ	実行するユーザー出口を識別するハーフワード。 2=LIBEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 2=GET; 4=FIND
08	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の成功を示します。 0=操作は成功しました。 4=データ終わり。 12=操作は失敗しました。
12	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域。
16	データ長	出口モジュールが設定するフルワードで、GET 操作によって戻されるレコードの長さを指定します (80 でなければなりません)。
20	データ または *str2*	出口モジュールが設定するフルワードであり、GET 操作の際、ユーザー所有のバッファー内のレコードのアドレスが入ります。 *str2* は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
24	システム *library-name*	COPY ステートメントの *library-name* が入れられる 8 文字の区域。プログラム名の処理規則および変換規則が適用されます。必要に応じて、ブランクが埋め込まれます。OPEN、CLOSE、および FIND に適用されます。
28	システム *text-name*	COPY ステートメントの *text-name*（BASIS ステートメントの *basis-name*）が入れられる 8 文字の区域。*program name* の処理規則と変換規則が適用されます。必要に応じて、ブランクが埋め込まれます。FIND にのみ適用されます。
32	ライブラリー名	COPY ステートメントからの完全な *library-name* が入れられる 30 文字の区域。必要な場合にはブランクが埋め込まれ、そのまま (英大文字に変換されず) で使用されます。OPEN、CLOSE、および FIND に適用されます。

オフセット	次のものの アドレスが入る	項目の説明
36	テキスト名	COPY ステートメントからの完全な *text-name* が入れられる 30 文字の区域。必要な場合にはブランクが埋め込まれ、そのまま (英大文字に変換されず) で使用されます。FIND にのみ適用されます。

関連タスク

667 ページの『EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用』

PRTEXIT の処理

この出口モジュールは、SYSPRINT データ・セットの代わりに使用されます。

PRTEXIT の処理は、次のとおりです。

コンパイラーによるアクション	出口モジュールによる結果としての アクション
初期化時に出口モジュール (*mod3*) をロードします。	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します。	処理のためにその出力宛先を準備します。OPEN 要求の状況をコンパイラーに渡します。
行が印刷されるときに、印刷されるレコードのアドレスと長さを与え、PUT 命令コードを使用してこの出口モジュールを呼び出します。	PUT 要求の状況を戻りコードによってコンパイラーに渡します。印刷されるレコードの最初のバイトには、ANSI プリンター制御文字が入ります。
データ終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します。	その出力宛先に関連のあるすべてのリソースを解放します。

PRTEXIT のパラメーター・リスト

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・バッファー・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。次の表に、PRTEXIT で使用されるパラメーター・リストの内容を説明します。

オフセット	次のものの アドレスが入る	項目の説明
00	ユーザー出口タイプ	実行するユーザー出口を識別するハーフワード。 3=PRTEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 3=PUT

オフセット	次のものの アドレスが入る	項目の説明
08	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の成功を示します。 0=操作は成功しました。 12=操作は失敗しました。
12	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域。
16	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード (コンパイラーはこの値を 133 に設定します)。
20	データ・バッファー または *str3*	コンパイラーが PUT 操作によって印刷されるレコードを入れたデータ・バッファーのアドレスが含まれているフルワード。 *str3* は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

関連タスク
667 ページの『EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用』

関連参照
661 ページの『LIBEXIT の処理』

ADEXIT の処理

ADEXITモジュールを使用するには、以下のものが必要です。

- コンパイラー・オプション ADATA (SYSADATA 出力を作成するため)
- DD ステートメント SYSADATA

ADEXIT の処理は、次のとおりです。

コンパイラーによるアクション	出口モジュールによる結果としての アクション
初期化時に出口モジュール (*mod4*) をロードします。	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します。	処理のためにその出力宛先を準備します。 OPEN 要求の状況をコンパイラーに渡します。

コンパイラーによるアクション	出口モジュールによる結果としてのアクション
コンパイラーが SYSADATA レコードを書き込んだときに、その SYSADATA のアドレスと長さを与え、PUT 命令コードを使用してこの出口モジュールを呼び出します。	PUT 要求の状況を戻りコードによってコンパイラーに渡します。
データ終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します。	すべてのリソースを解放します。

ADEXIT のパラメーター・リスト

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストはアドレスが入っている 10 個のフルワードで構成され、レジスター 1 にはパラメーター・リストのアドレスが入ります。戻りコード、データ長、およびデータ・バッファー・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。次の表に、ADEXIT で使用されるパラメーター・リストの内容を説明します。

オフセット	次のもののアドレスが入る	項目の説明
00	ユーザー出口タイプ	実行するユーザー出口を識別するハーフワード。 4=ADEXIT
04	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 3=PUT
08	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の成功を示します。 0=操作は成功しました。 12=操作は失敗しました。
12	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域。
16	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード。
20	データ・バッファー または *str4*	コンパイラーが PUT 操作によって印刷されるレコードを入れたデータ・バッファーのアドレスが含まれているフルワード。 *str4* は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
24	未使用	(LIBEXIT の専用)
28	未使用	(LIBEXIT の専用)
32	未使用	(LIBEXIT の専用)
36	未使用	(LIBEXIT の専用)

関連タスク

667 ページの『EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用』

関連参照
661 ページの『LIBEXIT の処理』

出口モジュールのエラー処理

出口モジュールをロードできないとき、または、出口モジュールが「操作失敗」または非ゼロの戻りコードを戻した場合は必ず、コンパイラーはエラー・メッセージを報告します。

次のいずれかが発生すると、メッセージ IGYSI5008 がオペレーターに出され、コンパイラーは戻りコード 16 で終了します。

- 出口モジュールをロードできない。
- OPEN 要求のときに、INEXIT からゼロ以外の戻りコードを受け取った。
- OPEN 要求のときに、PRTEXIT からゼロ以外の戻りコードを受け取った。

出口タイプと操作 (OPEN または LOAD) がメッセージで識別されます。

INEXIT または PRTEXIT からこれ以外のエラーが生じると、コンパイラーは終了します。

コンパイラーは、以下の条件を検出し、報告します。

5203 SYSPRINT ユーザー出口への PUT 要求が失敗し、戻りコード *nn* が戻されました。

5204 レコード・アドレスが *exit-name* ユーザー出口によって設定されていません。

5205 SYSIN ユーザー出口からの GET 要求が失敗し、戻りコード *nn* が戻されました。

5206 レコード長が *exit-name* ユーザー出口によって設定されていません。

EXIT コンパイラー・オプションと CICS および SQL ステートメントの併用

EXIT コンパイラー・オプションのサブオプションを使用してコンパイルするとき、および、プログラムに CICS ステートメントまたは SQL ステートメントが含まれているとき、出口モジュールにおいて実行できるアクションは、別個の、CICS トランスレーターと DB2 プリコンパイラーを使って CICS ステートメントまたは SQL ステートメントを変換したのか、それとも、組み込みの CICS トランスレーターと DB2 coprocessor を使って CICS ステートメントまたは SQL ステートメントを変換したのかによって、異なります。組み込みのトランスレーターを使用するときには、出口モジュールにおいて、EXEC CICS および EXEC SQL ステートメントを処理することができます。

INEXIT

別個の、CICS トランスレーターと DB2 プリコンパイラーを使って、EXEC CICS および EXEC SQL ステートメントをプログラムで変換し、続いて、そのプログラムを INEXIT サブオプションを使ってコンパイルするときには、EXEC ステートメントに対応して生成された COBOL ステートメントを INEXIT モジュールで処理すること

ができます。生成されたステートメントは INEXIT モジュールで変更することができます。ただし、この変更について、IBM ではサポートしません。

INEXIT サブオプションを使ってプログラムをコンパイルし、組み込みの CICS トランスレーターと DB2 coprocessor を使って (それぞれ、CICS および SQL コンパイラー・オプションを使用可能)、EXEC CICS および EXEC SQL ステートメントを変換するときには、EXEC CICS および EXEC SQL ステートメントを INEXIT モジュールで処理することができます。INEXIT モジュールは、組み込みのトランスレーターと coprocessor によって EXEC ステートメントに対応して生成された COBOL ステートメントの制御を得ることはありません。

LIBEXIT

LIBEXIT サブオプションを使ってプログラムをコンパイルし、組み込みの DB2 coprocessor を使用するときには、プログラム内の EXEC SQL INCLUDE ステートメントは、COBOL COPY ステートメントと同じように処理されます。渡されたステートメントは、EXEC SQL INCLUDE ステートメントによって LIBEXIT モジュールで処理することができます。(別個の DB2 プリコンパイラーを使用するときには、INEXIT サブオプションだけを使用して、EXEC SQL INCLUDE ステートメントで渡された入力ステートメントを処理することができます。)

別個の CICS トランスレーターを使って、EXEC CICS ステートメントをプログラムとそのコピーブックで変換し、続いて、そのプログラムを LIBEXIT サブオプションを使ってコンパイルするときには、EXEC CICS ステートメントに対応して生成された COBOL ステートメントを LIBEXIT モジュールで処理することができます。

LIBEXIT サブオプションを使ってプログラムをコンパイルし、組み込みの CICS トランスレーターを使って EXEC CICS ステートメントを変換するときには、EXEC CICS ソース・ステートメントを LIBEXIT モジュールで処理することができます。

PRTEXIT

別個の、CICS トランスレーターと DB2 プリコンパイラーを使って、EXEC CICS および EXEC SQL ステートメントをプログラムで変換し、そのプログラムを PRTEXIT サブオプションを使ってコンパイルするときには、EXEC ステートメントに対応して生成された COBOL SOURCE リスト・ステートメントを PRTEXIT モジュールで処理することができます。

PRTEXIT サブオプションを使ってプログラムをコンパイルし、組み込みの CICS トランスレーターと DB2 coprocessor を使って EXEC CICS および EXEC SQL ステートメントを変換するときには、SOURCE リストからの EXEC CICS および EXEC SQL ソース・ステートメントを PRTEXIT モジュールで処理することができます。PRTEXIT モジュールは、組み込みのトランスレーターと coprocessor によって生成された COBOL ソース・ステートメントへのアクセスを得ることはありません。

ADEXIT

別個の、CICS トランスレーターと DB2 プリコンパイラーを使って、EXEC CICS および EXEC SQL ステートメントをプログラムで変換し、そのプログラムを ADEXIT

サブオプションを使ってコンパイルするときには、EXEC ステートメントに対応して生成された COBOL SYSADATA ソース・ステートメントを ADEXIT モジュールで処理することができます。

ADEXIT サブオプションを使ってプログラムをコンパイルし、組み込みの CICS トランスレーターと DB2 coprocessor を使って EXEC CICS および EXEC SQL ステートメントを変換するときには、EXEC CICS および EXEC SQL ソース・ステートメントを ADEXIT モジュールで処理することができます。ADEXIT モジュールは、組み込みのトランスレーターと coprocessor によって生成された COBOL ソース・ステートメントへのアクセスを得ることはありません。

関連概念
422 ページの『DB2 coprocessor』
413 ページの『組み込みの CICS トランスレーター』

関連タスク
421 ページの『SQL オプションを使用したコンパイル』
411 ページの『CICS オプションを使用したコンパイル』

関連参照
659 ページの『INEXIT の処理』
661 ページの『LIBEXIT の処理』
664 ページの『PRTEXIT の処理』
665 ページの『ADEXIT の処理』

例: INEXIT ユーザー出口

次の例は、COBOL で書かれた INEXIT ユーザー出口モジュールを示しています。

```
************************************************************
*                                                          *
* Name:  SKELINX                                           *
*                                                          *
* Function:  Example of an INEXIT user exit written        *
*            in the COBOL language.                        *
*                                                          *
************************************************************

 Identification Division.
    Program-ID.  Skelinx.

 Environment Division.

 Data Division.

   WORKING-STORAGE Section.

*   ******************************************************
*   *                                                    *
*   *  Local variables.                                  *
*   *                                                    *
*   ******************************************************

    01  Record-Variable    Pic X(80).

*   ******************************************************
*   *                                                    *
*   *  Definition of the User-Exit Parameter List, which  *
```

```
*   *  is passed from the COBOL compiler to the user exit *
*   *  module.                                             *
*   *                                                      *
*   ******************************************************

   Linkage Section.
    01  Exit-Type         Pic 9(4)   Binary.
    01  Exit-Operation    Pic 9(4)   Binary.
    01  Exit-ReturnCode   Pic 9(9)   Binary.
    01  Exit-WorkArea.
        05  INEXIT-Slot   Pic 9(9)   Binary.
        05  LIBEXIT-Slot  Pic 9(9)   Binary.
        05  PRTEXIT-Slot  Pic 9(9)   Binary.
        05  Reserved-Slot Pic 9(9)   Binary.
    01  Exit-DataLength   Pic 9(9)   Binary.
    01  Exit-DataArea     Pointer.
    01  Exit-Open-Parm    Redefines  Exit-DataArea.
        05  String-Len    Pic 9(4)   Binary.
        05  Open-String   Pic X(64).
    01  Exit-Print-Line   Redefines  Exit-DataArea  Pic X(133).
    01  Exit-LIBEXIT      Pic X(8).
    01  Exit-Systext      Pic X(8).
    01  Exit-CBLLibrary   Pic X(30).
    01  Exit-CBLText      Pic X(30).

************************************************************
*                                                          *
*  Begin PROCEDURE DIVISION                                *
*                                                          *
*  Invoke the section to handle the exit.                  *
*                                                          *
************************************************************

 Procedure Division Using Exit-Type       Exit-Operation
                          Exit-ReturnCode Exit-WorkArea
                          Exit-DataLength Exit-DataArea
                          Exit-LIBEXIT    Exit-Systext
                          Exit-CBLLibrary Exit-CBLText.

     Evaluate Exit-type
       When (1) Perform Handle-INEXIT
       When (2) Perform Handle-LIBEXIT
       When (3) Perform Handle-PRTEXIT
     End-Evaluate
     Move 16 To Exit-ReturnCode
     Goback.

************************************************
*    I N E X I T   E X I T   P R O C E S S O R *
************************************************
 Handle-INEXIT.

     Evaluate Exit-Operation
       When (0) Perform INEXIT-Open
       When (1) Perform INEXIT-Close
       When (2) Perform INEXIT-Get
     End-Evaluate

     Move 16 To Exit-ReturnCode
     Goback.

 INEXIT-Open.
*    ------------------------------------------------------
*    Prepare for reading source
*    ------------------------------------------------------
     Goback.
```

```
 INEXIT-Close.
*    -------------------------------------------------------
*    Release resources
*    -------------------------------------------------------
     Goback.

 INEXIT-Get.
*    -------------------------------------------------------
*    Retrieve next source record
*    -------------------------------------------------------

*    -------------------------------------------------------
*    Return the address of the record to the compiler.
*    -------------------------------------------------------
     Set Exit-DataArea to Address of Record-Variable

*    -------------------------------------------------------
*    Set length of record in User-Exit Parameter List
*    -------------------------------------------------------
     Move 80 To Exit-DataLength

     Goback.

*****************************************************
*    L I B E X I T   P R O C E S S O R           *
*****************************************************
 Handle-LIBEXIT.
     Display "**** This module for INEXIT only"
     Move 16 To Exit-ReturnCode
     Goback.

*****************************************************
*    P R I N T  E X I T  P R O C E S S O R          *
*****************************************************
 Handle-PRTEXIT.
     Display "**** This module for INEXIT only"
     Move 16 To Exit-ReturnCode
     Goback.
*****************************************************

 End Program Skelinx.
```

付録 F. JNI.cpy

このリストには、Java インターオペラビリティーに対応するために OO 構文と併用する、コピーブック JNI.cpy を記述します。JNI.cpy を使用すると、COBOL プログラムから Java Native Interface (JNI) にアクセスすることができます。このコピーブックには、以下が含まれています。

- Java JNI タイプに対応するサンプル COBOL データ定義
- JNI 呼び出し可能サービス関数にアクセスするための関数ポインターが含まれている JNI 環境構造 JNINativeInterface

JNI.cpy は、JNI にアクセスするために C プログラマーが使用するヘッダー・ファイル jni.h に類似したものです。

```
*****************************************************************
* COBOL declarations for Java native method interoperation      *
*                                                               *
* To use the Java Native Interface callable services from a     *
* COBOL program:                                                *
* 1) Use a COPY statement to include this file into the         *
*    the Linkage Section of the program, e.g.                   *
*      Linkage Section.                                         *
*      Copy JNI                                                 *
* 2) Code the following statements at the beginning of the      *
*    Procedure Division:                                        *
*      Set address of JNIEnv to JNIEnvPtr                       *
*      Set address of JNINativeInterface to JNIEnv              *
*****************************************************************
*
* Sample JNI type definitions in COBOL
*
*01 jboolean1 pic X.
*  88 jboolean1-true  value X'01' through X'FF'.
*  88 jboolean1-false value X'00'.
*
*01 jbyte1 pic X.
*
*01 jchar1 pic N usage national.
*
*01 jshort1 pic s9(4)  comp-5.
*01 jint1   pic s9(9)  comp-5.
*01 jlong1  pic s9(18) comp-5.
*
*01 jfloat1  comp-1.
*01 jdouble1 comp-2.
*
*01 jobject1 object reference.
*01 jclass1  object reference.
*01 jstring1 object reference jstring.
*01 jarray1  object reference jarray.
*
*01 jbooleanArray1 object reference jbooleanArray.
*01 jbyteArray1    object reference jbyteArray.
*01 jcharArray1    object reference jcharArray.
*01 jshortArray1   object reference jshortArray.
*01 jintArray1     object reference jintArray.
*01 jlongArray1    object reference jlongArray.
*01 floatArray1    object reference floatArray.
*01 jdoubleArray1  object reference jdoubleArray.
*01 jobjectArray1  object reference jobjectArray.
```

© Copyright IBM Corp. 1991, 2001

```
* Possible return values for JNI functions.
 01 JNI-RC pic S9(9) comp-5.
* success
   88 JNI-OK        value  0.
* unknown error
   88 JNI-ERR       value -1.
* thread detached from the VM
   88 JNI-EDETACHED value -2.
* JNI version error
   88 JNI-EVERSION  value -3.
* not enough memory
   88 JNI-ENOMEM    value -4.
* VM already created
   88 JNI-EEXIST    value -5.
* invalid arguments
   88 JNI-EINVAL    value -6.

* Used in ReleaseScalarArrayElements
 01 releaseMode pic s9(9) comp-5.
   88 JNI-COMMIT value 1.
   88 JNI-ABORT  value 2.

 01 JNIenv pointer.

* JNI Native Method Interface - environment structure.
 01 JNINativeInterface.
   02 pointer.
   02 pointer.
   02 pointer.
   02 pointer.
   02 GetVersion                      function-pointer.
   02 DefineClass                     function-pointer.
   02 FindClass                       function-pointer.
   02 FromReflectedMethod             function-pointer.
   02 FromReflectedField              function-pointer.
   02 ToReflectedMethod               function-pointer.
   02 GetSuperclass                   function-pointer.
   02 IsAssignableFrom                function-pointer.
   02 ToReflectedField                function-pointer.
   02 Throw                           function-pointer.
   02 ThrowNew                        function-pointer.
   02 ExceptionOccurred               function-pointer.
   02 ExceptionDescribe               function-pointer.
   02 ExceptionClear                  function-pointer.
   02 FatalError                      function-pointer.
   02 PushLocalFrame                  function-pointer.
   02 PopLocalFrame                   function-pointer.
   02 NewGlobalRef                    function-pointer.
   02 DeleteGlobalRef                 function-pointer.
   02 DeleteLocalRef                  function-pointer.
   02 IsSameObject                    function-pointer.
   02 NewLocalRef                     function-pointer.
   02 EnsureLocalCapacity             function-pointer.
   02 AllocObject                     function-pointer.
   02 NewObject                       function-pointer.
   02 NewObjectV                      function-pointer.
   02 NewObjectA                      function-pointer.
   02 GetObjectClass                  function-pointer.
   02 IsInstanceOf                    function-pointer.
   02 GetMethodID                     function-pointer.
   02 CallObjectMethod                function-pointer.
   02 CallObjectMethodV               function-pointer.
   02 CallObjectMethodA               function-pointer.
   02 CallBooleanMethod               function-pointer.
   02 CallBooleanMethodV              function-pointer.
```

```
           02 CallBooleanMethodA              function-pointer.
           02 CallByteMethod                  function-pointer.
           02 CallByteMethodV                 function-pointer.
           02 CallByteMethodA                 function-pointer.
           02 CallCharMethod                  function-pointer.
           02 CallCharMethodV                 function-pointer.
           02 CallCharMethodA                 function-pointer.
           02 CallShortMethod                 function-pointer.
           02 CallShortMethodV                function-pointer.
           02 CallShortMethodA                function-pointer.
           02 CallIntMethod                   function-pointer.
           02 CallIntMethodV                  function-pointer.
           02 CallIntMethodA                  function-pointer.
           02 CallLongMethod                  function-pointer.
           02 CallLongMethodV                 function-pointer.
           02 CallLongMethodA                 function-pointer.
           02 CallFloatMethod                 function-pointer.
           02 CallFloatMethodV                function-pointer.
           02 CallFloatMethodA                function-pointer.
           02 CallDoubleMethod                function-pointer.
           02 CallDoubleMethodV               function-pointer.
           02 CallDoubleMethodA               function-pointer.
           02 CallVoidMethod                  function-pointer.
           02 CallVoidMethodV                 function-pointer.
           02 CallVoidMethodA                 function-pointer.
           02 CallNonvirtualObjectMethod      function-pointer.
           02 CallNonvirtualObjectMethodV     function-pointer.
           02 CallNonvirtualObjectMethodA     function-pointer.
           02 CallNonvirtualBooleanMethod     function-pointer.
           02 CallNonvirtualBooleanMethodV    function-pointer.
           02 CallNonvirtualBooleanMethodA    function-pointer.
           02 CallNonvirtualByteMethod        function-pointer.
           02 CallNonvirtualByteMethodV       function-pointer.
           02 CallNonvirtualByteMethodA       function-pointer.
           02 CallNonvirtualCharMethod        function-pointer.
           02 CallNonvirtualCharMethodV       function-pointer.
           02 CallNonvirtualCharMethodA       function-pointer.
           02 CallNonvirtualShortMethod       function-pointer.
           02 CallNonvirtualShortMethodV      function-pointer.
           02 CallNonvirtualShortMethodA      function-pointer.
           02 CallNonvirtualIntMethod         function-pointer.
           02 CallNonvirtualIntMethodV        function-pointer.
           02 CallNonvirtualIntMethodA        function-pointer.
           02 CallNonvirtualLongMethod        function-pointer.
           02 CallNonvirtualLongMethodV       function-pointer.
           02 CallNonvirtualLongMethodA       function-pointer.
           02 CallNonvirtualFloatMethod       function-pointer.
           02 CallNonvirtualFloatMethodV      function-pointer.
           02 CallNonvirtualFloatMethodA      function-pointer.
           02 CallNonvirtualDoubleMethod      function-pointer.
           02 CallNonvirtualDoubleMethodV     function-pointer.
           02 CallNonvirtualDoubleMethodA     function-pointer.
           02 CallNonvirtualVoidMethod        function-pointer.
           02 CallNonvirtualVoidMethodV       function-pointer.
           02 CallNonvirtualVoidMethodA       function-pointer.
           02 GetFieldID                      function-pointer.
           02 GetObjectField                  function-pointer.
           02 GetBooleanField                 function-pointer.
           02 GetByteField                    function-pointer.
           02 GetCharField                    function-pointer.
           02 GetShortField                   function-pointer.
           02 GetIntField                     function-pointer.
           02 GetLongField                    function-pointer.
           02 GetFloatField                   function-pointer.
           02 GetDoubleField                  function-pointer.
           02 SetObjectField                  function-pointer.
           02 SetBooleanField                 function-pointer.
```

```
                    02 SetByteField                  function-pointer.
                    02 SetCharField                  function-pointer.
                    02 SetShortField                 function-pointer.
                    02 SetIntField                   function-pointer.
                    02 SetLongField                  function-pointer.
                    02 SetFloatField                 function-pointer.
                    02 SetDoubleField                function-pointer.
                    02 GetStaticMethodID             function-pointer.
                    02 CallStaticObjectMethod        function-pointer.
                    02 CallStaticObjectMethodV       function-pointer.
                    02 CallStaticObjectMethodA       function-pointer.
                    02 CallStaticBooleanMethod       function-pointer.
                    02 CallStaticBooleanMethodV      function-pointer.
                    02 CallStaticBooleanMethodA      function-pointer.
                    02 CallStaticByteMethod          function-pointer.
                    02 CallStaticByteMethodV         function-pointer.
                    02 CallStaticByteMethodA         function-pointer.
                    02 CallStaticCharMethod          function-pointer.
                    02 CallStaticCharMethodV         function-pointer.
                    02 CallStaticCharMethodA         function-pointer.
                    02 CallStaticShortMethod         function-pointer.
                    02 CallStaticShortMethodV        function-pointer.
                    02 CallStaticShortMethodA        function-pointer.
                    02 CallStaticIntMethod           function-pointer.
                    02 CallStaticIntMethodV          function-pointer.
                    02 CallStaticIntMethodA          function-pointer.
                    02 CallStaticLongMethod          function-pointer.
                    02 CallStaticLongMethodV         function-pointer.
                    02 CallStaticLongMethodA         function-pointer.
                    02 CallStaticFloatMethod         function-pointer.
                    02 CallStaticFloatMethodV        function-pointer.
                    02 CallStaticFloatMethodA        function-pointer.
                    02 CallStaticDoubleMethod        function-pointer.
                    02 CallStaticDoubleMethodV       function-pointer.
                    02 CallStaticDoubleMethodA       function-pointer.
                    02 CallStaticVoidMethod          function-pointer.
                    02 CallStaticVoidMethodV         function-pointer.
                    02 CallStaticVoidMethodA         function-pointer.
                    02 GetStaticFieldID              function-pointer.
                    02 GetStaticObjectField          function-pointer.
                    02 GetStaticBooleanField         function-pointer.
                    02 GetStaticByteField            function-pointer.
                    02 GetStaticCharField            function-pointer.
                    02 GetStaticShortField           function-pointer.
                    02 GetStaticIntField             function-pointer.
                    02 GetStaticLongField            function-pointer.
                    02 GetStaticFloatField           function-pointer.
                    02 GetStaticDoubleField          function-pointer.
                    02 SetStaticObjectField          function-pointer.
                    02 SetStaticBooleanField         function-pointer.
                    02 SetStaticByteField            function-pointer.
                    02 SetStaticCharField            function-pointer.
                    02 SetStaticShortField           function-pointer.
                    02 SetStaticIntField             function-pointer.
                    02 SetStaticLongField            function-pointer.
                    02 SetStaticFloatField           function-pointer.
                    02 SetStaticDoubleField          function-pointer.
                    02 NewString                     function-pointer.
                    02 GetStringLength               function-pointer.
                    02 GetStringChars                function-pointer.
                    02 ReleaseStringChars            function-pointer.
                    02 NewStringUTF                  function-pointer.
                    02 GetStringUTFLength            function-pointer.
                    02 GetStringUTFChars             function-pointer.
                    02 ReleaseStringUTFChars         function-pointer.
                    02 GetArrayLength                function-pointer.
                    02 NewObjectArray                function-pointer.
```

```
    02 GetObjectArrayElement            function-pointer.
    02 SetObjectArrayElement            function-pointer.
    02 NewBooleanArray                  function-pointer.
    02 NewByteArray                     function-pointer.
    02 NewCharArray                     function-pointer.
    02 NewShortArray                    function-pointer.
    02 NewIntArray                      function-pointer.
    02 NewLongArray                     function-pointer.
    02 NewFloatArray                    function-pointer.
    02 NewDoubleArray                   function-pointer.
    02 GetBooleanArrayElements          function-pointer.
    02 GetByteArrayElements             function-pointer.
    02 GetCharArrayElements             function-pointer.
    02 GetShortArrayElements            function-pointer.
    02 GetIntArrayElements              function-pointer.
    02 GetLongArrayElements             function-pointer.
    02 GetFloatArrayElements            function-pointer.
    02 GetDoubleArrayElements           function-pointer.
    02 ReleaseBooleanArrayElements      function-pointer.
    02 ReleaseByteArrayElements         function-pointer.
    02 ReleaseCharArrayElements         function-pointer.
    02 ReleaseShortArrayElements        function-pointer.
    02 ReleaseIntArrayElements          function-pointer.
    02 ReleaseLongArrayElements         function-pointer.
    02 ReleaseFloatArrayElements        function-pointer.
    02 ReleaseDoubleArrayElements       function-pointer.
    02 GetBooleanArrayRegion            function-pointer.
    02 GetByteArrayRegion               function-pointer.
    02 GetCharArrayRegion               function-pointer.
    02 GetShortArrayRegion              function-pointer.
    02 GetIntArrayRegion                function-pointer.
    02 GetLongArrayRegion               function-pointer.
    02 GetFloatArrayRegion              function-pointer.
    02 GetDoubleArrayRegion             function-pointer.
    02 SetBooleanArrayRegion            function-pointer.
    02 SetByteArrayRegion               function-pointer.
    02 SetCharArrayRegion               function-pointer.
    02 SetShortArrayRegion              function-pointer.
    02 SetIntArrayRegion                function-pointer.
    02 SetLongArrayRegion               function-pointer.
    02 SetFloatArrayRegion              function-pointer.
    02 SetDoubleArrayRegion             function-pointer.
    02 RegisterNatives                  function-pointer.
    02 UnregisterNatives                function-pointer.
    02 MonitorEnter                     function-pointer.
    02 MonitorExit                      function-pointer.
    02 GetJavaVM                        function-pointer.
    02 GetStringRegion                  function-pointer.
    02 GetStringUTFRegion               function-pointer.
    02 GetPrimitiveArrayCritical        function-pointer.
    02 ReleasePrimitiveArrayCritical    function-pointer.
    02 GetStringCritical                function-pointer.
    02 ReleaseStringCritical            function-pointer.
    02 NewWeakGlobalRef                 function-pointer.
    02 DeleteWeakGlobalRef              function-pointer.
    02 ExceptionCheck                   function-pointer.
```

関連タスク

535 ページの『JNI サービスへのアクセス』

付録 G. COBOL SYSADATA ファイル内容

ADATA コンパイラー・オプションを使用すると、コンパイラーは、プログラム・データが含まれるファイルを生成します。このファイルを使用して (コンパイラー・リストではなく)、コンパイル済みプログラムに関する情報を取り出すことができます。たとえば、シンボリック・デバッグ・ツールや相互参照ツールに対応したコンパイル済みプログラムに関する情報を取り出すことができます。

682 ページの『例: SYSADATA』

関連参照

『SYSADATA ファイルに影響する既存のコンパイラー・オプション』
680 ページの『レコード・タイプ』
683 ページの『SYSADATA レコード記述』
315 ページの『ADATA』

SYSADATA ファイルに影響する既存のコンパイラー・オプション

以下のコンパイラー・オプションは、SYSADATA ファイルの内容に影響を与える可能性があります。

COMPILE

NOCOMPILE(W|E|S) を使用すると、コンパイルが実行途中で停止され、その結果、特定のメッセージが失われる可能性があります。

EVENTS EVENTS を使用すると、ADATA オプションと同じ結果を得ます (すなわち、ADATA または EVENTS、あるいはその両方が指定され、SYSADATA ファイルが生成されます)。(互換性のために。)

EXIT INEXIT を使用すると、コンパイル・ソース・ファイルの識別が禁止されます。

LANGUAGE

LANGUAGE は、メッセージ・テキスト (大文字英語、英大 / 小文字混合、または日本語) を制御します。日本語を選択すると、そのために DBCS 文字がエラー識別レコードに書き込まれることになる場合があります。

NUM NUM オプションを使用すると、コンパイラーは、生成されたシーケンス番号ではなく、ソース・レコードの 1 ～ 6 列の内容を行番号に使用します。無効な (非数値の) または不正なシーケンス番号は、直前のレコードより 1 だけ大きな数値で置き換えられます。

以下の SYSADATA フィールドには、NUM/NONUM 設定によってその内容が異なる、行番号が含まれています。

```
Type   Field
===    ======
0020   AE_LINE         External Symbol record
0030   ATOK_LINE       Token record
0032   AF_STMT         Source Error record
0038   AS_STMT         Source record
```

© Copyright IBM Corp. 1991, 2001

```
0039   AS_REP_EXP_SLIN COPY REPLACING record
0039   AS_REP_EXP_ELIN COPY REPLACING record
0042   ASY_STMT        Symbol record
0044   AX_DEFN         Symbol Cross Reference record
0044   AX_STMT         Symbol Cross Reference record
0046   AN_STMT         Nested Program record
```

タイプ 0038 ソース・レコードには、次のように、行番号とレコード番号に関連する 2 つのフィールドが含まれています。

- AS_STMT には、NUM および NONUM の両方に、コンパイラー行番号が含まれています。
- AS_CUR_REC# には、物理ソース・レコード番号が含まれています。

上記の 2 つのフィールドは常に、上記フィールドのすべてにおいて使われるコンパイラー行番号と物理ソース・レコード番号を相関させるために使用されます。

TEST TEST オプションを使用すると、SYSADATA ファイルの内容にも影響を与える、追加のオブジェクト・テキスト・レコードが作成されます。

残りのコンパイラー・オプションは、SYSADATA ファイルに直接的な影響は与えませんが、FLAGSAA、FLAGSTD、SSRANGE など、特定のオプションに関連付けられた、別のエラー・メッセージの生成をトリガーする可能性があります。

682 ページの『例: SYSADATA』

関連参照

『レコード・タイプ』
319 ページの『COMPILE』
331 ページの『LANGUAGE』
336 ページの『NUMBER』
349 ページの『TEST』

レコード・タイプ

SYSADATA ファイルには、別々のレコード・タイプに分類されるレコードが含まれています。各レコード・タイプには、コンパイルされる COBOL 言語プログラムに関する情報が提供されます。各レコードは、以下の 2 つのセクションで構成されます。

- 全レコード・タイプに対して同一の構造を有する、12 バイトのヘッダー・セクション
- レコード・タイプによって異なる、可変長データ・セクション

ヘッダー・セクションには、他の項目のほかに、レコードのタイプを識別するレコード・コードが含まれています。

次の表には、関連データ・ファイルに記述されるレコードのタイプをリストします。

レコード・タイプ	アクション
ジョブ識別レコード - X’0000’	ソース・データの処理に使用する環境に関する情報を記述します。
ADATA 識別レコード - X’0001’	SYSADATA ファイルのレコードに関する共通情報を記述します。
コンパイル単位の開始 / 終了レコード - X’0002’	ソース・ファイル内のコンパイル単位の開始と終了のマーク付けを行います。
オプション・レコード - X’0010’	コンパイルに使用するコンパイラー・オプションを記述します。
外部シンボル・レコード - X’0020’	プログラム内のすべての外部名、定義、および参照を記述します。
構文解析ツリー・レコード - X’0024’	プログラムの構文解析ツリーにノードを定義します。
トークン・レコード - X’0030’	ソース・トークンを定義します。
ソース・エラー・レコード - X’0032’	ソース・プログラム・ステートメントのエラーを記述します。
ソース・レコード - X’0038’	単一のソース行を記述します。
COPY REPLACING レコード - X’0039’	コピーするコピーブック内のテキストとの、COPY...REPLACING オペランド 1 の突き合わせの結果として、テキスト置換のインスタンスを記述します。
記号レコード - X’0042’	プログラムに定義される、単一の記号を記述します。プログラムに定義される、それぞれの記号ごとに 1 つの記号レコードがあります。
記号相互参照レコード - X’0044’	単一の記号への参照を記述します。
ネストされたプログラム・レコード - X’0046’	プログラムの名前とネスト・レベルを記述します。
ライブラリー・レコード - X’0060’	各ライブラリーで使用されるライブラリー・ファイルとメンバーを記述します。
統計レコード - X’0090’	コンパイルに関する統計を記述します。
EVENTS レコード - X’0120’	EVENTS レコードは、SYSADATA ファイルに記録され、コンパイラーの前のリリースとの互換性を記述します。EVENTS レコードのデータ・フォーマットは、コンパイラーの前のリリースのものと同一ですが、レコードの先頭に置かれる標準 ADATA ヘッダー、および EVENTS レコード・データの長さを示すフィールドが追加されます。

682 ページの『例: SYSADATA』

例: SYSADATA

次の例に、COBOL プログラムのリストの一部を示します。この COBOL プログラムを ADATA オプションを使ってコンパイルした場合には、関連データ・ファイルに生成されるレコードは、以下に示すシーケンスで記述されます。他のプログラムを使用した場合には、以下に記述されていないレコードが生成される可能性があり、レコードの配列に影響を与えることになります。

次は、COBOL プログラムのサンプル関連データ出力です。

```
LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-·--+----8  Map and Cross Reference
  000001               IDENTIFICATION DIVISION.                                        AD000020
  000002               PROGRAM-ID. AD04202.                                            AD000030
  000003               ENVIRONMENT DIVISION.                                           AD000040
  000004               DATA DIVISION.                                                  AD000050
  000005               WORKING-STORAGE SECTION.                                        AD000060
  000006               77   COMP3-FLD2  pic  S9(3)v9.                                  AD000070
  000007               PROCEDURE DIVISION.                                             AD000080
  000008                   STOP RUN.
```

タイプ	説明
X'0120'	EVENTS タイム・スタンプ・レコード
X'0120'	EVENTS プロセッサー・レコード
X'0120'	EVENTS ファイル ID レコード
X'0120'	EVENTS プログラム・レコード
X'0001'	ADATA 識別レコード
X'0000'	ジョブ識別レコード
X'0010'	オプション・レコード
X'0038'	ステートメント 1 のソース・レコード `IDENTIFICATION DIVISION.`
X'0038'	ステートメント 2 のソース・レコード `PROGRAM-ID。AD04202.`
X'0038'	ステートメント 3 のソース・レコード `ENVIRONMENT DIVISION.`
X'0038'	ステートメント 4 のソース・レコード `DATA DIVISION.`
X'0038'	ステートメント 5 のソース・レコード `WORKING-STORAGE SECTION.`
X'0038'	ステートメント 6 のソース・レコード `77 COMP3-FLD2 pic S9(3)v9.`
X'0038'	ステートメント 7 のソース・レコード `PROCEDURE DIVISION.`
X'0038'	ステートメント 8 のソース・レコード `STOP RUN.`
X'0020'	AD04202 の外部シンボル・レコード
X'0044'	STOP の記号相互参照レコード
X'0044'	COMP3-FLD2 の記号相互参照レコード
X'0044'	AD04202 の記号相互参照レコード
X'0042'	AD04202 の記号レコード
X'0042'	COMP3-FLD2 の記号レコード
X'0090'	統計レコード
X'0120'	EVENTS ファイル終わりレコード

関連参照

683 ページの『SYSADATA レコード記述』

SYSADATA レコード記述

関連データ・ファイルに書き込まれるレコードの形式については、以下の関連参照を参照してください。フィールドは、各レコード・タイプを説明する次の略語で表されています。

C 文字 (EBCDIC または ASCII) データを表す

H 2 バイトの 2 進整数データを表す

F 4 バイトの 2 進整数データを表す

A 4 バイトの 2 進整数アドレスとオフセット・データを表す

X 16 進数 (ビット) データまたは 1 バイトの 2 進整数データを表す

データ型には、境界合わせは一切含まれていません。したがって、上記の暗黙の長さは、長さ指標 (Ln) を含めることで変更される可能性があります。すべての整数データは、ヘッダー・フラグ・バイトの指標ビットによって、ビッグ・エンディアン形式、またはリトル・エンディアン 形式になっています。ビッグ・エンディアン形式では、ビット 0 が常に最上位ビットで、ビット 'n' が最下位ビットであることを意味します。リトル・エンディアン形式では、Intel ベース・ハードウェアに見られるような、「バイト反転」の整数を指します。

未定義フィールドおよび未使用値はすべて、予約済みです。

関連参照

『共通ヘッダー・セクション』
685 ページの『ジョブ識別レコード - X'0000'』
686 ページの『ADATA 識別レコード - X'0001'』
687 ページの『コンパイル単位の開始 / 終了レコード - X'0002'』
687 ページの『オプション・レコード - X'0010'』
698 ページの『外部シンボル・レコード - X'0020'』
699 ページの『構文解析ツリー・レコード - X'0024'』
713 ページの『トークン・レコード - X'0030'』
714 ページの『ソース・エラー・レコード - X'0032'』
714 ページの『ソース・レコード - X'0038'』
715 ページの『COPY REPLACING レコード - X'0039'』
715 ページの『記号レコード - X'0042'』
727 ページの『記号相互参照レコード - X'0044'』
728 ページの『ネストされたプログラム・レコード - X'0046'』
729 ページの『ライブラリー・レコード - X'0060'』
730 ページの『統計レコード - X'0090'』
731 ページの『EVENTS レコード - X'0120'』

共通ヘッダー・セクション

全レコード・タイプに共通のヘッダー・セクションです。MVS、VM、および VSE の場合、各レコードの前に 4 バイトの RDW (レコード記述子ワード) が置かれます。この RDW は通常、アクセス方式によってのみ使用され、ダウンロード・ユーティリティーによって、ストリップされます。

ヘッダー・セクションの形式は次のとおりです。

フィールド	サイズ	説明
言語コード	XL1	**16** 高水準アセンブラー **17** すべてのプラットフォームの COBOL **40** サポートされるプラットフォームの PL/I
レコード・タイプ	HL2	以下のいずれかのレコード・タイプ。 **X'0000'** ジョブ識別レコード[1] **X'0001'** ADATA 識別レコード **X'0002'** コンパイル単位の開始 / 終了レコード **X'0010'** オプション・レコード [1] **X'0020'** 外部シンボル・レコード **X'0024'** 構文解析ツリー・レコード **X'0030'** トークン・レコード **X'0032'** ソース・エラー・レコード **X'0038'** ソース・レコード **X'0039'** COPY REPLACING レコード **X'0042'** 記号レコード **X'0044'** 記号相互参照レコード **X'0046'** ネストされたプログラム・レコード **X'0060'** ライブラリー・レコード **X'0090'** 統計レコード[1] **X'0120'** EVENTS レコード
関連データ・アーキテクチャー・レベル	XL1	**3** ヘッダー構造の定義レベル

フラグ	XL1	**.... ..1.** ADATA レコード整数は、リトル・エンディアン (Intel) 形式です。 **.... ...1** このレコードは、次のレコードに続行されます。 **1111 11..** 将来の利用のために予約済み。
関連データ・レコード・エディション・レベル	XL1	特定のレコード・タイプの新規形式を表すために使用され、通常は 0 です。
予約済み	CL4	将来の利用のために予約済み
関連データ・フィールド長	HL2	ヘッダーの後に置かれるデータの長さ (バイト単位)
1. バッチ・コンパイル (一連のプログラム) が ADATA オプションで実行されるときには、それぞれのコンパイルごとに、複数ジョブ識別、オプション、および統計レコードがあります。		

12 バイト・ヘッダーのマッピングには、MVS、VM、および VSE のアクセス方式が必要とする、可変長レコード記述子ワードに使用される領域は組み込まれません。

ジョブ識別レコード - X'0000'

次のテーブルに、ジョブ識別レコードの内容を示します。

フィールド	サイズ	説明
日付	CL8	YYYYMMDD 形式のコンパイルの日付
時刻	CL4	HHMM 形式のコンパイルの時刻
プロダクト番号	CL8	関連データ・ファイルを生成したコンパイラーのプロダクト番号
プロダクト・バージョン	CL8	V.R.M 形式の、関連データ・ファイルを生成したプロダクトのバージョン番号
PTF レベル	CL8	関連データ・ファイルを生成したプロダクトの PTF レベル番号 (PTF 番号が利用不能であるとき、このフィールドはブランクです。)
システム ID	CL24	コンパイルが実行されたシステムのシステム識別
ジョブ名	CL8	コンパイル・ジョブの MVS ジョブ名
ステップ名	CL8	コンパイル・ステップの MVS ステップ名
PROC ステップ	CL8	コンパイル・プロシージャー・ステップの MVS プロシージャー・ステップ名
入力ファイル数[1]	HL2	このレコードに記録した入力ファイルの数 以下の、7 つのフィールドのグループでは、このフィールドの値に従って、'n' 回発生します。
...入力ファイル番号	HL2	ファイルの割り当てシーケンス番号

フィールド	サイズ	説明
...入力ファイル名長	HL2	次の入力ファイル名の長さ
...ボリューム通し番号長	HL2	ボリューム通し番号の長さ
...メンバー名長	HL2	メンバー名の長さ
...入力ファイル名	CL(n)	コンパイルの入力ファイルの名前
...ボリューム通し番号	CL(n)	入力ファイルが常駐する (最初の) ボリュームのボリューム通し番号
...メンバー名	CL(n)	該当する場合には、入力ファイル内のメンバーの名前

1. 入力ファイル数が、関連データ・ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。入力ファイルの現行数 (そのレコードの) は、レコードに保管され、レコードは、関連データ・ファイルに書き込まれます。次のレコードには、入力ファイルの残りが含まれます。入力ファイルの数のカウントは、現行レコードのカウントです。

ADATA 識別レコード - X'0001'

次の表に、ADATA 識別レコードの内容を示します。

フィールド	サイズ	説明
時刻 (2 進数)	XL8	世界時 (UT) は、真夜中のグリニッジ標準時を基準としたマイクロ秒数を表す 2 進数で、下位ビットは 1 マイクロ秒を表します。この時刻は、時間帯独立のタイム・スタンプとして使用することができます。 ワークステーション・コンパイラーでは、フィールドの 5 ～ 8 バイトのみが、時刻が含まれるフルワード・バイナリー・フィールドとして使用されます。
CCSID[1]	XL2	コード化文字セット ID
文字セット・フラグ	XL1	**X'80'** EBCDIC (IBM-037) **X'40'** ASCII (IBM-850)
コード・ページ名長	XL2	... の後に続く、コード・ページ名の長さ
コード・ページ名	CL(n)	コード・ページの名前

1. 適切な CCS フラグが常に設定されます。

   CCSID がゼロ以外に設定されると、コード・ページ名の長さはゼロとなります。CCSID がゼロに設定されると、コード・ページ名の長さはゼロ以外の値となり、コード・ページ名が表示されます。

コンパイル単位の開始 / 終了レコード - X'0002'

次の表に、ADATA コンパイル単位の開始 / 終了レコードの内容を示します。

フィールド	サイズ	説明
タイプ	HL2	コンパイル単位タイプ。以下のどちらかが使用できます。 **X'0000'** 開始コンパイル単位 **X'0001'** 終了コンパイル単位
予約済み	CL2	将来の利用のために予約済み
予約済み	FL4	将来の利用のために予約済み

オプション・レコード - X'0010'

次の表に、オプション・レコードの内容を示します。

フィールド	サイズ	説明
オプション・バイト 0	XL1	**1111 1111** 将来の利用のために予約済み
オプション・バイト 1	XL1	**1... ....** ビット 1 = DECK、ビット 0 = NODECK **.1.. ....** ビット 1 = ADATA、ビット 0 = NOADATA **..1. ....** ビット 1 = COLLSEQ(EBCDIC) (ワークステーションのみ) **...1 ....** ビット 1 = SEPOBJ、ビット 0 = NOSEPOBJ (ワークステーションのみ) **.... 1...** ビット 1 = NAME、ビット 0 = NONAME **.... .1..** ビット 1 = OBJECT、ビット 0 = NOOBJECT **.... ..1.** ビット 1 = SQL、ビット 0 = NOSQL **.... ...1** ビット 1 = CICS、ビット 0 = NOCICS

フィールド	サイズ	説明
オプション・バイト 2	XL1	**1... ....** ビット 1 = OFFSET、ビット 0 = NOOFFSET **.1.. ....** ビット 1 = MAP、ビット 0 = NOMAP **..1. ....** ビット 1 = LIST、ビット 0 = NOLIST **...1 ....** ビット 1 = DBCSXREF、ビット 0 = NODBCSXREF **.... 1...** ビット 1 = XREF(SHORT)、ビット 0 = XREF(SHORT) ではない。このフラグは、ビット 7 のフラグと組み合わせて使用します。このフラグによって、XREF(FULL) はオフ状態で示され、ビット 7 のフラグはオン状態で示されます。 **.... .1..** ビット 1 = SOURCE、ビット 0 = NOSOURCE **.... ..1.** ビット 1 = VBREF、ビット 0 = NOVBREF **.... ...1** ビット 1 = XREF、ビット 0 = XREF ではない。ビット 4 以降のフラグも参照。

フィールド	サイズ	説明
オプション・バイト 3	XL1	**1... ....** ビット 1 = FLAG、ビット 0 = NOFLAG **.1.. ....** ビット 1 = FLAGSTD、ビット 0 = NOFLAGSTD **..1. ....** ビット 1 = NUM、ビット 0 = NONUM **...1 ....** ビット 1 = SEQUENCE、ビット 0 = NOSEQUENCE **.... 1...** ビット 1 = SOSI、ビット 0 = NOSOSI (ワークステーションのみ) **.... .1..** ビット 1 = NSYMBOL(NATIONAL)、ビット 0 = NSYMBOL(DBCS) **.... ..1.** ビット 1 = PROFILE、ビット 0 = NOPROFILE (ワークステーションのみ) **.... ...1** ビット 1 = WORD、ビット 0 = NOWORD
オプション・バイト 4	XL1	**1... ....** ビット 1 = ADV、ビット 0 = NOADV **.1.. ....** ビット 1 = APOST、ビット 0 = QUOTE **..1. ....** ビット 1 = DYNAM、ビット 0 = NODYNAM **...1 ....** ビット 1 = AWO、ビット 0 = NOAWO **.... 1...** ビット 1 = 指定済み RMODE、ビット 0 = RMODE(AUTO) **.... .1..** ビット 1 = RENT、ビット 0 = NORENT **.... ..1.** ビット 1 = RES - このフラグは、常に COBOL に設定されます。 **.... ...1** ビット 1 = RMODE(24)、ビット 0 = RMODE(ANY)

フィールド	サイズ	説明
オプション・バイト 5	XL1	**1... ....** 互換性用に予約済み **.1. .....** ビット 1 = OPT、ビット 0 = NOOPT **..1. ....** ビット 1 = LIB、ビット 0 = NOLIB **...1 ....** ビット 1 = DBCS、ビット 0 = NODBCS **.... 1...** ビット 1 = OPT(FULL)、ビット 0 = OPT(FULL) ではない **.... .1..** ビット 1 = SSRANGE、ビット 0 = NOSSRANGE **.... ..1.** ビット 1 = TEST、ビット 0 = NOTEST **.... ...1** ビット 1 = PROBE、ビット 0 = NOPROBE (OS/2 のみ)
オプション・バイト 6	XL1	**1... ....** ビット 1 = CMPR2、ビット 0 = NOCMPR2 **..1. ....** ビット 1 = NUMPROC(PFD)、ビット 0 = NUMPROC(NOPFD) **...1 ....** ビット 1 = NUMCLS(ALT)、ビット 0 = NUMCLS(PRIM) **.... .1..** ビット 1 = BINARY(S370)、ビット 0 = BINARY(NATIVE) (ワークステーションのみ) **.... ..1.** ビット 1 = TRUNC(STD)、ビット 0 = TRUNC(OPT) **.... ...1** ビット 1 = ZWB、ビット 0 = NOZWB **.1.. 1...** 将来の利用のために予約済み

フィールド	サイズ	説明
オプション・バイト 7	XL1	**1... ....** ビット 1 = ALOWCBL、ビット 0 = NOALOWCBL (ワークステーションのみ) **.1.. ....** ビット 1 = TERM、ビット 0 = NOTERM **..1. ....** 1 = DUMP、ビット 0 = NODUMP **...1 11..** 予約済み **.... ..1.** ビット 1 = CURRENCY、ビット 0 = NOCURRENCY **.... ...1** 予約済み
オプション・バイト 8	XL1	**1111 1111** 将来の利用のために予約済み
オプション・バイト 9	XL1	**1... ....** ビット 1 = DATA(24)、ビット 0 = DATA(31) **.1.. ....** ビット 1 = FASTSRT、ビット 0 = NOFASTSRT **..1. ....** ビット 1 = SIZE(MAX)、ビット 0 = SIZE(nnnn) または SIZE(nnnnK) **.... .1..** ビット 1 = THREAD、ビット 0 = NOTHREAD **...1 1.11** 将来の利用のために予約済み
オプション・バイト A	XL1	**1111 1111** 将来の利用のために予約済み
オプション・バイト B	XL1	**1111 1111** 将来の利用のために予約済み

フィールド	サイズ	説明
オプション・バイト C	XL1	**1... ....** ビット 1 = TYPECHK、ビット 0 = NOTYPECHK **.1.. ....** ビット 1 = IDLGEN、ビット 0 = NOIDLGEN **..1. ....** ビット 1 = INTDATE(LILIAN)、ビット 0 = INTDATE(ANSI) **...1 ....** 将来の利用のために予約済み **.... 1...** ビット 1 = CHAR(EBCDIC)、ビット 0 = CHAR(NATIVE) (ワークステーションのみ) **.... .1..** ビット 1 = FLOAT(HEX)、ビット 0 = FLOAT(NATIVE) (ワークステーションのみ) **.... ..1.** ビット 1 = COLLSEQ(BIN) (ワークステーションのみ) **.... ...1** ビット 1 = COLLSEQ(NATIVE) (ワークステーションのみ)

フィールド	サイズ	説明
オプション・バイト D	XL1	**1... ....** ビット 1 = DLL、ビット 0 = NODLL (ホストのみ) **.1.. ....** ビット 1 = EXPORTALL、ビット 0 = NOEXPORTALL (ホストのみ) **..1. ....** ビット 1 = ANALYZE、ビット 0 = NOANALYZE **...1 ....** ビット 1 = DATEPROC、ビット 0 = NODATEPROC **.... 1...** ビット 1 = DATEPROC(FLAG)、ビット 0 = DATEPROC(NOFLAG) **.... .1..** YEARWINDOW **.... ..1.** ビット 1 = WSCLEAR、ビット 0 = NOWSCLEAR (ワークステーションのみ) **.... ...1** ビット 1 = BEOPT、ビット 0 = NOBEOPT (ワークステーションのみ)
オプション・バイト E	XL1	**1... ....** ビット 1 = DATEPROC(TRIG)、ビット 0 = DATEPROC(NOTRIG) **.1.. ....** ビット 1 = DIAGTRUNC、ビット 0 = NODIAGTRUNC **..11 1111** 将来の利用のために予約済み
オプション・バイト F	XL1	**1111 1111** 将来の利用のために予約済み
フラグ・レベル	XL1	**X'00'** フラグ(I) **X'04'** フラグ(W) **X'08'** フラグ(E) **X'0C'** フラグ(S) **X'10'** フラグ(U) **X'FF'** Noflag

フィールド	サイズ	説明
組み込み診断レベル	XL1	**X'00'** フラグ(I) **X'04'** フラグ(W) **X'08'** フラグ(E) **X'0C'** フラグ(S) **X'10'** フラグ(U) **X'FF'** Noflag
FLAGSTD (FIPS) 指定	XL1	**1... ....** 最小 **.1.. ....** 中間 **..1. ....** 高 **...1 ....** 将来の利用のために予約済み **.... 1...** レベル 1 セグメント化 **.... .1..** レベル 2 セグメント化 **.... ..1.** デバッグ **.... ...1** 廃止
フラグ用に予約済み	XL1	**1111 1111** 将来の利用のために予約済み
コンパイラー・モード	XL1	**X'00'** 無条件 Nocompile、Nocompile(I) **X'04'** Nocompile(W) **X'08'** Nocompile(E) **X'0C'** Nocompile(S) **X'FF'** コンパイル
スペース値	CL1	
3 値オプション用のデータ	XL1	**1... ....** NAME(ALIAS) 指定 **.1.. ....** NUMPROC(MIG) 指定 **..1. ....** TRUNC(BIN) 指定 **...1 1111** 将来の利用のために予約済み

フィールド	サイズ	説明
TEST(hook,sym, sep) サブオプション	XL1	**1... ....** TEST(ALL,x) **.1.. ....** TEST(NONE,x) **..1. ....** TEST(STMT,x) **...1 ....** TEST(PATH,x) **.... 1...** TEST(BLOCK,x) **.... .1..** TEST(x,SYM) **.... ..1.** ビット 1 = SEPARATE、ビット 0 = NOSEPARATE **.... ...1** TEST サブオプションのために予約済み
OUTDD 名の長さ	HL2	OUTDD 名の長さ
RWT ID の長さ	HL2	予約語テーブル ID の長さ
LVLINFO	CL4	ユーザー指定 LVLINFO データ
PGMNAME サブオプション	XL1	**1... ....** ビット 1 = PGMNAME(COMPAT) **.1.. ....** ビット 1 = PGMNAME(LONGUPPER) **..1. ....** ビット 1 = PGMNAME(LONGMIXED) **...1 1111** 将来の利用のために予約済み
記入項目インターフェース・サブオプション	XL1	**1... ....** ビット 1 = EntryInterface(System) (ワークステーションのみ) **.1.. ....** ビット 1 = EntryInterface(OptLink) (ワークステーションのみ) **..11 1111** 将来の利用のために予約済み

フィールド	サイズ	説明
CallInterface サブオプション	XL1	**1... ....** ビット 1 = CallInterface(System) (ワークステーションのみ) **.1.. ....** ビット 1 = CallInterface(OptLink) (ワークステーションのみ) **..1. ....** ビット 1 = CallInterface(Far16) (ワークステーションのみ) **...1 1111** 将来の利用のために予約済み
ARITH サブオプション	XL1	**1... ....** ビット 1 = ARITH(COMPAT) **.1.. ....** ビット 1 = ARITH(EXTEND) **11 1111** 将来の利用のために予約済み
DBCS 要件	FL4	DBCS XREF ストレージ要件
DBCS ORDPGM 長	HL2	DBCS 配列プログラムの名前の長さ
DBCS ENCTBL 長	HL2	DBCS エンコード・テーブルの名前の長さ
DBCS ORD TYPE	CL2	DBCS 配列型
予約済み	CL6	将来の利用のために予約済み
変換済み SO	CL1	変換済み SO 16 進数値
変換済み SI	CL1	変換済み SI 16 進数値
言語 ID	CL2	このフィールドには、LANGUAGE オプションからの 2 文字の省略形 (EN、UE、JA、または JP のいずれか) が保持されます。
予約済み	CL8	将来の利用のために予約済み
INEXIT 名長	HL2	SYSIN ユーザー出口名の長さ
PRTEXIT 名長	HL2	SYSPRINT ユーザー出口名の長さ
LIBEXIT 名長	HL2	'Library' ユーザー出口名の長さ
ADEXIT 名の長さ	HL2	ADATA ユーザー出口名の長さ
CURROPT	CL5	通貨オプション値
予約済み	CL1	将来の利用のために予約済み
YEARWINDOW	HL2	YEARWINDOW オプション値
CODEPAGE	HL2	CODEPAGE CCSID オプション値
予約済み	CL50	将来の利用のために予約済み
LINECNT	HL2	Linecount 値
予約済み	CL2	将来の利用のために予約済み
BUFSIZE	FL4	BUFSIZE オプション値
サイズ値	FL4	SIZE オプション値
予約済み	FL4	将来の利用のために予約済み

フィールド	サイズ	説明
フェーズ常駐ビット バイト 1	XL1	**1... ....** ビット 1 = ユーザー領域内の IGYCLIBR **.1.. ....** ビット 1 = ユーザー領域内の IGYCSCAN **..1. ....** ビット 1 = ユーザー領域内の IGYCDSCN **...1 ....** ビット 1 = ユーザー領域内の IGYCGROU **.... 1...** ビット 1 = ユーザー領域内の IGYCPSCN **.... .1..** ビット 1 = ユーザー領域内の IGYCPANA **.... ..1.** ビット 1 = ユーザー領域内の IGYCFGEN **.... ...1** ビット 1 = ユーザー領域内の IGYCPGEN
フェーズ常駐ビット バイト 2	XL1	**1... ....** ビット 1 = ユーザー領域内の IGYCOPTM **.1.. ....** ビット 1 = ユーザー領域内の IGYCLSTR **..1. ....** ビット 1 = ユーザー領域内の IGYCXREF **...1 ....** ビット 1 = ユーザー領域内の IGYCDMAP **.... 1...** ビット 1 = ユーザー領域内の IGYCASM1 **.... .1..** ビット 1 = ユーザー領域内の IGYCASM2 **.... ..1.** ビット 1 = ユーザー領域内の IGYCDIAG **.... ...1** 将来の利用のために予約済み
フェーズ常駐ビット バイト 3 ～ 4	XL2	予約済み
予約済み	CL8	将来の利用のために予約済み
OUTDD 名	CL(n)	OUTDD 名
RWT	CL(n)	予約語テーブル ID
DBCS ORDPGM	CL(n)	DBCS 配列プログラム名
DBCS ENCTBL	CL(n)	DBCS エンコード・テーブル名
INEXIT 名	CL(n)	SYSIN ユーザー出口名
PRTEXIT 名	CL(n)	SYSPRINT ユーザー出口名

フィールド	サイズ	説明
LIBEXIT 名	CL(n)	'Library' ユーザー出口名
ADEXIT 名	CL(n)	ADATA ユーザー出口名

外部シンボル・レコード - X'0020'

次の表に、外部シンボル・レコードの内容を示します。

フィールド	サイズ	説明
セクション・タイプ	XL1	**X'00'** PROGRAM-ID 名 メイン入り口点名 **X'01'** ENTRY 名 2 次入り口点名 **X'02'** 外部参照 参照済み外部入り口点 **X'04'** COBOL では N/A **X'05'** COBOL では N/A **X'06'** COBOL では N/A **X'0A'** COBOL では N/A **X'12'** 内部参照 参照済み内部サブプログラム **X'C0'** 外部 CLASS 名 OO COBOL CLASS 定義 **X'C1'** METHOD-ID 名 OO COBOL メソッド定義 **X'C6'** メソッド参照 OO COBOL メソッド定義 **X'FF'** COBOL では N/A タイプ X'12'、X'C0'、X'C1'、および X'C6X' は、COBOL のみ対応です。
フラグ	XL1	COBOL では N/A
予約済み	HL2	将来の利用のために予約済み
記号 ID	FL4	参照を含むプログラムの記号 ID (タイプ x'02' および x'12' の場合のみ)
行番号	FL4	参照を含むステートメントの行番号 (タイプ x'02' および x'12' の場合のみ)
セクション長	FL4	COBOL では N/A
LD ID	FL4	COBOL では N/A
予約済み	CL8	将来の利用のために予約済み
外部名の長さ	HL2	外部名の文字数
別名の長さ	HL2	COBOL では N/A

フィールド	サイズ	説明
外部名	CL(n)	外部名
別名セクション名	CL(n)	COBOL では N/A

構文解析ツリー・レコード - X'0024'

次の表に、構文解析ツリー・レコードの内容を示します。

フィールド	サイズ	説明
ノード番号	FL4	コンパイラーが生成するノード番号。1 から開始。
ノード・タイプ	HL2	ノードのタイプ: **001** プログラム **002** クラス **003** メソッド
		101 見出し部 **102** 環境部 **103** データ部 **104** 手続き部 **105** 終了プログラム / メソッド / クラス
		201 宣言本文 **202** 非宣言本文
		301 セクション **302** プロシージャー・セクション
		401 段落 **402** 手順段落
		501 文 **502** ファイル定義 **503** ソート・ファイル定義 **504** プログラム名 **505** プログラム属性 **508** ENVIRONMENT DIVISION 文節 **509** CLASS 属性 **510** METHOD 属性 **511** USE ステートメント

フィールド	サイズ	説明
		601 ステートメント **602** データ記述文節 **603** データ入力項目 **604** ファイル記述文節 **605** データ入力項目名 **606** データ入力項目レベル **607** EXEC 記入項目
		701 EVALUATE サブジェクト句 **702** EVALUATE WHEN 句 **703** EVALUATE WHEN OTHER 句 **704** SEARCH WHEN 句 **705** INSPECT CONVERTING 句 **706** INSPECT REPLACING 句 **707** INSPECT TALLYING 句 **708** PERFORM UNTIL 句 **709** PERFORM VARYING 句 **710** PERFORM AFTER 句 **711** ステートメント・ブロック **712** 範囲終了符号 **713** INITIALIZE REPLACING 句 **714** EXEC CICS コマンド **720** DATA DIVISION 句
		801 句 **802** ON 句 **803** NOT 句 **804** THEN 句 **805** ELSE 句 **806** 条件 **807** 式 **808** 相対索引付け **809** EXEC CICS オプション **810** 予約語 **811** INITIALIZE REPLACING カテゴリー

フィールド	サイズ	説明
		901 セクションまたは段落名 **902** ID **903** 英字名 **904** クラス名 **905** 条件名 **906** ファイル名 **907** 索引名 **908** 簡略名 **910** シンボリック文字 **911** リテラル **912** 関数 ID **913** データ名 **914** 特殊レジスター **915** プロシージャー参照 **916** 算術演算子 **917** 全プロシージャー **918** INITIALIZE リテラル (トークンなし) **919** ALL リテラルまたは形象定数 **920** キーワード・クラス名 **921** ID レベルの予約語 **922** 単項演算子 **923** 条件演算子
		1001 添え字 **1002** 参照変更
ノード・サブタイプ	HL2	ノードのサブタイプ。 セクション・タイプの場合: **0001** CONFIGURATION セクション **0002** INPUT-OUTPUT セクション **0003** FILE セクション **0004** WORKING-STORAGE セクション **0005** LINKAGE セクション **0006** LOCAL-STORAGE セクション **0007** REPOSITORY セクション

フィールド	サイズ	説明
		段落タイプの場合: **0001** PROGRAM-ID 段落 **0002** AUTHOR 段落 **0003** INSTALLATION 段落 **0004** DATE-WRITTEN 段落 **0005** SECURITY 段落 **0006** SOURCE-COMPUTER 段落 **0007** OBJECT-COMPUTER 段落 **0008** SPECIAL-NAMES 段落 **0009** FILE-CONTROL 段落 **0010** I-O-CONTROL 段落 **0011** DATE-COMPILED 段落 **0012** CLASS-ID 段落 **0013** METHOD-ID 段落 **0014** REPOSITORY 段落
		環境部文節タイプの場合: **0001** WITH DEBUGGING MODE **0002** MEMORY-SIZE **0003** SEGMENT-LIMIT **0004** CURRENCY-SIGN **0005** DECIMAL POINT **0006** PROGRAM COLLATING SEQUENCE **0007** ALPHABET **0008** SYMBOLIC-CHARACTER **0009** CLASS **0010** ENVIRONMENT NAME **0011** SELECT

フィールド	サイズ	説明
		データ記述文節タイプの場合: **0001** BLANK WHEN ZERO **0002** DATA-NAME OR FILLER **0003** JUSTIFIED **0004** OCCURS **0005** PICTURE **0006** REDEFINES **0007** RENAMES **0008** SIGN **0009** SYNCHRONIZED **0010** USAGE **0011** VALUE **0023** GLOBAL **0024** EXTERNAL

フィールド	サイズ	説明
		ファイル記述文節タイプの場合: **0001** FILE STATUS **0002** ORGANIZATION **0003** ACCESS MODE **0004** RECORD KEY **0005** ASSIGN **0006** RELATIVE KEY **0007** PASSWORD **0008** PROCESSING MODE **0009** RECORD DELIMITERr **0010** PADDING CHARACTER **0011** BLOCK CONTAINS **0012** RECORD CONTAINS **0013** LABEL RECORDS **0014** VALUE OF **0015** DATA RECORDS **0016** LINAGE **0017** ALTERNATE KEY **0018** LINES AT TOP **0019** LINES AT BOTTOM **0020** CODE-SET **0021** RECORDING MODE **0022** RESERVE **0023** GLOBAL **0024** EXTERNAL **0025** LOCK

フィールド	サイズ	説明
		ステートメント・タイプの場合: **0002** NEXT SENTENCE **0003** ACCEPT **0004** ADD **0005** ALTER **0006** CALL **0007** CANCEL **0008** CLOSE **0009** COMPUTE **0010** CONTINUE **0011** DELETE **0012** DISPLAY **0013** DIVIDE (INTO) **0113** DIVIDE (BY) **0014** ENTER **0015** ENTRY **0016** EVALUATE **0017** EXIT **0018** GO **0019** GOBACK **0020** IF **0021** INITIALIZE **0022** INSPECT

フィールド	サイズ	説明
		0023 INVOKE **0024** MERGE **0025** MOVE **0026** MULTIPLY **0027** OPEN **0028** PERFORM **0029** READ **0030** READY **0031** RELEASE **0032** RESET **0033** RETURN **0034** REWRITE **0035** SEARCH **0036** SERVICE **0037** SET **0038** SORT **0039** START **0040** STOP **0041** STRING **0042** SUBTRACT **0043** UNSTRING **0044** EXEC SQL **0144** EXEC CICS **0045** WRITE **0046** XML

フィールド	サイズ	説明
		句タイプの場合: **0001** INTO **0002** DELIMITED **0003** INITIALIZE. . .REPLACING **0004** INSPECT. . .ALL **0005** INSPECT. . .LEADING **0006** SET. . .TO **0007** SET. . .UP **0008** SET. . .DOWN **0009** PERFORM. . .TIMES **0010** DIVIDE. . .REMAINDER **0011** INSPECT. . .FIRST **0012** SEARCH. . .VARYING **0013** MORE-LABELS **0014** SEARCH ALL **0015** SEARCH. . .AT END **0016** SEARCH. . .TEST INDEX **0017** GLOBAL **0018** LABEL **0019** DEBUGGING **0020** SEQUENCE **0021** 将来の利用のために予約済み **0022** 将来の利用のために予約済み **0023** 将来の利用のために予約済み **0024** TALLYING **0025** 将来の利用のために予約済み **0026** ON SIZE ERROR **0027** ON OVERFLOW **0028** ON ERROR **0029** AT END **0030** INVALID KEY

フィールド	サイズ	説明
		0031 END-OF-PAGE **0032** USING **0033** BEFORE **0034** AFTER **0035** EXCEPTION **0036** CORRESPONDING **0037** 将来の利用のために予約済み **0038** RETURNING **0039** GIVING **0040** THROUGH **0041** KEY **0042** DELIMITER **0043** POINTER **0044** COUNT **0045** METHOD **0046** PROGRAM **0047** INPUT **0048** OUTPUT **0049** I-O **0050** EXTEND **0051** RELOAD **0052** ASCENDING **0053** DESCENDING **0054** DUPLICATES **0055** NATIVE (USAGE) **0056** INDEXED **0057** FROM **0058** FOOTING **0059** LINES AT BOTTOM **0060** LINES AT TOP

フィールド	サイズ	説明
		関数 ID タイプの場合: **0001** COS **0002** LOG **0003** MAX **0004** MIN **0005** MOD **0006** ORD **0007** REM **0008** SIN **0009** SUM **0010** TAN **0011** ACOS **0012** ASIN **0013** ATAN **0014** CHAR **0015** MEAN **0016** SQRT **0017** LOG10 **0018** RANGE **0019** LENGTH **0020** MEDIAN **0021** NUMVAL **0022** RANDOM **0023** ANNUITY **0024** INTEGER **0025** ORD-MAX **0026** ORD-MIN **0027** REVERSE **0028** MIDRANGE **0029** NUMVAL-C **0030** VARIANCE **0031** FACTORIAL **0032** LOWER-CASE

フィールド	サイズ	説明
		0033 UPPER-CASE **0034** CURRENT-DATE **0035** INTEGER-PART **0036** PRESENT-VALUE **0037** WHEN-COMPILED **0038** DAY-OF-INTEGER **0039** INTEGER-OF-DAY **0040** DATE-OF-INTEGER **0041** INTEGER-OF-DATE **0042** STANDARD-DEVIATION **0043** YEAR-TO-YYYY **0044** DAY-TO-YYYYDDD **0045** DATE-TO-YYYYMMDD **0046** UNDATE **0047** DATEVAL **0048** YEARWINDOW **0049** DISPLAY-OF **0050** NATIONAL-OF
		特殊レジスター・タイプの場合: **0001** ADDRESS OF **0002** LENGTH OF
		予約語クラス・タイプの場合: **0001** ALPHABETIC **0002** ALPHABETIC-LOWER **0003** ALPHABETIC-UPPER **0004** DBCS **0005** KANJI **0006** NUMERIC **0007** NEGATIVE **0008** POSITIVE **0009** ZERO
		予約語タイプの場合: **0001** TRUE **0002** FALSE **0003** ANY **0004** THRU

フィールド	サイズ	説明
		ID、データ名、索引名、条件名、または簡略名タイプの場合: **0001** REFERENCED **0002** CHANGED **0003** REFERENCED & CHANGED
		初期化リテラル・タイプの場合: **0001** ALPHABETIC **0002** ALPHANUMERIC **0003** NUMERIC **0004** ALPHANUMERIC-EDITED **0005** NUMERIC-EDITED **0006** DBCS/EGCS **0007** NATIONAL
		プロシージャー名タイプの場合: **0001** SECTION **0002** PARAGRAPH
		ID レベルの予約語タイプの場合: **0001** ROUNDED **0002** TRUE **0003** ON **0004** OFF **0005** SIZE **0006** DATE **0007** DAY **0008** DAY-OF-WEEK **0009** TIME **0010** WHEN-COMPILED **0011** PAGE **0012** DATE YYYYMMDD **0013** DAY YYYYDDD

フィールド	サイズ	説明
		算術演算子タイプの場合: **0001** PLUS **0002** MINUS **0003** TIMES **0004** DIVIDE **0005** DIVIDE REMAINDER **0006** EXPONENTIATE **0007** NEGATE
		比較演算子タイプの場合: **0008** LESS **0009** LESS OR EQUAL **0010** EQUAL **0011** NOT EQUAL **0012** GREATER **0013** GREATER OR EQUAL **0014** AND **0015** OR **0016** CLASS CONDITION **0017** NOT CLASS CONDITION
親ノード番号	FL4	ノードの親のノード番号。
左方兄弟ノード番号	FL4	ノードの左方兄弟のノード番号 (ある場合)。なしの場合、値はゼロ。
記号 ID	FL4	以下のタイプのいずれかのユーザー名の場合は、ノードの記号 ID。 • データ入力項目 • ID • ファイル名 • 索引名 • プロシージャー名 • 条件名 • 簡略名 段落 ID に対応するプロシージャー名を除き、この値は記号 (タイプ 42) レコード内の記号 ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。
セクション記号 ID	FL4	修飾段落名参照の場合は、ノードが含まれているセクションの記号 ID。この値は記号 (タイプ 42) レコード内のセクション ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。

フィールド	サイズ	説明
最初のトークン番号	FL4	ノードに関連付けられた最初のトークンの番号。
最後のトークン番号	FL4	ノードに関連付けられた最後のトークンの番号。
予約済み	FL4	将来の利用のために予約済み。
フラグ	CL1	ノードに関する情報は、以下を参照してください。 **X'80'** 予約済み **X'40'** 生成されたノード、トークンなし
予約済み	CL3	将来の利用のために予約済み。

トークン・レコード - X'0030'

次の表に、トークン・レコードの内容を示します。

フィールド	サイズ	説明
トークン番号	FL4	コンパイラーが生成するソース・ファイル内のトークン番号。1 から開始。ソースにはコピーブックを組み込み済みです。
トークン・コード	HL2	トークンの型 (ユーザー名、リテラル、予約語など)。 予約語の場合は、コンパイラー予約語テーブル値が使用されます。 PICTURE ストリングの場合は、特別コード 0000 が使用されます。 継続されるトークンのピース (最後のピース以外) の場合は、特殊コード 3333 が使用されます。
トークンの長さ	HL2	トークンの長さ。
トークン列	FL4	ソース行上のトークンの開始列番号。
トークン行	FL4	トークンの行番号。
フラグ	CL1	トークンに関する情報は、以下を参照してください。 **X'80'** トークンは継続 **X'40'** 継続されるトークンの最後のピース PICTURE ストリングの場合は、ソース・トークンが継続されても、1 つのトークン・レコードしか生成されません。トークン・コード 0000、最初のピースのトークン列と行、完全ストリングの長さ、継続なしフラグ・セット、完全ストリングのトークン・ストリングが与えられます。
予約済み	CL7	将来の利用のために予約済み。
トークン・テキスト	CL(n)	実トークン・ストリングです。

ソース・エラー・レコード - X'0032'

次の表に、ソース・エラー・レコードの内容を示します。

フィールド	サイズ	説明
ステートメント番号	FL4	エラーのあるステートメントのステートメント番号
エラー ID	CL16	エラー・メッセージ ID (左寄せ、ブランク埋め込み)
エラーの重大度	HL2	エラーの重大度
エラー・メッセージ長	HL2	エラー・メッセージ・テキストの長さ
行位置	XL1	FIPS メッセージに指定される行位置標識
予約済み	CL7	将来の利用のために予約済み
エラー・メッセージ	CL(n)	エラー・メッセージ・テキスト

ソース・レコード - X'0038'

次の表に、ソース・レコードの内容を示します。

フィールド	サイズ	説明
行番号	FL4	ソース・レコードのリスト行番号。
入力レコード番号	FL4	現行入力ファイル内の入力ソース・レコード番号。
1 次ファイル番号	HL2	このレコードが基本入力ファイルからの場合は、入力ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイルを参照してください)。
ライブラリー・ファイル番号	HL2	このレコードが COPY/BASIS 入力ファイルからの場合は、ライブラリー入力ファイルの割り当てシーケンス番号 (ライブラリー・レコード内のメンバー・ファイル ID n を参照してください)。
予約済み	CL8	将来の利用のために予約済み。
親レコード番号	FL4	親ソース・レコード番号。COPY/BASIS ステートメントのレコード番号となります。
親 1 次ファイル番号	HL2	このレコードの親が 1 次入力ファイルからの場合は、親ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイル n を参照してください)。
親ライブラリー割り当てファイル番号	HL2	このレコードの親が COPY/BASIS 入力ファイルからの場合は、親ライブラリー・ファイルの割り当てシーケンス番号 (ライブラリー・レコード内の COPY/BASIS メンバー・ファイル ID n を参照してください)。
予約済み	CL8	将来の利用のために予約済み。
ソース・レコードの長さ	HL2	後に続く実ソース・レコードの長さ。
予約済み	CL10	将来の利用のために予約済み。
ソース・レコード	CL(n)	

COPY REPLACING レコード - X'0039'

REPLACING アクションが実行されるごとに、1 つの COPY REPLACING タイプのレコードが出力されます。すなわち、REPLACING 句のオペランド 1 がコピーブックのテキストと一致するごとに、COPY REPLACING TEXT レコードが書き込まれます。

次の表に、COPY REPLACING レコードの内容を示します。

フィールド	サイズ	説明
REPLACED ストリングの開始行番号	FL4	REPLACING の結果であるテキストの開始のリスト行番号
REPLACED ストリングの開始列番号	FL4	REPLACING の結果であるテキストの開始のリスト列番号
REPLACED ストリングの終了行番号	FL4	REPLACING の結果であるテキストの終了のリスト行番号
REPLACED ストリングの終了列番号	FL4	REPLACING の結果であるテキストの終了のリスト列番号
オリジナル・ストリングの開始行番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル行番号
オリジナル・ストリングの開始列番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル列番号
オリジナル・ストリングの終了行番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル行番号
オリジナル・ストリングの終了列番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル列番号

記号レコード - X'0042'

次の表に、記号レコードの内容を示します。

フィールド	サイズ	説明
記号 ID	FL4	固有の記号 ID。
行番号	FL4	記号が定義または宣言されるソース・レコードのリスト行番号。
レベル	XL1	記号の真のレベル番号 (または構造内のデータ項目の相対レベル番号)。COBOL の場合、これは、01 ～ 49 の範囲、66 (名前変更項目用)、77、または 88 (条件項目用) です。
修飾標識	XL1	**X'00'** 固有の名前。修飾は不要。 **X'01'** このデータ項目は修飾が必須。名前はプログラム内で固有ではありません。このフィールドは、データ項目がレベル 01 名でないときにのみ 適用されます。

フィールド	サイズ	説明
記号タイプ	XL1	**X'68'** クラス名 (クラス ID) **X'58'** メソッド名 **X'40'** データ名 **X'20'** プロシージャー名 **X'10'** 簡略名 **X'08'** プログラム名 **X'81'** 予約済み 以下は、上記のタイプの ORed です (該当する場合)。 **X'04'** 外部 **X'02'** グローバル
記号属性	XL1	**X'01'** 数値 **X'02'** 英数字 **X'03'** グループ **X'04'** ポインター **X'05'** 索引データ項目 **X'06'** 索引名 **X'07'** 条件 **X'0F'** ファイル **X'10'** ソート・ファイル **X'17'** クラス名 (リポジトリー) **X'18'** オブジェクト・リファレンス

フィールド	サイズ	説明
文節	XL1	記号定義で指定されている文節。 数値、英数字、グループ、および索引記号の場合 **1... ....** 値 **.1.. ....** 索引付き **..1. ....** 再定義 **...1 ....** 名前変更 **.... 1...** 発生 **.... .1..** Occurs キーあり **.... ..1.** ケースにより発生 **.... ...1** 親で発生 両方のファイル・タイプの場合 **1... ....** 選択 **.1.. ....** 割り当て **..1. ....** 再実行 **...1 ....** 同一領域 **.... 1...** 同一レコード域 **.... .1..** 記録モード **.... ..1.** 予約済み **.... ...1** レコード

フィールド	サイズ	説明
		簡略名記号の場合 **01** CSP **02** C01 **03** C02 **04** C03 **05** C04 **06** C05 **07** C06 **08** C07 **09** C08 **10** C09 **11** C10 **12** C11 **13** C12 **14** S01 **15** S02 **16** S03 **17** S04 **18** S05 **19** CONSOLE **20** SYSIN/SYSIPT **22** SYSOUT/SYSLST/SYSLIST **24** SYSPUNCH/SYSPCH **26** UPSI-0 **27** UPSI-1 **28** UPSI-2 **29** UPSI-3 **30** UPSI-4 **31** UPSI-5 **32** UPSI-6 **33** UPSI-7 **34** AFP-5A

フィールド	サイズ	説明
データ・フラグ 1	XL1	両方のファイル・タイプ、数値、英数字、グループ、および索引記号の場合 **1... ....** 再定義 **.1.. ....** 名前変更 **..1. ....** 同期化 **...1 ....** 暗黙的再定義 **.... 1...** 日付フィールド **.... .1..** 暗黙の再定義 **.... ..1.** FILLER **.... ...1** レベル 77

フィールド	サイズ	説明
データ・フラグ 2	XL1	数値記号の場合 **1... ....** 2 進数 **.1.. ....** 外部浮動小数点 **..1. ....** 内部浮動小数点 **...1 ....** 圧縮 **.... 1...** ゾーン **.... .1..** 負の位取り **.... ..1.** 編集数値 **.... ...1** 将来の利用のために予約済み 英数字記号の場合 (グループ項目を含む) **1... ....** 英字 **.1.. ....** 英数字 **..1. ....** 編集英数字 **...1 ....** グループは独自の ODO オブジェクトを含む **.... 1...** DBCS 項目 **.... .1..** グループ可変長 **.... ..1.** EGCS 項目 **.... ...1** 編集 EGCS

フィールド	サイズ	説明
		両方のファイル・タイプの場合 **1... ....** レコード内の ODO のオブジェクト **.1.. ....** レコード内の ODO のサブジェクト **..1. ....** 順次アクセス **...1 ....** ランダム・アクセス **.... 1...** 動的アクセス **.... .1..** 位置指定モード **.... ..1.** レコード域 **.... ...1** 将来の利用のために予約済み 他のすべてのデータ型の場合、フィールドはゼロです。
データ・フラグ 3	XL1	両方のファイル・タイプの場合 **1... ....** すべてのレコードが同じ長さ **.1.. ....** 固定長 **..1. ....** 可変長 **...1 ....** 未定義 **.... 1...** スパン **.... .1..** ブロック **.... ..1.** 書き込みのみ適用 **.... ...1** 同一ソート・マージ領域 他のすべてのデータ型の場合、フィールドはゼロです。

フィールド	サイズ	説明
ファイル編成	XL1	両方のファイル・タイプの場合 **1... ....** QSAM **.1.. ....** ASCII **..1. ....** 標準ラベル **...1 ....** ユーザー・ラベル **.... 1...** VSAM 順次 **.... .1..** VSAM 索引 **.... ..1.** VSAM 相対 **.... ...1** 行順次 他のすべてのデータ型の場合、フィールドはゼロです。
USAGE 文節	FL1	**X'00'** USAGE IS DISPLAY **X'01'** USAGE IS COMP-1 **X'02'** USAGE IS COMP-2 **X'03'** USAGE IS PACKED-DECIMAL または USAGE IS COMP-3 **X'04'** USAGE IS BINARY、USAGE IS COMP、または USAGE IS COMP-4 **X'05'** USAGE IS DISPLAY-1 **X'06'** USAGE IS POINTER **X'07'** USAGE IS INDEX **X'08'** USAGE IS PROCEDURE-POINTER **X'09'** USAGE IS OBJECT-REFERENCE **X'0B'** NATIONAL **X'0A'** FUNCTION-POINTER
記号文節	FL1	**X'00'** SIGN 文節なし。 **X'01'** SIGN IS LEADING **X'02'** SIGN IS LEADING SEPARATE CHARACTER **X'03'** SIGN IS TRAILING **X'04'** SIGN IS TRAILING SEPARATE CHARACTER

フィールド	サイズ	説明
標識	FL1	**X'01'** JUSTIFIED 文節あり。右寄せ属性が有効。 **X'02'** BLANK WHEN ZERO 文節あり。
サイズ	FL4	このデータ項目のサイズ。この項目がストレージで占有する実バイト数。DBCS 項目の場合、この数は、文字数ではなく、バイト数で示されます。可変長項目の場合、このフィールドには、コンパイラーによってこの項目に予約されるストレージの最大サイズが反映されます。「長さ属性」とも呼ばれます。
精度	FL1	FIXED または FLOAT データ項目の精度。
スケール	FL1	FIXED データ項目のスケール因数。小数点の右方の桁数。

フィールド	サイズ	説明
ベース・ロケーター・タイプ	FL1	S/390 の場合 **01** ベース・ロケーター・ファイル **02** ベース・ロケーター作業用ストレージ **03** ベース・ロケーター・リンケージ・セクション **05** ベース・ロケーター特殊レジスター **07** 変数別索引付き **09** COMREG 特殊レジスター **10** UPSI スイッチ **13** Varloc 項目のベース・ロケーター **14** 外部データのベース・ロケーター **15** ベース・ロケーター英数字 FUNC **16** ベース・ロケーター英数字 EVAL **17** オブジェクト・データのベース・ロケーター **19** ローカル・ストレージのベース・ロケーター **20** ファクトリー・データ **21** XML-TEXT ワークステーションの場合 **01** ベース・ロケーター・ファイル **02** ベース・ロケーター・リンケージ・セクション **03** Varloc 項目のベース・ロケーター **04** 外部データのベース・ロケーター **05** オブジェクト・データのベース・ロケーター **10** ベース・ロケーター作業用ストレージ **11** ベース・ロケーター特殊レジスター **12** ベース・ロケーター英数字 FUNC **13** ベース・ロケーター英数字 EVAL **14** 変数別索引付き **16** COMREG 特殊レジスター **17** UPSI スイッチ **22** ローカル・ストレージのベース・ロケーター

フィールド	サイズ	説明
日付形式	FL1	日付形式は以下のとおりです。 **01** YY **02** YYXX **03** YYXXXX **04** YYXXX **05** YYYY **06** YYYYXX **07** YYYYXXXX **08** YYYYXXX **09** YYX **10** YYYYX **22** XXYY **23** XXXXYY **24** XXXYY **26** XXYYYY **27** XXXXYYYY **28** XXXYYYY **29** XYY **30** XYYYY
予約済み	FL4	将来の利用のために予約済み。
住所情報	FL4	ベース・ロケーター番号および変位は以下のとおりです。 **ビット 0 ～ 4** 未使用 **ビット 5 ～ 19** ベース・ロケーター (BL) 番号 **ビット 20 ～ 31** ベース・ロケーターの変位
構造変位	AL4	構造内の記号のオフセット。このオフセットは変数位置指定項目に 0 を設定します。
親変位	AL4	定義中の項目の即時親からのバイト・オフセット。
親 ID	FL4	定義中の項目の即時親からの記号 ID。
再定義 ID	FL4	この項目が再定義するデータ項目の記号 ID (該当する場合)。
開始名前変更 ID	FL4	この項目が 66 レベル項目の場合は、この項目が名前変更した開始 COBOL データ項目の記号 ID。66 レベル項目でない場合、このフィールドは 0 に設定されます。
終了名前変更 ID	FL4	この項目が 66 レベル項目の場合は、この項目が名前変更した終了 COBOL データ項目の記号 ID。66 レベル項目でない場合、このフィールドは 0 に設定されます。

フィールド	サイズ	説明
プログラム名記号 ID	FL4	この記号が定義されている、プログラムのプログラム名の ID またはクラスのクラス名の ID。
OCCURS 最小 / 段落 ID	FL4	OCCURS の最小値 段落名の PROC 名 ID
OCCURS 最大 / セクション ID	FL4	OCCURS の最大値 セクション名の PROC 名 ID
寸法	FL4	寸法の数
予約済み	CL12	将来の利用のために予約済み
値の組のカウント	HL2	値の組のカウント
記号名の長さ	HL2	記号名の文字数
データ名のピクチャー・データの長さ ファイル名の割り当て名の長さ	HL2	ピクチャー・データの文字数 — 関連する PICTURE 文節が記号にない場合は、ゼロ (PICTURE フィールドの長さ)。長さは、ソース入力にあるフィールドを表します。この長さは、レプリカの生成係数を含むピクチャー項目の拡張フィールドを表しません。ピクチャー・ストリングの最大 COBOL 長は、50 バイトです。このフィールドがゼロの場合は、PICTURE が指定されていないことを示します。 これがファイル名の場合は、外部ファイル名の文字数。割り当て名の DD 名の部分です。ファイル名および ASSIGN USING が指定されている場合は、ゼロ。
データ名の初期値の長さ CLASS-ID の外部 CLASS 名の長さ	HL2	記号値の文字数 — 記号に初期値がない場合は、ゼロ。 CLASS-ID の外部 CLASS 名の文字数。
データ名の ODO 記号名の ID ファイル名の場合は、ASSIGN 日付名の ID	FL4	データ名の場合は、ODO 記号名の ID — ODO が指定されていない場合は、ゼロ。 ファイル名の場合は、ASSIGN USING データ名の記号 ID — ASSIGN TO が指定されている場合は、ゼロ。
キー・カウント	HL2	定義されたキーの数
索引カウント	HL2	索引記号 ID のカウント。指定がない場合は、ゼロ。
記号名	CL(n)	
データ名のピクチャー・データ・ストリング ファイル名の割り当て名	CL(n)	ユーザーが入力したとおりの PICTURE 文字ストリング。文字ストリングには、全記号、括弧、およびレプリカの生成係数が含まれます。 これがファイル名の場合は、外部ファイル名。割り当て名の DD 名の部分。
索引 ID リスト	(n)FL4	各索引記号名の ID
キー	(n)XL8	このフィールドには、配列に対して指定するキーを記述するデータが含まれます。以下の 3 つのフィールドは、「キー・カウント」フィールドで指定する回数だけ繰り返されます。

フィールド	サイズ	説明
...キー・シーケンス	FL1	昇順 / 降順標識。 **X'00'** DESCENDING **X'01'** ASCENDING
...充てん文字	CL3	予約済み
...キー ID	FL4	配列内のキー・フィールドであるデータ項目の記号 ID。
データ名の初期値データ CLASS-ID の外部 CLASS 名	CL(n)	このフィールドには、この記号の INITIAL VALUE 文節で指定するデータが含まれます。以下の 4 つのサブフィールドは、「値の組カウント」フィールドのカウントだけ繰り返されます。このフィールドのデータの全長は、「初期値の長さ」フィールドに入れられます。 CLASS-ID の外部 CLASS 名。
...第 1 値の長さ	HL2	最初の値の長さ
...第 1 値の値	CL(n)	最初の値。 このフィールドには、ソース・ファイルの VALUE 文節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始 / 終了区切り文字、組み込み引用、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、その語に関連付けられた値ではなく、実際の予約語が入ります。
...第 2 値の長さ	HL2	2 番目の値の長さ — THRU 値の組でない場合は、ゼロ。
...第 2 値のデータ	CL(n)	2 番目の値。 このフィールドには、ソース・ファイルの VALUE 文節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始 / 終了区切り文字、組み込み引用、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、その語に関連付けられた値ではなく、実際の予約語が入ります。

記号相互参照レコード - X'0044'

次の表に、記号相互参照レコードの内容を示します。

フィールド	サイズ	説明
記号長	HL2	記号の長さ。
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号。 **VERB XREF の場合のみ** 動詞カウント - この動詞への参照の合計数。
参照数[1]	HL2	このレコード内の、後に続く記号への参照の数。

フィールド	サイズ	説明
相互参照タイプ	XL1	**X'01'** プログラム **X'02'** プロシージャー **X'03'** 動詞 **X'04'** 記号またはデータ名 **X'05'** メソッド **X'06'** クラス
予約済み	CL7	将来の利用のために予約済み。
記号名	CL(n)	記号。可変長。
...参照フラグ	CL1	記号またはデータ名参照の場合。 **C' '** ブランクは参照のみの意味 **C'M'** 修正参照フラグ プロシージャー型記号参照の場合 **C'A'** ALTER (プロシージャー名) **C'D'** GO TO (プロシージャー名) DEPENDING ON **C'E'** (プロシージャー名) を介した、(PERFORM) の範囲の末尾 **C'G'** GO TO (プロシージャー名) **C'P'** PERFORM (プロシージャー名) **C'T'** (ALTER) TO PROCEED TO (プロシージャー名) **C'U'** デバッグ用に使用 (プロシージャー名)
...ステートメント番号	XL4	記号または動詞が参照されているステートメント番号。

1. 参照フラグ・フィールドおよびステートメント番号フィールドは、参照フィールドで指示される回数分だけ、発生します。

   たとえば、参照フィールドの数に 10 の値が入っているとき、データ名、プロシージャー、またはプログラム式シンボルの場合には、参照フラグおよびステートメント番号ペアは 10 回発生し、動詞の場合には、ステートメント番号が 10 回発生します。

   参照数が、SYSADATA ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。レコードの共通ヘッダー・セクションに、継続フラグが設定されます。

ネストされたプログラム・レコード - X'0046'

次の表に、ネストされたプログラム・レコードの内容を示します。

フィールド	サイズ	説明
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号
ネスティング・レベル	XL1	プログラム・ネスティング・レベル

フィールド	サイズ	説明
プログラム属性	XL1	**1... ....** 初期 **.1.. ....** 共通 **..1. ....** PROCEDURE DIVISION を使用 **...1 1111** 将来の利用のために予約済み
予約済み	XL1	将来の利用のために予約済み
プログラム名長	XL1	次のフィールドの長さ
プログラム名	CL(n)	プログラム名

ライブラリー・レコード - X'0060'

次の表に、ライブラリー・レコードの内容を示します。

フィールド	サイズ	説明
メンバーの数[1]	HL2	このレコードに記述される COPY/INCLUDE コード・メンバーの数のカウント
ライブラリー名長	HL2	ライブラリー名の長さ
ライブラリー・ボリューム長	HL2	ライブラリー・ボリューム ID の長さ
連結番号	XL2	ライブラリーの連結番号
ライブラリー DD 名長	HL2	ライブラリー DD 名の長さ
予約済み	CL4	将来の利用のために予約済み
ライブラリー名	CL(n)	COPY/INCLUDE メンバーが取り出された元のライブラリーの名前
ライブラリー・ボリューム	CL(n)	ライブラリーが常駐するボリュームのボリューム識別
ライブラリー DD 名	CL(n)	このライブラリーに使用される DD 名 (または同義)
...COPY/BASIS メンバー・ファイル ID[2]	HL2	... の後の名前のライブラリー・ファイル ID
...COPY/BASIS 名長	HL2	... の後の名前の長さ
...COPY/BASIS 名	CL(n)	使用されてきた COPY/BASIS メンバーの名前

1. 合計で 10 の COPY メンバーがライブラリーから取り出された場合には、「メンバー数」フィールドに 10 が入り、「COPY/BASIS メンバー・ファイル ID」フィールド、「COPY/BASIS 名前長」フィールド、および「COPY/BASIS 名」フィールドが 10 回発生します。
2. COPY/BASIS メンバーが別のライブラリーから取り出された場合には、それぞれの一意のライブラリーごとに、ライブラリー・レコードは SYSADATA ファイルに書き込まれます。

統計レコード - X'0090'

次の表に、統計レコードの内容を示します。

フィールド	サイズ	説明
ソース・レコード	FL4	処理されたソース・レコードの数。
DATA DIVISION ステートメント	FL4	処理されたデータ部ステートメントの数。
PROCEDURE DIVISION ステートメント	FL4	処理された手続き部ステートメントの数。
コンパイル番号	HL2	バッチ・コンパイル番号。
エラーの重大度	XL1	最高のエラー・メッセージ重大度。
フラグ	XL1	**1... ....** ジョブ終了標識 **.1.. ....** クラス定義標識 **..11 1111** 将来の利用のために予約済み
EOJ 重大度	XL1	コンパイル・ジョブの最大戻りコード。
プログラム名長	XL1	プログラム名の長さ。
プログラム名	CL(n)	プログラム名。

EVENTS レコード - X'0120'

Events レコードは、ADATA ファイルに組み込まれ、コンパイラーの前のリリースとの互換性を記述します。

次の表に、EVENTS レコードの内容を示します。

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	次の EVENTS レコード・データの長さ (このハーフワードは除外)
EVENTS データ	CL(n)	以前に SYSEVENT ファイルに記述された EVENTS データ

付録 H. サンプル・プログラム

この付録には、プロダクト・テープに入っているサンプル・プログラムに関する情報を記載しています。

- プログラムの概要 (2 つのサンプルに関するプログラム図表を含む)
- 入力データの形式とサンプル
- 作成される報告書のサンプル
- プログラムの実行方法についての情報
- 例示されている言語エレメントと概念のリスト

これらのプログラムに関する疑似コードとその他のコメントは、プログラムのプロローグに収められており、プログラム・リストで入手できます。

この付録にあるサンプル・プログラムは、たくさんの言語エレメントと COBOL の概念を示しています。

- IGYTCARA は、QSAM ファイルと VSAM 索引ファイルの使用例を提供し、多くの COBOL 組み込み関数の使用方法を示します。
- IGYTCARB は、IBM 対話式システム生産性向上機能 (ISPF) を使用する例です。
- IGYTSALE は、言語環境プログラム呼び出し可能サービス機能のいくつかの使用例です。

関連概念
『IGYTCARA: バッチ・アプリケーション』
738 ページの『IGYTCARB: 対話式プログラム』
740 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』

IGYTCARA: バッチ・アプリケーション

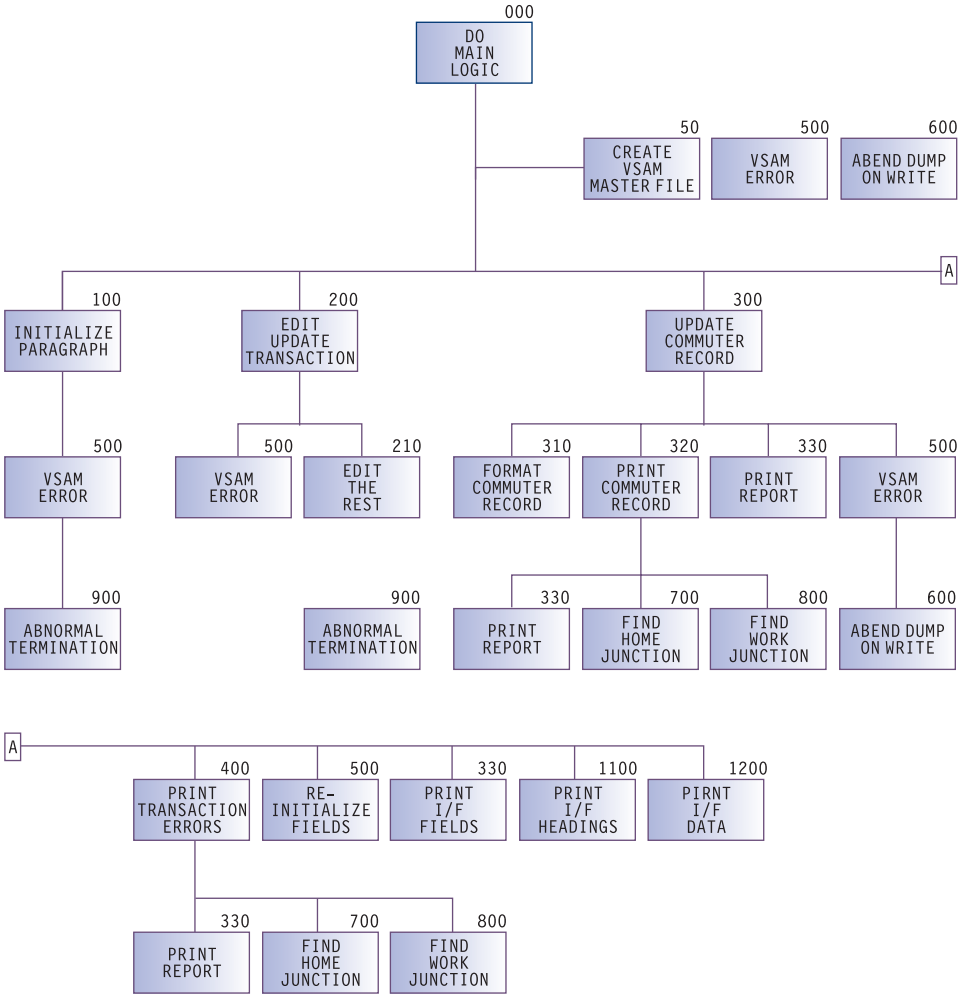
いくつかの地方事務所を持つ会社が従業員のための駐車場を作りたいとします。このバッチ・アプリケーションでは、2 つのタスクを実行しなければなりません。

- 同じ居住地域から同じ事務所まで同乗できる従業員の報告書を作成します。
- 駐車場データを更新します。
  - 新しい従業員のデータを追加します。
  - 加入している従業員の情報を変更します。
  - 従業員レコードを削除します。
  - 無効な更新要求をリストします。

このプログラムは、QSAM ファイルと VSAM 索引付きファイルを使用して、トランザクション・ファイル記入項目を妥当性検査し (順次ファイル処理)、マスター・ファイルを更新します (索引付きファイル処理)。

次の図は、アプリケーションの各部分とそれらがどのように編成されているかを示しています。

© Copyright IBM Corp. 1991, 2001



関連タスク

736 ページの『IGYTCARA の実行準備』

関連参照

『IGYTCARA の入力データ』
735 ページの『IGYTCARA によって作成される報告書』
749 ページの『例示されている言語エレメントおよび概念』

IGYTCARA の入力データ

プログラムへの入力として、会社は関連のある従業員から情報を集め、その情報をコーディングして、入力ファイルを作成しました。以下に、入力ファイルの形式の例 (フィールド相互間のスペースは、入力ファイルの場合と同様、省かれています) を示し、その後に各項目を説明しています。

```
A10111ROBERTS  AB1021  CRYSTAL  COURTSAN  FRANCISCOCA99901415555019041555513 87H1W1D
↑↑ ↑ ↑└──────┘└──────────────────────────────────────┘└──────┘└──────┘└┘└┘↑
12 3 4     5                     6                     7        8     9 10 11
```

1. トランザクション・コード
2. シフト
3. ホーム・コード
4. 業務コード
5. 通勤者名
6. 自宅の住所
7. 自宅の電話番号
8. 事務所の電話番号
9. 居住地域コード
10. 事務所の位置コード
11. 運転状況コード

以下は、入力ファイルのセクションの例です。

```
A10111ROBERTS  AB1021 CRYSTAL COURTSAN FRANCISCOCA99901415555019041555513871H1W1D
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555525892H2W2D
P48899                                          99ASDFG0005557890123ASDFGHJ    T
R10111ROBERTS  AB1221 CRYSTAL COURTSAN FRANCISCOCA99901415555019041555513871H1W1D
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555525892H2W2D
D20212KAHN     DE
D20212KAHN     DE
A20212KAHN     DE789 EMILY LANE    SAN FRANCISCOCA99921415555189041555525892H2W2D
A10111BONNICK  FD1025 FIFTH AVENUE SAN FRANCISCOCA99905415555959041555578958H8W3
A10111PETERSON SW435 THIRD AVENUE  SAN FRANCISCOCA99905415555469041555537173H3W4
   . . .
```

IGYTCARA によって作成される報告書

IGYTCARA によって作成される出力報告書の最初のページを、以下に示しています。実際の出力は、システムによって形式が多少変わることがあります。

```
1REPORT  #: IGYTCAR1                              COMMUTER FILE UPDATE LIST                             PAGE #:    1
-PROGRAM #: IGYTCAR1       RUN TIME: 01:40                                              RUN DATE: 11/30/1999
-=====================================================================================================================
      | RE-|  SHIFT    |               |                        |                |                        |STA- |
 TRANS|CORD|HOME CODE|   COMMUTER     |           HOME          |   HOME PHONE    | HOME LOCATION JUNCTION  |TUS  |  TRANS. ERROR
 CODE |TYPE|WORK CODE|     NAME       |          ADDRESS        |   WORK PHONE    | WORK LOCATION JUNCTION  |CODE |
 =====================================================================================================================
   A    NEW   1 01 11  ROBERTS    AB 1021 CRYSTAL COURT       (415) 555-0190 RODNEY/CRYSTAL            D
                                   SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
 ---------------------------------------------------------------------------------------------------------------------
 A    NEW   2 02 12  KAHN       DE 789 EMILY LANE           (415) 555-1890 COYOTE                  D
                                   SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 ---------------------------------------------------------------------------------------------------------------------
 P          4 88 99                                          (000) 555-7890 HOME CODE '  ' NOT FOUND.  T
                                                    99 ASDFG (123) ASD-FGHJ WORK CODE '  ' NOT FOUND.      TRANSACT. CODE
                                                                                                            SHIFT CODE
                                                                                                            HOME LOC. CODE
                                                                                                            WORK LOC. CODE
                                                                                                            LAST NAME
                                                                                                            INITIALS
                                                                                                            ADDRESS
                                                                                                            CITY
                                                                                                            STATE CODE
                                                                                                            ZIPCODE
                                                                                                            HOME PHONE
                                                                                                            WORK PHONE
                                                                                                            HOME JUNCTION
                                                                                                            WORK JUNCTION
                                                                                                            DRIVING STATUS
 ---------------------------------------------------------------------------------------------------------------------
 R    OLD   1 01 11  ROBERTS    AB 1021 CRYSTAL COURT       (415) 555-0190 RODNEY/CRYSTAL          D
                                   SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
        NEW   1 01 11  ROBERTS    AB 1221 CRYSTAL COURT       (415) 555-0190 RODNEY/CRYSTAL          D
                                   SAN FRANCISCO   CA 99901 (415) 555-1387 BAYFAIR PLAZA
 ---------------------------------------------------------------------------------------------------------------------
 A          2 02 12  KAHN       DE 789 EMILY LANE           (415) 555-1890 COYOTE                  D
                                   SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE      DUPLICATE REC.
 ---------------------------------------------------------------------------------------------------------------------
 D    OLD   2 02 12  KAHN       DE 789 EMILY LANE           (415) 555-1890 COYOTE                  D
                                   SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 ---------------------------------------------------------------------------------------------------------------------
 D          2 02 12  KAHN       DE                                                                      REC. NOT FOUND
 ---------------------------------------------------------------------------------------------------------------------
 A    NEW   2 02 12  KAHN       DE 789 EMILY LANE           (415) 555-1890 COYOTE                  D
                                   SAN FRANCISCO   CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE
 ---------------------------------------------------------------------------------------------------------------------
 A    NEW   1 01 11  BONNICK    FD 1025 FIFTH AVENUE        (415) 555-9590 RODNEY
                                   SAN FRANCISCO   CA 99905 (415) 555-7895 17TH FREEWAY SAN LEANDRO
 ---------------------------------------------------------------------------------------------------------------------
 A    NEW   1 01 11  PETERSON   SW 435 THIRD AVENUE         (415) 555-4690 RODNEY/THIRD AVENUE
```

IGYTCARA の実行準備

IGYTCARA プログラムに必要なすべてのファイルは、プロダクト・インストール・テープで提供されます。これらのファイル (IGYTCARA、IGYTCODE、および IGYTRANX) は、IGY.V3R1M0.SIGYSAMP データ・セットに入っています。

データ・セット名とプロシージャー名は、インストール時に変更することができます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTCARA のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

NOADV
NODYNAM
NONAME
NONUMBER
QUOTE
SEQUENCE

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

関連概念

733 ページの『IGYTCARA: バッチ・アプリケーション』

関連タスク
『IGYTCARA の実行』

関連参照
734 ページの『IGYTCARA の入力データ』
735 ページの『IGYTCARA によって作成される報告書』
749 ページの『例示されている言語エレメントおよび概念』

IGYTCARA の実行

以下で説明するプロシージャーでは、IGYTCARA プログラムのコンパイル、リンク・エディット、および実行を組み合わせて行います。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャーを変更しなければなりません。

z/OS のもとで IGYTCARA を実行するには、JCL を使用して VSAM クラスターを定義し、プログラムをコンパイルします。小文字で示されているフィールド (会計情報、ボリューム通し番号、装置名、クラスター接頭部) には、システムおよびインストール先固有の情報を挿入しなければなりません。以下の例では、名前 IGYTCAR.MASTFILE を使用しましたが、別の名前を使用しても構いません。

1. 次の JCL を使用して、必要な VSAM クラスターを作成します。

```
//CREATE   JOB (acct-info),'IGYTCAR CREATE VSAM',MSGLEVEL=(1,1),
// TIME=(0,29)
//CREATE   EXEC PGM=IDCAMS
//VOL1     DD VOL=SER=your-volume-serial,UNIT=your-unit,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
 DELETE your-prefix.IGYTCAR.MASTFILE -
     FILE(VOL1) -
     PURGE
 DEFINE CLUSTER -
    (NAME(your-prefix.IGYTCAR.MASTFILE) -
     VOLUME(your-volume-serial) -
     FILE(VOL1) -
     INDEXED -
     RECSZ(80 80) -
     KEYS(16 0) -
     CYLINDERS(2))
/*
```

   既存のクラスターを除去するために、VSAM クラスターの作成前に、DELETE が出されます。

2. 次の JCL を使用して、IGYTCARA プログラムをコンパイル、リンク・エディット、および実行します。

```
//IGYTCARA JOB (acct-info),'IGYTCAR',MSGLEVEL=(1,1),TIME=(0,29)
//TEST  EXEC IGYWCLG
//COBOL.SYSLIB  DD DSN=IGY.V3R1M0.SIGYSAMP,DISP=SHR
//COBOL.SYSIN   DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTCARA),DISP=SHR
//GO.SYSOUT     DD SYSOUT=A
//GO.COMMUTR    DD DSN=your-prefix.IGYTCAR.MASTFILE,DISP=SHR
//GO.LOCCODE    DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTCODE),DISP=SHR
//GO.UPDTRANS   DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTRANX),DISP=SHR
//GO.UPDPRINT   DD SYSOUT=A,DCB=BLKSIZE=133
//
```

関連タスク
165 ページの『第 10 章 VSAM ファイルの処理』

IGYTCARB: 対話式プログラム

IGYTCARB プログラムには、IBM 対話式システム生産性向上機能 (ISPF) を使用してダイアログ・マネージャーと Enterprise COBOL を呼び出し、画面を介して駐車場データを入力するための対話式プログラムが入っています。これは、IGYTCARA のような突き合わせプログラムまたは駐車場リストへの入力として使用できるファイルを作成します。

IGYTCARB の入力データは、IGYTCARA の場合と同じです。IGYTCARB により、ISPF パネルを介して入力ファイルに情報を追加することができます。IGYTCARB によって使用されるパネルの例を、以下に示します。

```
----------------------------- CARPOOL DATA ENTRY -----------------------------------
              New Data Entry                                      Previous Entry
Type =======> -                                    A, R, or D   A
Shift ======> -                                    1, 2, or 3   1
Home Code ==> --                                   2  Chars     01
Work Code ==> --                                   2  Chars     11
Name =======> ---------                            9  Chars     POPOWICH
Initials ===> --                                   2  Chars     AD
Address ====> ------------------                   18 Chars     134 SIXTH AVENUE
City =======> -------------                        13 Chars     SAN FREANCISCO
State ======> --                                   2  Chars     CA
Zip Code ===> -----                                5  Chars     99903
Home Phone => ----------                           10 Chars     4155553390
Work Phone => ----------                           10 Chars     4155557855
Home Jnc code > --                                 2  Chars     H3
Work Jnc Code > --                                 2  Chars     W7
Commuter Stat > -                                D, R or blank
```

関連タスク
『IGYTCARB の実行準備』

IGYTCARB の実行準備

対話式システム生産性向上機能 (ISPF) を使用して、IGYTCARB を実行します。

IGYTCARB プログラムに必要なファイルはすべて、プロダクト・インストール・テープで提供されます。これらのファイル (IGYTCARB、IGYTRANB、および IGYTPNL) は、IGY.V3R1M0.SIGYSAMP データ・セットに入っています。

データ・セット名とプロシージャー名は、インストール時に変更することができます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTCARB のソース・ファイルの中の、CBL カードの以下のオプションは変更しないでください。

```
NONUMBER
QUOTE
SEQUENCE
```

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、言語エレメントを見つけることができます。

関連概念
738 ページの『IGYTCARB: 対話式プログラム』

関連タスク
『IGYTCARB の実行』

関連参照
749 ページの『例示されている言語エレメントおよび概念』

IGYTCARB の実行

以下のプロシージャーでは、IGYTCARB プログラムのコンパイル、リンク・エディット、および実行を組み合わせて行います。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、このプロシージャーを変更しなければなりません。

z/OS のもとで IGYTCARB を実行するには、以下の手順を実行してください。

1. ISPF エディターを使用して、IGYTCARB 呼び出しを含めるよう、ISPF/PDF 基本オプション・パネル (ISR@PRIM) または他のパネルを変更します。パネル ISR@PRIM は、設置場所の PDF パネル・データ・セット (通常は ISRPLIB) に入っています。

   次の例は、2 つの指示された位置で IGYTCARB 呼び出しを含めるよう変更された ISR@PRIM パネルを示しています。パネル定義の上部にオプションを追加 (または変更) する場合は、パネルの下部にも対応する行を追加 (または変更) しなければなりません。

```
%----------------------  ISPF/PDF PRIMARY OPTION PANEL  -------------------------
%OPTION  ===>_ZCMD                                                              +
%                                                          +USERID   - &ZUSER
%   0 +ISPF PARMS  - Specify terminal and user parameters   +TIME     - &ZTIME
%   1 +BROWSE      - Display source data or output listings +TERMINAL - &ZTERM
%   2 +EDIT        - Create or change source data           +PF KEYS  - &ZKEYS
%   3 +UTILITIES   - Perform utility functions
%   4 +FOREGROUND  - Invoke language processors in foreground
%   5 +BATCH       - Submit to batch for language processing
%   6 +COMMAND     - Enter CMS command or EXEC
%   7 +DIALOG TEST - Perform dialog testing
%   8 +LM UTILITIES- Perform library management utility functions
%   C +IGYTCARB    - Run IGYTCARB UPDATE TRANSACTION PROGRAM        (1)
%   T +TUTORIAL    - Display information about ISPF/PDF
%   X +EXIT        - Terminate using console, log, and list defaults
%
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP = ISR00003
  &ZPRIM = YES         /* ALWAYS A PRIMARY OPTION MENU */
  &ZHTOP = ISR00003    /* TUTORIAL TABLE OF CONTENTS   */
  &ZHINDEX = ISR91000  /* TUTORIAL INDEX - 1ST PAGE    */
  VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
  &Z1 = TRUNC(&ZCMD,1)
  IF (&Z1 &notsym.= '.')
    &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                  0,'PANEL(ISPOPTA)'
                  1,'PGM(ISRBRO)  PARM(ISRBRO01)'
                  2,'PGM(ISREDIT) PARM(P,ISREDM01)'
                  3,'PANEL(ISRUTIL)'
                  4,'PANEL(ISRFPA)'
```

```
                  5,'PGM(ISRJB1) PARM(ISRJPA) NOCHECK'
                  6,'PGM(ISRPCC)'
                  7,'PGM(ISRYXDR) NOCHECK'
                  8,'PANEL(ISRLPRIM)'
                  C,'PGM(IGYTCARB)'                                (2)
                  T,'PGM(ISPTUTOR) PARM(ISR00000)'
                ' ',' '
                  X,'EXIT'
                  *,'?' )
   &ZTRAIL = .TRAIL
  IF (&Z1 = '.') .msg = ISPD141
)END
```

この例の **(1)** で示されているように、以下を入力して、パネルの上部に IGYTCARB を追加します。

```
%   C +IGYTCARB    - Run IGYTCARB UPDATE TRANSACTION PROGRAM
```

(2) で示されているように、以下を入力して、パネルの下部に対応する行を追加します。

```
C,'PGM(IGYTCARB)'
```

2. ISR@PRIM (または他の変更したパネル) と IGYTPNL をライブラリーに入れ、このライブラリーを ISPPLIB 連結の最初のライブラリーにします。
3. 必要に応じて、IGYTCARB 内のシーケンス行 IB2200 にコメントを加え、シーケンス行 IB2210 のコメントを省きます。z/OS のもとでは、OPEN EXTEND 動詞がサポートされます。
4. IGYTCARB をコンパイルおよびリンク・エディットし、結果として生じたロード・モジュールを LOADLIB に入れます。
5. 次のコマンドを使用して、ISPLLIB を割り振ります。

   ```
   ALLOCATE FILE(ISPLLIB) DATASET(DSN1, SYS1.COBLIB, DSN2) SHR REUSE
   ```

   ここで、*DSN1* は、ステップ 4 の LOADLIB のライブラリー名です。*DSN2* は、インストールされた ISPLLIB です。
6. 次のコマンドを使用して、入力データ・セットと出力データ・セットを割り振ります。

   ```
   ALLOCATE FILE(UPDTRANS) DA('IGY.V3R1M0.SIGYSAMP(IGYTRANB)) SHR REUSE
   ```
7. 次のコマンドを使用して、ISPPLIB を割り振ります。

   ```
   ALLOCATE FILE(ISPPLIB) DATASET(DSN3, DSN4) SHR REUSE
   ```

   ここで、*DSN3* は、変更されたパネルが含まれているライブラリーです。*DSN4* は、ISPF パネル・ライブラリーです。
8. 変更されたパネルを使用して、IGYTCARB を呼び出します。

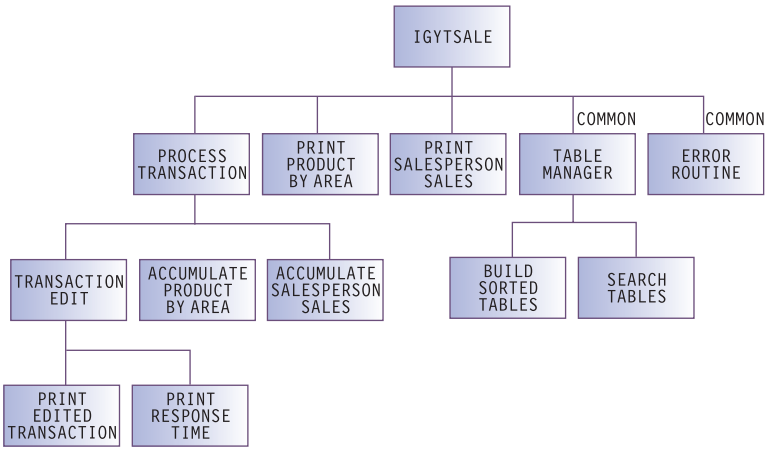
関連参照
ISPF ダイアログ開発者 ガイドとリファレンス

IGYTSALE: ネストされたプログラム・アプリケーション

スポーツ用品の卸売り会社で、製品の売上高と販売手数料を記録したいとします。このネストされたプログラム・アプリケーションでは、次のタスクを実行する必要があります。

1. 仕入れ先、顧客、および販売員の人数を記録します。このデータは、IGYTABLE と呼ばれるファイルに保管されます。
2. 有効なトランザクションとトランザクション・エラーを記録するファイルを保守します。無効なトランザクションにはすべてフラグが立てられ、結果は報告書に印刷されます。処理されるトランザクションは、IGYTRANA と呼ばれるファイルの中にあります。
3. トランザクションを処理して、地域ごとの売上高を報告します。
4. 各販売員の販売実績と手数料を記録して、その結果を報告書に印刷します。
5. 販売日と出荷日をローカル時間と UTC (世界協定時) でそれぞれ報告し、応答時間を計算します。

次の図は、アプリケーションの各部分を階層として示しています。



次の図は、各部分がどのようにネストされているかを示します。

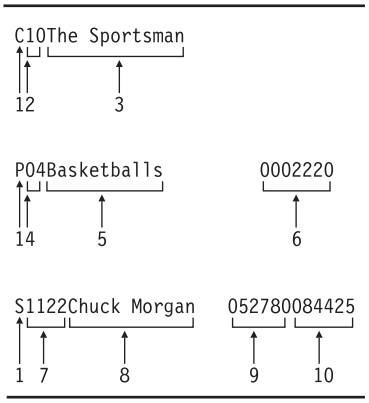
```
IGYTSALE
  Process-transactions.
    Transaction-edit is Initial.
      Print-edited-Transactions and Response-Time.
      End Program Print-edited-transactions.
    End Program transaction-edit.
    Accumulate-product-by-area.
    End Program Accumulate-product-by-area.
    Accumulate-salesperson-sales.
    End Accumulate-salesperson-sales.
  End Program Process-transactions.
  Print-product-by-area.
  End Program Print-product-by-area.
  Print-salesperson-sales.
  End Program Print-salesperson-sales.
  Table-manager is common.
    Build-sorted-tables.
    End Program Build-sorted-tables.
    Search-tables.
    End Program Search-tables.
  End Program Table-manager.
  Error-routine is Common.
  End Program Error-routine is Common.
End Program IGYTSALE.
```

関連タスク
748 ページの『IGYTSALE の実行準備』

関連参照
『IGYTSALE の入力データ』
744 ページの『IGYTSALE によって作成される報告書』
749 ページの『例示されている言語エレメントおよび概念』

IGYTSALE の入力データ

プログラムへの入力として、この流通業者は、その顧客、販売員、および製品についての情報を集め、その情報をコーディングし、入力ファイルを作成しました。IGYTABLE と呼ばれるこの入力ファイルは、トランザクション処理時に使用できるように 3 つの異なるテーブルにロードされます。以下に、このファイルの形式を示し、その後に各項目を説明しています。



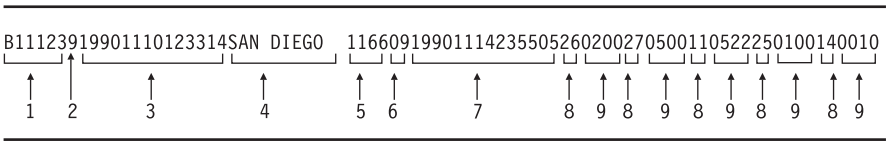
1. レコード・タイプ
2. 顧客コード
3. 顧客名
4. 製品コード
5. 製品説明
6. 製品の単価
7. 販売員の番号
8. 販売員名
9. 雇用日付
10. 手数料のレート

フィールド **1** の値 (C、P、または S) によって、入力レコードの形式が決まります。IGYTABLE のセクションの例を、以下に示します。

```
S1111Edyth Phillips 062484042327
S1122Chuck Morgan   052780084425
S1133Art Tung       022882061728
S1144Billy Jim Bob  010272121150
S1155Chris Preston  122083053377
S1166Al Willie Roz  111276100000
P01Footballs           0000620
P02Football Equipment  0032080
P03Football Uniform    0004910
```

```
P04Basketballs         0002220
P05Basketball Rim/Board0008830
P06Basketball Uniform  0004220
C01L. A. Sports
C02Gear Up
C03Play Outdoors
C04Sports 4 You
C05Sports R US
C06Stay Active
C07Sport Shop
C08Stay Sporty
C09Hot Sports
C10The Sportsman
C11Playing Ball
C12Sports Play
   . . .
```

さらに、この会社では販売トランザクションに関する情報を集めました。各トランザクションは、特定の顧客に対する個々の販売員の売上高を示しています。顧客は、各トランザクション時に 1 ～ 5 個の品目を購入することができます。トランザクション情報がコーディングされ、IGYTRANA と呼ばれる入力ファイルに入れられます。以下に、このファイルの形式を示し、その後に各項目を説明しています。



1. 販売注文番号
2. 送り状を送った品目 (発注された各品目の数)
3. 販売の日付 (年月日時分秒)
4. 販売区域
5. 販売員の番号
6. 顧客コード
7. 出荷の日付 (年月日時分秒)
8. 製品コード
9. 売上数量

フィールド **8** および **9** は、発注された各品目の数 (フィールド **2**) に応じて 1 ～ 8 回発生します。IGYTRANA のセクションの例を、以下に示します。

```
A000011199002270101 01CNTRL VALLEY1144201990022825 9999
A000041199003101005 30CNTRL VALLEY1144101990040315 0099
A000051199004182224 09CNTRL VALLEY1144121990041905 9900
A000061199005231510 10CNTRL VALLEY1144201990062325 0004
      419990324591515SAN DIEGO    11615        6020013220011052204 5100
B111144199011110033 01SAN DIEGO   1166151990111426020013220011052204 1100
A000071199011150032 05CNTRL VALLEY1133201990111712 0023
C001254199001181015 27SF BAY AREA 1133151990012016020011220025052214 5111
B111164199012011320 13SF BAY AREA 1133151990120306020010220011052204 5102
B111173199012010708 33SAN Diego   1165661990120333020o132200120522041100
B111184199012211915 44SAN DIEGO   1166141990122316020014220013052204 0300
B111194199012102115 44SAN DIEGO   1122121990121406020015220016052205 0500
B111204199012120008 16SAN DIEGO   1122041990121315020005220016052204 0100
B111214199012011315 44SAN DIEGO   1133021990120312020011220014052225 0100
B111224199011120733 12SAN DIEGO   1122101990111310020016220026052225 0100
```

```
B1112391990111012331 4SAN DIEGO   1166091990111426020027050011052225010014001 0
B1112421990131351000 0SAN DIEGO   116611        1 0200042200120a22141100
B1112541990121501251 0SAN DIEGO   1166151990121611020016220013052214111 1
B1112611990111100003 4SAN DIEGO   11331619901113260022
B1112711990111015410 0SAN DIEGO   11221219901113122000
B1112841990111017500 1SAN DIEGO   1166151990111326020013220016052104110 4
    . . .
```

IGYTSALE によって作成される報告書

次の図は、IGYTSALE 出力の例です。プログラムは以下のデータを報告書に記録します。

- トランザクション・エラー
- 製品および区域別の販売実績
- 個々の販売員の販売実績と歩合
- 販売日から、販売された製品が配送された日付までの応答時間

実際の出力は、システムによって形式が多少変わることがあります。

745 ページの『例: IGYTSALE トランザクション・エラー』
746 ページの『例: 区域および製品別による IGYTSALE 販売分析』
747 ページの『例: IGYTSALE の販売実績と歩合』
748 ページの『例: 販売から配送までの IGYTSALE 応答時間』

例: IGYTSALE トランザクション・エラー

IGYTSALE 出力の次の例では、最後の欄にトランザクション・エラーがあることを示しています。

```
Day of Report: Tuesday            C O B O L   S P O R T S       11/30/1999    03:12    Page:   1
                                          Invalid Edited Transactions
  Sales  Inv.  Sales           Sales        Sales  Cust. Product And Quantity Sold       Ship
  Order  Items Time Stamp      Area         Pers   Code                                  Date Stamp
  -----  ----- --------------  -----------  -----  ----- --------------------------      ------------
           4 19990324591515  SAN DIEGO      116     15 60200132200110522045100                Error Descriptions
                                                                                          -Sales order number is missing
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Salesperson number not numeric
                                                                                          -Product code not in product-table
                                                                                          -Date of ship time stamp is invalid
  B11117     3 19901201070833  SAN Diego     1165     66 33020o1322001205220411OO       19901203 Error Descriptions
                                                                                          -Sales area not in area-table
                                                                                          -Salesperson not in sales-per-table
                                                                                          -Customer code not in customer-table
                                                                                          -Product code not in product-table
                                                                                          -Quantity sold not numeric
  B11123     9 19901110123314  SAN DIEGO     1166     09 26020027050011052225010014001O 19901114 Error Descriptions
                                                                                          -Invoiced items is invalid
                                                                                          -Product and quantity not checked
                                                                                          -Date of ship time stamp is invalid
  B11124     2 19901313510000  SAN DIEGO     1166     11 1 020004220012Oa22141100               Error Descriptions
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Product code is invalid
                                                                                          -Date of ship time stamp is invalid
     133          81119110000  LOS ANGELES   1166     10 040112110210160321251104               Error Descriptions
                                                                                          -Sales order number is invalid
                                                                                          -Invoiced items is invalid
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Product and quantity not checked
                                                                                          -Date of ship time stamp is invalid
  C11133     4 1990111944                    1166     10 040112110210160321251104               Error Descriptions
                                                                                          -Date of sale time stamp is invalid
                                                                                          -Sales area is missing
                                                                                          -Date of ship time stamp is invalid
  C11138     4 19901117091530  LOS ANGELES   1155        1132001020102603212500O4       19901119 Error Descriptions
                                                                                          -Customer code is invalid
  DOOOO9     9 19901201222222  CNTRL COAST   115      19 141  1131221                   19901202 Error Descriptions
                                                                                          -Invoiced items is invalid
```

例: 区域および製品別による IGYTSALE 販売分析

IGYTSALE 出力の次の例では、製品および区域別による販売成績を示しています。

Day of Report: Tuesday C O B O L S P O R T S 11/30/1999 03:12 Page: 1

Sales Analysis By Product By Area

Product Codes	Areas of Sale: CNTRL COAST	CNTRL VALLEY	LOS ANGELES	NORTH COAST	SAN DIEGO	SF BAY AREA	Product Totals
Product Number 04 Basketballs							
Units Sold			433		2604	5102	8139
Unit Price			22.20		22.20	22.20	
Amount of Sale			$9,612.60		$57,808.80	$113,264.40	$180,685.80
Product Number 05 Basketball Rim/Board							
Units Sold		9900	2120	11	2700		14731
Unit Price		88.30	88.30	88.30	88.30		
Amount of Sale		$874,170.00	$187,196.00	$971.30	$238,410.00		$1,300,747.30
Product Number 06 Basketball Uniform							
Units Sold				990	200	200	1390
Unit Price				42.20	42.20	42.20	
Amount of Sale				$41,778.00	$8,440.00	$8,440.00	$58,658.00
Product Number 10 Baseball Cage							
Units Sold	45		3450	16	200	3320	7031
Unit Price	890.00		890.00	890.00	890.00	890.00	
Amount of Sale	$40,050.00		$3,070,500.00	$14,240.00	$178,000.00	$2,954,800.00	$6,257,590.00
Product Number 11 Baseball Uniform							
Units Sold	10003		3578		2922	2746	19249
Unit Price	45.70		45.70		45.70	45.70	
Amount of Sale	$457,137.10		$163,514.60		$133,535.40	$125,492.20	$879,679.30
Product Number 12 Softballs							
Units Sold	10	137	2564	13	2200	22	4946
Unit Price	1.40	1.40	1.40	1.40	1.40	1.40	
Amount of Sale	$14.00	$191.80	$3,589.60	$18.20	$3,080.00	$30.80	$6,924.40
Product Number 13 Softball Bats							
Units Sold	3227		3300	1998	5444	99	14068
Unit Price	12.60		12.60	12.60	12.60	12.60	
Amount of Sale	$40,660.20		$41,580.00	$25,174.80	$68,594.40	$1,247.40	$177,256.80
Product Number 14 Softball Gloves							
Units Sold	1155		136	3119	3833	5152	13395
Unit Price	12.00		12.00	12.00	12.00	12.00	
Amount of Sale	$13,860.00		$1,632.00	$37,428.00	$45,996.00	$61,824.00	$160,740.00
Product Number 15 Softball Cage							
Units Sold	997	99	2000		2400		5496
Unit Price	890.00	890.00	890.00		890.00		
Amount of Sale	$887,330.00	$88,110.00	$1,780,000.00		$2,136,000.00		$4,891,440.00
Product Number 16 Softball Uniform							
Units Sold	44		465	16	6165	200	6890
Unit Price	45.70		45.70	45.70	45.70	45.70	
Amount of Sale	$2,010.80		$21,250.50	$731.20	$281,740.50	$9,140.00	$314,873.00
Product Number 25 RacketBalls							
Units Sold	1001	10003	1108	8989	200	522	21823
Unit Price	0.60	0.60	0.60	0.60	0.60	0.60	
Amount of Sale	$600.60	$6,001.80	$664.80	$5,393.40	$120.00	$313.20	$13,093.80
Product Number 26 Racketball Rackets							
Units Sold	21		862	194	944	31	2052
Unit Price	12.70		12.70	12.70	12.70	12.70	
Amount of Sale	$266.70		$10,947.40	$2,463.80	$11,988.80	$393.70	$26,060.40
Total Units Sold	16503	20139	20016	15346	29812	17394 *	119210 *
Total Sales	$1,441,929.40	$968,473.60	$5,290,487.50	$128,198.70	$3,163,713.90	$3,274,945.70 *	$14,267,748.80 *

例: IGYTSALE の販売実績と歩合

IGYTSALE 出力の次の例では、販売員別の販売実績と歩合を示しています。

```
Day of Report: Tuesday           C O B O L   S P O R T S      11/30/1999    03:12    Page:   1
                                              Sales and Commission Report
  Salesperson: Billy Jim Bob
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Sports Stop              3          10117         $6,161.40       2.25%        $138.63         $746.45
  The Sportsman            1             99        $88,110.00       5.06%      $4,458.36      $10,674.52
  Sports Play              1           9900       $874,170.00       7.59%     $66,349.50     $105,905.69
                       ---------     --------    --------------                -----------     -----------
  Totals:                  5          20116       $968,441.40                 $70,946.49     $117,326.66
  Salesperson: Willie Al Roz
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Winners Club             4          13998     $1,572,775.90       7.59%    $119,373.69     $157,277.59
  Winning Sports           1           3222        $48,777.20       3.38%      $1,648.66       $4,877.72
  The Sportsman            1           1747        $27,415.50       3.38%        $926.64       $2,741.55
  Play Outdoors            1           2510        $18,579.60       3.38%        $627.99       $1,857.96
                       ---------     --------    --------------                -----------     -----------
  Totals:                  7          21477     $1,667,548.20                $122,576.98     $166,754.82
  Salesperson: Art Tung
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Sports Stop              1             23            $32.20       2.25%           $.72           $1.98
  Winners Club             2          16057     $2,274,885.00       7.59%    $172,663.77     $140,424.10
  Gear Up                  1           3022       $107,144.00       7.59%      $8,132.22       $6,613.78
  Sports Club              1             22           $279.40       2.25%          $6.28          $17.24
  Sports Fans Shop         1           1044        $20,447.30       3.38%        $691.11       $1,262.17
  L. A. Sports             1           1163       $979,198.10       7.59%     $74,321.13      $60,443.94
                       ---------     --------    --------------                -----------     -----------
  Totals:                  7          21331     $3,381,986.00                $255,815.23     $208,763.21
  Salesperson: Chuck Morgan
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Sports Play              3           7422     $3,817,245.40       7.59%    $289,728.92     $322,270.94
  Sports 4 You             1           3022       $398,335.40       7.59%     $30,233.65      $33,629.46
  The Sportsman            1           3022       $285,229.40       7.59%     $21,648.91      $24,080.49
  Sports 4 Winners         1           1100        $68,509.40       5.06%      $3,466.57       $5,783.90
  Sports Club              1          12027     $1,324,256.10       7.59%    $100,511.03     $111,800.32
                       ---------     --------    --------------                -----------     -----------
  Totals:                  7          26593     $5,893,575.70                $445,589.08     $497,565.11
  Salesperson: Chris Preston
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Playing Ball             1           5535     $1,939,219.10       7.59%    $147,186.72     $103,509.69
  Play Sports              1           5675       $225,130.80       7.59%     $17,087.42      $12,016.80
  Winners Club             1            631        $14,069.70       2.25%        $316.56         $750.99
  The Jock Shop            1           2332        $28,716.60       3.38%        $970.62       $1,532.80
                       ---------     --------    --------------                -----------     -----------
  Totals:                  4          14173     $2,207,136.20                $165,561.32     $117,810.28
  Salesperson: Edyth Phillips
  Customers:           Number of     Products      Total for       Discount     Discount       Commission
                        Orders       Ordered         Order         (if any)      Amount          Earned
  -------------------- ---------     --------    --------------    --------    -----------     -----------
  Sports Play              2           3575        $92,409.90       5.06%      $4,675.94       $3,911.43
  Winning Sports           1          11945        $56,651.40       5.06%      $2,866.56       $2,397.88
                       ---------     --------    --------------                -----------     -----------
  Totals:                  3          15520       $149,061.30                  $7,542.50       $6,309.31
    Grand Totals:            33        119210    $14,267,748.80              $1,068,031.60   $1,114,529.39
```

例: 販売から配送までの IGYTSALE 応答時間

IGYTSALE 出力の次の例では、米国での販売日から、販売された製品がヨーロッパに向けて配送された日付までの応答時間を示しています。

```
Day of Report: Tuesday            C O B O L   S P O R T S       11/30/1999    03:12    Page:   1
                                     Response Time from USA Sale to European Ship
                     Prod    Units    Sale Date/Time(PST)     Ship Date     Ship  Response Time
                     Code    Sold     YYYYMMDD  HHMMSS        YYYYMMDD      Day     Days
                     ----    -----    --------  ------        --------      ----  -------------
                     25      9999     19900226  010101        19900228      WED         .95
                     15        99     19900310  100530        19900403      TUE       23.57
                     05      9900     19900418  222409        19900419      THU         .06
                     25         4     19900523  151010        19900623      SAT       30.36
                     04      1100     19901110  003301        19901114      WED        2.97
                     12        23     19901114  003205        19901117      SAT        1.97
                     14      5111     19900118  101527        19900120      SAT        1.57
                     04      5102     19901201  132013        19901203      MON        1.44
                     04       300     19901221  191544        19901223      SUN        1.19
                     05       500     19901210  211544        19901214      FRI        3.11
                     04       100     19901211  000816        19901213      THU         .99
                     25       100     19901201  131544        19901203      MON        1.44
                     25       100     19901112  073312        19901113      TUE         .68
                     14      1111     19901214  012510        19901216      SUN         .94
                     26        22     19901110  000034        19901113      TUE        1.99
                     12      2000     19901110  154100        19901113      TUE        2.34
                     04      1104     19901110  175001        19901113      TUE        2.25
                     12       114     19901229  115522        19901230      SUN         .50
                     15      2000     19901110  190113        19901114      WED        3.20
                     10      1440     19901112  001500        19901115      THU        1.98
                     25      1104     19901118  120101        19901119      MON         .49
                     25         4     19901118  110030        19901119      MON         .54
                     12       144     19901114  010510        19901119      MON        3.95
                     14       112     19901119  010101        19901122      THU        1.95
                     26       321     19901117  173945        19901119      MON        1.26
                     13      1221     19901101  135133        19901102      FRI         .42
                     10        22     19901029  210000        19901030      TUE         .12
                     14        35     19901130  160500        19901201      SAT         .32
                     11      9005     19901211  050505        19901212      WED         .78
                     06       990     19900511  214409        19900515      TUE        3.09
                     13      1998     19900712  150100        19900716      MON        3.37
                     26        31     19901010  185559        19901011      THU         .21
                     14        30     19901210  195500        19901212      WED        1.17
```

IGYTSALE の実行準備

IGYTSALE プログラムに必要なファイルはすべて、プロダクト・インストール・テープで提供されます。これらのファイル (IGYTSALE、 IGYTCRC、 IGYTPRC、 IGYTSRC、 IGYTABLE、 IGYTRANA) は、 IGY.V3R1M0.SIGYSAMP データ・セットに入っています。

データ・セット名とプロシージャー名は、インストール時に変更することができます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTSALE のソース・ファイルの中の、CBL カードの以下のオプションは変更しないでください。

```
LIB
NONUMBER
SEQUENCE
NONUMBER
QUOTE
```

これらのオプションを適用すると、プログラムは診断メッセージを発行しなくなる可能性があります。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

IGYTSALE を実行すると、次のメッセージが SYSOUT データ・セットに印刷されます。

```
Program IGYTSALE Begins
There were 00041 records processed in this program
Program IGYTSALE Normal End
```

関連概念
740 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』

関連タスク
『IGYTSALE の実行』

関連参照
742 ページの『IGYTSALE の入力データ』
744 ページの『IGYTSALE によって作成される報告書』
『例示されている言語エレメントおよび概念』

IGYTSALE の実行

以下のプロシージャーでは、IGYTSALE プログラムのコンパイル、リンク・エディット、および実行を組み合わせて行います。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャーを変更しなければなりません。

次の JCL を使用して、IGYTSALE プログラムをコンパイル、リンク・エディット、および実行します。小文字で示されているフィールド (会計情報) には、システムおよびインストール先固有の情報を挿入しなければなりません。

```
//IGYTSALE JOB (acct-info),'IGYTSALE',MSGLEVEL=(1,1),TIME=(0,29)
//TEST  EXEC IGYWCLG
//COBOL.SYSLIB  DD DSN=IGY.V3R1M0.SIGYSAMP,DISP=SHR
//COBOL.SYSIN   DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTSALE),DISP=SHR
//GO.SYSOUT     DD SYSOUT=A
//GO.IGYTABLE   DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTABLE),DISP=SHR
//GO.IGYTRANS   DD DSN=IGY.V3R1M0.SIGYSAMP(IGYTRANA),DISP=SHR
//GO.IGYPRINT   DD SYSOUT=A,DCB=BLKSIZE=133
//GO.IGYPRT2    DD SYSOUT=A,DCB=BLKSIZE=133
//
```

例示されている言語エレメントおよび概念

サンプル・プログラムに適正な言語エレメントを見つけるためには、シーケンス・ストリングの欄でプログラムの省略形を探します。

サンプル・プログラム	省略形
IGYTCARA	IA
IGYTCARB	IB
IGYTSALE	IS

次の表に、サンプル・プログラムで例示されている言語エレメントとプログラミングの概念を示します。言語エレメントまたは概念を説明し、シーケンス・ストリングを示しています。シーケンス・ストリングとは、ソース・ファイルのシーケンス・フィールドに現れる特殊な文字ストリングのことを言います。このストリングを検索引き数として使用して、リスト内のエレメントを見つけることができます。

言語エレメントまたは概念	シーケンス・ストリング
ACCEPT . . . FROM DAY-OF-WEEK	IS0900

言語エレメントまたは概念	シーケンス・ストリング
ACCEPT . . . FROM DATE	IS0901
ACCEPT . . . FROM TIME	IS0902
ADD . . . TO	IS4550
AFTER ADVANCING	IS2700
AFTER PAGE	IS2600
ALL	IS4200
ASSIGN	IS1101
AUTHOR	IA0040
CALL	IS0800
呼び出し可能サービス (言語環境プログラム) CEEDATM - 日付および時刻出力の形式設定 CEEDCOD - サービス呼び出しからのフィードバック・コード検査 CEEGMTO - 地方時からの UTC オフセット CEELOCT - ローカル日付および時刻 CEESECS - タイム・スタンプの秒への変換	 IS0875, IS2575 IS0905 IS0904 IS0850 IS2350, IS2550
ファイルの CLOSE	IS1900
交換可能なコンマ、セミコロン、およびスペース	IS3500, IS3600
ネストされたプログラムの COMMON ステートメント	IS4600
複合 OCCURS DEPENDING ON	IS0700, IS3700
COMPUTE	IS4501
COMPUTE ROUNDED	IS4500
CONFIGURATION SECTION	IA0970
CONFIGURATION SECTION (オプショナル)	IS0200
CONTINUE ステートメント	IA5310, IA5380
COPY ステートメント	IS0500
DATA DIVISION (オプショナル)	IS5100
データ妥当性検査	IA5130-6190
Do-until (PERFORM . . . TEST AFTER)	IA4900-5010, IA7690-7770
Do-while (PERFORM . . . TEST BEFORE)	IS1660
END-ADD	IS2900
END-COMPUTE	IS4510
END-EVALUATE	IA6590, IS2450
END-IF	IS1680
END-MULTIPLY	IS3100
END-PERFORM	IS1700
END PROGRAM	IA9990
END-READ	IS1800
END-SEARCH	IS3400
ENVIRONMENT DIVISION (オプショナル)	IS0200
エラー処理、プログラムの終了	IA4620, IA5080, IA7800-7980
EVALUATE ステートメント	IA6270-6590

言語エレメントまたは概念	シーケンス・ストリング
EVALUATE . . . ALSO	IS2400
EXIT PROGRAM は段落内で唯一のステートメントである必要はない	IS2000
べき乗計算	IS4500
EXTERNAL 文節	IS1200
順次ファイルの FILE-CONTROL 記入項目	IA1190-1300
VSAM 索引付きファイルの FILE-CONTROL 記入項目	IA1070-1180
FILE SECTION (オプショナル)	IS0200
FILE STATUS コード検査	IA4600-4630, IA4760-4790
FILLER (オプショナル)	IS0400
フラグ、レベル 88、定義	IA1730-1800, IA2440-2480, IA2710
フラグ、レベル 88、テスト	IA4430, IA5200-5250
FLOATING POINT	IS4400
GLOBAL ステートメント	IS0300
ネストされたプログラムの INITIAL ステートメント	IS2300
INITIALIZE	IS2500
DATA DIVISION 内でのテーブルの初期化	IA2920-4260
インライン PERFORM ステートメント	IA4410-4520
I-O-CONTROL 段落 (オプショナル)	IS0200
INPUT-OUTPUT SECTION (オプショナル)	IS0200
組み込み関数 CURRENT-DATE MAX MEAN MEDIAN MIN STANDARD-DEVIATION UPPER-CASE VARIANCE WHEN-COMPILED	 IA9005 IA9235 IA9215 IA9220 IA9240 IA9230 IA9015 IA9225 IA9000
IS (すべての文節でオプショナル)	IS0700
LABEL RECORDS (オプショナル)	IS1150
LINKAGE SECTION	IS4900
指標と添え字の混合	IS3500
簡略名	IA1000
MOVE	IS0903
MOVE CORRESPONDING ステートメント	IA4810, IA4830
MULTIPLY . . . GIVING	IS3000
END-IF を使用した、ネストされた IF ステートメント	IA5460-5830
ネストされたプログラム	IS1000
NEXT SENTENCE	IS4300
NOT AT END	IS1600

言語エレメントまたは概念	シーケンス・ストリング
NULL	IS4800
OBJECT-COMPUTER (オプショナル)	IS0200
OCCURS DEPENDING ON	IS0710
ODO は項目を受け取るのに最大長を使用する	IS1550
OPEN EXTEND	IB2210
OPEN INPUT	IS1400
OPEN OUTPUT	IS1500
ORGANIZATION (オプショナル)	IS1100
ページ替え	IA7180-7210
簡略化された条件の中の括弧	IS4850
PERFORM . . . WITH TEST AFTER (Do-until)	IA4900-5010, IA7690-7770
PERFORM . . . WITH TEST BEFORE (Do-while)	IS1660
PERFORM . . . UNTIL	IS5000
PERFORM . . . VARYING ステートメント	IA7690-7770
POINTER 関数	IS4700
印刷ファイル FD 記入項目	IA1570-1620
印刷報告書	IA7100-7360
PROCEDURE DIVISION . . . USING	IB1320-IB1650
PROGRAM-ID (30 文字まで許可される)	IS0120
READ . . . INTO . . . AT END	IS1550
REDEFINES ステートメント	IA1940, IA2060, IA2890, IA3320
参照変更	IS2425
関係演算子 <= (より小さい、または等しい)	IS4400
関係演算子 >= (より大きい、または等しい)	IS2425
相対添え字付け	IS4000
REPLACE	IS4100
SEARCH ステートメント	IS3300
SELECT	IS1100
順序番号には任意の文字を入れることができる	IA, IB, IS
順次ファイル処理	IA4480-4510, IA4840-4870
PERFORM を使用した、順次テーブル探索	IA7690-7770
SEARCH を使用した、順次テーブル探索	IA5270-5320, IA5340-5390
SET INDEX	IS3200
SET . . . TO TRUE ステートメント	IA4390, IA4500, IA4860, IA4980
SOURCE-COMPUTER (オプショナル)	IS0200
SPECIAL-NAMES 段落 (オプショナル)	IS0200
STRING ステートメント	IA6950, IA7050
小文字のサポート	IS0100
TALLY	IS1650
ネストされたプログラムの TITLE ステートメント	IS0100
通勤者のレコードの更新	IA6200-6610

言語エレメントまたは概念	シーケンス・ストリング
トランザクション作業値スペースの更新	IB0790-IB1000
USAGE BINARY	IS1300
USAGE PACKED-DECIMAL	IS1301
妥当性検査エレメント	IB0810, IB0860, IB1000
OCCURS が指定された VALUE	IS0600
VALUE SPACE (S)	IS0601
VALUE ZERO (S) (ES)	IS0600
可変長テーブルの制御変数	IA5100
可変長テーブルの定義	IA2090-2210
可変長テーブルのロード	IA4840-4990
VSAM 索引付きファイルのキー定義	IA1170
VSAM 戻りコードの表示	IA7800-7900
WORKING-STORAGE SECTION	IS0250

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 製以外の製品と組み合わせた場合、その操作の評価と検証については、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む。) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権の許諾については、下記の宛先に書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適切な記述や誤植など) は、本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

商標

以下は、IBM Corporation の商標です。

AIX
CICS
CICS/ESA
COBOL/370
DB2

© Copyright IBM Corp. 1991, 2001

DFSMS
DFSORT
IBM
IMS
IMS/ESA
Language Environment
MVS
OpenEdition
OS/390
RACF
System/390
VisualAge
VM/ESA
z/OS
z/VM

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Action Media、LANDesk、MMX、Pentium、および ProShare は、Intel Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

用語集

この用語集に記載されている用語は、COBOL における意味に従って定義されています。これらの用語は、他の言語での意味と同じ場合もあれば、異なる場合もあります。

この用語集には、以下の資料からの用語および定義が記載されています。

- *American National Standard Programming Language COBOL*、*ANSI X3.23-1985* (copyright 1985 American National Standards Institute, Inc。) (ISO 1989: 1985)、X3.23a-1989 (ISO 1989/修正 1) および X3.23b-1993 (ISO1989/修正 2) によって改訂されました。
- *American National Standard Dictionary for Information Systems* ANSI X3.172-1990、copyright 1990 by the American National Standards Institute (ANSI)。American National Standards Institute, 11 West 42nd Street, New York, New York 10036 から、コピーを購入することができます。

米国標準規格の定義の前にはアスタリスク (*) を付けています。

［ア行］

アクセス・モード (* access mode). ファイル内でレコードが操作される方式。

遊びバイト (slack bytes). ある種の数字項目を正しく境界合わせするために、データ項目相互間またはレコード相互間に挿入されるバイト。遊びバイトには、意味のあるデータは含まれない。コンパイラーによって挿入される場合もあれば、プログラマーが挿入しなければならない場合もある。SYNCHRONIZED 文節は、正しい境界合わせが必要な場合に遊びバイトを挿入するようにコンパイラーに指示する。レコード間遊びバイトは、プログラマーが挿入する。

暗黙の範囲終了符号 (* implicit scope terminator). 終了していないステートメントが何か前にある場合、その有効範囲を区切る区切りピリオド。または、前にある句の中に含まれるステートメントが何かある場合、そのステートメントの有効範囲の終わりをそれが現れることによって示すステートメントの句。

異常終了 (abend). プログラムの異常終了。

移植性 (portability). あるアプリケーション・プラットフォームから別のアプリケーション・プラットフォームに、ソース・プログラムに比較的わずかな変更を加えるだけでアプリケーション・プログラムを移行できる能力。

インスタンス・データ (instance data). (1) オブジェクトの状態を定義するデータ。クラスによって導入されるインスタンス・データは、クラス定義の OBJECT 段落の DATA DIVISION の WORKING-STORAGE SECTION に定義される。オブジェクトの状態には、基本クラスが導入した、現行クラスによって継承されているインスタンス変数の状態も含まれる。インスタンス・データの個々のコピーは、各オブジェクト・インスタンスごとに作成される。(2) Visual Builder では、特定のオブジェクトに属する private データで、それ以外のすべてのオブジェクトによる直接アクセスから隠されている。データ・メンバーは、定義クラスとそのサブクラスのメソッドによってしかアクセスできない。

隠蔽 (hide). サブクラスのファクトリーまたは静的メソッド (親クラスから継承された) を再定義すること。

インライン (inline). プログラムでは、ルーチン、サブルーチン、または他のプログラムに分岐することなく、順次に実行される命令。

ウィンドウ化西暦年 (windowed year). 2 桁の年号だけから構成される日付フィールド。この 2 桁の年号は、世紀ウィンドウを使用して解釈することができる。たとえば、05 は 2005 と解釈することができる。「*世紀ウィンドウ (century window)*」も参照。「*拡張西暦年 (expanded year)*」と対比。

ウィンドウ化日付フィールド (windowed date field). ウィンドウ操作年号 (2 桁の年号) を含む日付フィールド。「*日付フィールド (date field)*」と「*ウィンドウ化西暦年 (windowed year)*」も参照。

埋め込み文字 (* padding character). 物理レコードの中の未使用の文字位置を埋めるために使用される英数字。

英字 (* alphabetic character). 文字または空白文字。

© Copyright IBM Corp. 1991, 2001

英字名 (* alphabet-name). ENVIRONMENT DIVISION の SPECIAL-NAMES 段落のユーザー定義語であり、特定の文字セットまたは照合シーケンス (あるいはその両方) に名前を割り当てるもの。

英数字 (* alphanumeric character). コンピューターの文字セットの任意の文字。

英数字関数 (* alphanumeric function). コンピューターの有する文字セットの文字を 1 つまたは複数個使って作られたストリングで構成された値を持つ関数。

英数字項目 (alphanumeric item). 記号 X を含む、USAGE DISPLAY および PICTURE 文字ストリングで記述されたデータ項目。

英数字編集文字 (alphanumeric-edited character). B、0 (ゼロ)、または / (スラッシュ) を少なくとも 1 つ含む英数字文字ストリングの中の文字。

英数字リテラル (alphanumeric literal). 次のセットからの開始区切り文字を有するリテラル。'、"、X'、X"、Z'、または Z"。この文字ストリングには、コンピューターの有する文字セットの任意の文字を含めることができる。

エレメント (テキスト・エレメント) (element (text element)). 1 つのデータ項目または動詞の記述などのようなテキスト・ストリングの 1 つの論理単位で、その前にエレメント・タイプを識別する固有のコードが付けられたもの。

エンクレーブ (enclave). 言語環境プログラム製品のもとで実行される場合、エンクレーブは実行単位に類似している。エンクレーブは、z/OS および CMS では LINK によって、CMS では CMSCALL によって、および C の system() 関数を使用すれば、他のエンクレーブを作成することができる。

エンコード・ユニット (encoding unit). 「*文字エンコード・ユニット (character encoding unit)*」を参照。

演算、操作 (operation). オブジェクトに関して要求できるサービス。

演算符号 (* operational sign). 値が正であるか負であるかを示すために数字データ項目または数字リテラルに付けられる代数符号。

オーバーフロー条件 (overflow condition). ある演算結果の一部が意図した記憶単位の容量を超えた場合に発生する条件。

オーバーライド (override). サブクラスのインスタンス・メソッド (親クラスから継承された) を再定義すること。

オープン・データベース・コネクティビティー (Open Database Connectivity (ODBC)). さまざまなデータベースおよびファイル・システム内にあるデータへのアクセスを提供する、アプリケーション・プログラミング・インターフェース (API) の仕様。

オープン・モード (* open mode). ファイルに対する OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。個々のオープン・モードは、OPEN ステートメントの中で、INPUT、OUTPUT、I-O、または EXTEND のいずれかとして指定する。

オブジェクト (object). (1) 状態 (そのデータ値) と操作 (そのメソッド) を持つエンティティー。オブジェクトは状態と動作をカプセル化する手段である。クラス内の各オブジェクトは、そのクラスの 1 つのインスタンスであると言われる。(2) Visual Builder では、ユーザーがタスクを実行するために作業を行う対象のコンピューター表現。オブジェクトは、テキストまたはアイコンとして表示することができる。

オブジェクト時 (* object time). オブジェクト・プログラムが実行される時点。「*実行時 (run time)*」と同義。

オブジェクト指向プログラミング (object-oriented programming). カプセル化および継承の概念に基づいたプログラミング・アプローチ。プロシージャー型プログラミング技法とは異なり、オブジェクト指向プログラミングでは、何かが達成される方法ではなく、問題を含むデータ・オブジェクトとその操作方法に重点を置く。

オブジェクト・インスタンス (object instance). 「*オブジェクト (object)*」を参照。

オブジェクト・コード (object code). コンパイラーまたはアセンブラーからの出力。それ自体が実行可能なマシン・コードか、またはその種のコードの作成を目的として処理するのに適する。

オブジェクト・コンピューター記入項目 (* object computer entry). ENVIRONMENT DIVISION の OBJECT-COMPUTER 段落内の記入項目。この記入項目には、オブジェクト・プログラムが実行されるコンピューター環境を記述する文節が入っている。

オブジェクト・デック (object deck). オブジェクト・プログラムのうち、リンケージ・エディターへの入力と

して適切な部分。「オブジェクト・モジュール (*object module*)」および「テキスト・デック (*text deck*)」と同義。

オブジェクト・プログラム (* object program). 問題を解決するためにデータと相互に作用することを目的とする実行可能なマシン言語命令とその他の要素の集合またはグループ。本書の場合で言えば、オブジェクト・プログラムは、普通、ソース・プログラムを COBOL コンパイラーでコンパイルした結果として得られるマシン言語であり、あいまいになる恐れがない場合には、*オブジェクト・プログラム* という用語の代わりに *プログラム* という語だけが使用される。

オブジェクト・モジュール (object module). 「*オブジェクト・デック (object deck)*」または「*テキスト・デック (text deck)*」の同義語。

オブジェクト・リファレンス (object reference). クラスのインスタンスを識別する値。クラスが指定されなかった場合、オブジェクト・リファレンスは一般的なものとなり、任意のクラスのインスタンスに適用できる。

オプショナル・ファイル (* optional file). オブジェクト・プログラムが実行されるたびに必ずしも存在しなくてもよいものとして宣言されているファイル。オブジェクト・プログラムが実行されると、そのファイルの存在の有無が調べられる。

オプショナル・ワード (* optional word). 言語を読みやすくする目的でのみ特定の形式で含められる予約語。このようなワードが現れる形式をソース・プログラムで使用する場合、そのワードの有無はユーザーが選択できる。

オペランド (* operand). (1) オペランドの一般的な定義は、「操作対象のコンポーネント」である。(2) 本書の目的に沿った言い方をすれば、ステートメントや記入項目の形式中に現れる小文字または日本語で書かれた語 (または語群) はオペランドと見なされ、そのオペランドによって指示されたデータに対して暗黙の参照を行う。

[カ行]

ガーベッジ・コレクション (garbage collection). Java ランタイム・システムによる、参照されなくなったオブジェクトのメモリーの自動解放。

下位終了 (* low-order end). 文字ストリングの右端の文字。

階層ファイル・システム (hierarchical file system). 階層構造で編成されたファイルとディレクトリーの集合であり、UNIX システム・サービスを使用してアクセスできる。

外部 10 進数項目 (external decimal item). 数値を表す形式であり、数字は右端バイトのビット 4 〜 7、符号はビット 0 〜 3 に入る。その他のバイトのビット 0 〜 3 には 1 (16 進数の F) が入る。たとえば、+123 の 10 進値は 1111 0001 1111 0010 1111 0011 と表される。「*ゾーン 10 進数項目 (zoned decimal item)*」と同義。

外部スイッチ (* external switch). インプリメントする人によって定義され指名されたハードウェアまたはソフトウェア装置であり、2 つの代替状態のいずれかが存在していることを示す。

外部データ (* external data). プログラムの中で外部データ項目および外部ファイル結合子として記述されるデータ。

外部データ項目 (* external data item). 実行単位の 1 つまたは複数のプログラムにおいて外部レコードの一部として記述されるデータ項目であり、その項目が記述されている任意のプログラムから参照することができる。

外部データ・レコード (* external data record). 実行単位の 1 つまたは複数のプログラムにおいて記述される論理レコードであり、そのデータ項目は、それらが記述されている任意のプログラムから参照できる。

外部ファイル結合子 (* external file connector). 実行単位の 1 つまたは複数のオブジェクト・プログラムにアクセス可能なファイル結合子。

外部浮動小数点項目 (external floating-point item). 実数が一対の特殊数値によって表される数字を表記するための形式。浮動小数点表記では、実数は固定小数点部分 (最初の数表示) であり、暗黙的な浮動小数点の底を指数 (2 番目の数表示) で累乗することで獲得される値である。たとえば、0.0001234 の浮動小数点表記は 0.1234 -3 である。この場合、0.1234 は小数部であり、-3 は指数である。

外部プログラム (external program). 最外部のプログラムであり、ネストされていないプログラム。

カウンター (* counter). 数値または数表現を保管するために使用されるデータ項目で、これらの数値は別の数値によって増分または減分されたり、0 または任意の正の値または負の値に変更、あるいはリセットされたりする。

拡張西暦年 (expanded year). 4 桁の年号だけからなる日付フィールド。この値には世紀が含まれる。たとえば、1998。「*ウィンドウ化西暦年 (windowed year)*」と対比。

拡張日付フィールド (expanded date field). 拡張 (4 桁) 年号を含む日付フィールド。「*日付フィールド (date field)*」と「*拡張西暦年 (expanded year)*」も参照。

拡張部分 (extensions). ANSI 規格で記述されるもの以外で、IBM コンパイラーがサポートする特定の COBOL 構文とセマンティクス。

拡張プログラム間通信機能 (advanced program-to-programcommunication (APPC)). ワークステーションとホストの間の通信プロトコル。リモート開発用の CICS、IMS、および SMARTdataUtilities が APPC 通信プロトコルを使用する。

拡張モード (* extend mode). ファイルに対する EXTEND 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

型式化オブジェクト・リファレンス (typed object reference). 指定されたクラスまたはそのサブクラスのオブジェクトだけを参照できるデータ名。

カタログ式プロシージャー (cataloged procedure). プロシージャー・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに置かれた一連のジョブ制御ステートメント。カタログ式プロシージャーを使用すると、JCL をコーディングする時間を節約して、エラーを減らすことができる。

カプセル化 (encapsulation). オブジェクト指向プログラミングでは、オブジェクトに関する固有の詳細を隠すために使用される技法。オブジェクトは、基礎構造を露出しなくても、データの照会と操作を行うインターフェースを提供する。「*情報隠蔽 (information hiding)*」と同義。

可変位置グループ (* variably located group). 同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに従属するわけではない)。

可変位置項目 (* variably located item). 同じレコード内の可変長テーブルに続くデータ項目 (可変長テーブルに従属するわけではない)。

可変オカレンス・データ項目 (* variable-occurrence data item). 可変オカレンス・データ項目とは、反復される回数が可変であるテーブル・エレメントを言う。そのような項目は、そのデータ記述記入項目内に OCCURS DEPENDING ON 文節を持っているか、またはそのような項目に従属していなければならない。

可変長レコード (* variable-length record). ファイル記述記入項目またはソート / マージ・ファイル記述記入項目が、文字位置の数が可変であるレコードを許容しているファイルに関連付けられているレコード。

環境文節 (* environment clause). ENVIRONMENT DIVISION 記入項目の一部として現れる文節。

環境変数 (environment variable). オペレーティング・システムを実行に移す方法、およびオペレーティング・システムに装置を認識させる方法を記述した一連の変数。

環境名 (environment-name). IBM が指定する名前であり、システム論理装置、プリンターおよびカード穿孔装置の制御文字、報告書コード、またはプログラム・スイッチ、あるいはそれらの組み合わせを識別する。環境名が ENVIRONMENT DIVISION の簡略名と関連付けられている場合は、その簡略名を、置換が有効な任意の形式で置き換えることができる。

関係 (* relation). 「*関係演算子 (relational operator)*」または「*比較条件 (relation condition)*」を参照。

関係演算子 (* relational operator). 予約語、比較文字、連続する予約語のグループ、または連続する予約語と比較文字のグループ。いずれも比較条件の構文中に使われる。許容されている演算子とその意味は次のとおり。

文字	意味
IS GREATER THAN	より大きい
IS >	より大きい
IS NOT GREATER THAN	より大きくない (以下)
IS NOT >	より大きくない (以下)
IS LESS THAN	より小さい
IS <	より小さい
IS NOT LESS THAN	より小さくない (以上)
IS NOT <	より小さくない (以上)
IS EQUAL TO	に等しい
IS =	に等しい
IS NOT EQUAL TO	に等しくない
IS NOT =	に等しくない
IS GREATER THAN OR EQUAL TO	より大きいか等しい (以上)
IS >=	より大きいか等しい (以上)
IS LESS THAN OR EQUAL TO	より小さいか等しい (以下)
IS <=	より小さいか等しい (以下)

関数 ID (* function-identifier). 関数を参照する文字ストリングと区切り文字の構文的に正しい組み合わせ。関数で表現されるデータ項目は、引き数 (ある場合) が指定された関数名によって一意に識別される。関数 ID には参照修飾子を入れることができる。英数字関数を参照する関数 ID は、ID を指定する一般形式で、任意の場所に指定できるが、特定の制約を受ける。整数関数または数字関数を参照する関数 ID は、算術式を指定する一般形式で、任意の場所で参照できる。

関数 (* function). ステートメントの実行中に参照された時点で決定される値を持つ、一時的なデータ項目。

関数ポインター・データ項目 (function-pointer data item). 入り口点へのポインターを格納することができるデータ項目。USAGE IS FUNCTION-POINTER 文節で定義されるデータ項目に、関数入り口点のアドレスが含められる。一般的に、C および Java プログラムと通信するために使用される。

関数名 (function-name). 必要な引き数を指定した呼び出しによって、関数の値が決定されるメカニズムを指名するワード。

簡略複合比較条件 (* abbreviated combined relation condition). 連続した一連の比較条件において、共通サブジェクトの明示的な省略、または共通サブジェクトと共通関係演算子の明示的な省略によって生じる複合条件。

簡略名 (* mnemonic-name). ENVIRONMENT DIVISION において、指定されたインプリメントする人の名前に関連したユーザー定義語。

キー (* key). レコードの位置を識別するデータ項目、またはデータの順序付けを識別するための一連のデータ項目。

キーワード (* keyword). 予約語または関数名で、その語の現れる形式がソース・プログラムの中で使用されるときには必須である。

疑似テキスト (* pseudotext). ソース・プログラムまたは COBOL ライブラリーの中にあって、その境界が疑似テキスト区切り文字によって区切られたテキストワード、コメント行、または分離符号のスペースからなる列 (その疑似テキスト区切り文字を含まない)。

疑似テキスト区切り文字 (* pseudotext delimiter). 疑似テキストを区切るために使用される 2 つの連続した等号文字 (==)。

記入項目 (* entry). 区切りピリオドで終了させられる連続する文節の記述セットであり、COBOL プログラムの IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、または DATA DIVISION に書き込まれる。

記入項目のオブジェクト (* object of entry). COBOL プログラムの DATA DIVISION 記入項目内の一連のオペランドと予約語であり、その記入項目のサブジェクトの直後に続く。

記入項目のサブジェクト (* subject of entry). DATA DIVISION の記入項目内において、レベル標識またはレベル番号の直後に現れるオペランドまたは予約語。

基本項目 (* elementary item). それ以上論理的に分割されないものとして記述されるデータ項目。

基本レコード・キー (* prime record key). 索引付きファイルのレコードを固有なものとして識別する内容を持つキー。

共通プログラム (* common program). 別のプログラムに直接的に含まれているにもかかわらず、その別のプログラムに直接的または間接的に含まれている任意のプログラムから呼び出すことができるプログラム。

記録モード (recording mode). ファイル内の論理レコードの形式。記録モードは、F (固定長)、V (可変長)、S (スパン)、または U (不定形式) とすることができる。

キロバイト (kilobyte (KB)). 1 キロバイトは 1024 バイトに相当する。

句 (* phrase). 句とは、COBOL プロシージャー・ステートメントまたは COBOL 文節の一部を形成する連続的な COBOL 文字ストリングを、1 つまたは複数個並べた集合。

区切りコンマ (* separator comma). 文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのコンマ (,)。

区切りセミコロン (* separator semicolon). 文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのセミコロン (;)。

区切り点 (breakpoint). 通常は命令によって指定されるコンピューター・プログラムの場所であり、プログラムの実行は外部からの介入またはモニター・プログラムによって割り込まれる場合がある。

区切りピリオド (* separator period). ピリオド (.) 文字ストリングを区切るために使用されるもの。

区切り文字 (* delimiter). 文字ストリングの終わりを示し、またその文字ストリングを後続の文字ストリングから切り離すために使われる、1 つの文字または一連の

連続する文字。区切り文字は、これを使って区切られる文字ストリングの一部ではない。

句読文字 (* punctuation character). 以下に示す集合に属する文字。

文字	意味
,	コンマ
;	セミコロン
:	コロン
.	ピリオド (終止符)
“	引用符
(	左括弧
)	右括弧
	スペース
=	等号

国別データ項目 (national data item). 暗黙的に、または明示的に USAGE NATIONAL として記述されるデータ項目。

国別文字 (national character). UTF-16 のエンコード・ユニット、または UTF-16 の 1 つのエンコード・ユニットで表す文字。

組み込み関数 (built-in function). 「*組み込み関数 (intrinsic function)* 」を参照。

組み込み関数 (intrinsic function). よく使用される算術関数のような事前定義関数で、組み込み関数参照によって呼び出される。

クライアント (client). オブジェクト指向プログラミングにおいて、クラス内の 1 つまたは複数のメソッドからサービスを要求するプログラムまたはメソッド。

クラス (* class). 0 個、1 個、または複数のオブジェクトに対して、共通の動作およびインプリメンテーションを定義するエンティティー。同一のインプリメンテーションを共用するオブジェクトは、同じクラスのオブジェクトと見なされる。クラスは階層として定義でき、あるクラスを別のクラスから継承することができる。

クラス階層 (class hierarchy). オブジェクト・クラス間の関係を示すツリーのような構造。最上部に 1 つのクラスが置かれ、その下に 1 つまたは複数のクラスの層が置かれる。「*継承階層 (inheritance hierarchy)* 」と同義。

クラス識別記入項目 (* class identification entry). IDENTIFICATION DIVISION の CLASS-ID 段落内の記入項目であり、クラス名を指定する文節と、選択した属性をクラス定義に割り当てる文節を含む。

クラス条件 (* class condition). 項目の内容がすべて英字であるか、すべて数字であるか、あるいはクラス名の定義にリストされている文字だけで構成されている命題 (それに対する真理値を判別できる)。

クラス定義 (* class definition). クラスを定義する COBOL ソース単位。

クラス名 (オブジェクト指向) (class-name (object-oriented)). オブジェクト指向 COBOL クラス定義の名前。

クラス名 (データの) (* class-name (of data)). ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語であり、真理値を定義できる命題に名前を割り当てる。データ項目の内容は、クラス名の定義にリストされている文字だけで構成される。

クラス・オブジェクト (class object). クラスを表すランタイム・オブジェクト。

クラス・ライブラリー (class library). クラスの集まり。

グループ項目 (* group item). 従属データ項目で構成されるデータ項目。

グローバル参照 (global reference). メソッドの有効範囲外にあるオブジェクトの参照。

グローバル名 (* global name). 1 つのプログラムにおいてのみ宣言されるが、そのプログラム、またはそのプログラム内に含まれている任意のプログラムから参照できる名前。グローバル名には、条件名、データ名、ファイル名、レコード名、報告書名、および特殊レジスターがある。

ケース構造 (case structure). 結果として生じた多数のアクションの中から選択を行うために、一連の条件をテストするプログラム処理ロジック。

形式 (* format). 一連のデータの特殊な配列。

継承 (* inheritance). 1 つまたは複数のクラスのインプリメンテーションを、別のクラスを基にして使用するメカニズム。サブクラスは、1 つまたは複数のスーパークラスから継承する。継承するクラスの定義は、継承されるクラスに準拠する。Enterprise COBOL は *多重継承* をサポートしない。サブクラスは、必ず 1 つの即時スーパークラスを有する。

継承階層 (inheritance hierarchy). 「*クラス階層 (class hierarchy)*」を参照。

形象定数 (* figurative constant). ある予約語を使用して参照されるコンパイラー生成の値。

桁位置 (* digit position). 1 つの桁を保管するために必要な物理ストレージの量。この大きさは、データ項目を定義するデータ記述記入項目に指定された用途によって異なる。

結果 ID (* resultant identifier). 算術演算の結果が収められるユーザー定義のデータ項目。

現行ボリューム・ポインター (* current volume pointer). 順次ファイルの現行のボリュームを指す概念的エンティティー。

現行レコード (* current record). ファイル処理では、ファイルに関連したレコード域に使用できるレコード。

言語環境プログラム準拠 (Language Environment-conforming). コンパイラー製品 (Enterprise COBOL、COBOL (OS/390 および VM 版)、COBOL (MVS および VM 版)、C/C++ for MVS and VM、PL/I for MVS and VM) の特性であり、言語環境プログラム形式に準拠したオブジェクト・コードを生成する。

言語間通信 (interlanguage communication (ILC)). 異なる言語で通信を行うように作成されたルーチンの機能。ILC サポートにより、アプリケーション開発者は、各種言語で書かれたコンポーネント・ルーチンからアプリケーションを簡単に構築することができる。

言語名 (* language-name). 特定のプログラミング言語を指定するシステム名。

コード化文字セット (coded character set). 文字セットを設定し、その文字セットの文字とコード化表現との間の関係を設定する明確な規則の集まり。コード化文字セットの例として、ASCII もしくは EBCDIC コード・ページで、または Unicode 対応の UTF-16 エンコード・スキームで表す文字セットがある。

コード化文字セット ID (coded character set identifier (CCSID)). エンコード・スキーム ID、文字セット ID、コード・ページ ID、およびコード化された図形文字表示を固有に識別するその他の情報を含む 16 ビットの数字 (5 桁の数)。

コード・ページ (code page). すべてのコード・ポイントに図形文字および制御機能の意味を割り当てるもの。たとえば、あるコード・ページでは、8 ビット・コードに対して 256 コード・ポイントに文字と意味を割り当て、別のコード・ページでは、7 ビット・コードに対して 128 コード・ポイントに文字と意味を割り当てることができる。ワークステーション上の英語の IBM コード・ページは IBM-850 で、ホストは IBM-1047 である。「*コード化文字セット (coded character set)*」を参照。

コード・ポイント (code point). コード化文字セット (コード・ページ) に定義する固有のビット・パターン。コード・ポイントは、コード化文字セットで図形文字に割り当てられている。

高位終了 (* high-order end). 文字ストリングの左端の文字。

降順キー (* descending key). 値に基づくキーであり、そのデータが、キーの最高値からキーの最低値まで、データ項目比較規則に従って順序付けられている。

構造化プログラミング (structured programming). コンピューター・プログラムの編成およびコーディングに関する技法の 1 つ。この技法では、プログラムはセグメントの階層から構成され、各セグメントは 1 つの入り口点と 1 つの出口点を有する。制御は、構造の下方へと渡され、階層内のより上位レベルへの無条件分岐は行われない。

構文 (syntax). (1) 意味や、解釈および使用の方法に依存しない、文字同士または文字のグループ同士の間の関係。(2) 言語における表現の構造。(3) 言語構造を支配する規則。(4) 記号相互の関係。(5) 文の構築についての規則。

互換性のある日付フィールド (compatible date field). *互換* という用語の意味は、日付フィールドに適用される場合は、それがどの COBOL 部で使用されるかによって異なる。

- DATA DIVISION (データ部)
  2 つの日付フィールドに互換性があると言われるのは、それらが同一の USAGE を持ち、以下の少なくとも 1 つの条件を満たす場合である。
  - 同じ日付形式を持つ。
  - ともにウィンドウ化日付フィールドであり、一方がウィンドウ化西暦年 DATE FORMAT YY だけで構成される。
  - ともに拡張日付フィールドであり、一方が拡張西暦年 DATE FORMAT YYYY だけで構成される。
  - 一方が DATE FORMAT YYXXXX で、他方が YYXX の形式である。
  - 一方が DATE FORMAT YYYYXXXX で、他方が YYYYXX の形式である。

  ウィンドウ化日付フィールドは、拡張日付グループであるデータ項目に従属することができる。2 つの日付フィールドに互換性があると言われるのは、従属日付

フィールドが USAGE DISPLAY を持ち、グループ拡張日付フィールドの開始より 2 バイト後で始まっており、2 つのフィールドが以下の少なくとも 1 つの条件を満たしている場合である。

- 従属日付フィールドの DATE FORMAT パターンが、グループ日付フィールドの DATE FORMAT パターンと同じ数の X を持つ。
- 従属日付フィールドが DATE FORMAT YY を持つ。
- グループ日付フィールドが DATE FORMAT YYYYXXXX を持ち、従属日付フィールドが DATE FORMAT YYXX を持つ。

- PROCEDURE DIVISION (手続き部)
  2 つの日付フィールドに互換性があると言われるのは、それらが年号部分以外は同じ日付形式を持つ場合である。年号部分は、ウィンドウ操作年号でも拡張年号でも可能。たとえば、DATE FORMAT YYXXX という形式のウィンドウ化日付フィールドは、以下のものと互換性がある。
  - DATE FORMAT YYXXX という形式の別のウィンドウ化日付フィールド。
  - DATE FORMAT YYYYXXX という形式の拡張日付フィールド。

固定小数点数 (fixed-point number). PICTURE 文節で定義される数値データ項目であり、オプションの符号の位置、その中に含まれる桁数、およびオプションの小数点の位置を指定するもの。その形式は 2 進数、パック 10 進数、または外部 10 進数。

固定長レコード (* fixed-length record). ファイル記述記入項目またはソート・マージ記述記入項目で、すべてのレコードに同じ数の文字位置が入っていなければならないファイルに関連付けられたレコード。

固定ファイル属性 (* fixed file attributes). ファイルに関する情報であり、ファイルの作成時に設定され、それ以降はファイルが存在する限り変更できない。これらの属性には、ファイル (順次、相対、または索引付き) の編成、基本レコード・キー、代替レコード・キー、コード・セット、最小および最大レコード・サイズ、レコード・タイプ (固定または可変)、索引付きファイルのキーの照合シーケンス、ブロック化因数、埋め込み文字、およびレコード区切り文字がある。

コピーブック (copybook). 一連のコードが含まれたファイルまたはライブラリー・メンバーであり、コンパイル時に COPY ステートメントを使用してソース・プログラムに組み込まれる。このファイルは、ユーザーが作成することもできるし、COBOL または別のプロダクトから提供されることもある。「*コピー・ファイル (copy file)*」と同義。

コメント記入項目 (* comment-entry).
IDENTIFICATION DIVISION 内の記入項目であり、コンピューターの文字セットから任意の文字を組み合わせることができる。

コメント行 (* comment line). 行の標識区域ではアスタリスク (*)、およびその行の区域 A および B ではコンピューターの文字セットの任意の文字で表されるソース・プログラム行。コメント行は、プログラムの文書化にのみ役立つ。行の標識区域では斜線 (/)、そしてその行の区域 A および B ではコンピューター文字セットの任意の文字で表される特殊形式のコメント行があると、コメントの印刷前に改ページが行われる。

固有照合シーケンス (* native collating sequence).
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した照合シーケンス。

固有文字セット (* native character set).
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した文字セット。

コンパイラー (compiler). 高水準言語で作成されたプログラムをマシン言語のオブジェクト・プログラムに変換するプログラム。

コンパイラー指示ステートメント (compiler-directing statement). コンパイラー指示動詞で始まるステートメント (または指示) で、これによって、コンパイラーはコンパイル時に特定のアクションを取る。コンパイラー指示は、COBOL ソース・プログラムに入っている。したがって、複数のコンパイラー指示ステートメントを使えば、ソース・プログラムにおいて指示のさまざまなサブオプションを指定できる。

コンパイル (* compile). (1) 高水準言語で表現されるプログラムを、中間言語、アセンブリー言語、またはコンピューター言語で表現されるプログラムに変換すること。(2) あるプログラミング言語で書かれたコンピューター・プログラムから、プログラムの全体的なロジック構造を利用することによって、または 1 つの記号ステートメントから 2 つ以上のコンピューター命令を作り出すことによって、またはアセンブラーの機能のようにこれら両方を使うことによって、マシン言語プログラムを生成すること。

コンパイル時間 (* compile time). COBOL ソース・プログラムが、COBOL コンパイラーによって COBOL オブジェクト・プログラムに変換される時間。

コンパイル用コンピューター記入項目 (* source computer entry). ENVIRONMENT DIVISION の SOURCE-COMPUTER 段落内の記入項目であり、ソース・プログラムがコンパイルされるコンピューター環境を記述する文節が入っている。

コンピューター名 (* computer-name). プログラムがコンパイルまたは実行されるコンピューターを識別するシステム名。

コンポーネント (component). (1) 関連ファイルからなる機能グループ化。(2) Visual Builder では、そのターゲットが EXE ではなく DLL として作成される GUI プロジェクト。

[サ行]

再帰 (recursion). 自分自身を呼び出すプログラム、または、自分で呼び出したプログラムによって直接あるいは間接に呼び出されるプログラム。

再帰可能 (recursively capable). PROGRAM-ID ステートメントで RECURSIVE 属性が指定されていれば、プログラムは再帰可能である (再帰的に呼び出すことができる)。

最後に使われた状態 (last-used state). 内部値がプログラム終了時と同じままで、初期値にリセットされない、プログラムの状態を言う。

再使用可能環境 (reusable environment). 再使用可能環境とは、事前初期設定用の古い COBOL インターフェース (ILBOSTP0 および IGZERRE 関数、および RTEREUS ランタイム・オプション)、または言語環境プログラム・インターフェース CEEPIPI のいずれかを使用して、アセンブラー・プログラムをメインプログラムとして設定した場合に作成される環境である。

再入可能 (reentrant). プログラムまたはルーチンの属性。この属性によって、ロード・モジュールの 1 つのコピーをコピーのユーザーが共用できる。

索引付きデータ名 (indexed data-name). データ名とそれに続く 1 つまたは複数の (括弧で囲まれた) 索引名で構成される ID。

索引付きファイル (* indexed file). 索引編成のファイル。

索引データ項目 (* index data item). 索引名および関連する値をインストール先指定の形式で保管できるデータ項目。

索引編成 (* indexed organization). 各レコードが、そのレコード内の 1 つまたは複数のキーの値で識別される、永続論理ファイル構造。

サブクラス (* subclass). 別のクラスから継承するクラス。継承関係にある 2 つのクラスをまとめて考える場合、継承する側、つまり継承先のクラスをサブクラスといい、継承される側、つまり継承元のクラスをスーパークラスという。

サブプログラム (* subprogram). 「*呼び出されるプログラム (called program)*」を参照。

算術演算 (* arithmetic operation). ある算術ステートメントが実行されることにより、またはある算術式が計算されることにより生じるプロセスで、そこで与えられている引き数に対して数学的に正しい解が求められる。

算術演算子 (* arithmetic operator). 次のセットに属する、1 つの文字、または固定の 2 文字の組み合わせ。

文字	意味
+	加算
-	減算
*	乗算
/	除算
**	べき乗計算

算術式 (* arithmetic expression). 数字基本項目の ID、数値リテラル、そのような ID とリテラルを算術演算子で区切ったもの、2 つの算術式を算術演算子で区切ったもの、または算術式を括弧で囲んだもの。

算術ステートメント (* arithmetic statement). 算術演算を実行させるステートメント。算術ステートメントには、ADD、COMPUTE、DIVIDE、MULTIPLY、および SUBTRACT の各ステートメントがある。

参照キー (* key of reference). 索引付きファイルの中のレコードをアクセスするために現在使用されている基本キーまたは代替キー。

参照形式 (* reference format). COBOL ソース・プログラムを記述するに際して標準的な方式を提供する形式。

参照修飾子 (* reference-modifier). 固有のデータ項目を定義する文字ストリングと区切り文字の構文的に正しい組み合わせ。区切り用の左括弧分離符号、左端の文字位置、分離符号のコロン、長さ (オプション)、および区切り用の右括弧分離符号を含む。

参照変更 (reference modification). 新しい英数字データ項目を定義する方法。左端の文字と別の英数字データ項目の左端の文字までの相対的な長さを指定することによって行う。

式 (* expression). 算術式または条件式。

シグニチャー (signature). (1) ある操作とそのパラメーターの名前。(2) あるメソッドの名前とその仮パラメーターの数と型。

指数 (exponent). 別の数 (底) を乗ずる指数を示す数。正の指数は乗算を示し、負の指数は除算を示し、小数の指数は数量の根を示す。COBOL では、指数式は記号 ** の後に指数を付けて表す。

システム名 (* system-name). オペレーティング環境と連絡し合うために使用される COBOL ワード。

事前初期設定 (preinitialization). プログラム (特に COBOL 以外のプログラム) からの複数の呼び出しに備えて、COBOL ランタイム環境を初期設定すること。この環境は、明示的に終了されるまで終了されない。

実行可能ファイル (executable file (EXE)). 取るべき操作または処置を実行するプログラムまたはコマンドが含まれたファイル。

実行時 (execution time). 「*実行時 (run time)*」を参照。

実行時 (* run time). オブジェクト・プログラムが実行される時点。「*オブジェクト時 (object time)*」と同義。

実行時環境 (execution-time environment). 「*ランタイム環境 (run-time environment)*」を参照。

実行単位 (* run unit). 1 つの独立型オブジェクト・プログラム、または COBOL の CALL ステートメントを介して相互作用する複数のオブジェクト・プログラムであり、実行時に 1 つのエンティティーとして機能するもの。

実行不成功 (* unsuccessful execution). あるステートメントの実行を試みたが、そのステートメントで指定したすべての演算の実行結果が得られないこと。あるステートメントの実行不成功は、そのステートメントによって参照されるデータには影響を及ぼさないが、状況表示には影響を与える可能性がある。

実際の小数点 (* actual decimal point). 10 進小数点文字のピリオド (.) を使用した物理表現。または、データ項目内の 10 進小数点のコンマ (,)。

指標 (* index). その内容が、テーブル内の特定エレメントの識別を表す、コンピューターのストレージ域またはレジスター。

指標付け (indexing). 指標名を使用しての*添え字付け*と同義。

指標名 (* index-name). 特定のテーブルに関係付けられた指標を指名するユーザー定義語。

修飾子 (* qualifier). (1) レベル標識と関連付けられるデータ名または名前であり、参照の際に、別のデータ名 (修飾子に従属する項目の名前) と一緒に、または条件名と一緒に使用される。(2) セクション名であり、参照の際に、そのセクションで指定された段落名と一緒に使用される。(3) ライブラリー名であり、参照の際に、そのライブラリーに関連付けられたテキスト名と一緒に使用される。

修飾データ名 (* qualified data-name). データ名と、その後に連結語の OF または IN とデータ名修飾子を続けたものが 1 セットまたは複数セット続いて構成される ID。

終了クラス・マーカー (end class marker). ワードの組み合わせに区切りのピリオドが続いたもので、COBOL クラス定義の終わりを示す。終了クラス・マーカーは次のとおり。

```
END CLASS class-name.
```

終了プログラム・マーカー (* end program marker). ワードの組み合わせに区切りのピリオドが続いたもので、COBOL ソース・プログラムの終わりを示す。終了プログラム・マーカーは次のとおり。

```
END PROGRAM program-name.
```

終了メソッド・マーカー (end method marker). ワードの組み合わせに区切りのピリオドが続いたもので、COBOL メソッド定義の終わりを示す。終了メソッド・マーカーは次のとおり。

```
END METHOD method-name.
```

出力ファイル (* output file). 出力モードまたは拡張モードのいずれかでオープンされるファイル。

出力プロシージャー (* output procedure). SORT ステートメントの実行中にソート機能が完了した後で制御が渡されるステートメントの集合、または MERGE ステートメントの実行中に、要求があればマージ機能がマージ済みの順序になっているレコードのうち次のレコードを選択できるようになった後で制御が渡されるステートメントの集合。

出力モード (* output mode). ファイルに対する OUTPUT または EXTEND 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

順次アクセス (* sequential access). ファイル内のレコードの順序によって規定されている、論理レコードの連続した前後関係順に、論理レコードをファイルから取り出したり、ファイルに書き込んだりするアクセス・モード。

順次ファイル (* sequential file). 順次編成のファイル。

順次編成 (* sequential organization). レコードがファイルに書き込まれるときに確定されたレコードの前後関係によって識別されるような永続的な論理ファイル構造。

順序構造 (sequence structure). 一連のステートメントが、順序どおりに実行されるプログラムの処理ロジック。

条件 (condition). 言語環境プログラムによって使用可能にされる、あるいは認識される例外。したがって、ユーザー条件処理ルーチンと言語条件処理ルーチンの活動化に適している。アプリケーションの通常のプログラム式フローの代替となるもの。条件は、ハードウェアまたはオペレーティング・システムによって検出され、その結果、割り込みが起こる。このほかにも、条件は言語特定の生成コードまたは言語ライブラリー・コードによっても検出できる。

条件 (* condition). 真理値を判別できる、実行時のプログラムの状況。条件がこれらの言語仕様または一般形式の '条件' (*condition-1, condition-2,...*) に関連して現れる場合は、次のいずれかである。オプションとして括弧で囲まれた単純条件からなる条件式、あるいは、単純条件、論理演算子、および括弧の構文的に正しい組み合わせ (真理値を判別できる) からなる複合条件。「*単純条件 (simple condition)*」、「*複合条件 (complex condition)*」、「*単純否定条件 (negated simple condition)*」、「*複合条件 (combined condition)*」、および「*複合否定条件 (negated combined condition)*」も参照。

条件式 (* conditional expression). EVALUATE、IF、PERFORM、または SEARCH ステートメントの中で指定される単純条件または複合条件。「*単純条件 (simple condition)*」と 「*複合条件 (complex condition)*」も参照。

条件ステートメント (* conditional statement). 条件の真理値を判別することと、オブジェクト・プログラムの次の処理がこの真理値によって決まることを指定するステートメント。

条件付き句 (* conditional phrase). ある条件ステートメントが実行された結果得られる条件の真理値の判定に基づいてとられるべき処置を指定する句。

条件変数 (* conditional variable). 1 つまたは複数の値に 1 つの条件名が割り当てられているデータ項目。

条件名 (* condition-name). 条件変数が想定できる値のサブセットに名前を割り当てるユーザー定義語。または、インプリメントする人が定義したスイッチまたは装置の状況に割り当てられるユーザー定義語。条件名が一般形式で使用されるとき、それは、条件名を修飾名と添え字と一緒に (参照の固有性の要件に応じて) 構文的に正しく組み合わせたものから構成される、固有のデータ項目参照を表す。

条件名条件 (* condition-name condition). 真理値を判別できる命題で、かつ、条件変数の値が、その条件変数と関連する条件名に属する一連の値のメンバーである命題。

照合シーケンス (* collating sequence). コンピューターに受け入れられる文字がソート、マージ、比較を行うため、また索引付きファイルを順次処理するために順序付けられているシーケンス。

昇順キー (* ascending key). データ項目を比較する際の規則に一致するように、最低のキー値から始めて最高のキー値へとデータを順序付けている値に即したキー。

初期状態 (* initial state). 実行単位で最初に呼び出されるときのプログラムの状態。

初期プログラム (* initial program). プログラムが実行単位で呼び出されるたびに初期状態に設定されるプログラム。

ジョブ制御言語 (job control language (JCL)). ジョブをオペレーティング・システムに識別させ、ジョブの要件を記述するために使われる制御言語。

シリアル検索 (serial search). 最初のメンバーから始めて最後のメンバーで終わるように、ある集合のメンバーが連続的に検査される探査方法。

シンボリック文字 (* symbolic-character). ユーザー定義の形象定数を指定するユーザー定義語。

真理値 (* truth value). 2 つの値 (真または偽) のどちらか一方によって、条件評価の結果を表したもの。

スーパークラス (* superclass). 別のクラスから継承されるクラス。「サブクラス *(subclass)*」も参照。

スイッチ状況条件 (switch-status condition). オンまたはオフに設定可能な UPSI スイッチが、特定の状況に設定されているという命題で、これに関して真理値を判別することができる。

数字 (digit). 0 ～ 9 までの任意の数字。COBOL では、この用語を用いて他の記号を参照することはない。

数字 (* numeric character). 次のような数字に属する文字: 0、1、2、3、4、5、6、7、8、9。

数字関数 (* numeric function). クラスとカテゴリーは数字だが、考えられる評価のいくつかにおいて整数関数の要件を満たさないような関数。

数字編集項目 (numeric-edited item). 印刷出力に使用できる形式になっている数値項目。外部 10 進数の 0 ～ 9 の数字、小数点、コンマ、ドル記号、編集符号制御記号、その他の編集記号から構成される。

数値項目 (* numeric item). 記述の内容が 0 ～ 9 の数字によって表される値に限定されるデータ項目。符号が付く場合は、+、-、または他の演算符号も含まれる。

数値リテラル (* numeric literal). 1 つまたは複数の数字からなるリテラルで、小数点または代数符号あるいはその両方を含むことができる。小数点は、右端の文字であることはできない。代数符号がある場合には、それが左端の文字でなければならない。

ステートメント (* statement). COBOL ソース・プログラムに書かれる、動詞を冒頭に置いた、ワード、リテラル、および分離符号の構文的に正しい組み合わせ。

スレッド (thread). 1 つのプロセスを制御する一連のコンピューター命令 (プロセス内のアプリケーションによって開始される)。

世紀ウィンドウ (century window). 世紀ウィンドウとは 100 年の間隔であり、その中では 2 桁の年号がすべて固有である。COBOL プログラマーが使用できる世紀ウィンドウには、いくつかのタイプがある。

- ウィンドウ化日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用する。
- ウィンドウ操作組み込み関数 DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、および YEAR-TO-YYYY については、*引き数-2 (argument-2)* によって世紀ウィンドウを指定する。
- 言語環境プログラム呼び出し可能サービスについては、CEESCEN で世紀ウィンドウを指定する。

整数 (* integer). (1) 小数点の右側に桁位置がない数字リテラル。(2) DATA DIVISION に定義される数値データ項目であり、小数点の右側に桁位置を含まないもの。(3) 関数の演算で戻される値の、小数点の右側の桁がすべて 0 であることが定義されている数字関数。

整数関数 (integer function). カテゴリーが数字であり、小数点の右側の桁位置が定義に入っていない関数。

セクション (* section). セクション本体と呼ばれる段落などのエンティティー (これらは省略したり、1 つだけまたは複数記述したりできる) からなる集合であり、その最初のものの前にセクション・ヘッダーが付いたもの。各セクションは、セクション・ヘッダーとそれに関連するセクション本体から構成される。

セクション名 (* section-name). PROCEDURE DIVISION の中にあるセクションに名前を付けるユーザー定義語。

セクション・ヘッダー (* section header). 後ろに区切りピリオドが付いたワードの組み合わせであり、ENVIRONMENT、DATA、または PROCEDURE の各部において、セクションの始まりを示すもの。ENVIRONMENT DIVISION および DATA DIVISION では、セクション・ヘッダーは、予約語の後に区切りピリオドを続けたものから構成される。ENVIRONMENT DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

DATA DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

PROCEDURE DIVISION では、セクション・ヘッダーは、セクション名、その後に続く予約語 SECTION、およびその後の区切りピリオドから構成される。

宣言部分 (* declaratives). PROCEDURE DIVISION の先頭に書き込まれた 1 つまたは複数の特殊目的セクションの集合であり、その先頭にはキーワード DECLARATIVE が付き、その最後にはキーワード END DECLARATIVES が続いている。宣言部分は、セクション・ヘッダー、USE コンパイラー指示文、および 0 個、1 個、または複数個の関連する段落で構成される。

宣言文 (* declarative sentence). 分離符号のピリオドによって終了する 1 つの USE ステートメントから構成されるコンパイラー指示文。

選択構造 (selection structure). 条件が真であるか偽であるかに応じて、ある一連のステートメントか、または別の一連のステートメントが実行されるというプログラムの処理ロジック。

ソース項目 (* source item). SOURCE 文節によって指定される ID で、印刷可能な項目の値を提供する。

ソース・プログラム (source program). ソース・プログラムは、他の形式や記号を使って表現することができるが、本書では、構文的に正しい COBOL ステートメントの集合を常に指している。COBOL ソース・プログラムは、IDENTIFICATION DIVISION または COPY ステートメントで開始され、指定された場合はプログラム終了マーカーで終了するか、または追加のソース・プログラム行なしで終了する。

ソート / マージ・ファイル記述記入項目 (* sort-merge file description entry). DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 SD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル文節から構成される。

ソート・ファイル (* sort file). SORT ステートメントによってソートされるレコードの集まり。ソート・ファイルは、ソート機能によってのみ作成され使用される。

ゾーン 10 進数項目 (zoned decimal item). 「*外部 10 進数項目 (external decimal item)*」を参照。

相互参照リスト (cross-reference listing). コンパイラー・リストの一部であり、プログラム内においてファイル、フィールド、および標識が定義、参照、および変更される場所に関する情報が入る。

相対キー (* relative key). 相対ファイルの中の論理レコードを識別するための内容を持つキー。

相対ファイル (* relative file). 相対編成のファイル。

相対編成 (* relative organization). 各レコードが、レコードのファイル内における論理的順序位置を指定する 0 より大きい整数値によって、固有なものとして識別される永続的な論理ファイル構造。

相対レコード番号 (* relative record number). 相対編成のファイル内におけるレコードの順序番号。この番号は、整数の数値リテラルとして扱われる。

想定小数点 (* assumed decimal point). データ項目中の実際の文字の存在とは無関係の小数点位置。想定小数点には、論理的な意味があり、物理的には表現されない。

添え字 (* subscript). 整数、(オプショナルで演算子 + または - 付きの整数が後ろにある) データ名、あるいは (オプショナルで演算子 + または - 付きの整数が後ろにある) 指標名のいずれかによって表されるオカレンス番号。これによりテーブル内の特定のエレメントを識別する。可変数の引き数を認める関数では、添え字付き ID を関数引き数として使用する場合は、添え字に ALL を使うことができる。

添え字付きデータ名 (* subscripted data-name). データ名とその後の括弧で囲まれた 1 つまたは複数の添え字から構成される ID。

［タ行］

代替レコード・キー (* alternate record key). 基本レコード・キー以外のキーであり、その内容が索引付きファイル内のレコードを識別する。

ダイナミック・リンク・ライブラリー (dynamic link library (DLL)). リンク時ではなく、ロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入ったファイル。複数のアプリケーションが DLL 内のコードおよびデータを同時に共用することができる。DLL はプログラムの実行可能 (.EXE) ファイルの一部ではないが、.EXE ファイルを正しく実行するためには必要となる可能性がある。

大容量記憶 (* mass storage). データを順次と非順次の 2 つの方法で編成して保管しておくことができるストレージ・メディア。

大容量記憶装置 (* mass storage device). 磁気ディスクや磁気ドラムのような、大きな記憶容量を持つ装置。

大容量記憶ファイル (* mass storage file). 大容量記憶メディアに割り当てられたレコードの集まり。

対話式システム生産性向上機能 (Interactive SystemProductivity Facility (ISPF)). TSO または VM ユーザーに対してメニュー方式のインターフェースを提供する IBM ソフトウェア・プロダクト。ISPF には、ライブラリー・ユーティリティー、強力なエディター、およびダイアログ管理が組み込まれている。

多重定義 (overload). 同じクラスで使用可能な別のメソッドと同一の名前を使い (ただし、異なるシグニチャーを使って)、メソッドを定義すること。「*シグニチャー (signature)*」を参照。

単項演算子 (* unary operator). 正符号 (+) または負符号 (-)。算術式の変数や算術式の左括弧の前に置き、それぞれ +1 または -1 を式に乗算する。

単純条件 (* simple condition). 次のような集合から選ばれた単一条件。

比較条件
クラス条件
条件名条件
スイッチ状況条件
符号条件

「*条件 (condition)*」と「*単純否定条件 (negated simple condition)*」も参照。

単純否定条件 (* negated simple condition). 論理演算子 NOT とその直後に単純条件を続けたもの。「*条件 (condition)*」と「*単純条件(simple condition)*」も参照。

段落 (* paragraph). PROCEDURE DIVISION では、段落名の後に区切りピリオドが続き、その後に 0 個、1 個、または複数個の文が続く。IDENTIFICATION DIVISION および ENVIRONMENT DIVISION では、段落ヘッダーの後に 0 個、1 個、または複数個の記入項目が続く。

段落ヘッダー (* paragraph header). 予約語の後に区切りピリオドが付いたもので、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION において段落の始まりを示すもの。IDENTIFICATION DIVISION で許可されている段落ヘッダーは次のとおり。

```
PROGRAM-ID. (Program IDENTIFICATION DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

ENVIRONMENT DIVISION で許可されている段落ヘッダーは次のとおり。

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

段落名 (* paragraph-name). PROCEDURE DIVISION の中の段落を識別し開始するユーザー定義語。

チェックポイント (checkpoint). ジョブ・ステップを後で再始動できるように、ジョブおよびシステムの状況に関する情報が記録される点。

中間結果 (intermediate result). 連続して行われる算術演算の結果を収める中間フィールド。

直接アクセス (* direct access). プロセスが、以前にアクセスされたデータへの参照ではなく、そのデータの位置にのみ依存する方法で、ストレージ・デバイスからデータを入手したり、ストレージ・デバイスにデータを入力したりする機能。

通貨記号 (currency symbol). 数字編集項目内の通貨記号値の部分を示すために、PICTURE 文節で使用される文字。通貨記号は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 文節によって定義することができる。CURRENCY SIGN 文節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値および通貨記号として使用される。通貨記号と通貨記号値は複数定義可能。「*通貨記号値 (currency sign value)*」も参照。

通貨記号値 (currency-sign value). 数字編集項目に保管される通貨単位を識別する文字ストリング。典型的な例としては、$、USD、EUR などがある。通貨記号値は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 文節によって定義することができる。CURRENCY SIGN 文節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値として使用される。「*通貨記号 (currency symbol)*」も参照。

次の実行可能ステートメント (* next executable statement). 現在のステートメントの実行完了後に制御が移される次のステートメント。

次の実行可能文 (* next executable sentence). 現在のステートメントの実行完了後に制御が移される次の文。

次のレコード (* next record). ファイルの現在のレコードに論理的に続くレコード。

データ記述記入項目 (* data description entry). COBOL プログラムの DATA DIVISION 内の記入項目であり、レベル番号の後に必要に応じてデータ名が続き、その後に必要に応じて一連のデータ文節が続いたもの。

データ項目 (* data item). COBOL プログラムによってまたは関数演算の規則によって定義されるデータ単位 (リテラルを除く)。

データ文節 (* data clause). COBOL プログラムの DATA DIVISION のデータ記述記入項目に現れる文節で、データ項目の特定の属性を記述する情報を提供する。

データ名 (* data-name). データ記述記入項目で記述されるデータ項目に名前を付けるユーザー定義語。一般形式で使用された場合、データ名は、その形式の規則で

特に許可されていない限り、参照変更、添え字付け、または修飾してはならないワードを表す。

テーブル (* table). DATA DIVISION の中で OCCURS 文節によって定義される、論理的に連続するデータ項目の集合。

テーブル・エレメント (* table element). テーブルを構成する反復項目の集合に属するデータ項目。

テキスト名 (* text-name). ライブラリー・テキストを識別するユーザー定義語。

テキスト・デック (text deck). 「*オブジェクト・デック (object deck)*」または「*オブジェクト・モジュール (object module)*」の同義語。

テキスト・ワード (* text word). COBOL ライブラリー、ソース・プログラム、または疑似テキスト内のマージン A およびマージン R の間の、1 文字または一連の連続する文字。

- スペース以外の分離符号、疑似テキスト区切り文字、非数値リテラルの開始と終了の区切り文字。右括弧と左括弧の文字は、ライブラリー、ソース・プログラム、または疑似テキストにおけるコンテキストに関係なく、常にテキスト・ワードと見なされる。
- 非数値リテラルの場合には、リテラルを囲む左引用符と右引用符を含むリテラル。
- コメント行および分離符号によって囲まれたワード COPY を除く、その他の連続する一連の COBOL 文字で、分離符号でもリテラルでもないもの。

デバッグ行 (* debugging line). 行の標識区域に文字 D がある行のこと。

デバッグ・セクション (* debugging section). USE FOR DEBUGGING ステートメントが含まれているセクション。

トークン (token). COBOL エディターでは、プログラムにおける意味の単位。トークンには、データ、言語キーワード、ID、またはその他の言語構文の一部を含めることができる。

トークン強調表示 (token highlighting). COBOL エディターでは、プログラム言語のトークン・タイプをさまざまなカラーやフォントで表示できるようにする機能。この機能は、プログラムの構造をより明確にする。「トークン属性 (Token Attributes)」ウィンドウを使用すれば、トークン・タイプの外観をカスタマイズすることができる。

動詞 (* verb). COBOL コンパイラーまたはオブジェクト・プログラムによってとられる処置を表すワード。

動的 CALL (dynamic CALL). DYNAM および NODLL オプションを使用してコンパイルしたプログラム内の CALL *literal* ステートメント、または NODLL オプションを使用してコンパイルしたプログラム内の CALL *identifier* ステートメント。

動的アクセス (* dynamic access). 1 つの OPEN ステートメントの実行範囲内において、特定の論理レコードを、大容量記憶ファイルからは順次アクセス以外の方法で取り出したりそのファイルに入れたりでき、またファイルからは順次アクセスの方法で取り出せるアクセス・モード。

動的ストレージ域 (dynamic storage area (DSA)). 動的に獲得されるストレージであり、レジスター保管域、および動的ストレージ割り振りに使用可能な区域 (プログラム変数など) から構成される。DSA は、通常、言語環境プログラムによって管理される STACK セグメント内に割り振られる。

特殊名記入項目 (* special names entry). ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の記入項目。この記入項目は、通貨記号を指定したり、小数点を選択したり、シンボリック文字を指定したり、インプリメントする人の名前をユーザー指定の簡略名と関連付けたり、英字名を文字セットまたは照合シーケンスと関連付けたり、クラス名を一連の文字と関連付けたりするための手段を提供する。

特殊文字 (* special character). 以下に示す集合に属する文字。

文字	意味
+	正符号
-	負符号 (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
$	通貨記号
,	コンマ (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
„	引用符
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン

特殊文字ワード (* special-character word). 算術演算子または比較文字である予約語。

特殊レジスター (* special registers). コンパイラー生成の特定のストレージ域で、その基本的な使用法は、

具体的な COBOL 機能を使用したときに作り出される情報を保管することである。

独立項目 (* noncontiguous items). WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目で、他のデータ項目と階層上の関係を持たないもの。

トップダウン開発 (top-down development). 「*構造化プログラミング (structured programming)*」を参照。

トップダウン設計 (top-down design). 関連付けられた諸機能が、構造の各レベルで実行されるようにする階層構造を使ったコンピューター・プログラムの設計。

トラブルシューティング (troubleshoot). コンピューター・ソフトウェアの使用中に問題を検出し、突き止め、除去すること。

トレーラー・ラベル (trailer-label). (1) 記録メディアの装置上でデータ・レコードの後に置かれるファイルまたはデータ・セットのラベル。(2) 「*ファイル終わりラベル (end-of-file label)*」の同義語。

[ナ行]

内部 10 進項目 (internal decimal item). フィールド内で右端のバイトを除く各バイトが 2 桁の数字を表現するような形式。右端のバイトには、1 桁の数字と符号が入れられる。たとえば 10 進数値の +123 は、0001 0010 0011 1111 として表記される。「*パック 10 進数 (packed decimal)*」と同義。

内部データ (* internal data). プログラムの中で記述されるデータで、すべての外部データ項目および外部ファイル結合子を除いたもの。プログラムの LINKAGE SECTION で記述された項目は、内部データとして扱われる。

内部データ項目 (* internal data item). 実行単位内の 1 つのプログラムの中で記述されるデータ項目。内部データ項目はグローバル名を持つことができる。

内部ファイル結合子 (* internal file connector). 実行単位内にあるただ 1 つのオブジェクト・プログラムのみがアクセスできるファイル結合子。

名前 (name). COBOL オペランドを定義する 30 文字以内で構成されたワード。

二分探索 (binary search). 二分探索では、探索の各ステップで、一連のデータ要素の集合が 2 つに分割される。要素の数が奇数の場合には、何らかの適切なアクションが取られる。

入出力状況 (* I-O status). 入出力操作の結果としての状況を示す 2 文字の値を収める概念上のエンティティー。この値は、そのファイルについてのファイル制御記入項目で FILE STATUS 文節を使用することによって、プログラムに使用可能にされる。

入出力ステートメント (* input-output statement). 個々のレコードに対して操作を行うことにより、またはファイルを 1 つの単位として操作することにより、ファイルの処理を行うステートメント。入出力ステートメントには、ACCEPT (ID 句付き)、CLOSE、DELETE、DISPLAY、OPEN、READ、REWRITE、SET (TO ON または TO OFF 句付き)、START、および WRITE がある。

入出力ファイル (* input-output file). I-O モードでオープンされるファイル。

入出力モード (* I-O mode). ファイルに対する I-O 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入力ファイル (* input file). 入力モードでオープンされるファイル。

入力プロシージャー (* input procedure). ソートすべき特定のレコードの解放を制御する目的で、SORT ステートメントの実行時に制御が渡されるステートメントの集合。

入力モード (* input mode). ファイルに対する INPUT 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

ヌル (null). 無効なアドレスの値をポインター・データ項目に割り当てるために使用される形象定数。NULL を使えるところならばどこでも、NULLS を使用できる。

ネストされたプログラム (nested program). ほかのプログラムの中に直接的に含まれているプログラム。

年フィールド拡張 (year field expansion). 2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用する方法。これは、2 桁の年を使用していたアプリケーションに対して確実に信頼できる日付処理を行う唯一の方法である。

［ハ行］

バイト (byte). 特定の数のビット (通常 8 ビット) からなるストリングであり、1 つの単位として処理され、1 つの文字を表す。

ハイパースペース (hiperspace). z/OS 環境で、プログラムがバッファーとして使用できる最大 2 GB までの連続する仮想記憶アドレス範囲。

パック 10 進数項目 (packed-decimal item). 「*内部 10 進項目 (internal decimal item)*」を参照。

パッケージ (package). 関連する Java クラスの集まり。個々に、または全体としてインポートすることができる。

バッファー (buffer). 入力データまたは出力データを一時的に保持するために使用されるストレージの一部分。

パラメーター (parameter). (1) 呼び出しプログラムと呼び出されるプログラムの間で受け渡されるデータ。(2) メソッド呼び出しの USING 文節内のデータ・エレメント。引き数によって、呼び出されたメソッドが要求された操作を実行するために使用できる追加情報を与える。

範囲区切りステートメント (* delimited scope statement). 明示的範囲終了符号を含んでいるステートメント。

範囲終了符号 (scope terminator). PROCEDURE DIVISION の特定のステートメントの終わりを示す COBOL 予約語。これは明示的なもの (たとえば、END-ADD など) であることもあれば、暗黙のもの (区切りピリオド) であることもある。

反復構造 (iteration structure). ある条件が真である間、あるいはある条件が真になるまで、一連のステートメントが繰り返して実行されるプログラムの処理ロジック。

汎用オブジェクト・リファレンス (universal object reference). どのクラスのオブジェクトでも参照できるデータ名。

比較条件 (* relation condition). ある算術式、データ項目、非数字リテラル、または指標名の値が、他の算術式、データ項目、非数字リテラル、または指標名の値と特定の関係を持つかどうかという命題で、それに関して真理値が判別され得る。「*関係演算子 (relational operator)*」も参照。

比較文字 (* relation character). 以下に示す集合に属する文字。

文字	意味
>	より大きい
<	より小さい
=	に等しい

引き数 (* argument). 関数の演算で使用する値を指定する ID、リテラル、算術式、または関数 ID。

非制限ストレージ (unrestricted storage). 2 GB バーより下のストレージ。16MB 境界より上または下がある。16MB 境界より上では、31 ビット・モードでのみ、アドレス可能。

ビッグ・エンディアン (big-endian). メインフレームおよび AIX ワークステーションが 2 進データを保管する際に使用するデフォルトの形式。この形式では、最も重要性の低い桁が最高位のアドレスになる。「*リトル・エンディアン (little-endian)* 」と対比。

日付形式 (date format). 次のいずれかの方法で指定される、日付フィールドの日付パターン。

- DATE FORMAT 文節または DATEVAL 組み込み関数 argument-2 によって明示的に。
- 日付フィールドを戻すステートメントおよび組み込み関数によって暗黙的に (詳細については、*Enterprise COBOL 言語解説書* の日付フィールドを参照)。

日付フィールド (date field). 次のいずれか。

- データ記述記入項目に DATE FORMAT 文節が含まれているデータ項目。
- 次のいずれかの組み込み関数で戻される値。

  DATE-OF-INTEGER
  DATE-TO-YYYYMMDD
  DATEVAL
  DAY-OF-INTEGER
  DAY-TO-YYYYDDD
  YEAR-TO-YYYY
  YEARWINDOW
- ACCEPT ステートメントの概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDD。
- 特定の算術演算の結果 (詳細については、*Enterprise COBOL 言語解説書* の日付フィールドの演算を参照)。

日付フィールド という用語は、*拡張日付フィールド* と *ウィンドウ化日付フィールド* の両方を指す。「*非日付 (nondate)*」も参照。

非日付 (nondate). 次のいずれか。

- 日付記述記入項目に DATE FORMAT 文節が含まれていないデータ項目
- リテラル
- UNDATE 関数を使用して変換された日付フィールド
- 参照変更された日付フィールド
- 日付フィールド・オペランドを含むことができる特定の算術演算の結果。たとえば、2 つの互換性のある日付フィールドの差

標準データ形式 (* standard data format). COBOL の DATA DIVISION においてデータの特性を記述する際に使われる概念。この概念の下では、データの特性は、そのデータがコンピューターの中または外部メディア上に内部的に保管される方式に合わせた形式ではなく、無限大の長さと幅を持つ印刷ページ上のデータの外観に合わせた形式で表現される。

部 (* division). 部の本体と呼ばれる、0 個、1 個、または複数個のセクションまたは段落の集合であり、特定の規則に従って形成および結合されたもの。部とは、部のヘッダーおよび関連した部の本体で構成される。COBOL プログラムには、見出し部、環境部、データ部、および手続き部の 4 つの部がある。

ファイル (* file). 論理レコードの集合。

ファイル位置標識 (*file position indicator). 概念的エンティティーであり、索引付きファイルの場合は参照キー内の現行キーの値、順次ファイルの場合は現行レコードのレコード番号、相対ファイルの場合は現行レコードの相対レコード番号が入っている。あるいは、次の論理レコードが存在しないことを示すか、オプションの入力ファイルが存在しないことを示すか、AT END 条件がすでに存在していることを示すか、もしくは有効な次のレコードが設定されていないことを示す。

ファイル記述項目 (* file description entry). DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 FD とそれに続くファイル名、必要に応じて次に続く一連のファイル文節から構成される。

ファイル結合子 (* file connector). ファイルに関する情報が入っており、ファイル名と物理ファイルの間のリンケージとして、さらにファイル名とその関連レコード域の間のリンケージとして使用されるストレージ域。

ファイル制御記入項目 (* file control entry). SELECT 文節と、ファイルの関連物理属性を宣言するすべての従属文節。

ファイル属性対立条件 (* file attribute conflict condition). ファイルに入出力操作の実行を試みて失敗した場合に、プログラムの中でそのファイルに対して指定されたファイル属性が、そのファイルの固定属性と一致しないこと。

ファイル文節 (* file clause). DATA DIVISION の記入項目であるファイル記述記入項目 (FD 記入項目) およびソート・マージ・ファイル記述記入項目 (SD 記入項目) のいずれかの一部として現れる文節。

ファイル編成 (* file organization). ファイルの作成時に確立される永続論理ファイル構造。

ファイル名 (* file-name). DATA DIVISION の FILE SECTION の中のファイル記述記入項目またはソート・マージ・ファイル記述記入項目で記述されるファイル結合子に名前を付けるユーザー定義語。

ファイル・システム (file system). ディスケットまたはミニディスクなどの物理または論理の大容量記憶装置上におけるファイルおよびファイル管理構造の集合。

ファクトリー・データ (factory data). いったんクラスに割り振られ、クラスのすべてのインスタンスに共用されるデータ。ファクトリー・データは、クラス定義の FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION 内に宣言される。Java private 静的データと同義。

ファクトリー・メソッド (factory method). オブジェクト・インスタンスとは無関係に、クラスによってサポートされるメソッド。ファクトリー・メソッドは、クラス定義の FACTORY 段落に宣言される。Java public 静的メソッドと同義。一般的に、オブジェクトの作成をカスタマイズするために使用される。

複合 (complex ODO). 次のような OCCURS DEPENDING ON 文節の特定の形式。

- 可変位置項目またはグループ: DEPENDING ON オプションを指定した OCCURS 文節によって記述されたデータ項目の後に、非従属データ項目またはグループが続く。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 文節によって記述されたデータ項目の後に、OCCURS 文節によって記述された非従属データ項目が続く。
- 可変長エレメントを持つテーブル: OCCURS 文節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 文節によって記述された従属データ項目が含まれている。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

複合条件 (* combined condition). 2 つ以上の条件を AND または OR 論理演算子で結合した結果生じる条件。「*条件 (condition)* 」と「*複合否定条件 (negated combined condition)* 」も参照。

複合条件 (* complex condition). 1 つまたは複数の論理演算子が 1 つまたは複数の条件に基づいて作動する条件。「*条件 (condition)*」、「*単純否定条件 (negated simple condition)*」、および「*複合否定条件 (negated combined condition)*」も参照。

複合否定条件 (* negated combined condition). 論理演算子 NOT とその直後に括弧で囲んだ複合条件を続けたもの。「*条件 (condition)*」と「*複合条件 (combined condition)*」も参照。

符号条件 (* sign condition). データ項目や算術式の代数値が、0 より小さいか、大きいか、または等しいかという命題で、それに関して真理値が判別できる。

物理レコード (* physical record). 「*ブロック (block)*」を参照。

浮動小数点数 (floating-point number). 小数部と指数が入っている数値データ項目。その値は、小数部に、指数で指定されただけ累乗された数字データ項目の底を乗算することによって得られる。

部のヘッダー (* division header). ワードとその後に続く、部の先頭を示す区切りピリオドの組み合わせ。部のヘッダーは次のとおり。

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

古くなったエレメント (* obsolete element). 標準 COBOL の COBOL 言語エレメントのうち、標準 COBOL の次回の改訂版から削除される予定のもの。

プログラム (program). (1) コンピューターで処理するのに適した一連の命令。処理には、コンパイラーを使用してプログラムを実行のために準備することや、ランタイム環境を使用してプログラムを実行することなどが含まれる可能性がある。(2) 1 つまたは複数の相互に関係のあるモジュールの論理アセンブリー。同じプログラムの複数のコピーを異なるプロセスで実行することができる。

プログラム識別記入項目 (* program identification entry). IDENTIFICATION DIVISION の PROGRAM-ID 段落内の記入項目であり、プログラム名を指定し、選択されたプログラム属性をプログラムに割り当てる文節が入っている。

プログラム名 (* program-name). IDENTIFICATION DIVISION およびプログラム終了マーカーにおいて、COBOL ソース・プログラムを識別するユーザー定義語。

プロシージャー (* procedure). PROCEDURE DIVISION 内にある 1 つの段落または論理的に連続する段落のグループ、あるいは 1 つのセクションまたは論理的に連続するセクションのグループ。

プロシージャー統合 (procedure integration). COBOL 最適化プログラムの機能の 1 つは、実行済みプロシージャーや含まれているプログラムへの呼び出しを簡略化することである。

PERFORM プロシージャー統合とは、PERFORM ステートメントがその実行済みプロシージャーによって置き換えられるプロセスのこと。含まれているプログラムのプロシージャー統合とは、含まれているプログラムへの呼び出しがプログラム・コードによって置き換えられるプロセスのこと。

プロシージャー分岐ステートメント (* procedure branching statement). ソース・プログラムの中にステートメントが書かれている順番どおりに次の実行可能ステートメントに制御の移動をせず、別のステートメントに明示的に制御の移動を引き起こすステートメント。プロシージャー分岐ステートメントは次のとおり。ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUT PROCEDURE 句付き)、PERFORM、および SORT (INPUT PROCEDURE または OUTPUT PROCEDURE 句付き)。

プロシージャー名 (* procedure-name). PROCEDURE DIVISION の中にある段落またはセクションに名前を付けるために使用されるユーザー定義語。プロシージャー名は、段落名 (これは修飾することができる) またはセクション名から構成される。

プロシージャー・ポインター・データ項目 (procedure-pointer data item). 入り口点へのポインターを収めることができるデータ項目。USAGE IS PROCEDURE-POINTER 文節で定義されるデータ項目には、プロシージャー入り口点のアドレスが入っている。 一般的に、COBOL および言語環境プログラムのプログラムと通信するために使用される。

プロジェクト環境 (project environment). エディターやジョブ・モニターのような COBOL ツールを立ち上げ、COBOL ファイルやデータ・セットの作業を行うための中央設置場所。

プロセス (process). プログラムの全部または一部の実行中に発生する一連のイベント。複数のプロセスを並行して実行することができ、1 つのプロセス内で実行されるプログラムはリソースを共用することができる。

ブロック (* block). 通常 1 つまたは複数の論理レコードで構成されるデータの物理装置。大容量記憶ファイルの場合、ブロックに論理レコードの一部が入れられることもある。ブロックのサイズは、そのブロックが入っているファイルのサイズや、そのブロックに入っているまたはそのブロックにオーバーラップする論理レコード (1 つまたは複数) のサイズとは直接関係しない。「*物理レコード (physical record)* 」と同義。

文 (* sentence). 1 つのステートメントまたは複数のステートメントの並びで、その最後のものは、分離符号のピリオドで終了する。

分散デバッガー (Distributed Debugger). クライアント / サーバー・アプリケーションの 1 つ。これを使用すると、ネットワーク接続を介してアクセス可能なシステムで実行されるか、またはワークステーションで実行されるプログラムのエラーを検出して診断することができる。分散デバッガーはグラフィカル・ユーザー・インターフェースを使用するので、ユーザーはコマンドを出してプログラムの実行 (リモートまたはローカル) を制御することができる。

文書タイプ定義 (document type definition (DTD)). XML 文書のクラスの文法。「XML 型定義 (XML type definition)」を参照

文節 (* clause). 記入項目の属性を指定するという目的で順番に並べられた連続する COBOL 文字ストリング。

分離文字 (* separator). 文字ストリングを区切るために使われる 1 文字または連続する 2 文字。

ページ (page). データの物理的分離を表す、出力データの垂直分割。分離は、内部論理要件または出力メディアの外部特性、あるいはその両方に基づいて行われる。

ページ本体 (* page body). 行を記述できる、または行送りすることができる (またはその両方ができる) 論理ページの部分。

別個にコンパイルされたプログラム (* separately compiled program). あるプログラムが、その中に含まれているプログラムと一緒に、それ以外のすべてのプログラムとは別個にコンパイルされること。

ヘッダー・ラベル (header label). (1) 記録メディア装置上のデータ・レコードの前に付いているファイル・ラベルまたはデータ・セット・ラベル。(2) 「*ファイル開始ラベル (beginning-of-file label)*」の同義語。

編集解除 (* de-edit). 項目の編集解除された数値を判別するために、数字編集データ項目からすべての編集文字を論理的に除去すること。

編集データ項目 (edited data item). 0 の抑止または編集文字の挿入、あるいはその両方を行うことによって変更されたデータ項目。

編集文字 (* editing character). 次のセットに属する 1 つの文字または固定された 2 文字の組み合わせ。

文字	意味
	スペース
0	ゼロ
+	正符号
-	負符号
CR	貸方
DB	借方
Z	ゼロ抑止
*	不渡り手形
$	通貨記号
,	コンマ (小数点)
.	ピリオド (小数点)
/	斜線 (スラッシュ)

変数 (* variable). オブジェクト・プログラムの実行によって変更を受ける可能性のある値を持つデータ項目。算術式で使われる変数は、数字基本項目でなければならない。

ポート、移植する (port). (1) 異なるプラットフォームで実行できるようにコンピューター・プログラムを変更すること。(2) インターネット・プロトコルでは、Transmission Control Protocol (TCP) プロトコルまたは User Datagram Protocol (UDP) プロトコルと高水準のプロトコルまたはアプリケーションの間の特定の論理結合子。ポートはポート番号によって識別される。

ポインター・データ項目 (pointer data item). アドレスの値を保管することのできるデータ項目。これらのデータ項目は、USAGE IS POINTER 文節を使用してポインターとして明示的に定義される。ADDRESS OF 特殊レジスターは、ポインター・データ項目として暗黙的に定義されている。ポインター・データ項目は、他のポインター・データ項目と等しいかどうか比較したり、他のポインター・データ項目に内容を移動したりできる。

ボリューム (volume). 外部ストレージのモジュール。テープ装置の場合はリール、直接アクセス装置の場合はユニット。

ボリューム切り替え処理手順 (volume switch procedures). ファイルの終わりに達する前にユニットまたはリールの終わりに達したとき、自動的に実行されるシステム固有の処理手順。

［マ行］

マージ・ファイル (* merge file). MERGE ステートメントによってマージされるレコードの集まり。マージ・ファイルは、マージ機能により作成され、マージ機能によってのみ使用できる。

マルチスレッド化 (multithreading). コンピューター内で複数のパスを使って実行を行う並行操作。「マルチプロセッシング *(multiprocessing)*」と同義。

マルチタスキング (multitasking). 2 つ以上のタスクの並行実行またはインターリーブド実行を可能にする操作モード。

無効キー条件 (* invalid key condition). 索引付きファイルまたは相対ファイルに関連するキーの特定値が無効であると判別された場合に生じる、実行時の条件。

明示的範囲終了符号 (* explicit scope terminator). 特定の PROCEDURE DIVISION ステートメントの有効範囲を終わらせる予約語。

命令ステートメント (* imperative statement). 命令の動詞で開始して、行うべき無条件の処置を指定するステートメント。または明示的範囲終了符号によって区切られた条件ステートメント (範囲区切りステートメント)。1 つの命令ステートメントは、一連の命令ステートメントから構成することができる。

メインプログラム (main program). プログラムとサブルーチンからなる階層内において、プログラムが実行されたときに最初に制御を受け取るプログラム。

メガバイト、MB (* megabyte (MB)). 1 メガバイトは 1,048,576 バイトに相当する。

メソッド (method). オブジェクトによってサポートされる操作の 1 つを定義し、そのオブジェクトに対する INVOKE ステートメントによって実行されるプロシージャー・コード。

メソッド識別記入項目 (* method identification entry). IDENTIFICATION DIVISION の METHOD-ID 段落内の記入項目。この記入項目には、メソッド名を指定する文節が入っている。

メソッド定義 (* method definition). メソッドを定義する COBOL ソース単位。

メソッドの起動 (method invocation). あるオブジェクトから別のオブジェクトへの通信で、受信オブジェクトにメソッドを実行するように要求するもの。Visual Builder では、メソッド呼び出しは、要求されたメソッドを示すメソッド名、そのメソッドを実行する際に使用すべきパラメーター、および必要に応じて戻り変数から構成される。

メソッド名 (method-name). オブジェクト指向操作の名前。メソッドを呼び出すために使用するときの名前は、リテラルまたはデータ項目のどちらでもよい。メソッドを定義するために METHOD-ID 段落で使用するときの名前は、リテラルでなければならない。

文字 (* character). 基礎的な分割できない言語単位。

文字 (* letter). 次のような 2 つの集合のどちらかに属している文字。

1. 英大文字: A、
   B、C、D、E、F、G、H、I、J、K、L、M、N、
   O、P、Q、R、S、T、U、V、W、X、Y、Z
2. 英小文字: a、
   b、c、d、e、f、g、h、i、j、k、l、m、n、
   o、p、q、r、s、t、u、v、w、x、y、z

文字位置 (character position). USAGE IS DISPLAY として記述される単一の標準データ形式文字を保管するために必要な物理ストレージの量。

文字エンコード・ユニット (character encoding unit). 文字を表すために使用されるコンピューター・メモリーの単位。1 つまたは複数のエンコード・ユニットを使用して、コード化文字セットで文字を表す。「エンコード・ユニット *(encoding unit)*」とも呼ばれる。

文字ストリング (character string). COBOL 語、リテラル、PICTURE 文字ストリング、またはコメント記入項目を形成する一連の連続した文字。文字ストリングは区切り文字で区切らなければならない。

文字セット (character set). テキスト情報を表すために使用されるエレメントの集合。ただし、コード化表現は想定されていない。「*コード化文字セット (coded character set)*」も参照。

モジュール定義ファイル (module definition file). ロード・モジュール内のコード・セグメントを記述するファイル。

［ヤ行］

ユーザー定義語 (* user-defined word). 文節やステートメントの形式を満たすためにユーザーが提供する必要のある COBOL ワード。

優先順位番号 (* priority-number). セグメント化の目的で、PROCEDURE DIVISION 内のセクションを分類する

ユーザー定義語。セグメント番号には 0 から 9 までの文字だけしか使えない。セグメント番号は 1 桁または 2 桁として表せる。

ユニット (unit). 直接アクセスのモジュール。その大きさは IBM によって規定されている。

呼び出されるプログラム (called program). CALL ステートメントの対象となるプログラム。呼び出されるプログラムと呼び出しプログラムが実行時に結合されて、1 つの実行単位が作成される。

呼び出し可能サービス (callable services). 言語環境プログラムでは、従来の言語環境プログラム定義の呼び出しインターフェースを使用して COBOL プログラムが呼び出すことができる一連のサービス。言語環境プログラムの規則を共用するプログラムはすべて、これらのサービスを使用できる。

呼び出しプログラム (* calling program). 別のプログラムへの CALL を実行するプログラム。

予約語 (* reserved word). COBOL ソース・プログラムの中で使用することができるが、ユーザー定義語またはシステム名としてプログラムの中で使用されてはならないワードのリスト中に挙げられている COBOL ワード。

[ラ行]

ライブラリー名 (* library-name). COBOL ライブラリーの名前を表すユーザー定義語。与えられたソース・プログラムをコンパイルするためにコンパイラーが使用するライブラリーを識別する。

ライブラリー・テキスト (* library text). COBOL ライブラリーの中にある一連のテキスト語、コメント行、分離符号のスペース、または分離符号の疑似テキスト区切り文字。

ラッパー (wrapper). オブジェクト指向コードとプロシージャー指向コード間のインターフェースを提供するオブジェクト。ラッパーを使用すると、他のシステムがプログラムを再利用したりアクセスしたりできるようになる。

ランタイム環境 (run-time environment). COBOL プログラムが実行される環境。

ランダム・アクセス (* random access). キー・データ項目のプログラム指定値を使って、相対ファイルまたは索引付きファイルから取り出したり、削除したり、またはそこに入れたりする論理レコードを識別するアクセス・モード。

リール (reel). ストレージ・メディアの個別部分。その大きさはインストールする人によって決定され、1 つのファイルの一部、1 つのファイルの全部、または任意の個数のファイルが収容される。「*ユニット (unit)*」および「*ボリューム (volume)*」と同義。

リソース (* resource). オペレーティング・システムの制御下に置かれており、実行中のプログラムによって使用できる機能またはサービス。

リテラル (literal). ストリングを構成するために並べられた文字によって、または形象定数を使用することによって、その値が決められる文字ストリング。

リトル・エンディアン (little-endian). PC が 2 進データを保管するのに使用するデフォルト形式。この形式では、最も重要な数字が最高位アドレスになる。「*ビッグ・エンディアン (big-endian)*」と対比。

リリアン日付 (Lilian date). グレゴリオ暦の開始以降の日数。第 1 日は 1582 年 10 月 15 日、金曜日。リリアン日付形式という呼び名は、グレゴリオ暦の作成者である Luigi Lilio に敬意を表して付けられている。

リンカー (linker). DFSMS リンケージ・エディターまたは DFSMS バインダーのいずれかを指す。

リンク (link). (1) リンク接続 (伝送メディア) と、それぞれがリンク接続の終端にある 2 つのリンク・ステーションの組み合わせ。1 つのリンクは、マルチポイントまたはトークンリング構成において、複数のリンク間で共用できる。(2) データ項目あるいは 1 つまたは複数のコンピューター・プログラムの部分を相互接続すること。たとえば、リンケージ・エディターによってオブジェクト・プログラムをリンクして実行可能ファイルを作成すること。

リング (ring). COBOL エディターでは、ファイル相互間でもっと簡単に編集できるようにしたファイルの集合。

ルーチン (routine). 1 つの演算または関連付けられた一連の演算をコンピューターに行わせる COBOL プログラムの中のステートメントの集合。言語環境プログラムでは、プロシージャー、関数、またはサブルーチンのいずれかを指す。

ルーチン名 (* routine-name). COBOL 以外の言語で記述されたプロシージャーを識別するユーザー定義語。

レコード (* record). 「*論理レコード (logical record)*」を参照。

レコード域 (* record area). DATA DIVISION の FILE SECTION 内のレコード記述項目で記述されるレコードを

処理する目的で割り振られるストレージ域。FILE SECTION では、レコード域の現行の文字位置の数は、明示または暗黙の RECORD 文節によって決められる。

レコード記述 (* record description). 「*レコード記述項目 (record description entry)*」を参照。

レコード記述項目 (* record description entry). 特定のレコードに関連付けられたデータ記述記入項目の全体集合。「*レコード記述 (record description)*」と同義。

レコード内データ構造 (* intrarecord data structure). 連続したデータ記述記入項目のサブセットによって定義される、1 つの論理レコードから得られるグループ・データ項目および基本データ項目の集合全体。これらのデータ記述記入項目には、レコード内データ構造を記述している最初のデータ記述記入項目のレベル番号より大きいレベル番号を持つすべての記入項目が含まれる。

レコード番号 (* record number). 編成が順次であるファイル内のレコードの順序数。

レコード名 (* record-name). COBOL プログラムの DATA DIVISION 内のレコード記述項目で記述されるレコードに名前を付けるユーザー定義語。

レコード・キー (record key). 索引付きファイル内のレコードを識別する内容を持つキー。

列 (* column). 印刷行における文字位置。列は、印刷行の左端の文字位置から印刷行の右端の位置まで、1 から番号が付けられ、1 ずつ増加する。

レベル番号 (* level-number). 階層構造におけるデータ項目の位置を示すか、またはデータ記述記入項目の特性を示す、2 桁の数字で表されたユーザー定義語。1 ～ 49 のレベル番号は、論理レコードの階層構造におけるデータ項目の位置を示す。1 ～ 9 のレベル番号は、1 桁の数字として書くことも、0 の後に有効数字を付けて書くこともできる。レベル番号 66、77、および 88 は、データ記述記入項目の特性を識別する。

レベル標識 (* level indicator). 特定の種類のファイルや階層内における位置を識別する 2 つの英字。DATA DIVISION 内のレベル標識には、CD、FD、および SD がある。

連続項目 (* contiguous items). DATA DIVISION 内の連続する記入項目によって記述され、相互に一定の階層関係を持っている項目。

ローカル参照 (local reference). メソッドの有効範囲内にあるオブジェクトの参照。

ロケール (locale). プログラム実行環境の一連の属性であり、文化的に重要な考慮事項を示す。たとえば、文字コード・ページ、照合シーケンス、日時形式、通貨表記、数値表記、または言語など。

論理演算子 (* logical operator). 予約語 AND、OR、または NOT のいずれか。条件の形成において、AND または OR、あるいはその両方を論理連結語として使用できる。NOT は論理否定に使用できる。

論理レコード (* logical record). 最も包括的なデータ項目。レコードのレベル番号は 01。レコードは、1 つの基本項目または項目のグループのどちらでもよい。「*レコード (record)*」と同義。

[ワ行]

ワークステーション (workstation). エンド・ユーザーが使用するコンピューターの総称 (パーソナル・コンピューター、3270端末、インテリジェント・ワークステーション、および UNIX 端末を含む)。ワークステーションはメインフレームまたはネットワークに接続されることがよくある。

ワード (* word). ユーザー定義語、システム名、予約語、または関数名を形成する、30 文字以内の文字ストリング。

割り当て名 (assignment-name). COBOL ファイルの編成を識別する名前で、システムがこれを認識する際に使う。

[数字]

1 バイト文字セット (single-byte character set (SBCS)). 各文字が 1 バイトで表現される文字の集合。「*EBCDIC (拡張 2 進化 10 進コード) (Extended Binary-Coded Decimal Interchange Code)*」も参照。

16MB 境界より上 (above the 16-MB line). いわゆる 16MB 境界より上で、2 GB バーより下のストレージ。このストレージは、31 ビット・モードでのみアドレス可能である。1980 年代に MVS/XA アーキテクチャーが IBM によって導入される以前は、プログラムの仮想記憶域は 16MB に限られていた。24 ビット・モードでコンパイルしたプログラムは、あたかも架空のストレージ境界線の下に抑えられているように、16MB のスペースしかアドレスすることができない。VS COBOL II 以降、31 ビット・モードでコンパイルしたプログラムは、16MB 境界より上に配置することができる。

2 進数項目 (binary item). 2 進表記 (基数 2 の数体系) で表される数値データ項目。等価の 10 進数は、10

進数字 0 ～ 9 に演算符号を加えたもので構成される。項目の左端ビットは演算符号。

2 バイト文字セット (double-byte character set (DBCS)). それぞれの文字が 2 バイトで表現される 1 組の文字。256 のコード・ポイントで表現できる記号より多くの記号を含む日本語、中国語、および韓国語などの言語には、2 バイト文字セットが必要である。各文字に 2 バイトが必要なため、DBCS 文字の入力、表示、および印刷には、DBCS を受け入れ可能なハードウェアおよびサポートされるソフトウェアが必要。

77 レベル記述記入項目 (* 77-level-description-entry). レベル番号 77 を持つ非連続データ項目を記述するデータ記述記入項目。

A

ANSI (米国規格協会) (ANSI (American National Standards Institute)). 米国で認定された組織が自発的工業規格を作成して維持する手順を設定する組織であり、製造業者、消費者、および一般の利害関係者で構成される。

APPC. 「*拡張プログラム間通信機能 (advanced program-to-program communication (APPC))* 」を参照。

array (配列). 言語環境プログラムでは、データ・オブジェクトからなる集合体。それぞれのデータ・オブジェクトは、添え字付けにより固有に参照することができる。配列は、COBOL ではテーブルに相当する。

ASCII. 情報交換用米国標準コード (ASCII)。7 ビットのコード化文字をベースとする 1 つのコード化文字セット (パリティー・チェックを含む 8 ビット) を使用する標準コードであり、データ処理システム、データ通信システム、および関連装置の間での情報交換に使用される。ASCII セットは、制御文字と図形文字で構成される。

拡張: IBM では、ASCII コードへの拡張部分を定義している (文字 128 ～ 255)。

AT END 条件 (* AT END condition). 次のような特定の条件のもとで、READ、RETURN、または SEARCH ステートメントを実行した場合に引き起こされる条件。

- 順次アクセス・ファイルに対して READ ステートメントを実行中に、そのファイル内に次の論理レコードが存在しない場合、または相対レコード番号中の有効数字の桁数が相対キー・データ項目のサイズより大きい場合、またはオプションの入力ファイルが存在しない場合。
- RETURN ステートメントの実行中に、関連するソート・ファイルまたはマージ・ファイルについての次の論理レコードが存在しない場合。
- SEARCH ステートメントの実行中に、関連する WHEN 句のいずれかで指定された条件を満足することなく、探索操作が終了した場合。

B

Btrieve ファイル・システム (Btrieve file system). アプリケーションに、キー値、順次アクセス方式、またはランダム・アクセス方式によってレコードを管理させる、キー索引付きレコード管理システム。IBM COBOL では、Btrieve (Pervasive Software から入手可能) によって COBOL 順次ファイルまたは索引付きファイルの入出力言語がサポートされる。Btrieve ファイル・システムを使用して、VisualAge CICS エンタープライズ・アプリケーション開発によって作成されたファイルにアクセスできる。

C

CCSID. 「*コード化文字セット ID (Coded Character Set Identifier)*」を参照。

COBOL 文字セット (* COBOL character set). COBOL 構文を作成する際に使用される文字セット。次にリストする文字で構成される完全な COBOL 文字セット。

文字	意味
0,1, . . . ,9	数字
A,B, . . . ,Z	英大文字
a,b, . . . ,z	英小文字
	スペース
+	正符号
-	負符号 (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
$	通貨記号
,	コンマ (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
”	引用符
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン

COBOL ワード (* COBOL word). 「*ワード (word)* 」を参照。

*** CONFIGURATION SECTION.** ENVIRONMENT DIVISION のセクションであり、ソース・プログラムとオブジェクト・プログラムの全体的な仕様およびクラス定義を記述する。

CONSOLE. オペレーター・コンソールと関連する COBOL 環境名。

D

DATA DIVISION (データ部). COBOL プログラム、クラス定義、またはメソッド定義の主要な 4 つのコンポーネントの 1 つ。DATA DIVISION では、オブジェクト・プログラム、クラス、またはメソッドによって処理されるデータが記述される。すなわち、その中で使用するファイルと含められるレコード、必要となる内部作業用ストレージ・レコード、および COBOL 実行単位内の 2 つ以上のプログラムで使用可能にされるデータ。(クラスの DATA DIVISION には WORKING-STORAGE SECTION だけしか入れられない。)

DBCS. 「*2 バイト文字セット (double-byte character set (DBCS))*」を参照。

DLL. 「*ダイナミック・リンク・ライブラリー (dynamic link library (DLL))*」を参照。

DLL アプリケーション (DLL application). インポートしたプログラム、関数、または変数を参照するアプリケーション。

DLL リンケージ (DLL linkage). DLL および NODYNAM オプションを使用してコンパイルされたプログラム内の CALL。CALL は、別個のモジュール内のエクスポートされた名前に解決されるか、または、別個のモジュールに定義されたメソッドの INVOKE に解決される。

do 構造 (do construction). 構造化プログラミングでは、DO ステートメントを使えば、プロシージャー内の複数のステートメントをグループ化できる。COBOL では、インライン PERFORM ステートメントが同様に機能する。

do-until. 構造化プログラミングにおいて、do-until ループは、所定の条件が真になるまでに少なくとも 1 回は実行される。COBOL では、TEST AFTER 句を PERFORM ステートメントで使用すれば、同様に機能する。

do-while. 構造化プログラミングにおいて、do-while ループは、所定の条件が真になる場合、またはなっている間に実行される。COBOL では、TEST BEFORE 句を PERFORM ステートメントで使用すれば、同様に機能する。

E

EBCDIC (拡張 2 進化 10 進コード) (* EBCDIC (ExtendedBinary-Coded Decimal Interchange Code)). 8 ビット・コード化文字をベースとするコード化文字セット。

EBCDIC 文字 (EBCDIC character). EBCDIC (拡張 2 進化 10 進コード) セットに含まれているいずれかの記号。

ENVIRONMENT DIVISION (環境部). COBOL プログラム、クラス定義、またはメソッド定義の主要な 4 つのコンポーネントの 1 つ。ENVIRONMENT DIVISION では、ソース・プログラムがコンパイルされるコンピューターと、オブジェクト・プログラムが実行されるコンピューターを記述する。この部では、ファイルの論理概念とそのレコードの間のリンケージ、およびファイルが保管される装置の物理的局面を提供する。

EXE. 「*実行可能ファイル (executable file (EXE))*」を参照。

Extensible Markup Language. 「*XML*」を参照。

F

*** FILE SECTION.** DATA DIVISION のセクションであり、ファイル記述記入項目、ソート・マージ・ファイル記述記入項目、および関連するレコード記述が入っている。

FILE-CONTROL. ENVIRONMENT DIVISION 段落の名前であり、この中では、特定のソース・プログラムのデータ・ファイルが宣言される。

I

IBM COBOL 拡張部分 (IBM COBOL extension). ANSI 規格で記述されるもの以外で、IBM コンパイラーがサポートする特定の COBOL 構文とセマンティクス。

ID (* identifier). データ項目に名前を付けるための文字ストリングと区切り文字の構文的に正しい組み合わせ。関数ではないデータ項目を参照する際には、参照の固有性を保つために必要に応じて、ID は、修飾語、添え字、参照修飾子を伴ったデータ名から構成される。関数であるデータ項目を参照する際には、関数 ID が使われる。

IDENTIFICATION DIVISION (見出し部). COBOL プログラム、クラス定義、またはメソッド定義の主要な 4 つ

のコンポーネントの 1 つ。IDENTIFICATION DIVISION では、プログラム名、クラス名、またはメソッド名を識別する。IDENTIFICATION DIVISION には、作成者名、インストール、または日付を含めることができる。

IGZCBSO. Enterprise COBOL ブートストラップ・ルーチン。このルーチンは、Enterprise COBOL プログラムが含まれているモジュールとリンク・エディットしなければならない。

*** INPUT-OUTPUT SECTION.** ENVIRONMENT DIVISION のセクションであり、オブジェクト・プログラムまたはメソッドに必要なファイルおよび外部メディアに名前を付け、オブジェクト・プログラムまたはメソッド定義の実行時にデータを伝送または処理するために必要な情報を提供する。

ISPF. 「*対話式システム生産性向上機能 (Interactive System Productivity Facility (ISPF))*」を参照。

is-a. 継承階層におけるクラスおよびサブクラスの特徴を表す関係。あるクラスに対して is-a 関係を持つサブクラスは、そのクラスから継承する。

I-O-CONTROL. ENVIRONMENT DIVISION 段落の名前。この段落では、再実行開始点についてのオブジェクト・プログラム要件、複数データ・ファイルによる同じ区域の共用、および単一入出力装置上の複数のファイル・ストレージが指定される。

I-O-CONTROL 記入項目 (* I-O-CONTROL entry). ENVIRONMENT DIVISION の I-O-CONTROL 段落内の記入項目であり、プログラム実行中に指定のファイルへのデータの伝送と処理を行うために必要な情報を提供する文節が入っている。

K

K. 記憶容量に関連して使われるときは、2 の 10 乗。10 進表記では 1024。

L

*** LINAGE-COUNTER.** ページ本体内の現在位置を指す値を収めた特殊レジスター。

LINKAGE SECTION. 呼び出されるプログラムの DATA DIVISION の中にあるセクションであり、ここには呼び出しプログラムから使用可能なデータ項目が記述される。呼び出しプログラムと呼び出されるプログラムの両方がこれらのデータ項目を参照することができる。

*** LOCAL-STORAGE SECTION.** DATA DIVISION のセクションであり、VALUE 文節で割り当てられた値に応じて、呼び出し単位で割り振りまたは解放が行われるストレージを定義する。

M

make ファイル (make file). アプリケーションに必要なファイルのリストが収められたテキスト・ファイル。make ユーティリティーはこのファイルを使用して、ターゲット・ファイルを最新の変更で更新する。

O

*** OBJECT-COMPUTER.** ENVIRONMENT DIVISION にある段落の名前であり、ここではオブジェクト・プログラムが実行されるコンピューター環境が記述される。

ODBC. 「*オープン・データベース・コネクティビティー (Open Database Connectivity (ODBC))*」を参照。

ODO オブジェクト (ODO object). 次の例では、X が OCCURS DEPENDING ON 文節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE SECTION
01  TABLE-1.
    05  X                   PICS9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

ODO オブジェクトの値によって、テーブル内の ODO サブジェクトの数が決まる。

ODO サブジェクト (ODO subject). 上記の例では、Y が OCCURS DEPENDING ON 文節のサブジェクト (ODO サブジェクト) である。テーブル内の ODO サブジェクトの数である Y の値は、X の値によって決まる。

P

private. ファクトリー・データまたはインスタンス・データに適用されるため、そのデータを定義するクラスのメソッドだけがアクセス可能である。

PROCEDURE DIVISION (手続き部). COBOL プログラム、クラス定義、またはメソッド定義の主要な 4 つのコンポーネントの 1 つ。PROCEDURE DIVISION では、問題を解決するための命令が記述される。プログラムまたはメソッドの PROCEDURE DIVISION には、命令ステートメント、条件ステートメント、コンパイラー指示ステートメント、段落、プロシージャー、およびセクションを含めることができる。クラスの PROCEDURE DIVISION には、メソッド定義だけが入れられる。

PROCEDURE DIVISION の終わり (* end of PROCEDURE DIVISION). COBOL ソース・プログラムにおいて、それ以後にはプロシージャーが存在しない物理的な位置。

Q

QSAM (待機順次アクセス方式)(queued sequential access method). 基本順次アクセス方式 (BSAM) の拡張版。この方式を使用する場合、処理を待っている入力データ・ブロック、または処理が終わって補助記憶装置または出力装置への転送を待っている出力データ・ブロックのキューが形成される。

S

SBCS. 「*1 バイト文字セット (single-byte character set (SBCS))*」を参照。

*** SOURCE-COMPUTER.** ENVIRONMENT DIVISION にある段落の名前であり、ここではソース・プログラムがコンパイルされるコンピューター環境が記述される。

SPECIAL-NAMES. ENVIRONMENT DIVISION にある段落の名前。この段落では、環境名がユーザー指定の簡略名と関連付けられる。

STL ファイル・システム (STL file system). 標準言語ファイル・システムは、COBOL および PL/I の固有のワークステーション・ファイル・システムである。このシステムは、順次ファイル、相対ファイル、および索引付きファイルをサポートする (特に明示のない限り、完全な ANSI 85 COBOL 標準の入出力言語と、*Enterprise COBOL 言語解説書* に記載されている拡張部分もすべて含まれる。)

U

Unicode. 現代世界の各国の言語で記述されるテキストの交換、処理、表示をサポートする汎用文字エンコード標準。UTF-8、UTF-16、UTF-32 など、Unicode を表現する複数のエンコード・スキームがある。Enterprise COBOL では、国別データ型の表現として、ビッグ・エンディアン形式の UTF-16 (CCSID 01200) を使用する Unicode がサポートされている。

UPSI スイッチ (UPSI switch). ハードウェア・スイッチの機能を果たすプログラム・スイッチ。UPSI-0 ～ UPSI-7 の 8 つのスイッチがある。

V

VSAM ファイル・システム (VSAM file system). COBOL 順次編成、相対編成、および索引付き編成をサポートするファイル・システム。このファイル・システムは、IBM VisualAge COBOL の一部として利用可能であり、z/OS のようなリモート・システムでファイルの読み取り / 書き込みを行うために使用できる。

W

*** WORKING-STORAGE SECTION.** 独立項目または作業用ストレージ・レコードあるいはその両方から構成される、作業用ストレージ・データ項目を記述する DATA DIVISION。

X

x. PICTURE 文節内の記号であり、コンピューターの有する文字セットの任意の文字を含めることができる。

XML. Extensible Markup Language の略。マークアップ言語を定義するための標準メタ言語。SGML から派生した、SGML のサブセットである。XML では、SGML の複雑で使用頻度の低い部分が省略され、文書タイプを扱うアプリケーションの作成、構造化情報の作成および管理、異種コンピューター・システム間での構造化情報の伝送および共用がはるかに容易になっている。XML を使用するときに、SGML で必要とされるような堅固なアプリケーションや処理は不要である。XML は、World Wide Web Consortium (W3C) の主催で開発された。

XML 型定義 (XML type definition). あるクラスの文書に対する文法を規定するマークアップ宣言を含む、または指示する XML エレメント。この文法は、文書タイプ定義または DTD とも呼ばれる。

XML 宣言 (XML declaration). 使用する XML のバージョンや文書のエンコード方式など、XML 文書の特性を指定する XML テキスト。

リソース・リスト

Enterprise COBOL for z/OS and OS/390

コンパイラーおよび実行時プログラム 移行ガイド、GC88-9119

カスタマイズ・ガイド、GC88-9119

Debug Tool Reference Manual and Messages、SC27-1575

Debug Tool User's Guide、SC27-1573

Fact Sheet、GC27-1407

言語解説書、SC88-9117

Licensed Program Specifications、GC27-1411

プログラミング・ガイド、SC88-9121

関連資料

VisualAge COBOL

Fact Sheet、GC26-9052

ご使用になる前に、GD88-7042

言語解説書、SD88-7071

プログラミング・ガイド、SC88-8665

ビジュアル・ビルダー 使用者の手引き、SD88-7072

COBOL Set for AIX

Fact Sheet、GC26-8484

Getting Started、GC26-8425

言語解説書、SD88-7071

LPEX User's Guide and Reference、SC09-2202

Program Builder User's Guide、SC09-2201

プログラミングの手引き、SC88-7491

CICS Transaction Server

アプリケーション・プログラミング・ガイド、SC88-7689

アプリケーション・プログラミング・リファレンス、SC88-7690

カスタマイズ・ガイド、SC88-7686

外部インターフェース・ガイド、SD88-7026

z/OS C/C++

ランタイム・ライブラリー・リファレンス、SA88-8515

DB2 UDB (OS/390 および z/OS 版)

アプリケーション・プログラミングおよび SQL ガイド、SC88-8763

コマンド解説書、SC88-8764

SQL 解説書、SC88-8772

z/OS DFSMS

Access Method Services for Catalogs、SC26-7394

Checkpoint/Restart、SC26-7401

Macro Instructions for Data Sets、SC26-7408

Program Management、SC27-1130

Using Data Sets、SC26-7410

ユーティリティー、SC88-8979

DFSORT

アプリケーション・プログラミングの手引き、SC88-7061

導入およびカストマイズ、SC88-7163

IMS/ESA

© Copyright IBM Corp. 1991, 2001

Application Programming: Client Server Object Manager Client/Server Application Programming Guide and Reference、SC26-3483

Application Programming: Client Server Object Manager データストア・アプリケーション・プログラミングの手引きと解説書、SC88-7498

Application Programming: Database Manager Summary、SC26-8037

アプリケーション・プログラミング: データベース管理プログラム、SC88-7552

アプリケーション・プログラミング 設計の手引き、SC88-7542

アプリケーション・プログラミング EXEC DLI コマンド (CICS および IMS)、SC88-7554

アプリケーション・プログラミング トランザクション管理プログラム、SD88-7121

z/OS ISPF

ダイアログ開発者ガイドとリファレンス、SC88-8964

ユーザーズ・ガイド 第 1 巻、SC88-8965 および第 2 巻、SC88-8966

z/OS 言語環境プログラム

概念、SA88-8555

カスタマイズ、SA88-8552

デバッグのガイド、GA88-8548

プログラミング・ガイド、SA88-8549

プログラミング・リファレンス、SA88-8550

ランタイム・メッセージ、SA88-8554

ランタイム マイグレーション・ガイド、GA88-8553

ILC アプリケーションの作成、SA88-8551

z/OS MVS

JCL Reference、SA22-7597

JCL ユーザーズ・ガイド、SA88-8570

System Commands、SA22-7627

z/OS UNIX システム・サービス

UNIX システム・サービス コマンド解説書、SA88-8641

UNIX システム・サービス プログラミング: アセンブラー呼び出し可能サービス 解説書、SA88-8642

UNIX システム・サービス ユーザーズ・ガイド、SA88-8640

z/OS TSO/E

コマンド解説書、SA88-8628

ユーザーズ・ガイド、SA88-8638

z/Architecture

Principles of Operation、SA22-7832

Unicode および文字表現

Character Data Representation Architecture Reference and Registry、SC09-2190

OS/390 Unicode サポート: 変換サービスの使用法、SD88-6163、publibfp.boulder.ibm.com/pubs/pdfs/os390/cunuge00.pdf (英文資料)

Program Directory for OS/390 Support for Unicode、GI10-9760、publibfp.boulder.ibm.com/pubs/pdfs/os390/cunpde00.pdf (英文資料)

ソフトコピー資料 (z/O 版)

次のコレクション・キットには、z/OS および関連製品の文献が含まれています。

IBM オンライン・ライブラリー MVS, OS/390, z/OS コレクション・キット CD-ROM、SK88-9002-34

オンライン資料

Java 言語仕様、第 2 版 (Gosling 他著)、
http://software.nikkeibp.co.jp/software/books/books90.html

Java Native Interface、
http://java.sun.com/j2se/1.3/ja/docs/ja/guide/jni/

Java on the OS/390 platform、
www-1.ibm.com/ servers/eserver/zseries/software/java/

The Java 2 Enterprise Edition Developer's Guide、
java.sun.com/j2ee/j2sdkee/devguide1_2_1.pdf

Unicode、www.unicode.org

XML 仕様、www.w3c.org/XML/

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アクセス方式サービス・プログラム
  代替索引の事前作成 193
  VSAM データ・セットのロード 178
  z/OS での VSAM データ・セットの定義 186
アセンブラー
  プログラム
    からのコンパイル 278
    からの呼び出し (CICS での) 409
    リスト 333, 599
  LIST オプションから 599
  PROCEDURE DIVISION の拡張 386
値の割り当て 29
アドレス
  入り口点アドレスの受け渡し 452
  増分 460
  ヌル値 460
  プログラム間での受け渡し 460
アドレッシング・モード 39
暗黙の範囲終了符号 24
移植、プログラムの 44
異常終了、コンパイル時 325
一括表示表現 592
入り口点
  入り口アドレスの受け渡し 452
  代替 453
  プロシージャー・ポインター・データ項目 452
  ENTRY ラベル 438, 453
印刷されるページのサイズ、制御 147
インスタンス
  削除 518
  生成 516
  定義 491
インスタンス・データ
  それをアクセス可能にする 506
  定義 491, 498, 522
  private 499
インスタンス・メソッド
  オーバーライド 504
  多重定義 505
  定義 491, 500, 522
  呼び出し、オーバーライドした 516
インスタンス・メソッドの多重定義 505
インライン PERFORM 91
受け取りフィールド 97
埋められていないトラック 144
ウィンドウ化日付フィールド 214
受け渡し、プログラム間でのデータの
  アドレス 460
  オプションの考慮事項 40
  使用される言語 456
  呼び出されるプログラム 456
  呼び出しプログラム 456
  BY CONTENT 455
  BY REFERENCE 455
  BY VALUE 455
  EXTERNAL データ 464
  JAVA との 540
  JNI サービス 536
  OMITTED 引き数 457
英字データ
  国別との比較 122
英数字データ
  国別との比較 122
英数字の日付フィールドの縮小 587
英数字リテラル
  英数字から DBCS への変換 637
  混合 DBCS/EBCDIC の変換 637
  2 バイト文字を含む 637
  DBCS から英数字への変換 640
エラー
  行順次ファイルの処理エラー 201
  コンパイラー・オプションの競合 313
  算術 249
  処理 247
  処理、QSAM ファイル 149
  入出力の処理 131
  メッセージ、コンパイラー 294
    ソース・リストへの組み込み 373
    端末への送信 283
    フラグを立てる重大度の選択 373
  メッセージ・テーブル、例 72
  リスト 287
  VSAM ファイルの処理エラー 183
エラー・メッセージ
  形式 295
  コンパイラー指示 294
  作成する重大度レベルの判別 328
  重大度レベル 296
  訂正 293
  出口モジュールからの 667
  リストにおける位置 295
  リストの生成 294
エンクレーブ 437
エンコード
  言語文字 115
  説明 117
  XML 文書 240
オーバーフロー条件 248, 259
オーバーライド
  インスタンス・メソッド 504
  ファクトリー・メソッド 527
オープン、ファイルの
  行順次 198
  マルチスレッド化シリアライゼーション 485
  QSAM 145
  VSAM 176, 177
起こりうる問題、日付フィールドでの 587
大文字 105
送り出しフィールド 97
オブジェクト
  削除 518
  作成 516
  定義 491
オブジェクト指向 COBOL
  アプリケーションの作成 308
  コンパイルおよびリンク 307
  実行 309
  DLL 480
  DYNAM コンパイラー・オプションの制約事項 326
  OO プログラムの書き込み 491
オブジェクト・インスタンス、定義 491
オブジェクト・インスタンスの解放 518
オブジェクト・コード
  コンパイルおよびリスト 287
  作成 284
  生成 319
  80 桁カードでの作成 323
オブジェクト・リファレンス
  型式化 512
  設定 513
  汎用 512
  比較する 513
  ローカルからグローバルへの変換 517
オプショナル・ファイル 146, 177

[カ行]

ガーベッジ・コレクション 518
外部 10 進数データ項目 46
外部クラス名 497, 511

© Copyright IBM Corp. 1991, 2001

外部ファイル 16, 464
外部浮動小数点データ項目 47
階層、z/OS でのコンパイラー・オプションの 286
階層ファイル・システム (HFS)
環境変数によるファイルの定義 129
コンパイラー・データ・セット 266
その中での DLL の探索順序 475
ACCEPT によるファイルの読み取り 34
DISPLAY によるファイルの書き込み 35
QSAM によるファイルの処理 157
回転位置感知機構 144
開発、CICS 用のプログラム 407, 408
回避、コーディング・エラーの 591
カウント、データ項目の 104
書き込み、レコードのブロックの 142
拡張 IF ステートメント 83
拡張モード 43, 619
型式化オブジェクト・リファレンス 512
カタログ式プロシージャー
コンパイル (IGYWC) 265
コンパイル、プリリンク、リンク・エディット (IGYWCPL) 270
コンパイル、プリリンク、リンク・エディット、実行 (IGYWCPLG) 271
コンパイル、プリリンク、ロード、実行 (IGYWCPG) 273
コンパイル、リンク・エディット、実行 (IGYWCLG) 268
コンパイル、ロード、実行 (IGYWCG) 269
コンパイルおよびリンク・エディット (IGYWCL) 267
コンパイル用 JCL 264
プリリンクおよびリンク・エディット (IGYWPL) 272
各国語
実行時の使用 113
各国語サポート 331
仮定による世紀ウィンドウ、非日付用の 575
可変位置グループ 632
可変位置データ項目 632
可変長テーブル 74
可変長レコード
ソート 211
OCCURS DEPENDING ON (ODO) 文節 596
QSAM 136, 137
VSAM 166, 167, 173
空の VSAM ファイルのオープン 177
環境変数
行順次ファイルの定義 197
コピーブック 361
環境変数 *(続き)*
設定 299
設定およびアクセス 429
設定およびアクセスの例 430
ファイルの定義、例 12
ファイルを割り振るための使用 129
COBOPT 300
LIBPATH 429
library-name 300, 360
QSAM ファイルの定義 150
SYSLIB 300
text-name 300, 360
_CEE_RUNOPTS 429
_IGZ_SYSOUT 429
環境名 9
関数ポインター・データ項目
対比プロシージャー・ポインター 453
呼び出し DLL プログラム 例 478
CALL ステートメント 453
JNI サービスのアドレッシング 673
SET 関数ポインター 452
完全な日付フィールド拡張
利点 566
簡略名
SPECIAL-NAMES 段落 9
完了コード、ソート 215
関連データ・ファイル
作成 284
記号、LIST および MAP 出力で使用される 385
基本クラスター名 189
基本ライブラリー 283
逆順、テープ・ファイルの 146
逆方向分岐 592
行、テーブルの 66
行順次ファイル
オープン 198
書き込み 198
許可される制御文字 196
クローズ 200
再オープン防止のためのクローズ 198
ソート / マージ 203
入出力エラー処理 201
入出力ステートメント 198
ファイルの処理 195
ブロック化 16
読み取り 198
レコードの追加 199
レコードの読み取り 199
DATA DIVISION 記入項目 196
ENVIRONMENT DIVISION 記入項目 195
z/OS のもとでの
環境変数 197
ジョブ制御言語 (JCL) 197
行順次ファイル *(続き)*
z/OS のもとでの *(続き)*
定義 197
ファイルの作成 197
DD ステートメント 197
行番号 381
共用
データ 450, 464
JAVA との 540
ファイル 16, 450, 464
国別
リテラル 116
国別条件 88
国別データ
英字または英数字との比較 122
グループとの比較 122
計算および置換 104
結合 95
参照変更 102
指定 116
条件式における 88, 121
数値との比較 122
分割 97
変換 118
中国語 GB 18030 121
例 119
例外 119
UTF-8 120
ACCEPT による入力 34
DISPLAY による出力 35
DISPLAY-OF 組み込み 118
LENGTH OF 特殊レジスター 110
LENGTH 組み込み関数 110
MOVE ステートメント 32
NATIONAL-OF 組み込み 118
XML 文書における 240
組み込み MAP 要約 376, 383
組み込みエラー・メッセージ 373
組み込み関数
コーディングの簡略化 609
参照修飾子としての 103
紹介 37
数字関数
型 (整数、浮動小数点、混合) 55
言語環境プログラム呼び出し可能サービスとの違い 57
同等の言語環境プログラム呼び出し可能サービス 56
ネストされた 56
引き数としてのテーブル・エレメント 56
引き数としての特殊レジスター 56
用途 55
例 55
中間結果 625, 628
データ項目の評価 107

組み込み関数 *(続き)*
テーブル・エレメントの処理 80
ネスト 38
文字データ項目の変換 105
例
ANNUITY 60
CHAR 108
CURRENT-DATE 59
DISPLAY-OF 119
INTEGER 103
INTEGER-OF-DATE 59
LENGTH 59, 109, 110
LOG 60
LOWER-CASE 105
MAX 59, 81, 108, 109
MEAN 60
MEDIAN 60, 81
MIN 103
NATIONAL-OF 119
NUMVAL 106
NUMVAL-C 59, 106
ORD 108
ORD-MAX 81, 108
PRESENT-VALUE 59
RANGE 60, 81
REM 60
REVERSE 106
SQRT 60
SUM 81
UPPER-CASE 105
WHEN-COMPILED 110
DATEVAL 583
UNDATE 584
組み込み相互参照
説明 377
例 400
組み込みの CICS トランスレーター 413
クライアント
定義 509
クラス
インスタンス・データ 498
オブジェクト 524
JNI の参照の取得 536
定義 491, 495
名前
外部 497, 511
プログラムにおいて 496
ファクトリー・データ 525
クラス条件
コーディング 88
数値 53
テスト 88, 367
クラスター、VSAM 186
グループ項目
可変位置 632
国別との比較 122
クローズ、ファイルの
行順次 200
マルチスレッド化シリアライゼーション 485
QSAM 148
マルチスレッド化 149
VSAM 183
マルチスレッド化 183
クローズ、ファイルの自動的な
行順次 200
QSAM 148
VSAM 183
グローバル名 450
ケース構造 85
計算
算術データ項目 594
添え字 597
重複 593
定数データ項目 593
形式、レコードの
可変長 136, 137, 173
形式 D 136, 137, 162
形式 F 135, 162
形式 S 140
形式 U 141, 162
形式 V 136, 137, 162
固定長 135, 173
スパン 140
不定 141
QSAM ASCII テープの場合 162
形象定数
定義 29
としての国別文字 117
継承の階層の定義 493
継続
記入項目 222
構文検査 319
検査、データの 104
検査、有効データの
条件式 88
数値 53
限界、コンパイラーの 15
言語環境プログラム呼び出し可能サービス
概要 611
各国語サポート 612
組み込み関数との違い 57
サンプル・リスト 612
条件処理 611
使用例 614
数学 611
数学用 56
対応する組み込み関数 56
タイプ 611
等価の組み込み関数 56
動的ストレージ・サービス 611
日時計算用 56
言語環境プログラム呼び出し可能サービス *(続き)*
日時の計算 611
フィードバック・コード 613
メッセージ処理 612
戻りコード 613
CALL ステートメントを使用した呼び出し 613
RETURN-CODE 特殊レジスター 613
言語間通信
サブプログラム 437
マルチスレッド化 437
CICS のもとで 409
COBOL および Java 間の 535
PL/I タスク 437
言語間通信 (ILC) 437
言語機能、デバッグ用の
DISPLAY ステートメント 366
検出
コンパイルの日付 110
最大または最小データ項目の 108
データ項目の長さ 110
コーディング
インスタンス・メソッド 500, 522
技法 15, 591
クライアント 509
クラス定義 495
決定 83, 88
効率的 591
コンストラクター・メソッド 525
サブクラス定義 520
出力 408
条件テスト 88
相互運用可能データ型 540
テーブル 65
テスト条件 88
入出力ステートメント
行順次ファイル用 198
QSAM ファイル用 145
VSAM ファイル用 174
入出力の概要 128
入力 408
ファイル入出力の概要 125
ファクトリー定義 524
ファクトリー・メソッド 525
ループ 88, 91
CICS 用のプログラムの制約事項 408
DATA DIVISION (データ部) 15
ENVIRONMENT DIVISION (環境部) 9
EVALUATE ステートメント 85
IDENTIFICATION DIVISION (見出し部) 7
IF ステートメント 83
IMS のもとで実行されるプログラム 425

コーディング *(続き)*
IMS 用のプログラムの制約事項 425
OO プログラム 491
PROCEDURE DIVISION (手続き部) 21
コード
コピー 609
最適化 599, 601
コード・ページ
指定 319
ユーロ通貨サポート 63
更新、VSAM レコードの 180
構造化プログラミング 592
構文エラー
NOCOMPILE コンパイラー・オプションによる検出 371
構文解析
XML 文書 225, 227
効率、コーディングの 591
互換性のある日付
比較における 571
MLE を使用した 572
互換モード 43, 619
固定小数点演算
指数 622
比較 61
評価 60
評価例 62
固定小数点データ
外部 10 進数 46
固定小数点データと浮動小数点データ間での変換 50
使用計画 594
中間結果 621
パック 10 進数 48
2 進数 47
固定世紀ウィンドウ 567
固定長レコード
形式 173
QSAM 135
VSAM 166, 167, 173
コピーブック
使用 609
説明 360
探索 304, 361
ユーザー提供モジュールからの入手 657
コピー・ライブラリー
検索順序 361
指定 283
データ・セット 280
例 610
COPY ステートメント 360
SYSLIB 283
UNIX 検索順序 300, 304
混合 DBCS/EBCDIC リテラル
英数字から DBCS への変換 637
DBCS から英数字への変換 640
コンパイラー
エラー・メッセージのリストの生成 294
制限 15
中間結果の計算 620
UNIX シェルでの呼び出し 301
コンパイラー指示ステートメント
概要 24
説明 360
LIST 24
コンパイラー・オプション
競合 313
コンパイラー呼び出しでの 380
シグニチャー・バイト 389
指定 285
PROCESS (CBL) ステートメント 286
状況 380
デバッグ用 370
バッチ階層例 292
パフォーマンスについての考慮事項 603
有効な 389
優先順位 286
ADATA 315
APOST 343
ARITH 316
AWO 316
AWO およびパフォーマンス 603
BUFSIZE 317
CICS 317
CICS のもとで 411
COBOL 85 標準のための設定値 313
CODEPAGE 319
COMPILE 319
CURRENCY 320
DATA 321
DATEPROC 322
DBCS 323
DECK 323
DIAGTRUNC 324
DLL 324
DUMP 325
DYNAM 326, 603
EXIT 327, 657
EXPORTALL 327
FASTSRT 217, 327, 603
FLAG 328, 373
FLAGSTD 329
IMS および CICS のもとでの 410
IMS で推奨される 425
INTDATE 330
LANGUAGE 331
コンパイラー・オプション *(続き)*
LIB 332
LINECOUNT 332
LIST 333, 377
MAP 333, 376, 377
NAME 334
NOCOMPILE 371
NOFASTSRT 219
NOSOURCE 380
NSYMBOL 335
NUMBER 336, 378
NUMPROC 336
NUMPROC(PFD) 603
NUMPROC(PFD|NOPFD|MIG) 52
OBJECT 338
OFFSET 338
OPTIMIZE 339, 599, 603
OUTDD 340
PGMNAME 340
QUOTE 343
RENT 343, 603
RMODE 344
RMODE およびパフォーマンス 603
SEQUENCE 345, 372
SIZE 346
SOURCE 346, 377, 380
SPACE 347
SQL 347, 421
SSRANGE 348, 372, 603
TERMINAL 349
TEST 349, 380, 402, 603
THREAD 352
TRUNC 353
TRUNC(STD|OPT|BIN) 603
TSO での指定 287
UNIX での指定 300
VBREF 356, 377
WORD 357
XREF 357, 375
YEARWINDOW 358
ZWB 359
z/OS での指定 287
コンパイラー・データ・セット
コンパイルに必要な 280
入出力 280
cob2 を使用した 305
HFS における 265, 275
SYSADATA (ADATA レコード) 284
SYSDEBUG (デバッグ・レコード) 285
SYSIN (ソース・コード) 282
SYSJAVA 285
SYSLIB (ライブラリー) 283
SYSLIN (オブジェクト・コード) 284
SYSOUT (リスト) 283

コンパイラー・データ・セット *(続き)*
SYSPUNCH (オブジェクト・コード) 284
SYSTERM (メッセージ) 283
コンパイラー・メッセージ
分析 585
コンパイラー・リスト
入手 377
コンパイル
アセンブラー・プログラムからの 278
カタログ式プロシージャーによる 264
コンパイル 265
コンパイル、プリリンク、リンク・エディット 270
コンパイル、プリリンク、リンク・エディット、実行 271
コンパイル、プリリンク、ロード、実行 273
コンパイル、リンク・エディット、実行 268
コンパイル、ロード、実行 269
コンパイルおよびリンク・エディット 267
結果 287
シェル・スクリプトの使用 306
制御 285
データ・セット 280
統計 381
バッチ 289
cob2 コマンドの使用 301
COBOL 85 標準 313
HFS ファイル 266
Java/COBOL アプリケーション 307
JCL (ジョブ制御言語) による 264
TSO のもとでの 276
UNIX のもとでの 299
z/OS のもとでの 263
コンパイルおよびリンク、UNIX シェルでの 301
コンパイル時エラー・メッセージ
作成する重大度レベルの判別 328
ソース・リストへの組み込み 373
フラグを立てる重大度の選択 373
コンパイル時ダンプの生成 325
コンパイル時についての考慮事項
エラー・メッセージの重大度 296
コンパイラー指示エラー 294
コンパイルおよびリンク・ステップの表示 305
表示後のコンパイルおよびリンク・ステップの実行 305
コンパイル・エラー・メッセージ
端末への送信 283
出口モジュールからの 667
コンピューター、記述 9

[サ行]

再帰呼び出し
コーディング 451
識別 8
LINKAGE SECTION 19
最後に使われた状態 439
再実行依頼、ジョブの 560
再始動
自動 557
据え置き 558
ルーチン 553
再始動、プログラムの 557
最適化
一貫性のあるデータ 595
一括表示表現 592
逆方向分岐の回避 592
構造化プログラミング 592
コンパイラー・オプションの影響 602
指標計算 597
指標付け 595
添え字計算 597
添え字付け 595
重複計算 593
テーブル・エレメント 595
定数計算 593
定数データ項目 593
到達不能コード 600, 601
トップダウン・プログラミング 592
ネストされたプログラムの統合 600
パフォーマンスの含意 596
パフォーマンスへの影響 592
含まれているプログラムの統合 600
プロシージャー統合 600
未使用データ項目 339, 383
ライン外の PERFORM 592
ALTER ステートメントの回避 592
BINARY データ項目 594
OCCURS DEPENDING ON 596
packed-decimal データ項目 594
最適化プログラム 599, 601
再入可能プログラム 454
作業データ・セット 280
作業用ストレージ
位置とサイズの検出 397
索引
キーの誤りの検出 258
索引付きファイル編成 125, 169
削除、VSAM ファイルからのレコードの 182
作成
オブジェクト 516
オブジェクト・コード 284
関連データ・ファイル 284
SYSJAVA データ・セット 285
z/OS での QSAM ファイル 150, 152
作成 *(続き)*
z/OS での行順次ファイル 197
サブクラス
インスタンス・データ 522
定義 520
サブストリング
参照変更 101
テーブル項目の参照 101
サブプログラム
終了
影響 439
説明 438
手続き部 458
メインプログラム 438
リンケージ 437, 445
共通データ項目 457
算術
エラー処理 249
組み込み関数を使用した 55
コーディングが容易な COMPUTE ステートメント 54
算術演算
MLE を使用した 577, 581
算術式
括弧で囲まれた 55
参照修飾子としての 103
説明 55
非算術ステートメントでの 628
MLE を使用した 581
算術比較 61
算術評価
固定小数点と浮動小数点の対比 60
精度 619
中間結果 619
データ形式の変換 50
パフォーマンスについてのヒント 594
優先順位 55, 621
例 60, 62
参照修飾子
としての組み込み関数 103
としての算術式 103
としての変数 101
参照変更
国別データ 102
テーブル 68, 101
範囲外の値 102
例 102
サンプル・プログラム 733
シグニチャー
固有でなければならない 500
定義 500
シグニチャー・バイト
有効なコンパイラー・オプション 389
DATA DIVISION 391
ENVIRONMENT DIVISION 392
PROCEDURE DIVISION 392, 394

指数
固定小数点演算で評価される 622
パフォーマンスについてのヒント 595
浮動小数点演算で評価される 627
システム決定のブロック・サイズ 142, 282
システム日付
CICS のもとで 408
システム名 9
システム・ダンプ 248
事前初期設定
マルチスレッド化 484
実行時
パフォーマンスについての考慮事項 591
ファイル名の変更 13
マルチスレッド化制限 488
実行単位
説明 437
マルチスレッド化での役割 482
実動デバッグ 351
自動再始動 557
指標
データ項目 70
テーブル 68
範囲検査 372
指標付け
添え字付けより望ましい 595
テーブル 69
例 73
指標名添え字付け 69
終了 439
主記憶域のバッファーへの割り振り 317
縮小、英数字日付の 587
出力
行順次ファイル用のコーディング 198
データ・セット 283
ファイル 125
CICS でのコーディング 408
QSAM ファイル用のコーディング 145
VSAM ファイル用のコーディング 174
z/OS でのコンパイラーからの 280
出力ファイル、cob2 での 306
出力プロシージャー
使用 208
制約事項 210
有効でない FASTSRT オプション 217
RETURN または RETURN INTO ステートメントが必要 208
順序、評価の
コンパイラー・オプション 313
算術演算子 55, 621
順次記憶装置 126
順次ファイル編成 125
処理
指標付けの使用 73
添え字付けの使用 72
チェーン・リスト 460, 461
テーブル 72
QSAM ファイルのラベル 158
初期化
インスタンス・データ 516
テーブル 71
状況キー
VSAM における重要性 183
条件式
国別データを伴う 88
EVALUATE ステートメント 84
IF ステートメント 83
PERFORM ステートメント 93
条件処理 148, 183, 611
入力または出力プロシージャーでの 210
条件ステートメント
オブジェクト・リファレンス 513
概要 23
リスト 23
EVALUATE ステートメント内の 84
NOT 句を指定した 23
条件テスト 88
条件名 574
照合シーケンス
指定 10
シンボリック文字 12
代替 11
非数値比較 10
文字の序数位置 107
EBCDIC 11
HIGH-VALUE 10
LOW-VALUE 10
MERGE 10
SEARCH ALL 10
SORT 10
常駐モード 39
ジョブ再実行要求 560
ジョブ・ストリーム 437
シリアライゼーション
マルチスレッド化によるファイルの 485
シリアル検索 78
診断、プログラムの 380
診断メッセージ
2000 年言語拡張からの 585
シンボリック定数 593
スイッチ状況条件 88
スイッチとフラグ
説明 88
定義 89
リセット 90
数字組み込み関数
型 (整数、浮動小数点、混合) 55
言語環境プログラム呼び出し可能サービスとの違い 57
同等の言語環境プログラム呼び出し可能サービス 56
ネストされた 56
引き数としてのテーブル・エレメント 56
引き数としての特殊レジスター 56
用途 55
例
ANNUITY 60
CURRENT-DATE 59
INTEGER 103
INTEGER-OF-DATE 59
LENGTH 59, 109
LOG 60
MAX 59, 81, 108, 109
MEAN 60
MEDIAN 60, 81
MIN 103
NUMVAL 106
NUMVAL-C 59, 106
ORD 108
ORD-MAX 81
PRESENT-VALUE 59
RANGE 60, 81
REM 60
SQRT 60
SUM 81
数値条件 88
数値データ
外部 10 進数
USAGE IS DISPLAY 46
外部浮動小数点
USAGE IS DISPLAY 47
国別との比較 122
固定小数点データと浮動小数点データ間での変換 50
固定小数点と浮動小数点の間での形式の変換 50
ゾーン 10 進数
USAGE IS DISPLAY 47
定義 43
内部浮動小数点
USAGE IS COMPUTATIONAL-1 (COMP-1) 49
USAGE IS COMPUTATIONAL-2 (COMP-2) 49
パック 10 進数
USAGE IS COMPUTATIONAL-3 (COMP-3) 48
USAGE IS PACKED-DECIMAL 48
編集記号 44

数値データ *(続き)*
保管形式 45
2 進数
USAGE IS BINARY 47
USAGE IS COMPUTATIONAL (COMP) 47
USAGE IS COMPUTATIONAL-4 (COMP-4) 47
USAGE IS COMPUTATIONAL-5 (COMP-5) 48
PICTURE 文節 43, 44
数値のクラス・テスト 53
数字編集記号 44
数値編集データ項目 44
数学
組み込み関数 55, 60
言語環境プログラム呼び出し可能サービス 57, 611
据え置き再始動 558
スキップ、レコードのブロックの 142
ステートメント
暗黙の範囲終了符号 24
コンパイラー指示 24
条件付き 23
定義 22
ネスト・レベル 381
範囲区切り 23
明示的範囲終了符号 24
命令 22
ストライプ拡張形式 QSAM ファイル 156
ストリング
ヌル終了 459
Java 541
取り扱い 543
ストレージ
管理、言語環境プログラム呼び出し可能サービス 611
ソート時の使用 222
装置
順次 126
直接アクセス 126
マッピング 377
スパン・レコード 140
スパン・レコード形式 140
スライディング世紀ウィンドウ 567
スレッド化についての考慮事項 427
制御
転送 438
ネストされたプログラム内の 447
プログラムのフロー 83
制御インターバル・サイズ (CISZ) のパフォーマンスの考慮事項 193, 607
世紀ウィンドウ
固定 567
スライディング 567
世紀ウィンドウ *(続き)*
非日付用の仮定による 575
静的 call ステートメント 441
静的データ、定義 491
静的データ域、ストレージの割り振り 40
静的メソッド
定義 491
呼び出し 527
制約事項
添え字付け 68
入出力プロシージャー 210
CICS コーディング 10
CICS 用のプログラムのコーディング 408
IMS コーディング 10, 410
IMS 用のプログラムのコーディング 425
セクション
グループ化 94
説明 21
宣言 25
セグメント化 602
設定
スイッチとフラグ 90
ゼロ比較 579
宣言型プロシージャー
EXCEPTION/ERROR 253
マルチスレッド化 254
LABEL 160
USE FOR DEBUGGING 368
ソース・コード
行番号 381, 382, 386
コンパイラー・データ・セット (z/OS) 282
プログラム・リスト 287
リストの説明 377
ソート
ウィンドウ化日付フィールド 214
オリジナル・シーケンスの保持 215
概念 204
可変長レコード 211
完了コード 215
基準 211
行順次ファイル 203
終了 216
出力プロシージャー 208
使用ストレージ 222
制御ステートメントの受け渡し 221
成功 215
説明 203
代替照合シーケンス 214
チェックポイント・リスタート 223
動作の制御 220
特殊レジスター 220
入出力プロシージャーに関する制約事項 210
ソート *(続き)*
入力プロシージャー 207
パフォーマンス 217
必要なデータ・セット、z/OS 210
ファイルの説明 205
複数の 204
マルチタスキング 203
ワークスペース 223
CICS のもとで 224
DD ステートメント、z/OS データ・セットの定義 210
FASTSRT コンパイラー・オプション 217
NOFASTSRT コンパイラー・オプション 219
z/OS のもとでの 210
ソート・ファイル記述 (SD) 記入項目
例 206
ゾーン 10 進数 47
添え字
計算 597
範囲検査 372
添え字付け
参照変更 68
指標名 69
制約事項 68
相対 68
テーブルの処理の例 72
変数 67
リテラル 67
装置
クラス 280
要件 280
相互運用可能データ型 540
相互参照
組み込み 377
データ名とプロシージャー名 375
動詞 377
動詞リスト 356
特殊定義記号 400
プログラム名 400
LIST 357
相対ファイル編成 125
増分、アドレスの 460
属性メソッド 506

［タ行］

対話式システム生産性向上機能 (ISPF) 738
対話式プログラムの例 738
代替入り口点 453
代替索引
作成 187
パス 188, 189
パスワード 185

代替索引 *(続き)*
パフォーマンスについての考慮事項 193
例 189
代替照合シーケンス 11, 214
代替予約語テーブル
指定 357
CICS 416
ダイナミック・リンク・ライブラリー
オブジェクト指向のための作成 309
オブジェクト指向プログラムでの使用 309
コンパイル 470
作成 469
動的呼び出しでの使用 475
に関して 469
プリリンク 472
リンク 471
CALL ID の使用 474
C/C++ プログラムでの使用 479
HFS における探索順序 475
Java とのインターオペラビリティーでの使用 309
Java とのインターオペラビリティーのための 309
OO COBOL アプリケーションで 480
多重継承、許可されていない 494, 520
探索、テーブルの 77
探索、名前宣言の 450
短縮リストの例 379
ダンプ
DUMP コンパイラー・オプションを指定した 287
端末へのメッセージの送信 349
段落
グループ化 94
紹介 22
チェーン・リスト処理 460, 461
チェックポイント
再始動ジョブ制御の例 560
再始動データ・セット 560
生成されるメッセージ 556
制約事項 554
設計 554
単一 554
テープ 556
ディスク 556
複数 554, 556
メソッド 554
レコード・データ・セット 555
COBOL 85 標準 554
DFSORT 時の再始動 223
置換
データ項目 104
QSAM ファイル内のレコード 147
VSAM ファイル内のレコード 182
チューニング考慮事項、パフォーマンス 602, 603
中間結果 619
中国語 GB 18030 データ
処理 121
変換 121
重複計算 593
直接アクセス
記憶装置 (DASD) 193
直接指標付け 69
ファイル編成 126
追加、レコードの
行順次ファイルへの 199
QSAM ファイルへの 147
VSAM ファイルへの 181
通貨記号
使用 63
複数文字 63
ユーロ 63
16 進数リテラル 63
データ
受け渡し 455
グループ化 459
形式、数値タイプ 45
形式の変換 50
結合 95
効率のよい実行 591
数値 43
妥当性検査 53
非互換 53
分割 97
変換、英数字を DBCS に 637
変換、DBCS を英数字に 637
命名 16
レコード・サイズ 16
連結 95
データ域、動的 326
データ記述記入項目 15
データ項目
カウント 104
可変位置 632
逆順、文字の 106
組み込み関数を使用した評価 107
組み込み関数を使用した変換 105
検出、グループ内の最小または最大の 108
サブストリングの参照 101
サブプログラム・リンケージで共通の 457
参照変更 101
指標 68
数値 43
置換 104
分割 97
変換、大文字または小文字への 105
マップ 287
データ項目 *(続き)*
未使用 339, 383
文字から数値への変換 106
連結 95
2 バイト文字を含む英数字 637
DBCS 637
INSPECT を使用した変換 104
Java の型のコーディング 540
データ操作
非数値データ 95
DBCS データ 637
データ定義 382
データ定義属性コード 382
データのグループ化 459
データ名とプロシージャー名相互参照の説明 375
データ・セット
環境変数を使用した定義 129
コンパイルに使用される 280
出力 283
ソース・コード 282
代替データ・セット名 278
チェックポイント・リスタート 560
チェックポイント・レコード 555
名前、代替 279
ファイルと同じ意味で使用 9
SYSADATA 284
SYSDEBUG 285
SYSIN 282
SYSJAVA 285
SYSLIB 283
SYSLIN 284
SYSPRINT 283
SYSPUNCH 284
SYSTERM 283
z/OS でのコンパイルに必要な 280
テーブル
値のロード 70
値の割り当て 71
可変長 74
行 66
組み込み関数 80
効率のよいコーディング 595, 597
参照 68
参照変更 68
指標 68
処理 65
初期化 71
説明 38
添え字 67
探索 77
テーブル記入項目のサブストリングの参照 101
定義 65
同一エレメント仕様 595
動的ロード 70

テーブル *(続き)*
配列との比較 38
深さ 66
ループ 93
列 65
1 次元 65
2 次元 66
3 次元 66
テープ・ファイル
逆順 146
パフォーマンス 143
定義
行順次ファイル
z/OS に対する 197
ソート・ファイル
z/OS のもとでの 210
デバッグ・データ・セット 285
ファイルの概要 12, 125
ライブラリー 283
QSAM ファイル
z/OS に対する 150, 152
VSAM ファイル 186
z/OS に対する 186
定数
計算 593
形象 29
データ項目 593
定様式ダンプ 247
ディレクトリー
エラー・リスト・ファイルが書き込まれる場所 294
パスの追加 304
適合要件
RETURNING 句 515
USING 句 515
出口モジュール
生成されるエラー・メッセージ 667
ライブラリー名の代わりに使用される 661
ロードおよび呼び出し 659
SYSADATA データ・セットのために呼び出される 665
SYSLIB の代わりに使用される 661
SYSPRINT の代わりに使用される 664
テスト
条件 92
数値オペランド 88
データ 88
UPSI スイッチ 88
デッドロック
入出力エラー宣言での 254
デバッグ
実動 351
データ・セットの定義 285
デバッグ *(続き)*
有用なコンパイラー・オプション 370
COBOL 言語機能の使用 366
デバッグ、言語機能の
クラス・テスト 367
デバッグ宣言部分 368
範囲終了符号 366
ファイル状況キー 367
INITIALIZE ステートメント 368
SET ステートメント 368
デバッグ・ツール
最大サポートのためのコンパイラー・オプション 402
説明 365
転送、制御権の
ネストされたプログラム 448
メインプログラムとサブプログラム 438
呼び出されるプログラム 438
呼び出しプログラム 438
COBOL プログラム相互間の 440, 447
COBOL プログラムと非 COBOL プログラムの間での 437
統計
組み込み関数 60
動詞、プログラムで使用される 377
動詞相互参照リスト
説明 377
到達不能コード 600, 601
動的データ域、ストレージの割り振り 40
動的ファイル割り振り 130, 146
動的呼び出し
DLL リンケージでの使用 475
特殊機能指定 9
特殊レジスター
組み込み関数の引き数 56
ADDRESS 456
JNIEnvPtr 535
LENGTH OF 110, 456
SORT-RETURN 215, 216
WHEN-COMPILED 110
XML-CODE 234
XML-EVENT 234
XML-NTEXT 234
XML-TEXT 234
特記事項 755
トップダウン・プログラミング
回避するための構成 592

[ナ行]

内部浮動小数点データ
定義 49
必要なバイト数 49
用途 49
内部ブリッジ
日付処理用 568
利点 566
例 569
長さを検出する、データ項目の 110
名前宣言
探索 450
日時操作
言語環境プログラム呼び出し可能サービス 611
二分探索、テーブルの 79
入出力
エラー後のロジック・フロー 250
エラーの検査 254
概要 125
コーディングの概要 128
処理エラー
行順次ファイル 201
QSAM ファイル 149, 250
VSAM ファイル 183, 250
FASTSRT オプションによる制御 327
入出力コーディング
誤った索引キーの検出 258
エラー処理技法 250
正常操作の検査 254
AT END (ファイル終わり) 句 252
EXCEPTION/ERROR 宣言 253
VSAM 戻りコードの検査 256
入力
行順次ファイル用のコーディング 198
ファイルから 125
CICS でのコーディング 408
QSAM ファイル用のコーディング 145
VSAM ファイル用のコーディング 174
z/OS のもとでのコンパイラーへの 280
入力プロシージャー
コーディング 207
制約事項 210
有効でない FASTSRT オプション 217
RELEASE または RELEASE FROM が必要 207
ヌル終了ストリング 100, 459
年のウィンドウ操作
サポートされない場合 573
制御方法 583
利点 566
MLE アプローチ 567
年フィールド拡張 569
ネストされた COPY ステートメント 609, 661
ネストされた IF ステートメント
コーディング 84

ネストされた IF ステートメント *(続き)*
望ましい EVALUATE ステートメント 84
CONTINUE ステートメント 84
NULL ブランチを伴う 84
ネストされた区切り範囲ステートメント 25
ネストされた組み込み関数 56
ネストされたプログラム
使用する際の規則 447
制御権の転送 447
説明 448
マップ 377, 386
有効範囲、名前の 450
呼び出し 447
ネストされたプログラムの統合 600
ネストされたプログラム・マップ
説明 377
例 386
ネスト・レベル
ステートメント 381
プログラム 381, 386
年末尾型日付フィールド 571

[ハ行]

バイト・ストリーム・ファイル
QSAM による処理 157
配列
COBOL での 38
Java 541
パス名
コピーブックの検索に使用 304, 360
パスワード
代替索引 185
例 185
VSAM ファイル 184
パック 10 進数データ項目
効率的使用 48, 594
説明 48
日付フィールドの起こりうる問題 587
バッチ・コンパイル
コンパイラー・オプション階層例 292
説明 289
バッファー、QSAM 用に獲得 157
バッファーの最適な使用 14
パフォーマンス
コーディング 591
コンパイラー・オプション
AWO 603
DYNAM 603
FASTSRT 603
NUMPROC 52, 603
OPTIMIZE 599, 603
RENT 603
RMODE 603
パフォーマンス *(続き)*
コンパイラー・オプション *(続き)*
SSRANGE 603
TEST 603
TRUNC 353, 603
コンパイラー・オプションの影響 602
最適化プログラム 599, 601
算術計算の計画 594
算術式の使用 595
指数の使用 595
実行時の考慮事項 591
データ使用 594
テープ、QSAM 143
テーブルのコーディング 595
テーブルの処理 597
バッファー・サイズの影響 317
プログラミング・スタイル 592
変数添え字のデータ形式 68
呼び出しの 444
ワークシート 606
AIXBLD ランタイム・オプション 607
APPLY WRITE-ONLY 文節 14
CICS 環境での 607
IMS 環境での 425, 607
OCCURS DEPENDING ON 596
QSAM ファイルのブロック化 142
VSAM ファイルの考慮事項 193
パラメーター
呼び出されるプログラムの中での記述 457
パラメーター・リスト
ADEXIT 用 665
INEXIT 用 659
LIBEXIT 用 663
PRTEXIT 用 664
UNIX におけるメインプログラム 432
範囲区切りステートメント
説明 23
ネストされた 25
範囲終了符号
暗黙 24
デバッグでの補助 366
明示 23, 24
汎用オブジェクト・リファレンス 512
比較条件 88
比較する
オブジェクト・リファレンス 513
国別データ 122
国別と英字または英数字 122
国別とグループ 122
国別と数値 122
日付フィールドの 571
引き数
呼び出しプログラムでの記述 456
OMITTED に関するテスト 457
引き数 *(続き)*
OMITTED の指定 457
OMITTED 引き数に関するテスト 457
引き数を省略するための OMITTED 句 457
非互換データ 53
日付算術演算 581
日付処理、内部ブリッジによる
利点 566
日付操作
組み込み関数 37
日付のウィンドウ操作
サポートされない場合 573
制御方法 583
利点 566
例 574
MLE アプローチ 567
日付の比較 571
日付フィールド
起こりうる問題 587
日付フィールド拡張
説明 569
利点 566
非同期シグナル
マルチスレッド化 488
非日付
MLE を使用した 576
評価、データ項目の内容の
組み込み関数 107
クラス・テスト 53, 88
INSPECT ステートメント 104
表現
データ 53
符号 52
標準ラベル、QSAM 163
標準ラベル形式 160
ピリオド、範囲終了符号としての 24
ファイル
オプショナル・ファイルの説明 146, 177
オペレーティング・システムへの定義 12
概要 126
処理
行順次 195
QSAM 133
VSAM 165
使用法の説明 13
説明 15
ソート・パフォーマンスの向上 217
データ・セットと同じ意味で使用 9
プログラム・ファイルを外部ファイルに関連付ける 9
マルチスレッド化 485
ラベル 163

ファイル *(続き)*
COBOL コーディング
概要 128
入出力ステートメント 145, 174, 198
DATA DIVISION 記入項目 134, 173, 196
ENVIRONMENT DIVISION 記入項目 133, 168, 195
z/OS に対する識別 150, 152, 186, 197
ファイル位置標識 (CRP) 176, 180
ファイル終わり句 (AT END) 252
ファイル拡張子
エラー・メッセージ・リストの場合 294
ファイル記述 (FD) 記入項目 16
ファイル状況キー
エラー処理の場合の設定 131, 367
正常 OPEN かどうかの検査 254, 255
入出力エラーの検査 254
VSAM 戻りコードとともに使用される 256
ファイル状況コード
使用 250
02 180
05 176
35 176
37 145
39 146, 154, 158, 177
49 182
90 143
92 431
ファイルの可用性
CMS のもとでの QSAM ファイルの 146
z/OS のもとでの QSAM ファイル 146
z/OS のもとでの VSAM ファイルの 185
ファイル変換
2000 年言語拡張での 570
ファイル編成
概要 125
行順次 195
索引付き 125, 169
順次 125, 168
相対 125
相対レコード 170
要約テーブル 127
ESDS、KSDS、RRDS の比較 167
QSAM 133
VSAM 166
ファイル名
指定 16
変更 13
ファイル割り振り 129
行順次 197
説明 129
QSAM 150
VSAM 189
ファイル・アクセス・モード
索引付きファイル (KSDS) の場合 172
順次 172
順次ファイル (ESDS) の場合 172
相対ファイル (RRDS) の場合 172
動的 172
パフォーマンスについての考慮事項 193
要約テーブル 127, 168
ランダム 172
例 172
ファクトリー定義、コーディング 524
ファクトリー・セクション、定義 524
ファクトリー・データ
それをアクセス可能にする 525
定義 491, 525
private 525
ファクトリー・メソッド
隠蔽 527
定義 491, 525
プロシージャー・プログラムのラップに使用 528
呼び出し 527
ファクトリー・メソッドの隠蔽 527
深さ、テーブルの 66
含まれているプログラムの統合 600
複合 OCCURS DEPENDING ON
可変位置グループ 632
可変位置データ項目 632
基本形式 631
複合 ODO 項目 631
複数の通貨記号
使用 63
例 64
符号条件 88, 579
符号表現 52
物理
ブロック 125
レコード 16, 125
不定形式レコード形式
説明 141
要求 141
レイアウト 141
QSAM 162
浮動小数点演算
指数 627
比較 61
評価 60
評価例 62
浮動小数点データ
外部浮動小数点 47
固定小数点データと浮動小数点データ間での変換 50
使用計画 594
中間結果 627
内部 49
フラグおよびスイッチ 88
ブランチ、暗黙の 91
プリリンク、カタログ式プロシージャー
コンパイル、プリリンク、リンク・エディット 270
コンパイル、プリリンク、リンク・エディット、実行 271
コンパイル、プリリンク、ロード、実行 273
プリリンクおよびリンク・エディット 272
プログラム
決定
スイッチとフラグ 88
ループ 92
EVALUATE ステートメント 84
IF ステートメント 83
PERFORM ステートメント 92
構造 7
再始動 557
再入可能 454
サブ 438
シグニチャー情報バイト 389
初期化コード 387
診断 380
制限 591
属性コード 386
統計 380
ネスト・レベル 381
メイン 438
cob2 を使用したコンパイル 301
UNIX のもとでのコンパイル 299
UNIX 用の開発 427
z/OS のもとでのコンパイル 263
プログラム終了
ステートメント 439
メインおよびサブプログラムで取られるアクション 439
プログラム処理テーブル 409
プログラム名
大文字小文字の処理 340
指定 7
プログラム名相互参照 400
プロシージャー統合 600
プロシージャー名とデータ名相互参照の説明 375
プロシージャー・ポインター・データ項目
入り口点の入り口アドレス 452
対比関数ポインター 453

プロシージャー・ポインター・データ項目 *(続き)*
呼び出し可能サービスへのパラメーターの受け渡し 452
DLL での 476
SET プロシージャー・ポインター 452
プロセス 482
ブロック化因数 135
ブロック・サイズ
システム決定の 142, 282
ASCII ファイル 162
QSAM ファイル 142
可変長 136
固定長 135
レコード・レイアウト 137
DCB の使用 151
文 22
分離桁符号 44
分割、データ項目の 97
文書化、プログラムの 9
ページ
制御 147
深さ 16
ヘッダー 380
カスタマイズ 380, 381
ベース・ロケーター 382, 384
ヘッダー、リストの 9
変換、データ項目の
大文字または小文字への 105
国別 118
DISPLAY-OF を使用した 118
MOVE を使用した 118
NATIONAL-OF を使用した 118
国別データ
ACCEPT による 34
DISPLAY による 35
組み込み関数を使用した 105
整数への 103
中国語 GB 18030 121
データ形式 50
文字から数値への 106
文字の逆順 106
INSPECT ステートメント 104
UTF-8 120
変換、ファイルの拡張日付形式への例 570
変更
ソース・リストのタイトル 9
ファイル名 13
文字から数値への 106
変数
参照修飾子としての 101
説明 27
変数、環境
library-name 360
保護、VSAM ファイルの 184
ポインター・データ項目
アドレスの受け渡しに使用 460
アドレスの増分 460
説明 38
チェーン・リストの処理に使用 460, 461
ヌル値 460
保持、ソートでのオリジナル・シーケンスの 215

[マ行]

マージ
概念 204
行順次ファイル 203
使用ストレージ 222
制御ステートメントの受け渡し 221
成功 215
説明 203
ファイルの説明 205
マルチタスキング 203
マッピング、DATA DIVISION 項目の 377
マルチスレッド化
概要 481
言語間通信 437
再帰 484
再帰的要件 487
再入可能性 487
再入可能性要件 487
事前初期設定 484
実行時の制約事項 488
制御転送問題 484
データ・セクションの選択 481
OO クライアントにおいて 512
入出力エラー宣言 254
ネストされたプログラム 487
非同期シグナル 488
ファイル入出力使用法の例 486
ファイル入出力のコーディング 485
古いコンパイラー 488
用語 482
リソースへのアクセスの同期化 487
AMODE 設定 488
COBOL に対する制限 487
COBOL プログラム 481
COBOL プログラムの準備 481
EXIT PROGRAM ステートメント 439
GOBACK ステートメント 439
IGZBRDGE 488
IGZEOPT 488
IGZETUN 488
PL/I タスクとの 437
QSAM ファイルのクローズ 149
マルチスレッド化 *(続き)*
STOP RUN ステートメント 439
THREAD コンパイラー・オプション
いつ選択するか 483
制約事項 352
UPSI スイッチ 488
VSAM ファイルのクローズ 183
マルチスレッド環境での実行 352
マルチタスキング
ソート 203
マージ 203
矛盾するコンパイラー・オプション 313
明示的範囲終了符号 24
命名
ファイル 12
プログラム 7
命令ステートメントのリスト 22
メインプログラム
サブプログラム 438
動的 CALL 442
UNIX でのパラメーター・リスト 432
メソッド
インスタンス 500, 522
オーバーライド 504, 527
から値を戻す 504
コンストラクター 525
シグニチャー 500
スーパークラスの呼び出し 516
多重定義 505
ファクトリー 525
ファクトリーの隠蔽 527
呼び出し 514, 527
渡された引き数の取得 503
Java アクセス制御 539
PROCEDURE DIVISION RETURNING 463
メッセージ
コンパイラー指示 294
コンパイラー・エラー
端末への送信 283
コンパイル時エラー
ソース・リストへの組み込み 373
フラグを立てる重大度の選択 373
作成する重大度レベルの判別 328
重大度レベル 296
出口モジュールからの 667
リストの生成 294
メッセージ処理、言語環境プログラム呼び出し可能サービス 612
メモリー・マップ
DSA 386
TGT 386
メモリー・マップ、TGT
例 396
文字の逆順 106
目標、2000 年言語拡張の 565

モジュール、出口
ロードおよび呼び出し 659
戻りコード
オペレーティング・システムに制御権が戻されるとき 463
言語環境プログラム・サービスからのフィードバック・コード 613
コンパイラー 296
CICS ECI からの 411
DB2 からの 420
RETURN-CODE 特殊レジスター 463, 613
VSAM ファイル 256

[ヤ行]

ユーザー定義の条件 88
ユーザー出口作業域 658
ユーザー・ラベル
出口 163
標準 161
QSAM 163
ユーザー・ラベル・トラック 159
ユーロ通貨記号 63
有効データ
数値 53
有効範囲、名前の 450
優先順位
算術演算子 55, 621
z/OS のもとでのコンパイラー・オプション 286
優先順位番号、セグメント化の 602
優先符号 52
用語
VSAM 165
用語、MAP 出力で使用される 384
呼び出し
インスタンス・メソッド 514
オーバーフロー条件 259
言語間 437
言語環境プログラム呼び出し可能サービス 613
言語環境プログラム呼び出し可能サービスへの 613
再帰的 451
静的 441
静的パフォーマンス 444
データの受け渡し 455
動的 442
動的パフォーマンス 444
パラメーターの受け取り 457
引き数の受け渡し 456
ファクトリーまたは静的メソッド 527
リンケージ・セクション 458
例外条件 259
呼び出し *(続き)*
24 ビット・プログラムの AMODE 切り替え 444
31 ビット・アドレッシング・モード 444
CICS の制約事項 409
COBOL UNIX プログラム 427
COBOL プログラム相互間の 437, 440
COBOL プログラムと非 COBOL プログラムの間での 437
JNI サービスへの 535
OMITTED 引き数の指定 457
読み取り、レコードの
行順次ファイルへの 199
ブロック 142
読み取り、VSAM ファイルからのレコードの
順次 180
動的 180
ランダム 180
予約語テーブル
代替、CICS 357, 416

[ラ行]

ラージ・ブロック・インターフェース 143
ライブラリー
定義 283
ディレクトリー記入項目 278
パスの指定 360
BASIS 283
COPY 283
ライン外 PERFORM 91
ラッパー、定義 528
ラッピング、プロシージャー指向プログラムの 528
ラベル
形式、標準 160
処理、QSAM ファイル 158
標準ユーザー 161
ASCII ファイル 163
乱数の生成 57
ランタイム・オプション
影響を与える DATA コンパイラー・オプション 40
AIXBLD 607
ALL31 444
CBLPSHPOP 416
CHECK(OFF) 603
COBOL 85 標準への適合性 313
MSGFILE 340
SIMVRD 170
TRAP
行順次のファイルのクローズ 200
ランタイム・オプション *(続き)*
TRAP *(続き)*
ON SIZE ERROR 249
QSAM のファイルのクローズ 148
VSAM のファイルのクローズ 183
リスト
組み込み MAP 要約 377
組み込みエラー・メッセージ 373
組み込み相互参照 377
ソース・コードの組み込み 377
短縮リストの生成 377
データ名およびプロシージャー名相互参照 375
手続き部のアセンブラー拡張 377
動詞相互参照 377
プログラム名のソート済み相互参照 400
ユーザー提供の行番号 378
用語、MAP 出力で使用される 384
DATA DIVISION 項目のマッピング 377
PROCEDURE DIVISION のアセンブラー拡張 386
リンカー
情報の受け渡し 303
CICS 413
リンク
Java/COBOL アプリケーション 307
ループ
一定の回数だけ実行 92
コーディング 91
条件付き 93
テーブルでの 93
do 92
列、テーブルの 65
例外処理
JAVA で 537
例外条件 259
レコード
形式 125
可変長 136, 137, 173
形式 D 136, 137, 162
形式 F 135, 162
形式 S 140
形式 U 141, 162
形式 V 136, 137, 162
固定長 135, 173
スパン 140
不定 141
QSAM ASCII テープ 162
説明 15
レコードのブロック化 142
レジスター 15 および CICS 411
レジスター、EXIT コンパイラー・オプションの影響 659

レベル 88 項目
ウィンドウ化日付フィールド用 574
条件式 88
スイッチとフラグ 88
制約事項 575
レベル定義 382
連結、データ項目の 95
ローカル参照、グローバルへの変換 517
ローカル名 450
ロード、テーブルの動的な 70
論理レコード
可変長形式 136, 137, 173
固定長形式 135, 173
説明 125

[ワ行]

ワークスペース
ソート時の使用 223
割り振り、TSO のもとでのデータ・セットの 276

[数字]

16 進数リテラル
国別 116
通貨符号として 63
16MB 境界
パフォーマンス・オプション 603
IMS プログラム 410
2 進数データ項目
一般説明 47
効率的使用 47, 594
中間結果 624
同義語 46
2000 年言語拡張
概念 564
仮定による世紀ウィンドウ 575
原則 564
互換性のある日付 572
日付のウィンドウ操作 563
非日付 576
目標 565
DATEPROC コンパイラー・オプション 322
YEARWINDOW コンパイラー・オプション 358
24 ビット・アドレッシング・モード 39
31 ビット・アドレッシング・モード 39
動的呼び出し 444
5203 - 5206 条件 667
64 ビットのアドレッシング
サポートなし 39

A

a 拡張子、cob2 での 305
ACCEPT ステートメント
入力データの割り当て 33
stdin からの読み取り 34
ADATA コンパイラー・オプション 315
ADDRESS 特殊レジスター、CALL ステートメント 456
ADEXIT サブオプション、EXIT コンパイラー・オプションの 658, 665
ADMODE 属性
マルチスレッド化 488
adt 拡張子、cob2 での 306
ADV コンパイラー・オプション 315
AIXBLD ランタイム・オプション
パフォーマンスへの影響 607
ALL 添え字 56, 80
ALL31 ランタイム・オプション
複数オプションの相互作用 39
AMODE 切り替え 444
ALLOCATE コマンド (TSO)
コンパイラー・データ・セット 276
HFS ファイル 276
ALPHABET 文節、照合シーケンスの確立 10
ALTERNATE RECORD KEY 188
AMODE
切り替え 444
説明 39
EXIT モジュールへの割り当て 659
AMP パラメーター 190
ANNUITY 組み込み関数 60
ANSI85 トランスレーター・オプション 414
API、UNIX、および POSIX
呼び出し 430
APOST コンパイラー・オプション 343
APPLY WRITE-ONLY 文節 14
ARITH コンパイラー・オプション 316
ASCII
アルファベット、QSAM 161
ジョブ制御言語 (JCL) 162
テープ・ファイル、QSAM 161
標準ラベル 163
ファイル・ラベル 163
ユーザー・ラベル 163
レコード形式、QSAM 162
ASCII ファイル
CODE-SET 文節 16
DCB 内の OPTCD= パラメーター 16
asm ファイル 333
ASSIGN 文節
DD 名に対応する 12
QSAM ファイル 134
AT END (ファイル終わり) 252
ATTACH マクロ 278
ATTRIBUTE-CHARACTER XML イベント 230
ATTRIBUTE-CHARACTERS XML イベント 230
ATTRIBUTE-NAME XML イベント 230
ATTRIBUTE-NATIONAL-CHARACTER XML イベント 231
AWO コンパイラー・オプション
説明 316
パフォーマンスについての考慮事項 603
APPLY-WRITE ONLY 文節のパフォーマンス 14
a.out ファイル、cob2 からの 306

B

Base クラス
java.lang.Object に相当 497
java.lang.Object のために使用 496
BASIS ステートメント 360
BLOCK CONTAINS 文節
FILE SECTION 記入項目 16
QSAM ファイル 135, 142
VSAM ファイルでは無意味 173
BPXBATCH ユーティリティー 427
BUFOFF= 162
BUFSIZE コンパイラー・オプション 317

C

CALL ID
DLL から行う 474
CALL コマンド (TSO) 276
CALL ステートメント
エラー処理に関する 259
オーバーフロー条件 259
関数ポインター 453
言語環境プログラム呼び出し可能サービスを呼び出すための 613
その中でのプログラム名の処理 340
代替入り口点への 453
例外条件 259
AMODE 処理 444
BY CONTENT 455
BY REFERENCE 456
BY VALUE 455
CANCEL を指定した 443
CICS の制約事項 409
DYNAM コンパイラー・オプションの影響 326
DYNAM コンパイラー・オプションの使用 444

CALL ステートメント *(続き)*
EXIT オプションのレジスターへの影響 659
ID 444
NODYNAM コンパイラー・オプションの使用 444
ON EXCEPTION を指定した 259
ON OVERFLOW を指定した 23, 259
. . . USING 456
CALLINT ステートメント 360
CANCEL ステートメント
その中でのプログラム名の処理 340
動的 CALL を使用した 443
cbl 拡張子、cob2 での 305
CBL ステートメント 286, 360
CBLPSHPOP ランタイム・オプション 416
CBLQDA ランタイム・オプション 146
CCSID
矛盾 244
PARSE ステートメントの 245
XML 文書の 240, 244
CHAR 組み込み関数 108
CHECK(OFF) ランタイム・オプション 603
CHKPT キーワード 223
CICS
組み込みのトランスレーター 413
組み込みのトランスレーターのもとでのコンパイル 413
コーディング制約事項 408
コマンドおよび PROCEDURE DIVISION 408
コマンド・レベル・インターフェース 407
コンパイラー制約事項 408, 413
システム日付 408
出力のコーディング 408
ステートメント
EXIT コンパイラー・オプション 667
ソート 224
入力のコーディング 408
のもとで実行するプログラムのコーディング 408
パフォーマンスについての考慮事項 607
プログラムの開発 407
分離型のトランスレーターのもとでのコンパイル 414
マクロ・レベル・インターフェース 407
マルチスレッド化環境において 487
呼び出し 409
予約語テーブル 416
CALL ステートメント 409
CICS *(続き)*
CICS HANDLE 416
例 417
LABEL 値 417
CICS オプション 411
COBOL 85 標準についての考慮事項 414
z/OS のもとでの制約事項 410
CICS ECI
戻りコード 411
CICS コンパイラー・オプション
説明 317
マルチオプションの相互作用 313
CISZ (制御インターバル・サイズ)、パフォーマンスの考慮事項 193, 607
CKPT キーワード 223
CLOSE ステートメント
行順次ファイル 198
QSAM 145
VSAM 174
cob2 コマンド
オプション 303
コンパイルおよびリンク 301
説明 303
入出力 305
例 303
Java およびオブジェクト指向での 308
COBOL
オブジェクト指向
コンパイルおよびリンク 307
Java 535
コンパイルおよびリンク 307
実行 309
COBOL 85 標準
チェックポイント 554
必要なコンパイラー・オプション 313
必要なランタイム・オプション 313
CICS についての考慮事項 414
COBOL 85 標準への準拠
必要なコンパイラー・オプション 313
COBOL DLL プログラムの呼び出し 477
COBOL クライアント、例 528
COBOL 言語、SQL ステートメントでの使用 420
COBOL 用語 27
COBOPT 環境変数
COBOPT 300
library-name 300
SYSLIB 300
text-name 300
CODEPAGE コンパイラー・オプション
国別リテラルの 117
説明 319
変換に対する 119
CODE-SET 文節 16
COMMENT XML イベント 229
COMMON 属性 8, 447
COMP (COMPUTATIONAL) 47
COMPAT サブオプション、PGMNAME の 341
COMPILE コンパイラー・オプション
構文エラーを見つけるための NOCOMPILE の使用 371
説明 319
COMPUTATIONAL (COMP) 47
COMPUTATIONAL-1 (COMP-1) 49, 595
COMPUTATIONAL-2 (COMP-2) 49, 595
COMPUTATIONAL-3 (COMP-3)
説明 48
日付フィールドの起こりうる問題 587
COMPUTATIONAL-4 (COMP-4) 47
COMPUTATIONAL-5 (COMP-5) 48
COMP-1 (COMPUTATIONAL-1) 49, 595
COMP-2 (COMPUTATIONAL-2) 49, 595
COMP-3 (COMPUTATIONAL-3) 48
COMP-4 (COMPUTATIONAL-4) 47
COMP-5 (COMPUTATIONAL-5 48
CONFIGURATION SECTION 9
CONTENT-CHARACTER XML イベント 232
CONTENT-CHARACTERS XML イベント 231
CONTENT-NATIONAL-CHARACTER XML イベント 232
CONTINUE ステートメント 84
CONTROL ステートメント 360
coprocessor、DB2 422
COPY ステートメント
ネストされた 609, 661
例 610
UNIX についての考慮事項 361
z/OS についての考慮事項 283
CRP (ファイル位置標識) 176, 180
CURRENCY コンパイラー・オプション 320
CURRENT-DATE 組み込み関数 59
C/C++ プログラム
COBOL DLL を使用した 479

D

D 形式レコード 136, 137
DASD (直接アクセス記憶装置) 193
DATA DIVISION (データ部)
インスタンス・データ 498, 522
インスタンス・メソッド 502
行順次ファイルについての記入項目 196
クライアント 511
コーディング 15
項目のマッピング 333, 377

DATA DIVISION (データ部) *(続き)*
シグニチャー・バイト 391
制限 15
制約事項 15
説明 15
その中に存在する項目 391
ファクトリー・データ 525
ファクトリー・メソッド 526
リスト 377
FD 記入項目 15
FILE SECTION 15
LINKAGE SECTION 19
OCCURS 文節 65
QSAM ファイルについての記入項目 134
VSAM ファイルについての記入項目 173
WORKING-STORAGE SECTION 15
DATA RECORDS 文節 16
DATA コンパイラー・オプション
説明 321
データの受け渡し時 40
データ・ロケーションへの影響 40
パフォーマンスについての考慮事項 603
複数オプションの相互作用 39
data-name
相互参照 399
相互参照リスト 287
MAP リスト内の 382
OMITTED 16
VSAM ファイルのパスワード 184
DATEPROC コンパイラー・オプション
警告レベル診断メッセージの分析 585
説明 322
DATEVAL 組み込み関数 583
DATE-COMPILED 段落 7
DATE-OF-INTEGER 組み込み関数 59
DB2
オプション 421
コーディング考慮事項 419
戻りコード 420
COBOL 言語、SQL ステートメントでの使用 420
coprocessor 422
SQL INCLUDE ステートメント 420
SQL ステートメント 419
DBCS
エンコード 117
コンパイラー・オプション 313, 323
変換 123, 637
XREF 出力でのユーザー定義語 375
DBCS コンパイラー・オプション
オブジェクト指向 COBOL のための 307
DBCS コンパイラー・オプション *(続き)*
Java とのインターオペラビリティーのための 307
DBCSXREF 376
dbg 拡張子、cob2 での 306
DCB 144
DD 制御ステートメント
行順次ファイルの作成 197
ソート・データ・セットの定義 210
ファイルの定義 12
AMP パラメーター 190
ASCII テープ・ファイル 162
DCB によるデータ・セット・ラベルのオーバーライド 151
QSAM ファイルの作成 150, 152
RLS パラメーター 191
SYSADATA 284
SYSDEBUG 285
SYSIN 282
SYSJAVA 285
SYSLIB 283
SYSLIN 284
SYSPRINT 283
SYSPUNCH 284
DD 名定義 12
DECK コンパイラー・オプション 323
DELETE ステートメント
コンパイラー指示 360
マルチスレッド化シリアライゼーション 485
VSAM、コーディング 174
DEPENDING ON オプション 137, 173
DFCOMMAREA パラメーター、CALL の 409
DFHEIBLK パラメーター、CALL の 409
DFSORT
使用 210
DIAGTRUNC コンパイラー・オプション 324
DISK コンパイラー・オプション 311
DISPLAY (USAGE IS)
エンコード 117
外部 10 進数 46
DISPLAY ステートメント
出力の指示 340
データ値の表示 34
デバッグでの使用 366
OUTDD との相互作用 35
stdout/stderr への書き込み 35
DISPLAY-OF 組み込み関数
使用 118
XML 文書における 240
DLL libjvm.x
EBCDIC サービスで 545
Java とのリンク 307
DLL コンパイラー・オプション
オブジェクト指向 COBOL のための 307
説明 324
マルチオプションの相互作用 313
Java とのインターオペラビリティーのための 307
DLL (ダイナミック・リンク・ライブラリーを参照) 469
DO ループ 92
DOCUMENT-TYPE-DECLARATION XML イベント 229
do-until 92
do-while 92
DSA メモリー・マップ 386
DUMP コンパイラー・オプション
出力 287
説明 325
マルチオプションの相互作用 313
DYNAM コンパイラー・オプション
説明 326
パフォーマンスについての考慮事項 603

E

E レベルのエラー・メッセージ 296, 373
EBCDIC
JNI サービス 544
ECI 呼び出しおよびレジスター 15 411
EJECT ステートメント 360
ENCODING-DECLARATION XML イベント 229
END-OF-CDATA-SECTION XML イベント 233
END-OF-DOCUMENT XML イベント 233
END-OF-ELEMENT XML イベント 231, 232
ENTER ステートメント 360
ENTRY ステートメント
その中でのプログラム名の処理 340
ENVIRONMENT DIVISION (環境部)
インスタンス・メソッド 501
行順次ファイルについての記入項目 195
クライアント 510
クラス 497
サブクラス 521
シグニチャー・バイト 392
照合シーケンスのコーディング 10
説明 9
その中に存在する項目、プログラム初期設定コード 392
CONFIGURATION SECTION 9
INPUT-OUTPUT SECTION 9

ENVIRONMENT DIVISION (環境部) *(続き)*
QSAM ファイルについての記入項目 133
VSAM ファイルについての記入項目 168
ERRMSG、エラー・メッセージのリストの生成 294
ESDS (入力順データ・セット)
ファイル・アクセス・モード 172
EVALUATE ステートメント
ケース構造 84, 85
構造化プログラミング 592
EXCEPTION XML イベント 233
EXCEPTION/ERROR 宣言
行順次エラー処理 201
説明 253
ファイル状況キー 255
QSAM エラー処理 149
VSAM エラー処理 184
EXEC 制御ステートメント 557
EXIT PROGRAM ステートメント
サブプログラムにおける 439
マルチスレッド化 439
EXIT コンパイラー・オプション
使用 657
説明 327
DUMP コンパイラー・オプションとの併用 313
SQL および CICS ステートメントについての考慮事項 667
EXPORTALL コンパイラー・オプション
説明 327
マルチオプションの相互作用 313
EXTERNAL データ
共用 464
ストレージの獲得 41
保管場所 41
EXTERNAL 文節
データ項目の 464
入出力で使用される 464
ファイルの 16
ファイルの場合の例 465, 466

F

F 形式レコード 135
FACTORY 段落
ファクトリー・データ 525
ファクトリー・メソッド 525
FASTSRT コンパイラー・オプション
説明 327
ソート・パフォーマンスの向上 217, 603
通知メッセージ 217
要件 217
FD (ファイル記述) 記入項目 16
FILE SECTION
説明 15
レコード記述 15
BLOCK CONTAINS 文節 16
CODE-SET 文節 16
DATA RECORDS 文節 16
EXTERNAL 文節 16
FD 記入項目 16
GLOBAL 文節 16
LABEL RECORDS 文節 16
LINAGE 文節 16
OMITTED 16
RECORD CONTAINS 文節 16
RECORD IS VARYING 16
RECORDING MODE 文節 16
VALUE OF 16
FILE STATUS 文節
行順次エラー処理 201
使用 254
説明 131
NOFASTSRT エラー処理 219
QSAM エラー処理 149
VSAM エラー処理 184
VSAM ファイルのロード 178
VSAM 戻りコードのある 256
FILEDEF コマンド
ASCII テープ・ファイル 162
FLAG コンパイラー・オプション
コンパイラー出力 374
使用 373
説明 328
FLAGSTD コンパイラー・オプション 329

G

get メソッドおよび set メソッド 506
GETMAIN のアドレスの保管 658
GLOBAL 文節、ファイルに対する 16, 20
GO TO MORE-LABELS 160
GOBACK ステートメント
サブプログラムにおける 439
マルチスレッド化 439
メインプログラムにおける 439

H

HEAP ランタイム・オプション
データ・ロケーションへの影響 40
複数オプションの相互作用 39

I

I レベルのエラー・メッセージ 296, 373
IDENTIFICATION DIVISION (見出し部)
エラー 8
クライアント 509
クラス 496
コーディング 7
サブクラス 521
必須段落 7
メソッド 501
リスト・ヘッダーの例 9
CLASS-ID 段落 496, 521
DATE-COMPILED 段落 7
PROGRAM-ID 段落 7
TITLE ステートメント 9
idl 拡張子、cob2 での 306
IF ステートメント
コーディング 83
ネストされた 84
NULL ブランチを伴う 83
IGZBRDGE マクロ
マルチスレッド化 488
IGZCA2D サービス・ルーチン 637
IGZCD2A サービス・ルーチン 640
IGZEOPT モジュール
マルチスレッド化 488
IGZETUN モジュール
マルチスレッド化 488
IGZSRTCD データ・セット 221
IMS
コーディング制約事項 425
推奨されるコンパイラー・オプション 425
のもとでのプログラムのコーディング 425
パフォーマンスについての考慮事項 607
プログラムのコーディング 10
INEXIT サブオプション、EXIT オプションの 657, 659
INITIAL 属性 9, 440, 443
INITIALIZE ステートメント
テーブルの値のロード 70
デバッグ用の使用 368
例 30
INPUT-OUTPUT SECTION 9
INSERT ステートメント 360
INSPECT ステートメント 104
INTDATE コンパイラー・オプション 330
INTEGER 組み込み関数 103
INTEGER-OF-DATE 組み込み関数 59
INVALID KEY 句 258
INVOKE ステートメント
オブジェクトの作成に使用 516

INVOKE ステートメント *(続き)*
メソッドの呼び出しに使用 514
ON EXCEPTION を指定した 515, 528
PROCEDURE DIVISION RETURNING との併用 463
ISAM データ・セット 165
ISPF (対話式システム生産性向上機能) 738

J

J2EE クライアント
例 546
Java
インターオペラビリティー 535
オブジェクト配列 542
クラス型 541
グローバル参照 538
受け渡し 538
オブジェクト 538
管理 538
JNI サービス 539
ストリング
宣言する 541
取り扱い 543
ストリング配列 541
相互運用可能データ型、コーディング 540
データ共用 540
長い配列 541
二重配列 541
バイト配列 541
配列
宣言する 541
取り扱い 542
例 543
配列クラス 540
ブール配列 541
浮動配列 541
短い配列 541
メソッド
アクセス制御 539
文字配列 541
例
配列の処理 543
例外処理 537
J2EE クライアント 546
例外
キャッチ 537
処理 537
スロー 537
例 537
レコード・クラス 540
ローカル参照 538
受け渡し 538
Java *(続き)*
ローカル参照 *(続き)*
オブジェクト 538
解放 539
管理 538
削除 538
保管 539
マルチスレッド化ごと 538
JNI サービス 539
boolean 型 540
byte 型 540
char 型 541
COBOL 535
コンパイルおよびリンク 307
COBOL での実行 309
double 型 541
float 型 541
int 型 540
int 配列 541
jstring クラス 540
long 型 541
short 型 540
java.lang.Object
Base として参照 496
JCL
カタログ式プロシージャー 264
行順次ファイル用 197
コンパイル用 264
ソート用 210
チェックポイント・リスタートの例 560
ASCII テープ・ファイル 162
FASTSRT の要件 217
HFS とのコンパイル用 266
QSAM ファイル用 151
VSAM データ・セット用 189
JNI
オブジェクト・リファレンスの比較 513
クラス・オブジェクト・リファレンスの取得 536
構造環境 535
アドレス可能度の場合 536
サービスへアクセス 535
使用した場合の制限 536
例外取り扱いサービス 537
ローカル参照をグローバルに変換 517
EBCDIC サービス 544
Java ストリング・サービス 543
Java 配列サービス 542
Unicode サービス 543
JNIEnvPtr 特殊レジスター 535
JNINativeInterface
構造環境 535
JNI.cpy 535
JNI.cpy
リスト 673
JNINativeInterface の場合 535
jstring Java クラス 540

K

KSDS (キー順データ・セット)
ファイル・アクセス・モード 172
編成 169

L

LABEL RECORDS 文節
FILE SECTION 記入項目 16
LABEL 宣言
説明 360
ユーザー・ラベルの処理 160
GO TO MORE-LABELS 160
LABEL= 162
LANGUAGE コンパイラー・オプション 331
LENGTH OF 特殊レジスター
受け渡し 456
国別データを伴う 110
使用 110
LENGTH 組み込み関数 107
可変長の結果 109
国別データを伴う 110
例 59, 110
LENGTH OF 特殊レジスターとの対比 110
LIB コンパイラー・オプション
説明と構文 332
LIBEXIT サブオプション、EXIT オプションの 657, 661
libjvm.x
EBCDIC サービスで 545
Java とのリンク 307
LIBPATH 環境変数
設定 429
library-name
指定されなかった場合の代替 304
使用されない場合 661
LINAGE 文節 16
LINECOUNT コンパイラー・オプション 332
LINK マクロ 278
LINKAGE SECTION
再帰呼び出し 19
実行単位 19
説明 458
GLOBAL 文節 20
THREAD オプションを指定した 19

LIST コンパイラー・オプション
コンパイラー出力 387, 389
出力で使用される記号 385
出力で使用される用語 384
出力の入手 377
出力の読み取り 386
説明 333
ソース・プログラムのアセンブラー・コード 386
マルチオプションの相互作用 313
DSA メモリー・マップ 386, 397
OFFSET オプションとの対立 377
TGT メモリー・マップ 386
WORKING-STORAGE の位置とサイズ 397
LOCAL-STORAGE SECTION
クライアント 511
ロケーションの決定 40
WORKING-STORAGE との比較 18, 512
LOG 組み込み関数 60
LONGMIXED サブオプション、PGMNAME の 342
LONGUPPER サブオプション、PGMNAME の 341
LOWER-CASE 組み込み関数 105
lst 拡張子、cob2 での 306
LST ファイル拡張子 294

M

MAP コンパイラー・オプション
組み込み MAP 要約 377
出力で使用される記号 385
出力で使用される用語 384
使用 376, 377
説明 333
データ項目と相対アドレス 287
ネストされたプログラム・マップ 377
ネストされたプログラム・マップの例 386
例 382, 386
MAX 組み込み関数
使用 108
例 59, 81
MEAN 組み込み関数 60, 81
MEDIAN 組み込み関数 60, 81
MERGE ステートメント
説明 204
METHOD-ID 段落 501
MIN 組み込み関数 103, 108
MLE 564
MOVE ステートメント
国別項目を伴う 32, 118
使用 32
MSGFILE ランタイム・オプション 340

N

NAME コンパイラー・オプション
使用 7
説明 334
NATIONAL-OF 組み込み関数
使用 118
XML 文書における 240
NOCOMPILE コンパイラー・オプション
構文エラーを見つけるための使用 371
NODYNAM コンパイラー・オプション
CICS のもとで 409
NOFASTSRT コンパイラー・オプション 219, 223
NSYMBOL コンパイラー・オプション
国別データ項目の 116
国別リテラルの 116
説明 335
NULL ブランチ 84
NUMBER コンパイラー・オプション
説明 336
デバッグ用 378
NUMCLS インストール・オプション 53
NUMPROC コンパイラー・オプション
説明 336
符号処理への影響 52
NUMCLS による影響 53
NUMVAL 組み込み関数 106
NUMVAL-C 組み込み関数
使用 106
例 59

O

o 拡張子、cob2 での 305, 306
OBJECT コンパイラー・オプション
説明 338
マルチオプションの相互作用 313
OBJECT 段落
インスタンス・データ 498, 522
インスタンス・メソッド 500
OBJECT-COMPUTER 段落 9
OCCURS DEPENDING ON (ODO) 文節
可変長テーブル 74
可変長レコード 137, 173
最適化 596
単純 74
複合 631
ODO エレメントの初期化 77
OCCURS 文節 65, 596
OFFSET コンパイラー・オプション
出力 401
説明 338
マルチオプションの相互作用 313
OMITTED LE パラメーター 613
OMITTED 文節、FILE SECTION 17
ON EXCEPTION 句
INVOKE ステートメント 515, 528
ON SIZE ERROR
ウィンドウ化日付フィールドでの 581
OPEN ステートメント
行順次ファイル 198
ファイル状況キー 254
ファイルの可用性 145, 176, 177, 198
QSAM ファイル 145
VSAM ファイル 174
OPEN 命令コード 659
OPTIMIZE コンパイラー・オプション
使用 599
説明 339
パフォーマンスについての考慮事項 603
パフォーマンスへの影響 599
マルチオプションの相互作用 313
ORD 組み込み関数 108
ORD-MAX 組み込み関数 81, 108
ORD-MIN 組み込み関数 108
OUTDD コンパイラー・オプション
説明 340
割り振られない DD 35
DISPLAY との対話 35

P

PACKED-DECIMAL
同義語 46
PASSWORD 文節 184
PERFORM ステートメント
一定の回数だけ実行 92
インライン 91
指標付け 70
テーブルの 72
ライン外 91
ループのコーディング 91
TEST AFTER 92
TEST BEFORE 92
TIMES 92
UNTIL 93
VARYING 93
VARYING WITH TEST AFTER 93
WITH TEST AFTER . . . UNTIL 93
WITH TEST BEFORE . . . UNTIL 93
. . .THRU 94
PGMNAME コンパイラー・オプション 340
PICTURE 文節
使用される記号の判別 320
数値データ 43
PL/I タスク
COBOL との 437
POSIX ランタイム・オプション 487

POSIX
スレッド 487
API の呼び出し 430
PRESENT-VALUE 組み込み関数 59
PROCEDURE DIVISION RETURNING
メソッドでの使用 463
PROCEDURE DIVISION (手続き部)
インスタンス・メソッド 503
クライアント 510
サブプログラムにおける 458
シグニチャー・バイト 392, 394
ステートメント
コンパイラー指示 24
条件付き 23
範囲区切り 23
命令 22
説明 21
その中に存在する動詞 392
追加情報 394
用語 21
RETURNING の指定 21
USING 21
PROCESS (CBL) ステートメント
対立するオプション 313
バッチ・コンパイル 291
優先順位 286
PROCESS ステートメント 286
PROCESSING-INSTRUCTION-DATA XML イベント 231
PROCESSING-INSTRUCTION-TARGET XML イベント 231
PROGRAM COLLATING SEQUENCE 文節 10
PROGRAM-ID 段落
コーディング 7
COMMON 属性 8
INITIAL 属性 9
PRTEXIT サブオプション、EXIT オプションの 657, 664

Q

QSAM ファイル
オープン 145
逆順でのファイル処理 146
クローズ 148
検索 152
再オープン防止のためのクローズ 146
処理 133
ストライプ拡張形式 156
テープのパフォーマンス 143
入出力エラー処理 149, 250
入出力ステートメント 145
バッファーの入手 157
パフォーマンスを向上させるブロック化 142
QSAM ファイル *(続き)*
ファイルの更新 147
プリンターへの書き込み 147
ブロック・サイズ 142
ラベル処理 158
レコードの置換 147
レコードの追加 147
レコードのブロック化 142, 157
ASCII テープ・ファイル 161
ASSIGN 文節 134
BLOCK CONTAINS 文節 142
CMS のもとでの
動的割り振り 146
ファイルの可用性 146
DATA DIVISION 記入項目 134
ENVIRONMENT DIVISION 記入項目 133
HFS ファイルの処理 157
z/OS のもとでの
環境変数 150
ジョブ制御言語 (JCL) 151
定義 150, 152
ファイルの可用性 146
ファイルの作成 150, 152
DD ステートメント 150, 152
QSAM ファイルのブロック化 142
QUOTE コンパイラー・オプション 343

R

RANGE 組み込み関数 60, 81
RD パラメーター 557
READ INTO. . . 174
READ NEXT ステートメント 174
READ ステートメント
行順次ファイル 198
マルチスレッド化シリアライゼーション 485
QSAM 145
VSAM 174
RECORD CONTAINS 文節
FILE SECTION 記入項目 16
RECORDING MODE 文節
可変長レコード、QSAM 136, 137
固定長レコード、QSAM 135
レコード形式の指定 134
QSAM ファイル 16
RELEASE FROM ステートメント
例 207
RELEASE との比較 207
RELEASE ステートメント
RELEASE FROM との比較 207
SORT での 207
REM 組み込み関数 60
RENT コンパイラー・オプション
アドレス可能度への影響 40
RENT コンパイラー・オプション *(続き)*
オブジェクト指向 COBOL のための 307
説明 343
データの受け渡し時 40
パフォーマンスについての考慮事項 603
複数オプションの相互作用 39
Java とのインターオペラビリティーのための 307
REPLACE ステートメント 360
REPOSITORY 段落
クライアント 510
クラス 497
コーディング 9
サブクラス 521
RERUN 文節
チェックポイント・リスタート 223
RETURN INTO ステートメント 208
RETURN ステートメント 208
RETURNING 句
メソッドでの使用 463
INVOKE ステートメント 515
PROCEDURE DIVISION ヘッダー 504
RETURN-CODE 特殊レジスター
オペレーティング・システムに制御権が戻されるとき 463
言語環境プログラム・サービスへの呼び出し後の値 613
DB2 についての考慮事項 420
INVOKE で設定しない 515
REVERSE 組み込み関数 106
REWRITE ステートメント
マルチスレッド化シリアライゼーション 485
QSAM 145
VSAM 174
RLS パラメーター 191
RMODE
説明 39
EXIT モジュールへの割り当て 659
RMODE コンパイラー・オプション
アドレス可能度への影響 39
説明 344
データの受け渡し時 40
パフォーマンスについての考慮事項 603
複数オプションの相互作用 39
RRDS (相対レコード・データ・セット)
可変長レコード 166
可変長レコードのシミュレート 170
固定長レコード 166
パフォーマンスについての考慮事項 193
ファイル・アクセス・モード 172

RRDS (相対レコード・データ・セット) *(続き)*
編成 168, 170

S

S 形式レコード 140
S レベルのエラー・メッセージ 296, 373
SEARCH ALL ステートメント
指標付け 69, 77
順序付けられたテーブル 79
二分探索 79
SEARCH ステートメント
指標付け 69, 70
シリアル検索 78
ネスト 78
例 79
SELECT OPTIONAL 146, 176
SELECT 文節
入出力ファイルの変更 13
ファイルの命名 12
ASSIGN 文節 12
SELF 513
SEQUENCE コンパイラー・オプション 345, 372
SERVICE LABEL ステートメント 360
SET 条件名 TO TRUE ステートメント
スイッチとフラグ 90
例 92, 94
SET ステートメント
オブジェクト・リファレンスの設定 513
関数ポインター・データ項目用 452
その中でのプログラム名の処理 340
デバッグ用の使用 368
プロシージャー・ポインター・データ項目用 452
SIMVRD ランタイム・オプション 170
SIZE コンパイラー・オプション 346
SORT ステートメント
説明 211
CICS アプリケーションの場合の制約事項 224
CICS のもとで 224
SORTCKPT DD ステートメント 223
SORT-CONTROL 特殊レジスター 221
SORT-CORE-SIZE 特殊レジスター 221
SORT-FILE-SIZE 特殊レジスター 221
SORT-MESSAGE 特殊レジスター 221
SORT-MODE-SIZE 特殊レジスター 221
SORT-RETURN 特殊レジスター 216, 221
SOURCE および NUMBER 出力の例 381
SOURCE コンパイラー・オプション 346, 377
SOURCE-COMPUTER 段落 9
SPACE コンパイラー・オプション 347
SPECIAL-NAMES 段落
コーディング 9
QSAM ファイル 162
SQL INCLUDE ステートメント 420
SQL コンパイラー・オプション
使用 421
説明 347
マルチオプションの相互作用 313
SQL ステートメント
DB2 サービスのための使用 419
EXIT コンパイラー・オプション 667
SQLCA 419
SQLCODE 420
SQLSTATE 420
SQRT 組み込み関数 60
SSRANGE コンパイラー・オプション
使用 372
説明 348
パフォーマンスについての考慮事項 603
CHECK(OFF) ランタイム・オプション 603
STACK ランタイム・オプション
データ・ロケーションへの影響 40
複数オプションの相互作用 39
STANDALONE-DECLARATION XML イベント 229
STANDARD 文節、FD 記入項目 16
START ステートメント
マルチスレッド化シリアライゼーション 485
VSAM 174
START-OF-CDATA-SECTION XML イベント 233
START-OF-DOCUMENT XML イベント 228
START-OF-ELEMENT XML イベント 230
stderr
DISPLAY の指示 35
DISPLAY の設定 429
stdin
ACCEPT による読み取り 34
stdout
DISPLAY の指示 35
DISPLAY の設定 429
STEPLIB 環境変数
設定 429
STOP RUN ステートメント
サブプログラムにおける 439
マルチスレッド化 439
メインプログラムにおける 439
STRING ステートメント
オーバーフロー条件 248
STRING ステートメント *(続き)*
使用 95
例 96
DBCS データを使う 637
SUM 組み込み関数 81
SUPER 516
SYMBOLIC CHARACTER 文節 12
SYSADATA
出力 315
レコード、出口モジュール 665
SYSADATA データ・セット 284
SYSADATA ファイル
ファイル内容 679
例 682
レコード記述 683
レコード・タイプ 680
SYSIN データ・セット
説明 280
定義 282
ユーザー出口エラー・メッセージ 667
SYSJAVA データ・セット、作成 285
SYSLIB データ・セット
使用されない場合 661
説明 280
定義 283
SYSLIN データ・セット 284
SYSPRINT データ・セット
使用されない場合 664
説明 280
定義 283
SYSPUNCH データ・セット
説明 280, 284
DECK コンパイラー・オプションの要件 323
SYSTERM データ・セット
説明 280
定義 283
メッセージの送信 349
SYSUT データ・セット 280

T

TALLYING オプション 104
TERMINAL コンパイラー・オプション 349
TEST AFTER 92
TEST BEFORE 92
TEST コンパイラー・オプション
説明 349
デバッグ・ツールの最大限の利用 402
パフォーマンスについての考慮事項 603
マルチオプションの相互作用 313
TGT メモリー・マップ
説明 386
例 396

THREAD コンパイラー・オプション
オブジェクト指向 COBOL のための 307
説明 352
Java とのインターオペラビリティーのための 307
LINKAGE SECTION 19
TITLE ステートメント
リストのヘッダーの制御 9
TRACK OVERFLOW オプション 144
TRAP ランタイム・オプション
行順次ファイルのクローズ 200
ON SIZE ERROR 249
QSAM ファイルのクローズ 148
VSAM ファイルのクローズ 183
TRUNC コンパイラー・オプション
説明 353
パフォーマンスについての考慮事項 603
TSO
コンパイラー・メッセージ用の SYSTERM 283
のもとでのコンパイル 276
ALLOCATE コマンド 276
CALL コマンド 276

U

U 形式レコード 141
U レベルのエラー・メッセージ 296, 373
UNDATE 組み込み関数 584
Unicode
エンコード 117
実行時サポート 113
説明 115
JNI サービス 543
UNIX
オブジェクト指向アプリケーションの作成 308
環境変数の設定 429
例 430
環境変数へのアクセス 429
例 430
コピーブック 361
コピーブック探索順序 300, 304, 361
コンパイラー環境変数 299
コンパイラー・オプションの指定 300
実行環境 427
ソート / マージ 203
添え字からのコンパイル 306
のもとでのコンパイル 299
プログラムの開発 427
メイン・パラメーターへのアクセス 432
例 433
API の呼び出し 430
UNIX API
呼び出し 430
UNKNOWN-REFERENCE-IN-ATTRIBUTE XML イベント 233
UNKNOWN-REFERENCE-IN-CONTENT XML イベント 233
UNSTRING ステートメント
オーバーフロー条件 248
使用 97
例 98
DBCS データを使う 637
UPPER-CASE 組み込み関数 105
UPSI スイッチ
マルチスレッド化 488
USAGE 文節
非互換データ 53
IS INDEX 70
OBJECT REFERENCE 512
USE AFTER STANDARD LABEL 163
USE FOR DEBUGGING 宣言 368
USE ステートメント 360
USE . . . LABEL 宣言 160
USING 句
INVOKE ステートメント 515
PROCEDURE DIVISION ヘッダー 503
UTF-16
エンコード方式、国別データの 115
UTF-8 データ
処理 120
変換 120

V

V 形式レコード 136, 137
VALUE IS NULL 460
VALUE OF 文節 16
VALUE 文節
大きな、TRUNC(BIN) での 355
データ記述記入項目 72
テーブルの値の割り当て 71
COMP-5 を指定したラージ・リテラル 48
VBREF コンパイラー・オプション
出力例 402
使用 377
説明 356
VERSION-INFORMATION XML イベント 229
VSAM ファイル
アクセス方式サービス・プログラムによる 178
エラー処理 250
オープン 176, 177
環境変数を使用した割り振り 189
クローズ 183
VSAM ファイル *(続き)*
代替索引の作成 187
動的ロード 178
入出力エラー処理 183
入出力ステートメントのコーディング 174
パスワードを使っての保護 184
パフォーマンスについての考慮事項 193
ファイル位置標識 (CRP) 176, 180
ファイル状況キー 183
ファイルの処理 165
ファイル編成の比較 167
マルチスレッド化環境において 487
戻りコード 256
ランダム・ロード 178
レコードの更新 180
レコードの削除 182
レコードの置換 182
レコードの追加 181
レコードの読み取り 179
レコードのロード 177
DATA DIVISION 記入項目 173
ENVIRONMENT DIVISION 記入項目 168
z/OS のもとでの
データ・セットの定義 186
ファイルの可用性 185
JCL 189
RLS モード 191
VSAM 用語
非 VSAM 用語との比較 165
BDAM データ・セット 165
BDAM に対応する RRDS 165
ISAM に対応する KSDS 165
QSAM に対応する ESDS 165

W

W レベルのエラー・メッセージ 296, 373
WHEN 句
EVALUATE ステートメント 85
SEARCH ステートメント 78
WHEN-COMPILED 組み込み関数
例 110
WHEN-COMPILED 特殊レジスターとの対比 110
WHEN-COMPILED 特殊レジスター 110
wlist ファイル 333
WORD コンパイラー・オプション 357
WORKING-STORAGE
データの保管場所 321
WORKING-STORAGE SECTION 18
インスタンス・データ 498, 522
インスタンス・メソッド 502

WORKING-STORAGE SECTION *(続き)*
クライアント 511
ファクトリー・データ 525
マルチスレッド化の考慮事項 512
LOCAL-STORAGE との比較 18, 512
WRITE ADVANCING ステートメント 148
WRITE ステートメント
行順次ファイル 198
マルチスレッド化シリアライゼーション 485
QSAM 145
VSAM 174

X

x 拡張子、cob2 での 305, 306
XML PARSE ステートメント
使用 227
説明 225
NOT ON EXCEPTION 241
ON EXCEPTION 241
XML イベント
処理 228
処理プロシージャー 227
説明 225
ATTRIBUTE-CHARACTER 230
ATTRIBUTE-CHARACTERS 230
ATTRIBUTE-NAME 230
ATTRIBUTE-NATIONAL-CHARACTER 231
COMMENT 229
CONTENT-CHARACTER 232
CONTENT-CHARACTERS 231
CONTENT-NATIONAL-CHARACTER 232
DOCUMENT-TYPE-DECLARATION 229
ENCODING-DECLARATION 229
END-OF-CDATA-SECTION 233
END-OF-DOCUMENT 233
END-OF-ELEMENT 231, 232
EXCEPTION 233
PROCESSING-INSTRUCTION-TARGET 231
STANDALONE-DECLARATION 229
START-OF-CDATA-SECTION 233
START-OF-DOCUMENT 228
START-OF-ELEMENT 230
UNKNOWN-REFERENCE-IN-ATTRIBUTE 233
UNKNOWN-REFERENCE-IN-CONTENT 233
VERSION-INFORMATION 229
XML 構文解析
概要 225
XML 構文解析 *(続き)*
終了 244
説明 227
特殊レジスター 234
CCSID 矛盾 244
XML 処理プロシージャー
書き込み 234
指定 227
特殊レジスターの使用 234
例 237
XML パーサー
説明 225
適合性 652
XML 文書
アクセス 227
エラーの処理 241
各国語 240
構文解析 227
例 237
処理 225
パーサー 225
XML 例外コード
処理可能でない 648
処理可能な 643
XML-CODE 特殊レジスター
使用 225
説明 234
例外のある 241
XML-EVENT 特殊レジスター
使用 225, 228
説明 234
XML-NTEXT 特殊レジスター 234
使用 225
XML-TEXT 特殊レジスター 234
使用 225
XREF コンパイラー・オプション 357, 375
XREF 出力
データ名相互参照 399
プログラム名相互参照 400

Y

YEARWINDOW コンパイラー・オプション
説明 358
ソート・マージへの影響 221

Z

ZWB コンパイラー・オプション 359
z/OS
のもとでのコンパイル 263

[特殊文字]

*CBL ステートメント 360
*CONTROL ステートメント 360
-b cob2 オプション 303
-c cob2 コマンド 304
-comprc_ok cob2 オプション 304
-e cob2 オプション
入り口点の指定 304
-g cob2 オプション
TEST オプション 304
-I cob2 オプション
コピーブックの検索 304
-L cob2 オプション
アーカイブ・ライブラリー・パスの指定 304
-l cob2 オプション
アーカイブ・ライブラリー名の指定 305
-o cob2 オプション
出力ファイルの指定 305
-q cob2 オプション
コンパイラー・オプションの指定 305
-v cob2 オプション 305
-# cob2 オプション 305
.asm ファイル 333
.wlist ファイル 333
>>CALLINT ステートメント 360
_CEE_RUNOPTS 環境変数
設定 429
_IGZ_SYSOUT 環境変数
設定 429
stdout/stderr への書き込み 35

Printed in Japan

SC88-9121-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12