Enterprise PL/I for z/OS

プログラミング・ガイド

バージョン **5** リリース **2**

GC43-3419-01

Enterprise PL/I for z/OS

プログラミング・ガイド

バージョン **5** リリース **2**

GC43-3419-01

お願い

本書および本書で紹介する製品をご使用になる前に、 587 ページの『特記事項』に記載されている情報をお読みください。

| 本書は、Enterprise PL/I for z/OS のバージョン 5 リリース 2、および新しい版またはテクニカル・ニュースレターで明記されていない限り、以降のすべてのリリースに適用されます。製品のレベルに合った正しい版をご使用ください。

資料のご注文方法については、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)

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： GI13-4536-01
Enterprise PL/I for z/OS
Programming Guide
Version 5 Release 2
First Edition (September 2017)

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

© Copyright IBM Corporation 1999, 2017.

目次

表 xi

図. xiii

はじめに xv

本書について xv
Enterprise PL/I for z/OS のランタイム環境 . . . xv
資料の使用 xv
本書で使用されている表記規則 xvi
使用する規則 xvi
構文記法の読み方 xvii
表記記号の読み方 xx
変更の要約 xx
本リリースにおける機能強化 xx
V5R1 からの機能拡張 xxii
V4R5 からの機能拡張 xxv
V4R4 からの機能拡張 xxvi
V4R3 からの機能拡張 xxvii
V4R2 からの機能拡張. xxviii
V4R1 からの機能強化 xxx
V3R9 からの機能強化 xxxi
V3R8 からの機能拡張. xxxiii
V3R7 からの機能拡張. xxxiv
V3R6 からの機能拡張. xxxvi
V3R5 からの機能拡張 xxxvii
V3R4 からの機能拡張 xxxviii
V3R3 からの機能拡張 xl
V3R2 からの機能拡張 xlii
V3R1 からの機能拡張 xliv
VisualAge PL/I からの機能拡張. xliv
ご意見の送付方法 xlv
アクセシビリティー xlvi

第 1 部 プログラムのコンパイル . . . 1

第 1 章 コンパイラー・オプションと機能の使用. 3

コンパイル時オプションの説明 3
AGGREGATE 7
ARCH 8
ASSERT 9
ATTRIBUTES 9
BACKREG. 10
BIFPREC 10
BLANK. 12
BLKOFF 12
BRACKETS 12
CASE 13
CASERULES 13
CEESTART 14
CHECK 14
CMPAT 16
CODEPAGE 17
COMMON 18
COMPILE 18
COPYRIGHT 19
CSECT 19
CSECTCUT 20
CURRENCY 20
DBCS 21
DBRMLIB 21
DD 22
DDSQL. 22
DECIMAL. 22
DECOMP 24
DEFAULT. 24
DEPRECATE 34
DEPRECATENEXT 36
DISPLAY 36
DLLINIT 37
EXIT. 37
EXPORTALL 37
EXTRN 38
FILEREF 38
FLAG 38
FLOAT 39
FLOATINMATH 40
GOFF 41
GONUMBER 42
GRAPHIC 43
HEADER 43
IGNORE 43
INCAFTER 44
INCDIR 44
INCLUDE 45
INCPDS 45
INITAUTO 46
INITBASED 47
INITCTL 47
INITSTATIC 48
INSOURCE 48
INTERRUPT 49
JSON 50
LANGLVL 50
LIMITS 51
LINECOUNT 52
LINEDIR 52
LIST 53
LISTVIEW 53
LP 54
MACRO 55

© Copyright IBM Corp. 1999, 2017

MAP 55
MARGINI 56
MARGINS. 56
MAXBRANCH 57
MAXGEN 58
MAXMEM. 58
MAXMSG 59
MAXNEST 59
MAXSTMT 60
MAXTEMP 60
MDECK 60
MSGSUMMARY. 61
NAME 62
NAMES 62
NATLANG 63
NEST 63
NOT. 63
NULLDATE 64
NUMBER 64
OBJECT 64
OFFSET 65
OFFSETSIZE 65
ONSNAP 65
OPTIMIZE. 66
OPTIONS 67
OR 68
PP 68
PPCICS. 70
PPINCLUDE 70
PPLIST 71
PPMACRO 71
PPSQL 72
PPTRACE 72
PRECTYPE 72
PREFIX. 73
PROCEED. 74
PROCESS 74
QUOTE. 75
REDUCE 75
RENT 76
RESEXP 78
RESPECT 78
RTCHECK. 78
RULES 79
SEMANTIC 95
SERVICE 95
SOURCE 96
SPILL 96
STATIC. 96
STDSYS 97
STMT 97
STORAGE. 98
STRINGOFGRAPHIC 98
SYNTAX 98
SYSPARM. 99
SYSTEM 99
TERMINAL 100
TEST 101
UNROLL. 104
USAGE 105
WIDECHAR. 106
WINDOW 107
WRITABLE 107
XINFO 108
XML 111
XREF 112
オプションの中のブランク、コメント、およびストリング. 112
デフォルト・オプションの変更 113
%PROCESS ステートメントまたは *PROCESS ステートメントでのオプションの指定 113
% ステートメントの使用 115
%INCLUDE ステートメントの使用 115
コンパイラー・リストの使用 117
見出し情報 117
コンパイルに使用するオプション. 118
プリプロセッサー入力 118
SOURCE プログラム. 118
ステートメントのネスト・レベル. 119
ATTRIBUTE と相互参照テーブル 119
集合長さテーブル 120
ステートメント・オフセット・アドレス . . . 121
ストレージ・オフセット・リスト. 123
式および属性のリスト 124
ファイル参照テーブル 124
メッセージと戻りコード. 125
例 126

第 2 章 PL/I プリプロセッサー 131
インクルード・プリプロセッサー. 131
マクロ・プリプロセッサー 132
マクロ・プリプロセッサーのオプション . . . 133
マクロ・プリプロセッサーの例 136
SQL プリプロセッサー 137
プログラミングとコンパイルに関する考慮事項 138
SQL プリプロセッサー・オプション 141
PL/I アプリケーション内での SQL ステートメントのコーディング 145
LOB データの操作 157
SQL プリプロセッサー・メッセージの抑止 . . 161
CICS プリプロセッサー. 162
プログラミングとコンパイルに関する考慮事項 162
CICS プリプロセッサー・オプション 163
PL/I アプリケーション内での CICS ステートメントのコーディング 163
PL/I を使用した CICS トランザクションの作成 164
エラー処理 165

第 3 章 PL/I カタログ式プロシージャーの用法. 167
IBM 提供のカタログ式プロシージャー 167
コンパイルのみ (IBMZC) 168

コンパイルおよびバインド (IBMZCB) 170
コンパイル、バインド、および実行 (IBMZCBG) 172
| コンパイルのみ - 64 ビット (IBMQC) 174
| コンパイルおよびバインド - 64 ビット (IBMQCB) 175
| コンパイル、バインド、および実行 - 64 ビット (IBMQCBG) 177
カタログ式プロシージャーの呼び出し 179
複数のカタログ式プロシージャー呼び出しを指定 179
PL/I カタログ式プロシージャーの変更. 180
EXEC ステートメント 181
DD ステートメント 182

第 4 章 プログラムのコンパイル . . . 183
z/OS UNIX の下でのコンパイラーの呼び出し . . 183
入力ファイル 183
z/OS UNIX の下でのコンパイル時オプションの指定. 184
-qoption_keyword. 185
単一フラグおよび複数文字フラグ. 185
JCL を使用した z/OS の下でのコンパイラーの呼び出し. 186
EXEC ステートメント 186
標準データ・セット用の DD ステートメント 187
リスト (SYSPRINT) 189
ソース・ステートメント・ライブラリー (SYSLIB) 190
オプションの指定 190
EXEC ステートメントでのオプションの指定 191
オプション・ファイルを使用した EXEC ステートメントでのオプションの指定 191

第 5 章 31 ビット・プログラムに対するリンク・エディットおよび実行. . . . 193
31 ビット・プログラムに関するリンク・エディットの考慮事項 193
31 ビット・プログラムでバインダーを使用 . . 193
ENTRY カードの使用 193
31 ビット・プログラムに関する実行時の考慮事項 194
PRINT ファイルのフォーマット設定に関する規則 194
PRINT ファイル上のフォーマットを 31 ビット・プログラム用に変更. 194
自動プロンプト. 196
自動プロンプトの指定変更 196
長い入力行の句読法 197
GET LIST ステートメントと GET DATA ステートメントの句読法 197
GET EDIT での自動埋め込み 198
端末入力での SKIP の使用. 198
ENDFILE. 198
31 ビット・プログラムに関する SYSPRINT の考慮事項 198
MSGFILE(SYSPRINT) の使用 200
31 ビット・アプリケーションで自分のルーチンにおいて FETCH を使用 201
31 ビット・アプリケーションで Enterprise PL/I ルーチンをフェッチ 201
31 ビット・アプリケーションで PL/I MAIN ルーチンをフェッチ 210
31 ビット・アプリケーションで z/OS C ルーチンをフェッチ. 211
31 ビット・アプリケーションでアセンブラー・ルーチンをフェッチ 211
TSO/E のもとでの MAIN の呼び出し. 211
z/OS UNIX を指定した場合の MAIN の呼び出し 213

第 6 章 64 ビット・プログラムに対するリンク・エディットおよび実行. . . . 215
64 ビット・プログラムに関するリンク・エディットの考慮事項 215
64 ビット・プログラムでバインダーを使用 . . 215
64 ビット・プログラムで ENTRY カードを使用 215
64 ビット・プログラムに関する実行時の考慮事項 216
64 ビット・プログラムに関する SYSPRINT の考慮事項 216
64 ビット・アプリケーションで自分のルーチンにおいて FETCH を使用 216
64 ビット・アプリケーションで Enterprise PL/I ルーチンをフェッチ 216
64 ビット・アプリケーションで PL/I MAIN ルーチンをフェッチ 217
64 ビット・アプリケーションでアセンブラー・ルーチンをフェッチ 217
TSO/E のもとでの MAIN の呼び出し. 217
z/OS UNIX を指定した場合の MAIN の呼び出し 218

第 7 章 64 ビット・アプリケーションを開発する場合の考慮事項 221
コンパイラー・オプションを使用して 64 ビット・アプリケーションをビルド 221
LP(64) で属性 HANDLE および POINTER を使用 222
HANDLE 属性. 222
POINTER 属性. 222
LP(64) で ENTRY 変数を使用 223
LP(64) で組み込み関数を使用 223
SQL プログラムの考慮事項 225
31 ビット・ルーチンとのやり取り 227

第 2 部 入出力機能の使用 231

第 8 章 データ・セットとファイルの使用 233
ファイルの割り振り 233
z/OS でのデータ・セットとファイルの関連付け 235
複数のファイルと 1 つのデータ・セットの関連付け 237
複数のデータ・セットと 1 つのファイルの関連付け 237
複数のデータ・セットの連結 238
z/OS での HFS ファイルへのアクセス . . . 239

z/OS UNIX でのデータ・セットとファイルの関連付け 240
環境変数の使用. 240
OPEN ステートメントの TITLE オプションの使用 241
データ・セットに関連付けられていないファイルの使用の試み 243
PL/I によるデータ・セットの検索方法. . . . 243
DD_DDNAME 環境変数を使用した特性の指定 243
データ・セット特性の設定 250
ブロックおよびレコード. 251
情報交換コード. 251
レコード・フォーマット. 251
データ・セットの編成 253
ラベル. 254
データ定義 (DD) ステートメント 255
OPEN ステートメントの TITLE オプションの使用 257
PL/I ファイルとデータ・セットの関連付け . . 257
ENVIRONMENT 属性での特性の指定 259

第 9 章 ライブラリーの使用 273
ライブラリーのタイプ 273
ライブラリーの使用 274
ライブラリーの作成 274
SPACE パラメーター. 274
ライブラリー・メンバーの作成と更新 275
例: コンパイルされたオブジェクト・モジュール用の新規ライブラリーの作成 276
例: ロード・モジュールの既存ライブラリーへの配置 277
例: ライブラリー・メンバーの更新 277
ライブラリー・ディレクトリーからの情報の取り出し 279

第 10 章 連続データ・セットの定義と使用 281
ストリーム指向データ伝送の用法. 281
ストリーム入出力を用いたファイルの定義. . . 282
PL/I 動的割り振りを使用したストリーム・ファイルの定義 282
ENVIRONMENT オプションの指定. 282
ストリーム入出力によるデータ・セットの作成 285
ストリーム入出力によるデータ・セットへのアクセス 290
ストリーム入出力による PRINT ファイルの使用 292
31 ビット・プログラムに SYSIN ファイルおよび SYSPRINT ファイルを使用 297
64 ビット・プログラムに SYSIN ファイルおよび SYSPRINT ファイルを使用 298
端末からの入力の制御 298
データのフォーマット 299
ストリーム・ファイルおよびレコード・ファイル 300
PL/I 動的割り振りを使用して QSAM ファイルを定義 301
大文字と小文字. 301
ファイルの終わり 301
GET ステートメントの COPY オプション . . . 301

第 11 章 端末への出力の制御 303
PRINT ファイルのフォーマット 303
ストリーム・ファイルおよびレコード・ファイル 303
PUT EDIT コマンドの出力. 304

第 12 章 レコード単位データ伝送の使用 305
レコード・フォーマットの指定 306
レコード入出力を使用したファイルの定義. . . . 306
ENVIRONMENT オプションの指定. 307
CONSECUTIVE 307
ORGANIZATION(CONSECUTIVE) 308
CTLASA | CTL360 308
LEAVE | REREAD 310
レコード入出力によるデータ・セットの作成 . . . 310
必須情報 311
レコード入出力によるデータ・セットのアクセスおよび更新 312
必須情報 313
連続データ・セットの例. 313

第 13 章 領域データ・セットの定義と使用 317
PL/I 動的割り振りを使用した REGIONAL(1) データ・セットの定義 319
領域データ・セット用ファイルの定義 319
ENVIRONMENT オプションの指定. 320
REGIONAL データ・セットでのキーの使用 . . 321
REGIONAL(1) データ・セットの使用 321
ダミー・レコード 322
REGIONAL(1) データ・セットの作成 322
REGIONAL(1) データ・セットへのアクセスと更新 323
領域データ・セットの作成時、および領域データ・セットへのアクセス時の必須情報. 327

第 14 章 VSAM データ・セットの定義と使用. 331
PL/I 動的割り振りを使用した VSAM ファイルの定義 331
VSAM データ・セットの使用 331
VSAM データ・セットを使用してプログラムを実行 331
代替索引パスとファイルをペア化. 332
VSAM 編成 332
VSAM データ・セットのキー 335
データ・セット・タイプの選択 336
VSAM データ・セットのファイルの定義 339
ENVIRONMENT オプションの指定. 339
パフォーマンス・オプション 343
代替索引パスのファイルを定義 343

VSAM データ・セットの定義 344
入力順データ・セット 344
ESDS のロード 345
SEQUENTIAL ファイルを使用した ESDS へのアクセス 345
キー順および索引付き入力順データ・セット . . . 348
KSDS または索引付き ESDS のロード 350
SEQUENTIAL ファイルを使用した KSDS または索引付き ESDS へのアクセス 352
DIRECT ファイルを使用した KSDS または索引付き ESDS へのアクセス 352
KSDS の更新 354
KSDS または索引付き ESDS の代替索引 . . . 355
相対レコード・データ・セット 363
RRDS のロード 364
SEQUENTIAL ファイルを使用した RRDS へのアクセス 367
DIRECT ファイルを使った RRDS へのアクセス 367
非 VSAM データ・セットに対して定義されたファイルの使用 369
共用データ・セットの使用 369

第 3 部 プログラムの改良 371

第 15 章 パフォーマンスの向上 373
最適なパフォーマンスのためのコンパイラー・オプションの選択 373
OPTIMIZE 373
GONUMBER 374
ARCH 374
REDUCE 374
RULES 374
PREFIX 376
CONVERSION 376
FIXEDOVERFLOW 376
DEFAULT 376
パフォーマンスを向上させるコンパイラー・オプションの要約 380
パフォーマンス向上のためのコーディング 381
DATA ディレクティブ入出力 381
入力専用パラメーター 381
GOTO ステートメント 382
ストリングの割り当て 382
ループ制御変数 382
PACKAGE 対ネストされた PROCEDURE . . 383
REDUCIBLE 関数 384
REPATTERN の使用 385
DESCLOCATOR または DESCLIST 385
DEFINED 対 UNION 385
名前付き定数対静的変数 386
ライブラリー・ルーチンの呼び出しの回避 . . . 387
ライブラリー・ルーチンのプリロード 388

第 4 部 他の製品に対するインターフェースの使用 389

第 16 章 ソート・プログラムの使用 391
ソート・プログラムの使用準備 392
ソート・タイプの選択 392
ソート・フィールドの指定 395
ソートするレコードの指定 397
ソート・プログラムに必要なストレージの決定 398
ソート・プログラムの呼び出し 399
例 1 400
例 2 400
例 3 401
例 4 401
例 5 401
ソートが成功したかどうかの判別 401
ソート・プログラム用のデータ・セットの確立 402
ソート・データの入出力 403
データ入出力処理ルーチン 403
E15 — 入力処理ルーチン (ソート出口 E15) . . 404
E35 — 出力処理ルーチン (ソート出口 E35) . . 407
PLISRTA の呼び出し例 408
PLISRTB の呼び出し例 409
PLISRTC の呼び出し例 410
PLISRTD の呼び出し例 411
可変長レコードのソートの例 413

第 17 章 C との ILC 415
同等なデータ・タイプ 415
単純なタイプの一致 415
struct タイプの一致 416
enum タイプの一致 416
ファイル・タイプの一致 417
C 関数を使用する 417
一致する単純パラメーター・タイプ 418
一致するストリング・パラメーター・タイプ . . 421
ENTRY を戻す関数 423
リンケージ 424
出力および入力の共有 426
出力の共用 426
入力の共有 427
ATTACH ステートメントの使用 427
C 標準ストリームのリダイレクト 427
要約 428

第 18 章 Java とのインターフェース 429
Java Native Interface (JNI) 429
Java からの PL/I プログラムの呼び出し 430
JNI サンプル・プログラム #1 -「Hello World」 430
ステップ 1: Java プログラムの作成 431
ステップ 2: Java プログラムのコンパイル . . 432
ステップ 3: PL/I プログラムの作成 432
ステップ 4: PL/I プログラムのコンパイルとリンク 434
ステップ 5: サンプル・プログラムの実行 . . . 434
JNI サンプル・プログラム #2 - ストリングを渡す 434
ステップ 1: Java プログラムの作成 435
ステップ 2: Java プログラムのコンパイル . . 437
ステップ 3: PL/I プログラムの作成 437

ステップ 4: PL/I プログラムのコンパイルとリンク 440
ステップ 5: サンプル・プログラムの実行 . . . 440
JNI サンプル・プログラム #3 - 整数の引き渡し 440
ステップ 1: Java プログラムの作成 441
ステップ 2: Java プログラムのコンパイル . . 443
ステップ 3: PL/I プログラムの作成. 443
ステップ 4: PL/I プログラムのコンパイルとリンク 445
ステップ 5: サンプル・プログラムの実行 . . . 445
JNI サンプル・プログラム #4 - Java 呼び出し API. 445
ステップ 1: Java プログラムの作成 445
ステップ 2: Java プログラムのコンパイル . . 446
ステップ 3: PL/I プログラムの作成. 446
ステップ 4: PL/I プログラムのコンパイルとリンク 450
ステップ 5: サンプル・プログラムの実行 . . . 450
既存の Java VM へのプログラムの接続 450
Java および PL/I の同等なデータ・タイプの判別 451

第 5 部 特殊プログラミング・タスク 453

第 19 章 PLISAXA および PLISAXB XML パーサーの使用 455
概説 455
PLISAXA 組み込みサブルーチン 456
PLISAXB 組み込みサブルーチン 457
SAX イベント構造体 457
start_of_document. 458
version_information 458
encoding_declaration. 458
standalone_declaration 458
document_type_declaration. 458
end_of_document 459
start_of_element 459
attribute_name 459
attribute_characters 459
attribute_predefined_reference. 459
attribute_character_reference 459
end_of_element. 460
start_of_CDATA_section 460
end_of_CDATA_section 460
content_characters. 460
content_predefined_reference 460
content_character_reference. 461
processing_instruction 461
comment 461
unknown_attribute_reference 461
unknown_content_reference 461
start_of_prefix_mapping. 461
end_of_prefix_mapping 461
exception. 461
イベント関数に渡されるパラメーター 462
XML 文書のコード化文字セット 463
サポートされる EBCDIC コード・ページ . . . 463
サポートされる ASCII コード・ページ. . . . 464
コード・ページの指定 464
例外 465
例 466
継続可能な例外コード 477
例外コードの終了 481

第 20 章 PLISAXC および PLISAXD XML パーサーの使用 485
概説 485
PLISAXC 組み込みサブルーチン 486
PLISAXD 組み込みサブルーチン 487
SAX イベント構造体 487
start_of_document. 488
version_information 488
encoding_declaration. 488
standalone_declaration 488
document_type_declaration. 488
end_of_document 488
start_of_element 489
attribute_name 489
attribute_characters 489
end_of_element. 489
start_of_CDATA_section 489
end_of_CDATA_section 489
content_characters. 490
processing_instruction 490
comment 490
namespace_declare 490
end_of_input 491
unresolved_reference. 491
exception. 491
イベント関数に渡されるパラメーター 491
イベントにおける差異 493
XML 文書のコード化文字セット 495
サポートされるコード・ページ 495
コード・ページの指定 496
例外 496
妥当性検査を伴う XML 文書の構文解析 497
XML スキーマ 497
OSR の作成 498
単純な文書での例 499
PLISAXC 組み込みサブルーチンの使用例 . . . 499
PLISAXD 組み込みサブルーチンの使用例 . . . 509

第 21 章 PLIDUMP の用法 521
PLIDUMP の使用上の注意 522
PLIDUMP 出力内の変数の検出 523
AUTOMATIC 変数の検出 523
STATIC 変数の検出 525
CONTROLLED 変数の検出 526
保存されたコンパイル・データ 529
Copyright 529
タイム・スタンプ 530

保存されたオプション・ストリング 530

第 22 章 割り込みとアテンションの処理 533
ATTENTION ON ユニットの使用 534
デバッグ・ツールとの対話 534

第 23 章 チェックポイント/再始動機能の使用. 535
チェックポイント・レコードの要求 535
チェックポイント・データ・セットの定義. . . . 536
再始動の要求 537
システム障害後の自動再始動 537
プログラム内の自動再始動 538
据え置き再始動. 538
チェックポイント/再始動活動の変更 538

第 24 章 ユーザー出口の用法 541
コンパイラー・ユーザー出口によって実行されるプロシージャー 541
グローバル制御ブロックの構造 542
IBM 提供のコンパイラー出口、IBMUEXIT . . . 544
コンパイラー・ユーザー出口の活動化 544
コンパイラー・ユーザー出口のカスタマイズ . . . 544
SYSUEXIT の変更. 544
独自のコンパイラー出口の作成 545
初期化プロシージャーの作成 545
メッセージ・フィルター操作プロシージャーの作成 546
終了プロシージャーの作成 548
SQL メッセージの抑止例 549

第 25 章 PL/I 記述子 557
引数の引き渡し. 557
記述子リストによる引数の引き渡し 557
ロケーター/記述子で引数を渡す 558
CMPAT(V*) 記述子 558
ストリング記述子 559
配列記述子 560
CMPAT(LE) 記述子 561
ストリング記述子 561
配列記述子 562

第 6 部 付録 563

付録. SYSADATA メッセージ情報 . . . 565
SYSADATA ファイルについて 565
サマリー・レコード 567
オプション・レコード 568
カウンター・レコード 568
リテラル・レコード 569
ファイル・レコード 569
メッセージ・レコード 570
SYSADATA シンボル情報について 571
序数タイプ・レコード 571
序数エレメント・レコード 572
シンボル・レコード 573
SYSADATA 構文情報について 576
ソース・レコード 576
トークン・レコード 577
構文レコード 578

特記事項. 587
商標 588

参考文献. 589
PL/I 資料 589
関連資料 589

用語集. 593

索引 613

表

1. Enterprise PL/I 付属資料の使用方法 xv
2. z/OS 言語環境プログラム 付属資料の使用方法 xvi
3. コンパイル時オプション、省略形、および IBM 提供のデフォルト値 4
4. サポート対象の CCSID 17
5. SYSTEM オプション・テーブル 100
6. リストされたメッセージの最低重大度を選択するための FLAG オプションの使用 . . . 126
7. PL/I エラー・コードと戻りコードの説明 126
8. SQL プリプロセッサー・オプション、および IBM 提供のデフォルト値 141
9. PL/I 宣言から生成される SQL データ・タイプ 153
10. SQL TYPE 宣言から生成される SQL データ・タイプ 153
11. SQL データ・タイプと PL/I 宣言の対応 154
12. SQL データ・タイプと SQL TYPE 宣言の対応 155
13. z/OS UNIX の下で Enterprise PL/I によりサポートされるコンパイル時オプション・フラグ 186
14. コンパイラーの標準データ・セット 187
15. PL/I ファイル宣言の属性 260
16. PL/I レコード入出力で使用できるデータ・セット・タイプの比較 269
17. ライブラリー作成時に必要な情報 274
18. 連続データ・セットの作成と連続データ・セットへのアクセスで使用できるステートメントとオプション 305
19. IBM マシン・コード印刷制御文字 (CTL360) 309
20. LEAVE および REREAD オプションの影響 310
21. レコード入出力による連続データ・セットの作成: DD ステートメントの必須パラメーター 311
22. レコード入出力による連続データ・セットへのアクセス: DD ステートメントの必須パラメーター 312
23. 領域データ・セットの作成と領域データ・セットへのアクセスで使用できるステートメントとオプション 318
24. 領域データ・セットの作成: DD ステートメントの必須パラメーター 328
25. 領域データ・セットの DCB サブパラメーター 329
26. 領域データ・セットへのアクセス: DD ステートメントの必須パラメーター 329
27. VSAM データ・セットのタイプ、および対応する PL/I データ・セット編成 332
28. VSAM データ・セットのタイプと利点 334
29. VSAM データ・セットと使用できるファイル属性 337
30. 代替索引パスで実行できる処理 338
31. VSAM 入力順データ・セットのロードと入力順データ・セットへのアクセスで使用できるステートメントとオプション 344
32. VSAM 索引付きデータ・セットのロードとそれへのアクセスに使用できるステートメントとオプション 348
33. VSAM 相対レコード・データ・セットのロードとそれへのアクセスに使用できるステートメントとオプション 363
34. PLISRT*x* (*x* = A、B、C、または D) に対するエントリー・ポイントおよび引数 399
35. C と PL/I の同等なタイプ 415
36. Java 基本タイプと、同等の PL/I ネイティブ・タイプ 451
37. 継続可能な例外 477
38. 終了例外 481

© Copyright IBM Corp. 1999, 2017

図

1. ライブラリーからのソース・ステートメントの組み込み 117
2. ステートメント番号の検索 (コンパイラー・リストの例) 122
3. ステートメント番号の検索 (ランタイム・メッセージの例) 122
4. コンパイラー・リスト例 127
5. ソース・デックを作成するためのマクロ・プリプロセッサーの使用 137
6. SQLCA の PL/I 宣言 146
7. SQL 記述子域の PL/I 宣言. 147
8. 標識変数を含む SQL ステートメント 157
9. pliclob サンプル・プログラム 160
10. カタログ式プロシージャーの呼び出し 168
11. カタログ式プロシージャー IBMZC 170
12. カタログ式プロシージャー IBMZCB 171
13. カタログ式プロシージャー IBMZCBG 173
14. カタログ式プロシージャー IBMQC (64 ビット) 175
15. カタログ式プロシージャー IBMQCB (64 ビット) 176
16. カタログ式プロシージャー IBMQCBG (64 ビット) 178
17. PLITABS の宣言 195
18. PAGELENGTH および PAGESIZE 196
19. 自動プロンプトを使用した出力 197
20. 自動プロンプトを使用しない出力 197
21. ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル JCL 204
22. SYSTEM(STD) オプションを使用したときに TSO のもとで CPPL からプログラム実引数を表示するサンプル・プログラム 212
23. z/OS UNIX 引数と環境変数を表示するサンプル・プログラム 214
24. SYSTEM(STD) オプションを使用したときに TSO のもとで CPPL からプログラム実引数を表示するサンプル・プログラム 218
25. z/OS UNIX 引数と環境変数を表示するサンプル・プログラム 219
26. 固定長レコード 252
27. オペレーティング・システムによる DCB への情報の組み込みの方法 258
28. コンパイルされたオブジェクト・モジュール用の新規ライブラリーの作成 276
29. ロード・モジュールの既存ライブラリーへの配置 277
30. PL/I プログラム内でのライブラリー・メンバーの作成 278
31. ライブラリー・メンバーの更新 278
32. ストリーム指向データ伝送によるデータ・セットの作成 287
33. グラフィック・データのストリーム・ファイルへの書き込み 289
34. ストリーム指向データ伝送によるデータ・セットへのアクセス 292
35. ストリーム・データ伝送による印刷ファイルの作成 295
36. 事前設定済みのタブ設定を変更する場合の PL/I 構造体 PLITABS 297
37. 米国標準規格の印刷およびカード穿孔制御文字 (CTLASA) 309
38. 連続データ・セットのマージ、ソート、作成と連続データ・セットへのアクセス 314
39. レコード単位データ伝送の印刷 316
40. REGIONAL(1) データ・セットの作成 323
41. REGIONAL(1) データ・セットの更新 326
42. ESDS の定義とロード. 347
43. ESDS の更新. 348
44. キー順データ・セット (KSDS) の定義とロード 351
45. KSDS の更新 353
46. ESDS の固有キー代替索引パスを作成 356
47. ESDS の非固有キー代替索引パスを作成 357
48. KSDS の固有キー代替索引パスを作成 358
49. ESDS での代替索引パスと逆方向読み取り 360
50. KSDS アクセス用の固有代替索引パスの使用 362
51. 相対レコード・データ・セット (RRDS) の定義とロード 366
52. RRDS の更新 368
53. ソート・プログラムの制御の流れ 394
54. 入力および出力処理サブルーチンのフローチャート 405
55. 入力プロシージャー用の骨組みコード 406
56. 出力処理プロシージャー用の骨組みコード 408
57. PLISRTA - 入力データ・セットから出力データ・セットへのソート 409
58. PLISRTB - 入力処理ルーチンから出力データ・セットへのソート 410
59. PLISRTC - 入力データ・セットから出力処理ルーチンへのソート 411
60. PLISRTD - 入力処理ルーチンから出力処理ルーチンへのソート 412
61. 入出力処理ルーチンを使った可変長レコードのソート 413
62. 単純なタイプの一致 416
63. struct タイプの一致の例 416
64. enum タイプの一致の例 417
65. FILE タイプの C 宣言の開始 417
66. C ファイルと一致する PL/I 417
67. fopen と fread を使用してファイルをダンプするコードの例 418
68. filedump プログラムの宣言. 418

© Copyright IBM Corp. 1999, 2017

69. fread の C 宣言 419
70. fread の誤った宣言 (その 1) 419
71. fread の誤った宣言 (その 2) 419
72. fread の誤った宣言 (その 3) 419
73. RETURNS BYADDR に対して生成されるコード 420
74. fread の正しい宣言 420
75. RETURNS BYVALUE に対して生成されるコード 421
76. fopen の誤った宣言 (その 1) 421
77. fopen の誤った宣言 (その 2) 421
78. fopen の正しい宣言 422
79. fopen の正しく最適な宣言 422
80. fclose の宣言 422
81. filedump をコンパイルして実行するためのコマンド 422
82. filedump の実行結果の出力. 423
83. C qsort 関数の比較ルーチンの例 423
84. C qsort 関数を使用するためのコード例 423
85. qsort の誤った宣言 424
86. qsort の正しい宣言 424
87. パラメーターが BYADDR である場合のコード 425
88. パラメーターが BYVALUE である場合のコード 426
89. Java サンプル・プログラム #2 - ストリングの引き渡し 436
90. PL/I サンプル・プログラム #2 - ストリングの引き渡し 439
91. Java サンプル・プログラム #3 - 整数の引き渡し 442
92. PL/I サンプル・プログラム #3 - 整数の引き渡し 444
93. Java サンプル・プログラム #4 - ストリングの受け取りおよび出力 446
94. PL/I サンプル・プログラム #4 - Java 呼び出し API の呼び出し 449
95. サンプル XML 文書 458
96. PLISAXA のコーディング例 - 型宣言 466
97. PLISAXA のコーディング例 - イベント構造体 467
98. PLISAXA のコーディング例 - メインルーチン 468
99. PLISAXA のコーディング例 - イベント・ルーチン 469
100. PLISAXA のコーディング例 - プログラム出力 477
101. サンプル XML 文書 488
102. PLISAXC のコーディング例 - 型宣言 500
103. PLISAXC のコーディング例 - イベント構造体 501
104. PLISAXC のコーディング例 - メインルーチン 502
105. PLISAXC のコーディング例 - イベント・ルーチン 503
106. PLISAXC のコーディング例 - プログラム出力 509
107. PLISAXD のコーディング例 - イベント・ルーチン 510
108. PLISAXD サンプルからの出力. 520
109. PLIDUMP を呼び出す PL/I ルーチンの例 521
110. ATTENTION ON ユニットの使用 534
111. PL/I コンパイラー・ユーザー出口のプロシージャー 542
112. ユーザー出口入力ファイルの例 545
113. SQL メッセージの抑止 549
114. 序数値としてエンコードされたレコード・タイプ 566
115. レコードのヘッダー部分の宣言 567
116. サマリー・レコードの宣言 568
117. カウンター・レコードの宣言 568
118. リテラル・レコードの宣言 569
119. ファイル・レコードの宣言 570
120. メッセージ・レコードの宣言 571
121. 序数タイプ・レコードの宣言 572
122. 序数エレメント・レコードの宣言 573
123. 構造体のエレメントに割り当てられたシンボル索引 574
124. 変数のデータ・タイプ 575
125. ソース・レコードの宣言 577
126. トークン・レコードの宣言 578
127. トークン・レコードの種類の宣言 578
128. プログラムのブロックに割り当てられるノード索引 579
129. 構文レコードの宣言 579
130. 構文レコードの種類の宣言 582
131. プログラムの構文レコードに割り当てられるノード索引 583
132. 式の種類の宣言 584
133. 数値の種類の宣言 585
134. 字句の種類の宣言 586

はじめに

本書について

本書は、PL/I プログラマーおよびシステム・プログラマーを対象とした資料です。PL/I プログラムをコンパイルするための Enterprise PL/I for z/OS® の使用方法を理解するのに役立ちます。また本書は、プログラム・パフォーマンスを最適化し、エラーに対する処理を行うのに必要となるオペレーティング・システムの各種機能についても説明しています。

重要**:** 本書では、Enterprise PL/I for z/OS を Enterprise PL/I と呼びます。

Enterprise PL/I for z/OS のランタイム環境

Enterprise PL/I は、ランタイム環境として言語環境プログラムを使用します。これは 言語環境プログラム体系に適合し、ランタイム環境を他の言語環境プログラム適合言語と共用することができます。

言語環境プログラムはランタイム・オプションおよび呼び出し可能サービスの共通セットを提供します。また、各 ILC 呼び出しの言語固有の初期化および終了をなくすことによって、高水準言語 (HLL) とアセンブラーの間の言語間通信 (ILC) も改善しています。

資料の使用

Enterprise PL/I に付属の資料は、PL/I を使用してプログラミングを行うときに役立つように設計されています。言語環境プログラムに付属の資料は、Enterprise PL/I を使用して生成されたアプリケーション用にランタイム環境を管理するときに役立つように設計されています。それぞれの資料がさまざまな作業に役立ちます。

下の表には、Enterprise PL/I および言語環境プログラムの資料の使用方法を示しています。 ご使用のコンパイラーとランタイム環境の両方に関する情報を知る必要があります。これらの資料および関連資料の正式名称および資料番号については、589 ページの『参考文献』を参照してください。

PL/I 情報

表 *1. Enterprise PL/I* 付属資料の使用方法

目的...	使用するもの...
Enterprise PL/I の評価	データ・シート (ファクト・シート)
保証情報の理解	Licensed Program Specifications
Enterprise PL/I の計画とインストール	Enterprise PL/I プログラム・ディレクトリー
コンパイラーおよびランタイム変更作業の理解と、プログラムの Enterprise PL/I および言語環境プログラムへの適合	コンパイラーおよびランタイム 移行ガイド

© Copyright IBM Corp. 1999, 2017

表 1. *Enterprise PL/I* 付属資料の使用方法 (続き)

目的...	使用するもの...
プログラムの準備とテスト、およびコンパイラー・オプションについての詳細情報の入手	プログラミング・ガイド
PL/I の構文および言語エレメントの仕様についての詳細情報の入手	言語解説書
コンパイラーの問題診断および IBM® への連絡	診断ガイド
コンパイル時メッセージについての詳細情報の入手	メッセージおよびコード

言語環境プログラム情報

表 2. *z/OS* 言語環境プログラム 付属資料の使用方法

目的...	使用するもの...
言語環境プログラムの評価	概念
言語環境プログラムの計画	概念 ランタイム・アプリケーション マイグレーション・ガイド
z/OS に言語環境プログラムをインストール	z/OS プログラム・ディレクトリー
z/OS 上で言語環境プログラムをカスタマイズ	カスタマイズ
言語環境プログラムのプログラム・モデルおよび概念の理解	概念 プログラミング・ガイド
言語環境プログラムランタイム・オプションおよび呼び出し可能サービスの構文の検索	プログラミング・リファレンス
言語環境プログラムで実行されるアプリケーションの開発	プログラミング・ガイドおよび該当する言語のプログラミング・ガイド
言語環境プログラムで実行されるアプリケーションのデバッグ、ランタイム・メッセージに関する詳細情報の入手、言語環境プログラムの問題の診断	デバッグのガイドとランタイム・メッセージ
言語間通信 (ILC) アプリケーションの開発	ILC アプリケーションの作成
言語環境プログラムへのアプリケーションの移行	「ランタイム・アプリケーション マイグレーション・ガイド」、および言語環境プログラムで使用できる各言語の移行ガイド

本書で使用されている表記規則

本書では、『使用する規則』および xx ページの『表記記号の読み方』に示す規則、構文図の書き方、および表記を用いて PL/I および非 PL/I のプログラミング構文を説明しています。

使用する規則

本書における一部のプログラミング構文では、各種エレメントを活字フォントで区別しています。

- 大文字で (UPPERCASE のように) 示した項目は、そのとおりにタイプする必要がある重要な項目です。
- 小文字で (lowercase のように) 示した項目は、適切な名前または値に置き換えてタイプする必要があるユーザー提供変数です。 変数は英字で始まり、ハイフン、数字、または下線文字 (_) を入れることができます。
- 数字 は、数字 (0 から 9 まで) に置き換えなければならないことを示します。
- *DO グループ* は、DO グループに置き換えなければならないことを示します。
- 下線付きの項目は、デフォルト・オプションです。
- 例は、上段専用文字で示されます。
- 特に指示がなければ、反復可能項目は 1 つ以上のブランクを使っておのおのを区切ります。

注: xx ページの『表記記号の読み方』の説明にあるような純粋な表記記号ではない記号が示されている場合、それらはすべてプログラミング構文自体の一部です。

これらの規則に従ったプログラミング構文の例については、 xx ページの『表記例』を参照してください。

構文記法の読み方

本書で使用されている構文図には以下の規則が適用されます。

矢印記号

構文図は、左から右、上から下へと線をたどって読んでください。

►►── ステートメントはここから始まります。

──► ステートメントの構文は次の行へ続きます。

►── ステートメントは前の行から続いています。

──►◄ ステートメントはここで終わります。

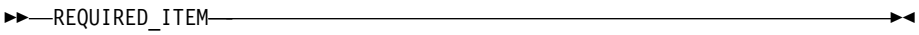
完結したステートメント以外の構文単位の図は、►── 記号で始まり、──► 記号で終わります。

規則

- キーワード、許容される同義語、および予約パラメーターは、MVS™ および OS/2 プラットフォームでは大文字で示され、UNIX プラットフォームでは小文字で示されます。これらの項目は示されたとおりに入力する必要があります。
- 変数は、小文字のイタリック体で示します (例えば、*column-name*)。これらはユーザー定義のパラメーターまたはサブオプションを表します。
- コマンドの入力において、パラメーターおよびキーワードを区切る句読記号がない場合は、少なくとも 1 つのブランクで区切る必要があります。
- 句読記号 (スラッシュ、コンマ、ピリオド、括弧、引用符、等号) と数字は、示されたとおりに入力する必要があります。
- 脚注は、(1) のように番号を括弧に入れて示します。
- ■ 記号は 1 つのブランク位置を示します。

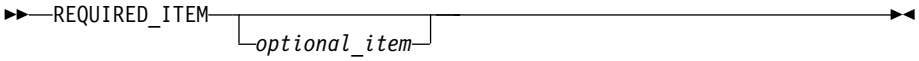
必須項目

必須項目は横線 (メインパス) に示します。

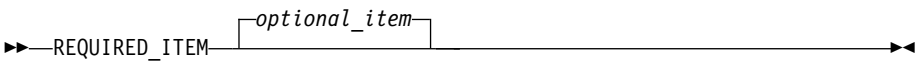


オプション項目

オプション項目は、メインパスの下に示します。

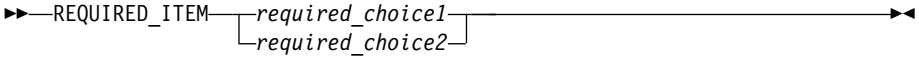


メインパスより上にオプション項目を示すこともあります。これは読みやすくするためで、ステートメントの実行には影響を及ぼしません。

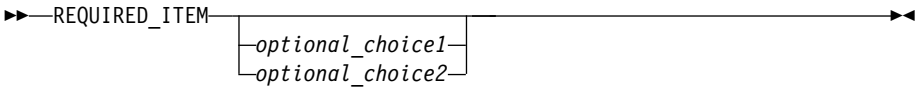


複数の必須項目またはオプション項目

複数の項目から選択する場合には、それらの項目が縦に重なって、スタックを形成しています。 複数の項目からいずれか 1 つを選択しなければならない 場合は、スタック上の項目のうち、1 つがメインパス上に置かれます。

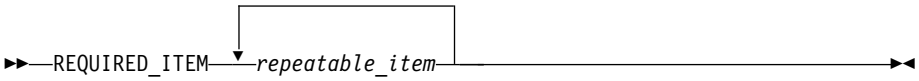


項目がオプショナルである場合、メインパスの下にある支線上に縦に並んだ項目として示されます。

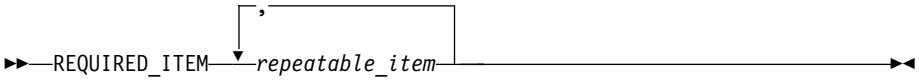


反復可能項目

メインパスの上方を通って左側へ戻る矢印は、項目が反復可能であることを示します。



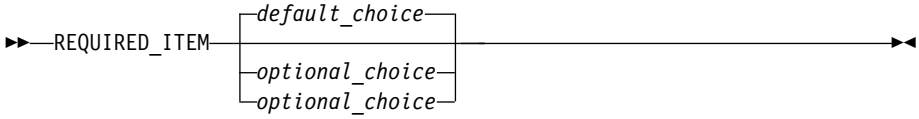
反復矢印の途中にコンマがある場合は、反復される項目をコンマで区切らなければなりません。



スタックの上の繰り返しを示す矢印は、スタックから複数の選択項目を指定できることを示しています。

デフォルト・キーワード

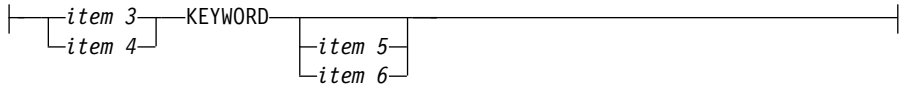
IBM 提供のデフォルト・キーワードはメインパスより上に示され、それ以外の選択項目はメインパスより下に示されます。 構文図の下にあるパラメーター・リストでは、デフォルト選択項目に下線を付けてあります。



フラグメント

構文図は、フラグメント (部分) に分割する必要がある場合があります。 フラグメントは文字またはフラグメント名を用いて | A | のように表します。フラグメントは主図のあとに置かれます。 次の例は、フラグメントの使い方を示したものです。

A:



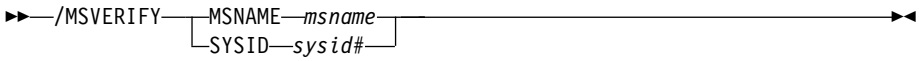
置換ブロック

いくつかのパラメーターの集合を **<A>** のような置換ブロックで表すことがあります。 例えば、/VERB という仮のコマンドで、/VERB LINE 1、/VERB EITHER LINE 1、または /VERB OR LINE 1 と入力することができます。

ここで、<A> は次のようになります。

パラメーターの終わり

数値を持つパラメーターは記号 '#' で終わり、名前であるパラメーターは 'name' で終わり、汎用とすることができるパラメーターは '*' で終わります。



この例の MSNAME キーワードは名前の値をサポートし、SYSID キーワードは数値をサポートします。

表記記号の読み方

本書のプログラミング構文の一部は、表記記号を用いて表されています。同一構文の他の IBM 資料における記述法に合わせるため、またはテーブルまたは見出し内で 1 行に構文を入れられるようにするためです。

- 中括弧 { } は、選択項目を示します。 項目の 1 つに下線が付いている (デフォルトを示す) 場合、または項目がすべて大括弧で囲まれている場合を除き、少なくとも 1 つの項目を選択する必要があります。
- 単一の縦線 | で区切られた項目は、代替項目です。 単一の縦線で区切られた項目 (または項目のグループ) では、その中の 1 つしか選択できません。 (2 重縦線 || は、代替項目ではなく、連結演算を指定します。 2 重縦線の詳細については、「*PL/I* 言語解説書」を参照してください。)
- 大括弧 [ ] に囲まれたものはすべてオプショナルです。 項目が縦に積み重ねられて大括弧で囲まれていれば、1 つの項目しか指定することはできません。
- 省略符号 ... は、直前の項目と同じタイプの項目を複数回指定できることを示します。

表記例

次の PL/I 構文は、『表記記号の読み方』の表記記号の実例です。

```
DCL file-reference FILE STREAM
                  {INPUT | OUTPUT [PRINT]}
                  ENVIRONMENT(option ...);
```

この例は、次のように解釈します。

- 最初の行は、*file-reference* を除き、このとおりのスペルで入力しなければなりません。 file-reference は、参照するファイルの名前に置き換えます。
- 2 行目では INPUT または OUTPUT を指定することができますが、両方を指定することはできません。 OUTPUT を指定した場合は、任意で PRINT も指定できます。 どちらも選択しなかった場合は、デフォルトにより INPUT が採用されます。
- 1 つ以上のブランクでおのおのを区切り 1 つ以上のオプションに置き換えなければならない *option ...* を除き、最後の行は示されたとおりのスペル (括弧とセミコロンも含む) で入力しなければなりません。

変更の要約

本リリースにおける機能強化

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- INLIST のコードが、最初の引数が CHAR(*n*) (1 <= *n* <= 4) で、すべての他の引数が長さ <= *n* の CHAR である場合に、その機能が強化されました。
- SELECT(*x*) の *x* が CHAR である場合に、そのコードが強化されました。これに該当する場合、コンパイラーは二分探索によって SELECT を実行します。

使いやすさの向上

- コンパイラーは、到達不能な ASSERT UNREACHABLE ステートメントに、他の到達不能なステートメントにフラグを立てるときとは異なるメッセージでフラグを立てるようになりました。
- コンパイラーは、より速い MEMCU12 または MEMCU21 に変換できる可能性がある、MEMCONVERT 組み込み参照にフラグを立てるようになりました。
- コンパイラーは、REPATTERN に変換しなければならない可能性がある、TRANSLATE 組み込み参照にフラグを立てるようになりました。
- コンパイラーは、他の構造のメンバーである AGGREGATE リスト・タイプの構造において、その機能が強化されました。
- コンパイラーは、PROCEDURE ブロックまたは BEGIN ブロックの INLINE をオフにするシチュエーションの多くで、通知メッセージを出すようになりました。
- コンパイラーは、ネストされた PROCEDURE および BEGIN ブロックで、INLINE オプションを受け入れるようになりました。
- 属性リストに、長さ 256 以下の定数 CHARACTER および BIT における VALUE 属性、また数値 PICTURE 定数における VALUE 属性の内容が表示されるようになりました。

コンパイラー・オプション機能拡張

- 新しい 9 ページの『ASSERT』 コンパイラー・オプションは、ASSERT ステートメントが、ASSERTION 条件を発生させるデフォルト・ライブラリー・ルーチンを呼び出すか、またはユーザーによって指定されたルーチンを呼び出すかを制御します。
- 新しい 13 ページの『CASE』 コンパイラー・オプションは、一部の名前が大文字で表示されるか、またはソース・プログラムでの表示と同じフォーマットで表示されるかを制御します。
- 16 ページの『CMPAT』コンパイラー・オプションでサブオプション CMPAT(LE) および CMPAT(V1) が復活しました。
- 21 ページの『DBRMLIB』新規コンパイラー・オプションは、SQL プリプロセッサーが UNIX システム・サービスで動作しているときにその SQL プリプロセッサーの DBRM データ・セットを作成する場所を制御します。
- 79 ページの『RULES』 コンパイラー・オプションには、以下の新しいサブオプションがあります。

  LAXCONV | NOLAXCONV
  RULES(NOLAXCONV) を使用して、オペランドに算術型がない演算式にフラグを立てます。

  LAXINTERFACE | NOLAXINTERFACE
  RULES(NOLAXINTERFACE) を使用して、外部 PROCEDURE のそれぞれに有効な明示宣言を含まないコードにフラグを立てることができます。

  MULTIENTRY | NOMULTIENTRY
  RULES(NOMULTIENTRY) を使用して、ENTRY ステートメントを含むコードにフラグを立てることができます。

MULTIEXIT | NOMULTIEXIT
RULES(NOMULTIEXIT) を使用して、複数の RETURN ステートメントを含むコードにフラグを立てることができます。

MULTISEMI | NOMULTISEMI
RULES(NOMULTISEMI) を使用すれば、複数のセミコロンを含むソース行にフラグを立てることができます。

UNREFCTL | NOUNREFCTL
RULES(NOUNREFCTL) を使用して、非参照 CTL 変数にフラグを立てることができます。

UNREFDEFINED | NOUNREFDEFINED
RULES(NOUNREFDEFINED) を使用して、非参照 DEFINED 変数にフラグを立てることができます。

UNREFENTRY | NOUNREFENTRY
RULES(NOUNREFENTRY) を使用して、非参照 ENTRY 定数にフラグを立てることができます。

UNREFFILE | NOUNREFFILE
RULES(NOUNREFFILE) を使用して、非参照 FILE 定数にフラグを立てることができます。

UNREFSTATIC | NOUNREFSTATIC
RULES(NOUNREFSTATIC) を使用して、非参照 STATIC 変数にフラグを立てることができます。

YY | NOYY
RULES(NOYY) を使用して、2 桁の年の使用に対してフラグを立てることができます。

- また、 79 ページの『RULES』 コンパイラー・オプションの NOELSEIF サブオプションは、単純な DO-END で囲まれた IF ステートメントの直後に続く ELSE ステートメントにフラグを立てるようになりました。
- 79 ページの『RULES』 コンパイラー・オプションの NOGOTO オプションは、LOOSEFORWARD サブオプションを受け入れるようになったため、前方に分岐する GOTO ステートメントには NOGOTO フラグ設定が適用されません。

CICS 機能強化

- CICS プリプロセッサー出力に、そのプリプロセッサーの稼働中に有効な CICS オプションすべてのリストが組み込まれるようになりました。

V5R1 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- Enterprise PL/I は、ベクトル機構をさらに活用するようになりました。
- 桁数の多い固定小数点の除算のいくつかが、10 進浮動小数点 (DFP) 機能を使用して行われるようになりました。これにより、一部の ZERODIVIDE 例外は INVALIDOP として報告される可能性があります。

使いやすさの向上

- Enterprise PL/I は、64 ビット・アプリケーションのコードをコンパイルするためのサポートを提供するようになりました。詳しくは、 221 ページの『第 7 章 64 ビット・アプリケーションを開発する場合の考慮事項』を参照してください。
- コンパイラーは、IBMZIOP におけるオプション・ストリング同士が相互に同じ場合、および指定変更できないいずれかのオプションと矛盾するオプションをユーザーが指定した場合に、警告メッセージを発行するようになりました。
- コンパイラーは、コンパイル時に使用するコンパイラー・オプションに関する情報も、生成する SYSADATA ファイルに組み込むようになりました。SYSADATA レコードの宣言はすべて、サンプル・データ・セット SIBMZSAM において組み込みファイルで提供されるようになりました。

コンパイラー・オプション機能拡張

- 新しい 12 ページの『BRACKETS』 コンパイラー・オプションを使用すれば、SQL プリプロセッサーが SQL 配列参照において左右の括弧として受け入れる記号を指定できます。このオプションを使用すれば、そのような言語をさらに利用しやすくなります。
- コンパイラーは、新しい 24 ページの『DECOMP』 コンパイラー・オプションを使用してリスト・セクションを生成します。このリスト・セクションには、ソース・プログラムで使用されるすべての式に対するすべての中間式とその属性が表示されます。
- 22 ページの『DECIMAL』 コンパイラー・オプションには、新しいサブオプション TRUNCFLOAT があります。このサブオプションを使用すれば、切り捨てが行われる可能性があるときに固定小数点への浮動小数点の代入をコンパイラーが処理する方法を制御できます。
- 新しい 37 ページの『EXPORTALL』 コンパイラー・オプションは、外部で定義されたすべてのプロシージャーと変数を、DLL アプリケーションで使用できるようにエクスポートするかどうかを制御します。
- 新しい 43 ページの『HEADER』 コンパイラー・オプションを使用すれば、コンパイラー・リスト内の各ヘッダー行の中央に表示されるものを制御できます。
- 48 ページの『INSOURCE』 コンパイラー・オプションには、FIRST および ALL という 2 つの新しいサブオプションがあります。これらは、リスト・ファイルに表示されるソース・リストの数を制御します。
- 新しい 54 ページの『LP』 コンパイラー・オプションを使用すれば、コンパイラーが 31 ビット・コードを生成するのか 64 ビット・コードを生成するのかを指定できます。
- 60 ページの『MDECK』 コンパイラー・オプションには、AFTERALL および AFTERMACRO という 2 つの新しいサブオプションがあります。これらは、MDECK が生成されるタイミングを制御します。
- 新しい 64 ページの『NULLDATE』 コンパイラー・オプションを使用すれば、ご使用のプログラムにおいて SQL ヌル日付を有効な日付として使用できます。
- 新しい 65 ページの『OFFSETSIZE』 コンパイラー・オプションは、64 ビット・アプリケーションにおける OFFSET 変数のサイズを判別します。

- 79 ページの『RULES』 コンパイラー・オプションには、以下の新しいサブオプションがあります。

  LAXSTMT | NOLAXSTMT
  RULES(NOLAXSTMT) を使用して、複数のステートメントを持つすべての行にフラグを立てることができます。

  UNREFBASED | NOUNREFBASED
  RULES(NOUNREFBASED) を使用して、BASED ストレージにある非参照 BASED 変数にフラグを立てることができます。
- 79 ページの『RULES』 コンパイラー・オプションの NOPROCENDONLY サブオプションには、ALL および SOURCE という 2 つの新しいサブオプションがあります。これらのサブオプションを使用すれば、各プロシージャーの END ステートメントにそのプロシージャーの名前が組み込まれていなければならないという規則をさらに実施しやすくなります。
- 112 ページの『XREF』 コンパイラー・オプションには、EXPLICIT および IMPLICIT という 2 つの新しいサブオプションがあります。これらのサブオプションを使用すれば、暗黙的変数参照が XREF リストに組み込まれているのかどうかを判別できます。

注:

リリースに対するコンパイラー・オプション・デフォルト設定はすべて、SIBMZSAM データ・セットにおいて IBMXO*vrm* という名前を持つメンバーにリストされます。 *v* はバージョン番号です。 *r* はリリース番号です。 *m* はモディフィケーション番号です。通常、モディフィケーション番号は 0 です。例えば、V5R1 の場合は IBMXO510 となり、V4R5 の場合は IBMXO450 となります。このデータ・セットには、すべてのサポート対象 PL/I リリースに対する IBMXO*vrm* メンバーが含まれています。あるリリースから次のリリースの間に変更されたデフォルト設定の内容および追加された新規オプションを確認するには、その 2 つのリリースの IBMXO*vrm* ファイルを比較します。 IBMXO*vrm* ファイルは、希望の設定を使用して独自のオプション・ファイルを作成するためのテンプレートとしても使用できます。

SQL 機能拡張

- SQL プリプロセッサーは、DEFINE ALIAS ステートメント、DEFINE ORDINAL ステートメント、および DEFINE STRUCTURE ステートメントを構文解析するようになりました。SQL ステートメントで使用できる PL/I タイプが DEFINE ALIAS ステートメントで定義される場合、そのタイプで宣言された変数も SQL ステートメントで使用できます。
- 64 ページの『NULLDATE』 コンパイラー・オプションが指定されている場合、*year*、*month*、および *day* がすべて 1 の値になっている SQL ヌル日付は、一部の日次処理組み込み関数で、有効な日付として受け入れられます。
- INDFOR 属性を使用すれば、別の PL/I 構造体と一致するように標識変数の構造体をより宣言しやすくなります。
- DSNH030 SQL プリプロセッサー・メッセージの重大度をより変更しやすくするために、固有のプリプロセッサー・メッセージ IBM3317 が指定されるようになりました。

V4R5 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- MVC が 1 つ除去されたため、EXEC CICS® ステートメント用に生成されたコードの実行速度が向上しました。
- FIXED DEC の MOD および REM に関して 15 を超える精度で非常に高速なコードが生成されるようになりました。
- ARCH オプションで最大値として 11 が受け入れられるようになりました。ARCH(11) が指定されると、コンパイラーは、対象の z システム上で新規ハードウェア命令を使用するコードを生成します。このコードにより、特に、SEARCH 組み込み関数および VERIFY 組み込み関数の一部インスタンスのパフォーマンスが向上します。

SQL 機能拡張

- 最初の無効ホスト変数が見つかった時点では EXEC SQL ステートメントの妥当性検査は停止せず、代わりにすべてのホスト変数参照が検査されます。
- 新規 SQL プリプロセッサー・オプション (NO)CODEPAGE により、SQL プリプロセッサーがコンパイラー CODEPAGE オプションをどのように指定するのかが決まります。
- 新規 SQL プリプロセッサー・オプション (NO)WARNDECP により、SQL プリプロセッサーによって生成される「ノイズ」の量を減らすことができます。
- ホスト変数として使用される構造体は、標識変数として使用される構造体も持つようになりました。
- 名前付き定数 (すなわち、VALUE 属性を持つ PL/I 変数) をホスト変数として使用できるようになりました (その設定の定数が SQL で許可される場合)。

使いやすさの向上

- 新規 MAXBRANCH コンパイラー・オプションにより、過度に複雑なコードを見つけやすくなりました。
- 新規コンパイラー・オプション FILEREF および NOFILEREF は、コンパイラーがファイル参照テーブルを生成するのかどうかを制御します。
- 新規 JSON コンパイラー・オプションにより、JSONPUT 組み込み関数によって生成されたり JSONGET 組み込み関数によって期待されたりする JSON テキスト内の名前の大/小文字を選択できるようになりました。
- LIMITS コンパイラー・オプションで、BIT 変数、CHARACTER 変数、または WIDECHAR 変数の長さのしきい値として値 32K、512K、8M、および 128M を持つサブオプションとして STRING が受け入れられるようになりました。
- XML コンパイラー・オプションにより、XMLCHAR 組み込み関数によって生成される XML 属性をアポストロフィで囲むのか引用符で囲むのかを選択できるようになりました。
- コードがプロシージャーにおいて RETURNS 属性を伴って END ステートメントに到達した場合にエラーになるように RULES(NOLAXRETURN) コンパイラー・オプションが拡張されました。

- コンパイラー・オプション RULES(NOLAXNESTED) および RULES(NOPADDING) に対する新規サブオプション ALL | SOURCE は、疑わしいコーディングにコンパイラーがフラグを立てるタイミングをより詳細に制御します。
- ユーザーは、新規 FORCE(NOLAXQUAL) 属性と、RULES(NOLAXQUAL) オプションの新規 FORCE サブオプションを使用すれば、構造体ごとに NOLAXQUAL 規則を強制できます。
- INITIAL 属性を持つ REFER オブジェクトに E レベル・メッセージのフラグが立てられるようになりました。
- マクロ・プリプロセッサーが DEPRECATE オプションと DEPRECATENEXT オプションをサポートするようになりました。これらのオプションを使用すれば、計画した段階にわたって、選択したマクロ・プロシージャーを除去できます。
- FIXED 変数や FLOAT 変数に VALUE 属性がある場合、コンパイラーはその値を ATTRIBUTES リストにおいて文字ストリングとしてリストします。
- 配列メンバーと配列メンバーの間に埋め込みが含まれる (構造体に CHAR(31) VARYING ALIGNED の配列が含まれるときなど) VARYING の配列にマークを付けるために集合リストが拡張されました。
- SUBSCRIPTRANGE が使用可能になっていて、複数の非定数境界を持つ配列が配列の割り当てに含まれる場合に、その境界同士が一致しなければ SUBSCRIPTRANGE 条件を発生させるコードをコンパイラーが生成するようになりました。境界がすべて定数の場合、コンパイラーは、その境界同士が一致することをコンパイル時に検査し続けます。
- いずれかの PLISAX イベント構造体におけるイベントが (NULLENTRY 組み込み関数または UNSPEC 疑似変数を使用して) ヌルに設定されている場合、XML パーサーはそのイベントを呼び出しません。これにより、関心のあるイベントのみに XML 構文解析コードを制限することができ、同時に構文解析全体のパフォーマンスを向上させることができます。
- コンパイラーは、現在コンパイル中のプロシージャーに対して FETCH や RELEASE を実行する試みにフラグを立てるようになりました。

V4R4 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- 疑似アセンブラー・リストの生成が、前のリリースよりも早くなりました。
- コンパイラーは、ARCH(7) でコンパイルされるようになりました。
- コンパイラーは、STATIC 属性を持たず、100 個を超える INITIAL 項目を持つ変数の宣言にフラグを立てるようになりました。

使いやすさの向上

- プログラムに間違った構文が含まれているときに SQL プリプロセッサーから発行されるメッセージが増加しました。これらのメッセージにより、エラーがあるソース・ステートメントを識別しやすくなります。

- 新しい NOINCLUDE オプションを使用して、マクロ・プリプロセッサーの外部での %INCLUDE ステートメントおよび %XINCLUDE ステートメントの使用を禁止することができます。
- %INCLUDE ステートメントが最後のコンパイラー・パスによって処理されるとき、組み込みファイルがソース・リスト内で大括弧に囲まれるようになりました。
- DEFAULT オプションの NULLSTRPTR サブオプションが持つ新しい STRICT サブオプションによって、POINTERS への '' の割り当ておよび比較に対して無効のフラグが立てられます。
- STMT オプションを指定すると、ソース・リストおよびメッセージ・リストには論理ステートメント番号とソース・ファイル番号の両方が入ります。

V4R3 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- ARCH(10) オプションを使用すると、IBM zEnterprise® EC12 System 命令を活用できます。
- VERIFY および SEARCH 用のインライン・コードがコンパイラーによって生成されるようになりました (VERIFY および SEARCH の引数が 3 つあって、その 2 番目の引数が 1 文字の場合)。
- PICTURE から DFP への追加変換用のインライン・コードがコンパイラーによって生成されるようになりました。
- BIT から CHAR への追加変換用のインライン・コードがコンパイラーによって生成されるようになりました。
- BIT から WIDECHAR への変換用のインライン・コードがコンパイラーによって生成されるようになりました。
- FIXED DEC の TRIM 用に生成されるコードが向上しました。

使いやすさの向上

- SQL プリプロセッサーにおいて以下の点が向上しました。
  - ONEPASS オプションがサポートされます。
  - ホスト変数宣言で一部の制限付き式を使用できます。
  - EXEC SQL ステートメントのリストが、元のソースによく似た読みやすいフォーマットで表示されます。
  - LIKE 属性を使用して宣言されたホスト変数を使用できます。
  - 新規 DEPRECATE オプションがサポートされます。このオプションを使用すると、プリプロセッサーが非推奨のステートメントのリストにフラグを立てます。
- ADATA ファイルに、以下の属性、組み込み関数、およびステートメントの使用が記録されるようになりました。
  - INONLY、INOUT、および OUTONLY 属性
  - XMLATTR および XMLOMIT 属性

  - ALLCOMPARE、UTF8、UTF8TOCHAR、および UTF8TOWCHAR 組み込み関数
  - ASSERT ステートメント
- 新規 CASERULES オプションを使用すれば、PL/I キーワードに対して大/小文字の規則を指定できます。 例えば、すべてのキーワードは大文字でなければならないという規則を指定できます。
- DEPRECATE オプションで新規 STMT サブオプションを使用できます。このサブオプションを使用すると、コンパイラーが非推奨のステートメントのリストにフラグを立てます。
- 新規 DEPRECATENEXT オプションを使用すれば、関数を段階的に非推奨にできます。
- IGNORE オプションで新規 ASSERT サブオプションを使用できます。このサブオプションを使用すると、コンパイラーがすべての ASSERT ステートメントを無視します。
- 新規 (NO)MSGSUMMARY オプションは、コンパイル中に発行されたすべてのメッセージの要約をコンパイラーがリストに追加するかどうかを制御します。
- RTCHECK オプションで新規 NULL370 サブオプションを使用できます。このサブオプションは、旧 NULL() 値に相当するポインターが逆参照されるかどうかを検査します。 旧 NULL() 値に相当するポインターとは、16 進値 'FF000000'x を持つポインターのことです。
- RULES オプションで、CONTROLLED 属性の使用に対してフラグを立てるかどうかを制御する (NO)CONTROLLED がサブオプションとして受け入れられるようになりました。
- RULES オプションで、実行可能コードのセクション間にネスト・プロシージャーが存在するプログラムにフラグを立てるかどうかを制御する (NO)LAXNESTED がサブオプションとして受け入れられるようになりました。
- RULES オプションで、RECURSIVE 属性の使用、または自分自身を直接呼び出すプロシージャーの使用に対してフラグを立てるかどうかを決定する (NO)RECURSIVE がサブオプションとして受け入れられるようになりました。
- RULES(NOUNREF) オプションで、すべての未参照変数にフラグを立てるかどうかを決定する SOURCE | ALL がサブオプションとして受け入れられるようになりました。
- RULES(NOLAXIF) では、x = y = z の形式の割り当てに対してコンパイラーがフラグを立てるようにもなりました。
- RULES(NOLAXSCALE) では、p < 0 の場合にコンパイラーが ROUND(x, p) にフラグを立てるようにもなりました。

V4R2 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- ARCH(9) オプションを使用すると、IBM zEnterprise 196 (z196) System 命令 (high-word facility、floating-point extension facility、および population count facility) をさらに活用することができます。

- 新しい UNROLL コンパイラー・オプションによって、ループのアンロールを制御できるようになりました。
- コンパイラーによってインライン・コードが生成され、 ULENGTH および USUBSTR 組み込み関数の文字ストリングが解決できるようになりました。
- コンパイラーによって MEMINDEX(*p, n, x*) 用のインライン・コードが生成されるようになりました。*x* は WCHAR(1) です。
- コンパイラーによって STG(*x*) 用のインライン・コードが生成されるようになりました。この *x* は、REFER を使用する BASED 変数であり、以下の条件の両方を満たします。
  - x 内の NONVARYING BIT はすべて、ALIGNED 属性で指定される。
  - x 内のその他のエレメントはすべて、UNALIGNED で指定される。

デバッグの向上

- コンパイラーが、デバッグ・ツール内のタイプ付き構造体をサポートするようになりました。

SQL サポートの強化

- SQL プリプロセッサーは、次のように大幅に変更されました。
  - 現時点で、ブロック・スコーピングを完全にサポートしています。
  - プリプロセッサー・ロード・モジュールが、8 分の 1 よりも小さくなりました。
  - プリプロセッサーの実行速度が速くなりました。
  - PL/I データ・タイプを指定できる任意の場所で、SQL TYPE 属性が使用できるようになりました。
  - ホスト変数の宣言を処理する際に、以下のコンパイラー・オプションをサポートするようになりました。 デフォルトが正しく適用され、不適切なホスト変数は適宜拒否されます。
    - DEFAULT(ANS | IBM)
    - DEFAULT(ASCII | EBCDIC)
    - DEFAULT((NO)EVENDEC)
    - DEFAULT((NON)NATIVE)
    - DEFAUT(SHORT(HEX | IEEE))
    - RULES((NO)LAXCTL)
  - PRECISION 属性を正しく処理するようになりました。
  - UNSIGNED 属性と COMPLEX 属性を認識し、任意のホスト変数でこれらの属性の使用を拒否するようになりました。
  - DSNHMLTR を必要とするコードが含まれる最外部のプロシージャーで、DSNHMLTR が確実に宣言されるようになりました。
  - パッケージを正しく処理するようになりました。
  - ソース・コード内の文字が、SQLAVDAID を設定するコードが生成されるときに確実に印刷できるようになりました。
  - 標識配列に下限 1 を設定する必要がなくなりました。

– SQL パラメーター・リスト構造体を生成するようになりました。この構造体では共用体や init 節の数が少なくなり、構造体のエレメントに基づく追加の宣言はなくなりました。

使いやすさの向上

- 新しい PPLIST コンパイラー・オプションは、メッセージを提供しないプリプロセッサー・フェーズで生成されるリスト部分を条件付きで削除します。
- 構造体宣言にコンマが欠落している場合、コンパイラーは、より良いメッセージを出します。
- ソース・コードに無効のシフトイン・バイトおよびシフトアウト・バイトが含まれている場合、コンパイラーは新規メッセージを出します。
- コンパイラーは、INONLY 属性で宣言される任意のパラメーターに NONASSIGNABLE 属性を適用します。 INONLY として宣言されるパラメーターの割り当てにはフラグを立てます。
- RULES(NOLAXENTRY) を指定すると、コンパイラーは DSN で始まる名前にはフラグを立てません。
- RULES(NOUNREF) を指定すると、コンパイラーは DSN または SQL で始まる名前にはフラグを立てません。
- RULES コンパイラー・オプションの新しいサブオプション NOSELFASSIGN を指定すると、コンパイラーは変数の自己への割り当てにフラグを立てます。
- RULES コンパイラー・オプションの新しいサブオプション NOLAXRETURN を指定すると、 RETURN ステートメントが何らかの無効な方法で使用された場合に、コンパイラーがエラー条件を引き起こすコードを生成します。

V4R1 からの機能強化

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

デバッグの向上

- TEST(SEPARATE) が指定されている場合、コンパイラーはオプションでステートメント番号テーブルをデバッグ・ファイルに組み込むため、生成されるオブジェクト・デックのサイズが削減されます。
- TEST(SEPARATE) が指定されている場合、コンパイラーは、宣言、参照、および割り当てのソース行を特定する情報を組み込みます。
- TEST(SEPARATE) が指定されている場合、BASED 変数が配列エレメントの ADDR またはその他の複合参照を基にしていると、コンパイラーは暗黙的ロケーター参照を特定するための情報を生成します。

パフォーマンス向上

- ARCH(9) オプションを使用すると、IBM zEnterprise System 命令を活用できます。
- REFER を使用する構造体のすべてのエレメントがバイト整合である場合、コンパイラーは、ライブラリー呼び出しを通じてではなく、コードのインライン化により、これらのエレメントに対する参照を解決します。

使いやすさの向上

- 新しい DEPRECATE コンパイラー・オプションは、非推奨の変数名および組み込みファイル名にフラグを立てます。
- GONUMBER オプションの新しい SEPARATE サブオプションが提供され、生成されたステートメント番号テーブルを別個のデバッグ・ファイルに組み込むことができます。
- RULES オプションの新しい NOGLOBALDO サブオプションは、親ブロックで宣言されているすべての DO ループ制御変数にフラグを立てます。
- RULES オプションの新しい NOPADDING サブオプションは、埋め込みを含むすべての構造体にフラグを立てます。
- SQL プリプロセッサーが XREF オプションをサポートするようになりました。
- 新しい PLISAXD 組み込みサブルーチンにより、XML System Services パーサーを使用して、スキーマに対する妥当性検査を伴う XML 文書の解析が可能になりました。

保守容易性の向上

- GOSTMT オプションまたは IMPRECISE オプションの使用には、非サポートのフラグが立てられるようになりました。
- コンパイラーは、有効なすべての SQL プリプロセッサー・オプションを常にリストするようになりました。

V3R9 からの機能強化

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- 長さが 256 以下のストリングに対する UVALID はインライン化されるようになりました。
- ARCH(7) 以上を指定すると、UTF-8、UFT-16、および UTF-32 相互間の高速変換に対応するために CU12、CU14、CU21、CU24、CU41、および CU42 命令が使用されます。
- ARCH(7) 以上を指定すると、1 バイト・バッファーと 2 バイト・バッファー間の高速変換に対応するために TRTT、TROT、TRTO、および TROO 命令が使用されます。
- スカラーの類似配列の割り当てはストレージ・コピー操作として扱われるようになりました。
- BIT VARYING から BIT VARYING への割り当てはすべてインライン化されるようになりました。
- バイト整合の BIT NONVARYING から BIT VARYING への割り当てはすべてインライン化されるようになりました。
- ROUND および ROUNDDEC 組み込み関数は、丸める引数が DFP であるときインライン化されるようになりました。

  注: PL/I for z/OS V5.1 以降、ROUNDDEC 組み込み関数は ROUNDAWAYFROMZERO に名前変更されています。

- 最高のパフォーマンスを得るためのオプション選択を簡略化するために、以下のことが行われました。
  - COMPACT オプションが除去されました。
  - DEFAULT(REORDER | ORDER) のデフォルト設定が DEFAULT (REORDER) に変更されました。
  - TUNE オプションが除去されました。
- NULL ポインターの逆参照の検出では、ARCH(8) のもとで新しい比較トラップ命令が活用されます。
- コンパイラーはパフォーマンスの向上のために ARCH(6) でビルドされました。

使いやすさの向上

- CICS プリプロセッサーは、ブロック・スコーピングをサポートするようになったため、必要なローカル CICS 宣言をすべてのネストなしプロシージャーに追加するようになりました。
- SQL プリプロセッサーは、新しい SCOPE オプションをによってホスト変数参照を解決するとき、宣言のスコープに関する PL/I 規則をサポートするようになりました。 NOSCOPE は前のリリースとの互換性を保つためのデフォルトです。
- マクロ・プリプロセッサーは、コンパイラー・リスト内の %include、%xinclude、%inscan、および %xinscan ステートメントをコメントとして残すことになりました。
- マクロ・プリプロセッサーは、%DO SKIP; ステートメントを介して、コンパイルからコードのセクションを省略する簡単かつ明瞭な方法を提供するようになりました。
- マクロ・プリプロセッサーは NAMEPREFIX というオプションをサポートするようになりました。ユーザーは、このオプションを使用して、マクロ・プロシージャーおよび変数を、指定した文字から強制的に開始することができます。
- IGNORE コンパイラー・オプションにより、PUT FILE または DISPLAY ステートメント、あるいはその両方を抑制することができます (いずれか一方のステートメントがデバッグ目的で使用されていたことがありますが、その場合は実動バージョンからコンパイルされました)。
- DEFAULT コンパイラー・オプションの NULLSTRPTR サブオプションにより、割り当てのソースがヌル・ストリングであるとき、ポインターに sysnull または null のどちらを割り当てるかをユーザーが制御できます。
- 新しい MAXGEN オプションは、任意の 1 つのユーザー・ステートメントに対して生成される中間言語ステートメントの最大数を指定するもので、コンパイラーがこの最大数を超える場合、ステートメントにフラグを立てるようにします。
- 新しい ONSNAP オプションにより、ユーザーは、MAIN または FROMALIEN プロシージャーのプロローグに ON STRINGRANGE SNAP; または ON STRINGSIZE SNAP; ステートメントを挿入するようにコンパイラーに要求できるようになりました。
- INITAUTO オプションの新しい SHORT サブオプションは、INITAUTO オプションがランタイム STORAGE オプションの全部を複写するのではなく、レジスターに最適化される可能性がある変数を初期化するように、INITAUTO オプションを制限します。

- 新しい RTCHECK オプションは、NULL ポインターの逆参照をテストするコードを生成します。
- 次のような、リスクを伴う可能性がある各種ステートメントに、コンパイラーがフラグを立てるようになりました。
  - FIXED 演算の結果がゼロより小さいスケール係数を持つコード
  - 関数として使用されたが RETURNS 属性なしで宣言された ENTRY
  - 不適切であるにも関わらず BYVALUE として宣言されたパラメーター (例えば、FIXED DEC パラメーターが BYVALUE であると宣言するなど)
  - FIXEDOVERFLOW を起こす可能性がある FIXED DECIMAL 加算および乗算演算
- 不良コードの制御強化およびフラグ設定を可能にするように RULES オプションが拡張されました。
  - NOPROCENDONLY は、終了する PROC を指定していない、PROC の END ステートメントにフラグを立てます。
  - NOSTOP は STOP および EXIT の使用にフラグを立てます。
  - NOLAXQUAL(STRICT) は、レベル 1 名で修飾されていない変数にフラグを立てます。
  - NOLAXSCALE は、FIXED DEC(p,q)、および FIXED BIN(p,q) の宣言にフラグを立てます (ただし q < 0 または p < q の場合)。
  - NOGOTO(LOOSE) は、同じブロック内にある場合にのみ GOTO を許可します。
  - 複数の DELAY STATEMENT を別々のプロシージャーで並行して実行することができます。

保守容易性の向上

- コンパイラーがファイルを開けないとき、可能であれば、コンパイラーが関連する C ランタイム・メッセージをリスト内のメッセージにも組み込むようになりました。
- ユーザー・コードがコンパイル時に DFP 変換を必要とするが、DFP ハードウェアを備えていないマシンでコンパイルが実行されている場合は、このエラーがトラップされ、意味のあるエラーが出されるようになりました。
- SQL プリプロセッサーがサブオプションとして INCONLY を指定しないで複数回呼び出された場合は、コンパイラーによって作成された DBRM ライブラリーが空になり、ユーザーにこの問題を警告するため E レベル・メッセージが出されるようになりました。

V3R8 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

パフォーマンス向上

- ARCH(8) および TUNE(8) オプションを指定すると、z/HE 命令が使用されます。
- HGPR オプションによって、32 ビット・コードでの 64 ビット・レジスターの使用がサポートされます。

- GOFF オプションによって、GOFF オブジェクトの生成がサポートされます。
- PFPO 命令は、異なる浮動フォーマットの変換で使用されます。
- SRSTU は、UTF-16 INDEX で生成されたコードで使用されます。
- ヌルの内部プロシージャーの呼び出しは、完全に除去されるようになりました。

使いやすさの向上

- PLISAXC によって、SAX インターフェースを使用して XML System Services パーサーにアクセスできます。
- INCDIR オプションは、バッチでサポートされます。
- LISTVIEW オプションによって、TEST の AFTERMACRO などのサブオプションで以前は提供されていたサポートが提供されるようになりました。
- RULES オプションの NOLAXENTRY サブオプションによって、非プロトタイプ ENTRY のフラグを立てることができます。
- DECIMAL オプションの (NO)FOFLONMULT サブオプションによって、FIXED DECIMAL の MULTIPLY で FOFL を発生させるかどうかを制御できます。
- USAGE オプションの HEX および SUBSTR サブオプションによって、ユーザーは、対応する組み込み関数の振る舞いをさらに制御できます。
- DDSQL コンパイラー・オプションによって、EXEC SQL INCLUDE で使用される代替 DD 名を指定できます。
- MACRO および SQL プリプロセッサーによって提供される INCONLY サブオプションを使用して、当該プリプロセッサーが INCLUDE のみを実行するように要求できます。
- LOB(DB2®) SQL プリプロセッサー・オプションが選択されている場合に、統合された SQL プリプロセッサーでは、既にサポート済みの BLOB、CLOB、および DBCLOB SQL 型以外にも、すべての *LOB_FILE、*LOCATOR、ROWID、BINARY、および VARBINARY SQL 型について、DB2 プリコンパイラー・スタイル宣言が生成されるようになりました。

V3R7 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

デバッグの向上

- TEST オプションが機能拡張され、ユーザーが指定したプリプロセッサーが実行された後 (または、すべてのプリプロセッサーが実行された後) にソースが表示されるのと同様に、ソースをリストおよび「デバッグ・ツール・ソース (Debug Tool source)」ウィンドウに表示することを選択できます。

パフォーマンス向上

- BASR 命令が、BALR 命令の代わりに使用されるようになりました。
- 仲介として FIXED BIN(63) を使用することにより、精度の高い FIXED DEC から FLOAT への変換が、インライン化され、高速化されました。
- CHAR 組み込み関数は、CHAR 式に適用される際には常にインライン化されるようになりました。

- FIXED BIN(p,q) からスケールのない FIXED DEC への変換用の生成コードが大幅に改良されました。
- ARCH(7) のもとで TRTR は、TRT が SEARCH および VERIFY に対して使用されたのと同じ状態で、SEARCHR および VERIFYR に対して使用されます。
- UNPKU は、一部の PICTURE を WIDECHAR に変換する (ライブラリー呼び出しをするのではなく) ために使用されます。

使いやすさの向上

- IEEE 10 進浮動小数点 (DFP) がサポートされます。
- 新規の MEMCONVERT 組み込み関数を使用すると、任意のコード・ページ間で任意の長さのデータを変換できます。
- 新規の ONOFFSET 組み込み関数を使用すると、以前は実行時のエラー・メッセージまたはダンプ、つまりある条件が発生したユーザー・プロシージャーのオフセットでのみ使用可能だった他の情報に簡単にアクセスすることができます。
- 新規の STACKADDR 組み込み関数は、現在の動的保存域 (z/OS 上のレジスター 13) のアドレスを戻し、ユーザー独自の診断コードの作成を容易にします。
- アセンブラー・リスト内の簡略記号フィールドの長さが拡大され、長い簡略記号を持つ新規 z/OS 命令のサポートが改善されました。
- 属性、相互参照、およびメッセージ・リストで、使用可能な右マージンが拡張されました。
- CODEPAGE オプションが、1026 (トルコ語コード・ページ) および 1155 (1026 コード・ページに加え、ユーロ記号) に対応できるようになりました。
- 新規の MAXNEST オプションを使用すると、BEGIN、DO、IF、および PROC ステートメントの過度なネストにフラグを立てることができます。
- RULES オプションの新規 (かつ、デフォルトでない) のサブオプション NOELSEIF を指定すると、直後に IF ステートメントが続く ELSE ステートメントにコンパイラーがフラグを立て、SELECT ステートメントとして書き直すように提案します。
- RULES オプションの新規 (かつ、デフォルトでない) のサブオプション NOLAXSTG を指定すると、コンパイラーは、変数 A が ADDR(B) および STG(A) > STG(B) の BASED として宣言されている場所にフラグを立てます。以前のように、B が定数の存在期間を指定した AUTOMATIC、BASED、または STATIC である場合のみでなく、B が定数の存在期間を指定して宣言されるパラメーターである場合にもフラグが立てられます。
- 新規の QUOTE オプションを使用すると、引用符 (") 記号に大体コード・ポイントを指定することができます。これは、この記号がコード・ページ・インバリアントではないためです。
- 新規の XML コンパイラー・オプションを使用すると、XMLCHAR 組み込み関数の出力内のタグを、すべて大文字にするか、大/小文字を宣言で使用したとおりに指定できます。
- メッセージを生成しないコンパイルであっても、コンパイラー・メッセージがリストされるはずの行に、「コンパイラー・メッセージはありません (no compiler messages)」というメッセージが表示されるようになりました。

- マクロ・プリプロセッサーは、%INCLUDE ステートメントのみを処理するのか、すべてのマクロ・ステートメントを処理する必要があるかを指定することのできる新規サブオプションをサポートします。
- LOB(DB2) SQL プリプロセッサー・オプションが選択されている場合に、統合された SQL プリプロセッサーでは、既にサポート済みの BLOB、CLOB、および DBCLOB SQL 型以外にも、すべての *LOB_FILE、*LOCATOR、ROWID、BINARY、および VARBINARY SQL 型について、DB2 プリコンパイラー・スタイル宣言が生成されるようになりました。

V3R6 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

DB2 V9 サポート

- STDSQL(YES/NO) のサポート
- CREATE TRIGGER (複数 SQL ステートメント) のサポート
- FETCH CONTINUE のサポート
- SQL ステートメントに組み込まれた SQL スタイルのコメント ('--') のサポート
- 以下の追加 SQL TYPE のサポート
  - SQL TYPE IS BLOB_FILE
  - SQL TYPE IS CLOB_FILE
  - SQL TYPE IS DBCLOB_FILE
  - SQL TYPE IS XML AS
  - SQL TYPE IS BIGINT
  - SQL TYPE IS BINARY
  - SQL TYPE IS VARBINARY
- SQL プリプロセッサーが DB2 コプロセッサー・オプションもリストするようになりました。

デバッグの向上

- TEST(NOSEPNAME) を指定すると、デバッグ・サイドのファイルの名前がオブジェクト・デックに保存されません。

パフォーマンス向上

- ARCH(7) での z/OS 拡張即値機能のサポート
- ARCH(6) での CLCLU、MVCLU、PKA、TP、および UNPKA 命令の活用
- ARCH(5) での CVBG および CVDG 命令の活用
- CLCLE の拡張使用
- DB2 日時パターンに関する変換をインライン化
- ALLOCATION 組み込み関数をインライン化
- 浮動 $ を含んだ変換をインライン化
- PIC'(n)Z' への割り当てから条件付きコードを除去
- FIXED BIN からスケール因数を指定する PICTURE への変換をインライン化

- 次元を継承したが 8 で割り切れるストライドを持つ BIT 変数への割り当てをインライン化

使いやすさの向上

- ブロックが使用する、AUTOMATIC ストレージのストレージ・オフセット (ブロックごと) 順のリストも、MAP 出力に含まれるようになりました。
- 規格合致検査が拡張されて構造体が含められました。
- リストには、ファイル内の行番号用に 7 カラムが含まれるようになります。
- z/OS 環境では THREADID 組み込み関数がサポートされるようになりました。
- PICSPEC 組み込み関数がサポートされるようになりました。
- 新規 CEESTART オプションにより、オブジェクト・デックの始めまたは終わりに CEESTART csect を配置できるようになりました。
- 新規 PPCICS、PPMACRO、および PPSQL オプションにより、対応するプリプロセッサーで使用されるデフォルト・オプションを指定できるようになりました。
- ATTRIBUTES リストに ENVIRONMENT オプションが組み込まれるようになりました。
- DISPLAY オプションがサポートする新規サブオプションによって、REPLY を指定した DISPLAY か REPLY を指定しない DISPLAY が、さまざまな DESC コードで可能になりました。
- コメント内のセミコロンに対するフラグ・メッセージに、セミコロンのある行の行番号が組み込まれるようになりました。
- REFER 項目が変更される可能性がある割り当てに対してフラグが立ちます。
- KEY/KEYFROM 文節が含まれない KEYED DIRECT ファイルの使用に対してフラグが立ちます。
- PICTURE をループ制御変数として使用することに対してフラグが立ちます。

V3R5 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

デバッグの向上

- TEST(SEPARATE) を指定すると、デバッグ情報の大部分が別個のデバッグ・ファイルに書き込まれます。
- AUTOMONITOR に割り当てのターゲットが含まれます。
- AUTOMATIC の初期化後に AT ENTRY フックが配置されるようになりました。それによって、変数を調べる前のブロックへのステップイントゥが必要なくなりました。

パフォーマンス向上

- 分岐相対命令の生成により、基底レジスターと制御権移動ベクトルの必要性が大幅に削減されます。
- ARCH(6) での z/OS 長変位機能のサポート。

- REFER を使用する単純構造体が、ライブラリー呼び出しを介さずにインラインでマップされます。
- REFER を使用する構造体のうち、これまでのようにライブラリー呼び出しを介してマップされるものについては、REFER が副構造体の配列の境界を指定していれば、生成されるコードが少なくなります。
- 重複 INCLUDE が高速で処理されるようになりました。
- 最終位置が I または R の PICTURE 変数への変換が、インライン化されました (最終文字が T の PICTURE 変数への変換は既にインライン化されていました)。
- B を含まないピクチャーがインライン化された場合、そのピクチャーに対応する、1 つ以上の B で終わる PICTURE 変数への変換がインライン化されるようになりました。
- CHARACTER 変数から X のみで構成される PICTURE 変数への変換が、インライン化されるようになりました。

使いやすさの向上

- リストなどのすべての部分で、ソース・ファイルがファイル 0、最初のインクルード・ファイルがファイル 1、2 番目の (固有の) インクルード・ファイルがファイル 2 のように数えられます。
- 規格合致検査が拡張されて配列が含められました。
- 呼び出されたプリプロセッサーのビルドの日付がリストに組み込まれます。
- COBOL との ILC を簡単にするために、1 バイト FIXED BINARY 引数を抑止できます。
- SYSADATA、SYSXMLSD、および SYSDEBUG の代替 DD 名を指定できます。
- XNUMERIC を指定した場合、RULES(NOLAXMARGINS) はシーケンス番号を許容します。
- RULES(NOUNREF) は、参照されない AUTOMATIC 変数に対してフラグを立てます。
- 変数への割り当てがライブラリー呼び出しを介して行われる場合、ライブラリー呼び出しに対するフラグ・メッセージには、ターゲット変数の名前が含まれます。
- 一回限りの DO ループに対してフラグが立ちます。
- 引数として使用されるラベルに対してフラグが立ちます。
- PRV が使用される場合、FETCHABLE の PARAMETER CONTROLLED 以外の ALLOCATE と FREE に対してフラグが立ちます。
- DEFINED および BASED がそれぞれの基数より大きい場合、基数が後で宣言されるとしても、それぞれに対してフラグが立ちます。
- FIXED DEC から 8 バイト整数への暗黙的型変換に対してフラグが立ちます。

V3R4 からの機能拡張

本リリースでは以下の機能が強化されています。強化された機能については、本書のほか、他の IBM PL/I ブックにも説明があります。

マイグレーションの強化

- 旧コードとの CONTROLLED の共用のサポート
- デフォルト初期化の向上
- ADD、DIVIDE、および MULTIPLY での小数部指定の容易化
- GRAPHIC の STRING に対する旧セマンティクスのサポート
- DEFAULT ステートメントに対する旧セマンティクスのサポート
- ストレージ・オーバーレイのある宣言に対するフラグ付け
- BEGIN 内の RETURN での制限の撤廃
- コメント内のセミコロンに対するフラグ付け (オプション)
- アセンブラーで初期化される EXT STATIC のサポート
- 無効な紙送り制御文字に対するフラグ付け
- 言語の誤用 (特に RETURN) に対するフラグ付けの強化
- REPLACEBY2 組み込み関数のサポート
- SIZE を発生させる 10 進代入における FOFL の抑止 (オプション)
- 言語で旧コンパイラーとは異なる処理をしたことに対するフラグ付けの強化

パフォーマンス向上

- INDEX および TRANSLATE でのコード生成の向上
- ピクチャーに対する代入のインライン化の向上
- ソースが FIXED DEC であった場合に、変換がインラインで行われるときの PICTURE に対する CHARACTER の変換で生成されるコードの向上
- パック 10 進数変換で生成されるコードの向上
- REFER の一部の使用法で生成されるコードの向上
- 長さが不明な文字ストリングの比較のインライン化
- 連結に使用されるスタック・ストレージの量の削減
- GET/PUT STRING EDIT ステートメントのインライン化の向上
- LE 条件処理の短縮の強化
- BIN FIXED の OR および AND のインライン化の向上
- ALIGNED BIT(8) に対する SIGNED FIXED BIN(8) のインライン化
- コンパイラーが実行時に構造にマップするためのライブラリー・ルーチン呼び出しを生成するステートメントに対するフラグ付け
- リスト生成に使用される I/O の量の削減

使いやすさの向上

- 16 進での AGGREGATE リストにおけるオフセットの提供 (オプション)
- LIMITS(FIXEDDEC(15,31)) オプションによる、必要な場合のみの DEC(31) のサポート
- オプション内でのコメントの許可
- 偶数精度での FIXED DEC 宣言に対するフラグ付け (オプション)
- SIZE を発生させる可能性のある DEC 代入に対する DEC のフラグ付け
- SIZE を発生させる可能性のある PIC 代入に対する DEC/PIC のフラグ付け

- NOINIT 属性による、INIT なしの LIKE のサポート
- z/OS UNIX での PDS からの組み込みの容易化
- マクロ・プリプロセッサー内での LOWERCASE、MACNAME、TRIM および LOWERCASE 組み込み関数のサポート
- PTF による、オプションの紹介の容易化
- *PROCESS の使用の不許可 (オプション)
- MDECK 内での *PROCESS の保持 (オプション)
- 一回限りの INCLUDE のサポート
- マクロで決定する INCLUDE 名のサポート
- 実行時のストリング・パラメーター検査のサポート
- 未初期化変数である可能性があることに対するフラグ付けの強化
- コーディング・エラーの可能性がある、異常な比較に対するフラグ付け
- STORAGE オプションの出力のフォーマットが使いやすくなり、LIST オプションの出力には、コンパイル単位から各ブロックまでの 16 進オフセットが組み込まれるようになりました。

デバッグの向上

- オーバーレイ・フックに対するより良いサポート
- LE ダンプ内の CONTROLLED 変数の解決の容易性の強化
- リストへのユーザー指定のオプションの常時組み込み

V3R3 からの機能拡張

このリリースには、以下を含む Enterprise PL/I V3R3 で拡張された機能がすべて備わっています。

XML サポートの強化

XMLCHAR 組み込み関数が、参照される構造体のエレメントの名前と値で XML をバッファーに書き込み、書き込まれたバイト数を戻します。 次にこの XML は、PL/I SAX パーサーを使用したコードとともに他のアプリケーションに渡され、実行されます。

パフォーマンスの改善

- OPT(2) でのコンパイル時間は、Enterprise PL/I V3R2 の場合よりも、特に大規模なプログラムで大幅に少なくなります。
- コンパイラーは、ED および EDMK 命令を使用して、PICTURE および CHARACTER へのインライン化された数値変換を行います。 これにより、より早く、短いコード・シーケンスおよびコンパイルの高速化が実現しました。
- コンパイラーは、ストリング比較をさらに効率よく行うコードを生成するようになりました。このことも、より早く、短いコード・シーケンスという結果をもたらしています。
- コンパイラーは、より短く高速なコードを生成し、FIXED DECIMAL から、末尾に overpunch という文字のついた PICTURE への変換を行います。

- ARCH および TUNE コンパイラー・オプションは、有効なサブオプションとして 5 つを受け入れます。 ARCH(5) のもとでは、コンパイラーが適切なときに、NILL、NILH、OILL、OILH、LLILL、および LLILH などの新規 z/Architecture® 命令を生成します。

容易なマイグレーション

- 新規の BIFPREC コンパイラー・オプションは、さまざまな組み込み関数によって戻された FIXED BIN の結果の精度を制御し、これにより OS PL/I コンパイラーとのよりよい互換性を提供します。
- 新規の BACKREG コンパイラー・オプションは、コンパイラーが、逆チェーン・レジスターとしてどのレジスターを使用するかを制御し、これにより、古いオブジェクト・コードと新しいオブジェクト・コードの混合を容易にします。
- 新規の RESEXP コンパイラー・オプションは、コードの中の制限つき式の評価を制御し、OS PL/I コンパイラーとのよりよい互換性を提供します。
- 新規の BLKOFF コンパイラー・オプションは、コンパイラーの疑似アセンブラー・リストにおけるオフセットの計算方法を制御します。
- STORAGE コンパイラー・オプションは、それぞれのプロシージャーおよび開始ブロックによって使用されるストレージの要約をコンパイラーに作成させます。これは、リストの一部として作成され、OS PL/I コンパイラーで作成されたものと似ています。

使いやすさの向上

- RULES コンパイラー・オプションの新規の LAXDEF サブオプションにより、いわゆる無許可定義を使用できるようになり、この際、コンパイラーが E レベル・メッセージを生成することはありません。
- 新規の FLOATINMATH コンパイラー・オプションにより、数学関数の評価に関する精度の制御が容易になりました。
- 新規の MEMINDEX、MEMSEARCH(R) および MEMVERIFY(R) 組み込み関数により、32K より大きいストリングの検索が可能です。
- DISPLAY(WTO) コンパイラー・オプションの、新規の ROUTCDE および DESC サブオプションは、対応する WTO の要素の制御を提供します。
- コンパイラーは、それぞれのオブジェクトの中に短ストリングを保管し、それが関連するコードが実行されている間もストレージにあり、そのオブジェクトを作成するために使用されるオプションをすべて記録します。 これにより、さまざまなツールによるより良い診断が可能になりました。
- コンパイラーは、ステートメントがマージまたは削除された場所をさらに識別するメッセージを発行します。
- PLIDUMP 出力は、静的ユーザーの 16 進ダンプをインクルードするようになりました。
- PLIDUMP 出力は、言語環境プログラムのトレースバックで、それぞれのプログラムをコンパイルするために使用するオプションをインクルードするようになりました。
- PLIDUMP 出力は、PL/I ファイルに関する情報をインクルードするようになりました。

デバッグ・サポートの向上

- REFER を使用した BASED 構造体は、DebugTool およびデータ指示 I/O ステートメントでサポートされるようになりました (制限は他のすべての BASED 変数上と同じです)。
- 他の構造体のスカラー・メンバーから (順番に BASED されるなどで) BASED された BASED 構造体は、DebugTool およびデータ指示 I/O ステートメントでサポートされるようになりました (制限は他のすべての BASED 変数上と同じです)。

V3R2 からの機能拡張

本リリースでは、次を含む Enterprise PL/I V3R2 での機能強化もすべて提供されます。

パフォーマンスの改善

- このコンパイラーでは、インライン・コードを生成してより多くの型変換を処理できるようになりました。これにより、型変換が以前に比べ、格段に早く行われるようになります。また、ライブラリー呼び出しによって行われるすべての型変換は、コンパイラーによってフラグが立てられるようになりました。
- コンパイラー生成コードが、さまざまな状況において使用するスタック・ストレージの容量が減少しました。
- コンパイラーが、TRANSLATE 組み込み関数を参照するために生成するコードが改善されました。
- SUBSCRIPTRANGE 検査用のコンパイラー生成コードは、既知の境界を持つ配列の場合は、処理速度が以前の倍になりました。
- ARCH と TUNE オプションは、サブオプション 4 をサポートするようになり、zSeries マシンで新たに命令の開発が行えるようになりました。
- ARCH(2)、FLOAT(AFP) および TUNE(3) がデフォルトになりました。

容易なマイグレーション

- マイグレーションを容易にし、互換性を持たせるために、コンパイラーのデフォルト値が変更されました。変更されたデフォルト値は、次のとおりです。
  - CSECT
  - CMPAT(V2)
  - LIMITS(EXTNAME(7))
  - NORENT
- コンパイラーは、OPTIONS(COBOL) を指定した PROC および ENTRY で、NOMAP、NOMAPIN および NOMAP 属性を指定できるようになりました。
- コンパイラーは、複数の ENTRY ステートメントを指定した PROC をサポートするようになりました。この ENTRY ステートメントは、前リリースのホスト・コンパイラーと同様、それぞれが異なる RETURNS 属性を保持することができます。
- コンパイラーは、OPTIONS(RETCODE) において、PROC と ENTRY には OPTIONS(COBOL) が指定されていることを想定しています。
- 未処理の場合、SIZE 条件は ERROR にプロモートされません。

- コンパイル時間とストレージ要件を減らすために、さまざまな変更が行われました。
- OFFSET オプションは、前リリースの PL/I コンパイラーで生成されたものとよく似たステートメント・オフセット・テーブルを生成します。
- FLAG オプションは前リリースのコンパイラーの場合と意味はまったく同じですが、新規の MAXMSG オプションは、指定された重大度において、メッセージが指定した回数発生した後にコンパイラーを終了すべきかどうかを、ユーザーが決定できるようになりました。 例えば、FLAG(I) MAXMSG(E,10) を指定すると、I レベル・メッセージはすべて確認し、E レベル・メッセージは 10 回発生したらコンパイルを終了するように指定できます。
- AGGREGATE リストには、調節可能エクステントを備えた構造を組み込めるようになりました。
- STMT オプションは、リストのいくつかのセクションをサポートするようになりました。
- LINESIZE で使用できる最大値は、F フォーマット・ファイルでは 32759、V フォーマット・ファイルでは 32751 に変更されました。

使いやすさの向上

- コンパイラー・オプションのデフォルトは、インストール時に変更できるようになりました。
- 内蔵 SQL プリプロセッサーは、DB2 Unicode をサポートするようになりました。
- コンパイラーは、デバッグ・ツールが Auto Monitor をサポートできるようにする情報を生成するようになりました。それで、各ステートメントが実行される直前に、ステートメントで使用されるすべての変数のすべての値が表示されます。
- 新規の NOWRITABLE コンパイラー・オプションを使用すると、NORENT を指定した場合に、最適なパフォーマンスを犠牲にしても、コンパイラーが FILE と CONTROLLED を操作するコードを生成する際に、書き込み可能な静的値を使用しないように指定できます。
- 新規の USAGE コンパイラー・オプションを使用すると、RULES(IBM|ANS) オプションの他の影響を受けずに、ROUND および UNSPEC 組み込み関数の IBM または ANS 動作を完全に制御できます。
- 新規の STDSYS コンパイラー・オプションは、コンパイラーに SYSPRINT ファイルと C stdout ファイルを同一にするように指定します。
- 新規の COMPACT コンパイラー・オプションが使用されると、コードが大きくなることを制限する最適化を使用するようにコンパイラーに指示します。
- SYSPRINT の LRECL は 137 に変更され、C/C++ コンパイラーの LRECL と一致するようになりました。
- PUT LIST と PUT EDIT ステートメントで POINTER が使用できるようになりました。8 バイトの 16 進値が出力されます。
- ABNORMAL 属性を STATIC 変数で指定すると、STATIC 変数が使用されていなくてもこの変数は保存されます。

V3R1 からの機能拡張

本リリースでは、次を含む Enterprise PL/I V3R1 での機能強化もすべて提供されます。

- z/OS でのマルチスレッド化のサポート
- z/OS での IEEE 浮動小数点のサポート
- マクロ・プリプロセッサーでの ANSWER ステートメントのサポート
- PLISAXA および PLISAXB 組み込みサブルーチンを介した SAX 形式 XML 構文解析
- 追加の組み込み関数
  - CS
  - CDS
  - ISMAIN
  - LOWERCASE
  - UPPERCASE

VisualAge PL/I からの機能拡張

本リリースでは、次を含む VisualAge® PL/I V2R2 での機能強化もすべて提供します。

- WIDECHAR 属性を介した初期 UTF-16 サポート

  以下については、まだサポートされていません。
  - ソース・ファイル内の WIDECHAR 文字
  - W ストリング定数
  - ストリーム入出力内の WIDECHAR 式の使用
  - レコード入出力での WIDECHAR からの暗黙の型変換または WIDECHAR への暗黙の型変換
  - レコード入出力での暗黙の endianness フラグ

    WIDECHAR ファイルを作成する場合は、ファイルの最初の 2 バイトとして endianness フラグ ('fe_ff'wx) を書き込んでください。
- DEFAULT ステートメントでサポートされる DESCRIPTORS オプションと VALUE オプション
- PUT DATA 機能強化
  - POINTER、OFFSET およびサポートされているその他の非計算変数
  - タイプ 3 DO 仕様が使用可能
  - 添え字が使用可能
- DEFINE ステートメントの機能強化
  - 指定されていない構造体の定義
  - CAST および RESPEC タイプ関数
- 追加の組み込み関数
  - CHARVAL
  - ISIGNED
  - IUNSIGNED
  - ONWCHAR
  - ONWSOURCE

  - WCHAR
  - WCHARVAL
  - WHIGH
  - WIDECHAR
  - WLOW
- プリプロセッサー機能強化
  - プリプロセッサー・プロシージャーでの配列のサポート
  - %DO ステートメントでの WHILE、UNTIL および LOOP キーワードのサポート
  - %ITERATE ステートメントのサポート
  - %LEAVE ステートメントのサポート
  - %REPLACE ステートメントのサポート
  - %SELECT ステートメントのサポート
  - 追加の組み込み関数
    - COLLATE
    - COMMENT
    - COMPILEDATE
    - COMPILETIME
    - COPY
    - COUNTER
    - DIMENSION
    - HBOUND
    - INDEX
    - LBOUND
    - LENGTH
    - MACCOL
    - MACLMAR
    - MACRMAR
    - MAX
    - MIN
    - PARMSET
    - QUOTE
    - REPEAT
    - SUBSTR
    - SYSPARM
    - SYSTEM
    - SYSVERSION
    - TRANSLATE
    - VERIFY

ご意見の送付方法

本書または PL/I の他のマニュアルについてご意見がありましたら、IBM 発行のマニュアルに関する情報の Web ページ (http://www.ibm.com/jp/manuals/) よりお送りください。今後の参考にさせていただきます。(URL は、変更になる場合があります)

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーが情報技術の内容を首尾良く使用できるように支援します。z/OS のアクセシビリティー機能は、Enterprise PL/I にアクセシビリティーを提供します。

アクセシビリティー機能

z/OS には、以下の主要アクセシビリティー機能が含まれています。

- スクリーン・リーダーおよび画面拡大ソフトウェアで一般的に使用されるインターフェース
- キーボードのみのナビゲーション
- 表示属性 (色、コントラスト、フォント・サイズなど) のカスタマイズ機能

z/OS では、US Section 508 (http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards) およびWeb Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/) に準拠するために、最新の W3C 標準である WAI-ARIA 1.0 (http://www.w3.org/TR/wai-aria/) が使用されます。アクセシビリティー機能を利用するには、この製品でサポートされる最新の Web ブラウザーと、最新リリースのスクリーン・リーダーを組み合わせて使用します。

IBM Knowledge Center にある Enterprise PL/I オンライン製品資料はアクセシビリティーに対応しています。IBM Knowledge Center のアクセシビリティー機能については、http://www.ibm.com/support/knowledgecenter/en/about/releasenotes.html に説明があります。

キーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS ユーザー・インターフェースにアクセスできます。

また、ユーザーは IBM Developer for z Systems を使用して z/OS サービスにアクセスできます。

このようなインターフェースへのアクセスについては、以下の資料を参照してください。

- *z/OS TSO/E Primer* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4p120)
- *z/OS TSO/E User's Guide* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c240/APPENDIX1.3)
- *z/OS ISPF User's Guide Volume I* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzug70)
- IBM Rational® Developer for System z® Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSQ2R2/rdz_welcome.html?lang=en)

上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む TSO/E および ISPF の使用方法が記載されています。 それ

ぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

インターフェース情報

Enterprise PL/I オンライン製品資料は IBM Knowledge Center で入手でき、標準の Web ブラウザーで表示できます。

PDF ファイルでのアクセシビリティー・サポートは限定的です。PDF 資料では、オプションのフォント拡大設定やハイコントラスト表示設定を使用したり、キーボードのみでナビゲートしたりできます。

ご使用のスクリーン・リーダーが構文図やソース・コード例を正確に読み上げたり、ピリオドやコンマといった PICTURE 記号を含むテキストを正確に読み上げたりできるようにするには、ご使用のスクリーン・リーダーがすべての句読点を読み上げるように設定する必要があります。

支援技術製品は、z/OS に備わっているユーザー・インターフェースと連動します。特定のガイダンス情報については、z/OS インターフェースへのアクセスに使用する支援技術製品の資料を参照してください。

関連アクセシビリティー情報

IBM は、標準の IBM ヘルプ・デスクとサポート Web サイトに加え、聴覚に障害を持つお客様が販売サービスやサポート・サービスへのアクセスに使用するための TTY 電話サービスを開設しました。

TTY サービス
800-IBM-3383 (800-426-3383)
(北アメリカ内)

IBM とアクセシビリティー

アクセシビリティーに対する IBM の取り組みについて詳しくは、「IBM Accessibility」(www.ibm.com/able) を参照してください。

第 1 部 プログラムのコンパイル

© Copyright IBM Corp. 1999, 2017

第 1 章 コンパイラー・オプションと機能の使用

この章では、コンパイラーに使用できるオプションと、その省略形および IBM 提供のデフォルトについて説明します。

重要: PL/I はコンパイル時に言語環境プログラムランタイムへのアクセスを要求します。

Enterprise PL/I を使用すれば、31 ビット・アプリケーションと 64 ビット・アプリケーションの両方を開発できます。コードをコンパイルするときは、31 ビット・アプリケーションには LP(32) オプションを使用し、64 ビット・アプリケーションには LP(64) オプションを使用します。ただし、LP(64) でのコンパイラーの動作は LP(32) でのコンパイラーの動作とは異なります。詳しくは、 221 ページの『第 7 章 64 ビット・アプリケーションを開発する場合の考慮事項』および 215 ページの『第 6 章 64 ビット・プログラムに対するリンク・エディットおよび実行』を参照してください。

大部分のデフォルトは、PL/I プログラムのコンパイル時に指定変更することができます。 コンパイラーのインストール時には、デフォルトを指定変更することもできます。

リリースに対するコンパイラー・オプション・デフォルト設定はすべて、SIBMZSAM データ・セットにおいて IBMXO*vrm* という名前を持つメンバーにリストされます。 *v* はバージョン番号です。 *r* はリリース番号です。 *m* はモディフィケーション番号です。通常、モディフィケーション番号は 0 です。例えば、V5R1 の場合は IBMXO510 となり、V4R5 の場合は IBMXO450 となります。このデータ・セットには、すべてのサポート対象 PL/I リリースに対する IBMXO*vrm* メンバーが含まれています。あるリリースから次のリリースの間に変更されたデフォルト設定の内容および追加された新規オプションを確認するには、その 2 つのリリースの IBMXO*vrm* ファイルを比較します。 IBMXO*vrm* ファイルは、希望の設定を使用して独自のオプション・ファイルを作成するためのテンプレートとしても使用できます。

コンパイル時オプションの説明

多くのコンパイラー・オプションに肯定形式と否定形式があります。否定形式は、肯定形式の初めに `NO` を付け加えたものです (例えば TEST および NOTEST)。オプションによっては、肯定形式のみを持つものもあります (例: SYSTEM)。

コンパイラー・オプションには以下の 3 つのタイプがあります。

1. キーワードの単純な組み合わせ:機能を要求する肯定形式、およびその機能を禁止する代替否定形式 (例えば、NEST および NONEST)
2. オプションを修飾する値リストを提供するためのキーワード (例えば、FLAG(W))
3. 上記の 1 と 2 を組み合わせたもの (例えば、NOCOMPILE(E))

© Copyright IBM Corp. 1999, 2017

表 3 に、すべてのコンパイラー・オプションと、その省略形 (存在する場合)、および IBM 提供のデフォルト値をリストします。省略できるサブオプションがオプションに含まれている場合、その省略形については、オプションの詳細説明のところで説明します。

簡便のために、テーブル内のいくつかのオプションは簡単に説明しています (例えば、LANGLVL で必要となるサブオプションは 1 つだけです。同様に、TEST でサブオプションを 1 つ指定したら、他を指定する必要はありません)。 完全に正確な構文全体については、後続のトピックで説明します。

表 3. コンパイル時オプション、省略形、および *IBM* 提供のデフォルト値

コンパイル時オプション	省略名	**z/OS** デフォルト
AGGREGATE[(DEC|HEX)] | NOAGGREGATE	AG | NAG	NOAGGREGATE
ARCH(n)	-	ARCH(9)
ASSERT(ENTRY | CONDITION)	-	ASSERT(ENTRY)
ATTRIBUTES[(FULL|SHORT)] | NOATTRIBUTES	A | NA	NA [(FULL)][1]
BACKREG(5 | 11)	-	BACKREG(5)
BIFPREC(15 | 31)	-	BIFPREC(15)
BLANK('c')	-	BLANK('t')[2]
BLKOFF | NOBLKOFF	-	BLKOFF
BRACKETS('*symbol_1symbol_2*')	-	BRACKETS('[]')
CASE(UPPER | ASIS)	-	CASE(UPPER)
CASERULES(KEYWORD(LOWER | MIXED | START | UPPER))	-	CASERULES(KEYWORD (MIXED))
CEESTART(FIRST | LAST)	-	CEESTART(FIRST)
CHECK(STORAGE | NOSTORAGE, CONFORMANCE | NOCONFORMANCE)	-	CHECK(NSTG, NOCONFORMANCE)
CMPAT(LE | V1 | V2 | V3)	CMP	CMPAT(V2)
CODEPAGE(n)	CP	CODEPAGE(1140)
COMMON | NOCOMMON	-	NOCOMMON
COMPILE | NOCOMPILE[(W | E | S)]	C | NC	NOCOMPILE(S)
COPYRIGHT('string') | NOCOPYRIGHT	-	NOCOPYRIGHT
CSECT | NOCSECT	CSE | NOCSE	CSECT
CSECTCUT(n)	-	CSECTCUT(4)
CURRENCY('c')	CURR	CURRENCY($)
DBCS | NODBCS	-	NODBCS
DBRMLIB(*'data-set-name'*) | NODBRMLIB	-	NODBRMLIB
DD(ddname-list)	-	DD(SYSPRINT,SYSIN, SYSLIB,SYSPUNCH, SYSLIN,SYSADATA, SYSXMLSD,SYSDEBUG)
DDSQL(ddname)	-	DDSQL('')
DECIMAL(FOFLONASGN | NOFOFLONASGN, FOFLONMULT | NOFOFLONMULT, FORCEDSIGN | NOFORCEDSIGN, TRUNCFLOAT | NOTRUNCFLOAT)	DEC	DEC(FOFLONASGN, NOFOFLONMULT, NOFORCEDSIGN, NOTRUNCFLOAT)
DECOMP | NODECOMP	-	NODECOMP
DEFAULT(*attribute* | *option*)	DFT	「デフォルト」を参照
DEPRECATE(BUILTIN(*built-in-name*) | ENTRY(*entry-name*) | INCLUDE(*filename*) | STMT(*statement-name*) | VARIABLE(*variable-name*))	-	DEPRECATE(BUILTIN() ENTRY() INCLUDE() STMT() VARIABLE())

表 3. コンパイル時オプション、省略形、および *IBM* 提供のデフォルト値 (続き)

コンパイル時オプション	省略名	**z/OS** デフォルト
DEPRECATENEXT(BUILTIN(*built-in-name*) | ENTRY(*entry-name*) | INCLUDE(*filename*) | STMT(*statement-name*) | VARIABLE(*variable-name*))	-	DEPRECATENEXT(BUILTIN() ENTRY() INCLUDE() STMT() VARIABLE())
DISPLAY (STD | WTO(ROUTCDE(x) DESC(y) REPLY(z)))	-	DISPLAY(WTO)
DLLINIT | NODLLINIT	-	NODLLINIT
EXIT | NOEXIT	-	NOEXIT
EXTRN(FULL | SHORT)	-	EXTRN(FULL)
EXPORTALL	-	EXPORTALL
FILEREF | NOFILEREF	-	FILEREF
FLAG[(I | W | E | S)]	F	FLAG(W)
FLOAT(DFP | NODFP)	-	FLOAT(NODFP)
FLOATINMATH(ASIS | LONG | EXTENDED)	-	FLOATINMATH(ASIS)
GOFF | NOGOFF	-	NOGOFF
GONUMBER(SEPARATE | NOSEPARATE) | NOGONUMBER	GN | NGN	NOGONUMBER
GRAPHIC | NOGRAPHIC	GR | NGR	NOGRAPHIC
IGNORE(ASSERT | DISPLAY | PUT) | NOIGNORE	-	NOIGNORE
HEADER(SOURCE | FILE)	-	SOURCE
INCAFTER([PROCESS(*filename*)])	-	INCAFTER()
INCDIR('*directory name*') | NOINCDIR	-	NOINCDIR
INCLUDE | NOINCLUDE	-	INCLUDE
INCPDS('*PDS name*') | NOINCPDS	-	NOINCPDS
INITAUTO([SHORT | FULL]) | NOINITAUTO	-	NOINITAUTO
INITBASED | NOINITBASED	-	NOINITBASED
INITCTL | NOINITCTL	-	NOINITCTL
INITSTATIC | NOINITSTATIC	-	NOINITSTATIC
INSOURCE[(FULL|SHORT)(ALL|FIRST)] | NOINSOURCE	IS | NIS	NOINSOURCE
INTERRUPT | NOINTERRUPT	INT | NINT	NOINTERRUPT
JSON(CASE(UPPER | ASIS))	-	JSON(CASE(UPPER))
LANGLVL(NOEXT | OS)	-	LANGLVL(OS)
LIMITS(*options*)	-	51 ページの『LIMITS』を参照
LINECOUNT(*n*)	LC	LINECOUNT(60)
LINEDIR | NOLINEDIR	-	NOLINEDIR
LIST | NOLIST	-	NOLIST
LISTVIEW(SOURCE | AFTERMACRO | AFTERCICS | AFTERSQL | AFTERALL)	-	LISTVIEW(SOURCE)
LP(32 | 64)	-	LP(32)
MACRO | NOMACRO	M | NM	NOMACRO
MAP | NOMAP	-	NOMAP
MARGINI('c') | NOMARGINI	MI | NMI	NOMARGINI
MARGINS(m,n[,c])| NOMARGINS	MAR(m,n)	MARGINS F-format: (2,72) V-format: (10,100)
MAXBRANCH(max)	-	MAXBRANCH(2000)
MAXGEN(n)	-	MAXGEN(100000)
MAXMEM(n)	MAXM	MAXMEM(1048576)
MAXMSG(I | W | E | S,n)	-	MAXMSG(W,250)
MAXNEST(BLOCK(x) DO(y) IF(z))	-	MAXNEST(BLOCK(17) DO(17) IF(17))
MAXSTMT(n)	-	MAXSTMT(4096)
MAXTEMP(n)	-	MAXTEMP(50000)

表 3. コンパイル時オプション、省略形、および *IBM* 提供のデフォルト値 (続き)

コンパイル時オプション	省略名	**z/OS** デフォルト
MDECK | NOMDECK	MD | NMD	NOMDECK
MSGSUMMARY[(XREF | NOXREF)] | NOMSGSUMMARY	-	NOMSGSUMMARY
NAME[('*external-name*')] | NONAME	N	NONAME
NAMES('*lower*'[,*upper*])	-	NAMES('#@$','#@$')
NATLANG(ENU | UEN)	-	NATLANG(ENU)
NEST | NONEST	-	NONEST
NOT	-	NOT('¬')
NULLDATE | NONULLDATE	-	NONULLDATE
NUMBER | NONUMBER	NUM | NNUM	NUMBER
OBJECT | NOOBJECT	OBJ | NOBJ	OBJECT
OFFSET | NOOFFSET	OF | NOF	NOOFFSET
OFFSETSIZE(n)[3]	-	OFFSETSIZE(4)
ONSNAP(STRINGRANGE, STRINGSIZE) | NOONSNAP	-	NOONSNAP
OPTIMIZE(0 | 2 | 3) | NOOPTIMIZE	OPT | NOPT	OPT(0)
OPTIONS[(ALL|DOC)] | NOOPTIONS	OP | NOP	NOOPTIONS
OR('c')	-	OR(' | ')
PP(*pp-name*) | NOPP	-	NOPP
PPCICS('string') | NOPPCICS	-	NOPPCICS
PPINCLUDE('string') | NOPPINCLUDE	-	NOPPINCLUDE
PPLIST(KEEP | ERASE)	-	PPLIST(KEEP)
PPMACRO('string') | NOPPMACRO	-	NOPPMACRO
PPSQL('string') | NOPPSQL	-	NOPPSQL
PPTRACE | NOPPTRACE	-	NOPPTRACE
PREFIX(*condition*)	-	「PREFIX」を参照
PRECTYPE(ANS | DECDIGIT | DECRESULT)	-	PRECTYPE(ANS)
PROCEED | NOPROCEED[(W | E | S)]	PRO | NPRO	NOPROCEED(S)
PROCESS[(KEEP | DELETE)] | NOPROCESS	-	PROCESS(DELETE)
QUOTE('"')	-	QUOTE('"')
REDUCE | NOREDUCE	-	REDUCE
RENT | NORENT	-	NORENT
RESEXP | NORESEXP	-	RESEXP
RESPECT([DATE])	-	RESPECT()
RTCHECK(NULLPTR | NONULLPTR | NULL370)	-	RTCHECK(NONULLPTR)
RULES(*options*)	-	79 ページの『RULES』を参照
SEMANTIC | NOSEMANTIC[(W | E | S)]	SEM | NSEM	NOSEMANTIC(S)
SERVICE('*service string*') | NOSERVICE	SERV | NOSERV	NOSERVICE
SOURCE | NOSOURCE	S | NS	NOSOURCE
SPILL(n)	SP	SPILL(512)
STATIC(FULL | SHORT)	-	STATIC(SHORT)
STDSYS | NOSTDSYS	-	NOSTDSYS
STMT | NOSTMT	-	NOSTMT
STORAGE | NOSTORAGE	STG | NSTG	NOSTORAGE
STRINGOFGRAPHIC(CHAR | GRAPHIC)	-	STRINGOFGRAPHIC (GRAPHIC)
SYNTAX | NOSYNTAX[(W | E | S)]	SYN | NSYN	NOSYNTAX(S)
SYSPARM('string')	-	SYSPARM('')
SYSTEM(MVS | CICS | IMS | TSO | OS)	-	SYSTEM(MVS)
TERMINAL | NOTERMINAL	TERM | NTERM	
TEST(*options*) | NOTEST	-	101 ページの『TEST』[4] を参照
UNROLL(AUTO | NO)		UNROLL(AUTO)

表 3. コンパイル時オプション、省略形、および *IBM* 提供のデフォルト値 (続き)

コンパイル時オプション	省略名	**z/OS** デフォルト
USAGE(*options*)	-	105 ページの『USAGE』を参照
WIDECHAR(BIGENDIAN | LITTLEENDIAN)	WCHAR	WIDECHAR(BIGENDIAN)
WINDOW(*w*)	-	WINDOW(1950)
WRITABLE | NOWRITABLE[(FWS|PRV)]	-	WRITABLE
XINFO(*options*)	-	XINFO(NODEF,NOMSG, NOSYM,NOSYN,NOXML)
XML(CASE(UPPER | ASIS))	-	XML(CASE(UPPER))
XREF[(FULL | SHORT)(EXPLICIT | IMPLICIT)] | NOXREF	X | NX	NX [(FULL)][1]

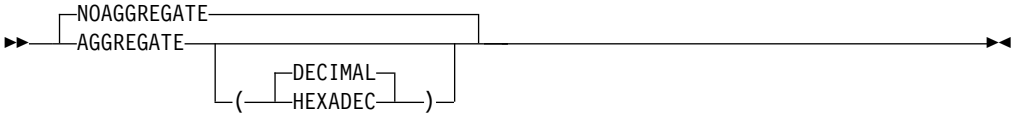
注:

1. FULL は、ATTRIBUTES または XREF の指定でサブオプションが省略された場合のデフォルト・サブオプションです。
2. BLANK 文字のデフォルト値は、 '05'x 値のタブ文字です。
3. OFFSETSIZE オプションは、LP(32) オプションが有効になっていると無視されます。
4. (ALL,SYM) は、TEST の指定でサブオプションを省略した場合のデフォルト・サブオプションです。

以下のトピックでオプションをアルファベット順に説明します。コンパイラーが情報をリストするように指定するオプションについては、簡単な説明のみが含まれています。生成されるリストについて詳しくは、 117 ページの『コンパイラー・リストの使用』を参照してください。

AGGREGATE

AGGREGATE オプションは、コンパイラーのリスト時にソース・プログラムの配列と大構造の長さを示す集合長さテーブルを作成します。



省略形: AG、NAG

AGGREGATE オプションのサブオプションは、集合長さテーブルにおいてサブエレメントのオフセットがどのように表示されるのかを決定します。

DECIMAL
すべてのオフセットが 10 進数で表示されます。

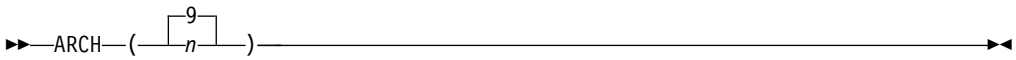
HEXADEC
すべてのオフセットが 16 進数で表示されます。

集合長さテーブルでは、次元設定のない大構造または小構造は、常にバイトで表現されますが、大構造または小構造に位置合わせされていないビット・エレメントが含まれていると、長さが不正確になる場合があります。

集合長さテーブルには、配列ではなく非固定エクステントを持つ構造が組み込まれています。 しかし、この構造体は非固定エクステントを保持し、構造体内部のエレメントのサイズとオフセットは、不正確であるか、または * として指定されます。

ARCH

ARCH オプションは、実行可能プログラムの命令が生成されるアーキテクチャーを指定します。このオプションにより、最適化プログラムは特定のハードウェア命令セットの利点を利用できます。



ARCH レベルに対して以下の値を指定できます。

9 2817-xxx モデル (IBM zEnterprise 196 (z196)) および 2818-xxx モデル (IBM zEnterprise 114 (z114)) で使用できる命令が z/Architecture モードで利用されるコードが生成されます。

特に、これら ARCH(9) マシンおよびその後継機は、コンパイラーで活用できる以下の機能によってサポートされる命令を追加します。

- high-word facility
- population count facility
- distinct-operands facility
- floating-point extension facility
- load/store-on-condition facility

これらの機能について詳しくは、「*z/Architecture* 解説書」を参照してください。

10 2827-xxx モデル (IBM zEnterprise EC12) で使用できる命令が z/Architecture モードで利用されるコードが生成されます。

特に、これら ARCH(10) マシンおよびその後継機は、コンパイラーで活用できる以下の機能によってサポートされる命令を追加します。

- execution-hint facility
- load-and-trap facility
- miscellaneous-instructions-extension facility
- transactional-execution facility

これらの機能の詳細については、「*z/Architecture* 解説書」を参照してください。

11 2964-xxxx モデル (IBM z13®) で使用できる命令が z/Architecture モードで利用されるコードが生成されます。

特に、その ARCH(11) マシンとその後継マシンは、以下の機能によってサポートされる命令を追加します。

- ベクトル機構

ARCH(11) またはより上位の ARCH レベルでコンパイルされたコードは、ベクトル命令とベクトル・コンテキスト切り替えをサポートするランタイム環境を必要とします。ベクトル命令は、以下の環境においてランタイム異常終了を引き起こす可能性があります。

- V2.1 対応 APAR PI12281 の PTF が適用される前の z/OS バージョン

- V6.3 対応 APAR VM65733 の PTF が適用される前の z/VM® バージョンで稼働している z/OS イメージ
- V5.3 対応 APAR PI59322 の PTF が適用される前の CICS Transaction Server バージョン

12 3906-xxxx モデル (IBM z14) で使用できる命令が z/Architecture モードで利用されるコードが生成されます。

特に、これらの ARCH(12) マシンとその後継では、ベクトル・パック 10 進機能をサポートする命令が追加されています。この機能は、中間結果をメモリーではなくベクトル・レジスターに保管することによって、パック 10 進計算およびゾーン 10 進計算を加速します。

注**:**

1. 9 より小さい ARCH 値を指定すると、コンパイラーがその値を 9 に再設定します。
2. 上記のモデル番号における x (2084-xxx など) はワイルドカードであり、そのタイプを持つ任意のマシン (英数字からなる) を表します。
3. 上位の ARCH レベルには、下位の ARCH レベルの機能が含まれています。例えば、ARCH(12) には、それよりも低い ARCH レベルの機能がすべて含まれています。
4. さまざまな ARCH レベルでコンパイルされたコードを制限なく混用できます。

ASSERT

ASSERT オプションは、ASSERT ステートメントが、ASSERTION 条件を発生させるデフォルト・ライブラリー・ルーチンを呼び出すか、またはユーザーによって指定されたルーチンを呼び出すかを制御します。

デフォルトは ASSERT(ENTRY) です。

ENTRY

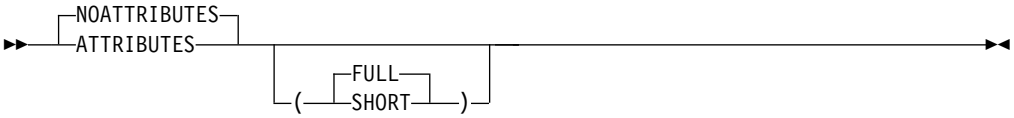
ASSERT ステートメントが、「*Enterprise PL/I for z/OS* 言語解説書」 に記述されているインターフェースでユーザー指定のルーチンを呼び出すことを指定します。

CONDITION

ASSERT ステートメントが、ASSERTION 条件を発生させるライブラリー・ルーチンを、該当する ONCODE 組み込み関数で呼び出すことを指定します。

ATTRIBUTES

ATTRIBUTES オプションは、コンパイラー・リスト内にソース・プログラム ID とそれぞれの属性のテーブルをコンパイラーが組み込むことを指定します。



省略形: A、NA、F、S

FULL
すべての ID と属性がコンパイラー・リストに組み込まれます。 FULL がデフォルトです。

SHORT
非参照 ID が省かれ、リストの取り扱いが簡単になります。

ATTRIBUTES と XREF (相互参照テーブルを作成する) を両方組み込むと、2 つのテーブルが結合されます。ただし SHORT サブオプションと FULL サブオプションが対立する場合は、後から指定したオプションが使用されます。 例えば、ATTRIBUTES (SHORT) XREF (FULL) と指定すると、組み合わされたリストには FULL が使用されます。

BACKREG

BACKREG オプションは逆チェーン・レジスターを制御するものです。逆チェーン・レジスターは、ネストされたルーチンが呼び出されたときに、親ルーチンが自動的に保管される親ルーチンのアドレスを渡すために使用されます。

注: LP(64) オプションでは、BACKREG オプションは無視されます。

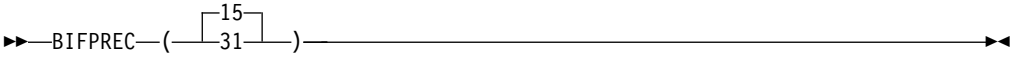
PL/I for MVS & VM、OS PL/I V2R3、およびそれ以前のコンパイラーとの最良の互換性を得るには、BACKREG(5) を使用します。

ENTRY VARIABLE を共用するルーチンはすべて同じ BACKREG オプションでコンパイルしなければなりません。また、アプリケーションの中のコードはすべて同じ BACKREG オプションでコンパイルすることを強くお勧めします。

事実上、VisualAge PL/I for OS/390® でコンパイルされたコードでは、BACKREG(11) オプションが使用されます。 Enterprise PL/I V3R1 または V3R2 でコンパイルされたコードでも、デフォルトで BACKREG(11) オプションが使用されます。

BIFPREC

BIFPREC オプションは、各種組み込み関数によって返される FIXED BIN 結果の精度を制御します。



PL/I for MVS & VM、OS PL/I V2R3、およびそれ以前のコンパイラーとの最良の互換性を得るには、BIFPREC(15) を使用します。

BIFPREC は次の組み込み関数に影響します。

- COUNT
- INDEX
- LENGTH
- LINENO
- ONCOUNT
- PAGENO
- SEARCH
- SEARCHR
- SIGN
- VERIFY
- VERIFYR

BIFPREC コンパイラー・オプションの影響が最も明らかに見えるのは、上記の組み込み関数の結果の 1 つが、パラメーター・リストなしに宣言された外部関数に受け渡されるときです。例えば、次のような部分コードがあるとします。

```
dcl parm char(40) var;
dcl funky ext entry( pointer, fixed bin(15) );
dcl beans ext entry;
call beans( addr(parm), verify(parm), ' ' );
```

実際に関数 beans が当該パラメーターを POINTER および FIXED BIN(15) として宣言するとします。上記のコードがオプション BIFPREC(31) でコンパイルされている場合、および上記のコードが z/OS のようなビッグ・エンディアン・システム上で実行される場合、コンパイラーは 2 番目の引数として 4 バイトの整数を渡し、2 番目のパラメーターはゼロとなります。

関数 funky は、すべてのシステム上でどちらのオプションでも機能することに注意してください。

BIFPREC オプションは、組み込み関数 DIM、HBOUND および LBOUND には影響しません。 CMPAT オプションは、これら 3 つの組み込み関数によって返される FIXED BIN 結果の精度を決定します。

- CMPAT(V1) では、これらの配列処理関数によって FIXED BIN(15) の結果が返されます。
- CMPAT(V2) および CMPAT(LE) では、FIXED BIN(31) の結果が返されます。
- CMPAT(V3) では、FIXED BIN(63) の結果が返されます。

BLANK

BLANK オプションは、ブランク文字の代替記号を 10 個まで指定します。

注: 引用符と引用符の間にブランクをコーディングしないでください。

BLANK 記号用の IBM 提供のデフォルト・コード・ポイントは、'05'X です。

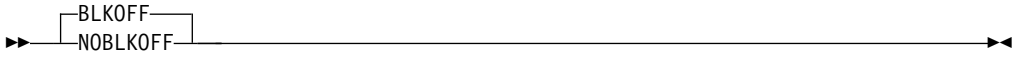
char
単一の SBCS 文字

英字も、数字も、「*PL/I* 言語解説書」に定義されている特殊文字も指定できません。

BLANK オプションを指定した場合でも、標準のブランク記号はブランクとして認識されます。

BLKOFF

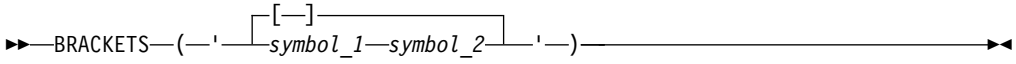
BLKOFF オプションは、疑似アセンブラー・リスト (LIST オプションによって生成される) とステートメント・オフセット・リスト (OFFSET オプションによって生成される) に示されるオフセットを、現行モジュールの開始位置を基準とするか、現行プロシージャーの開始位置を基準とするかを制御します。



疑似アセンブラー・リストには、現行モジュールの開始位置からの各ブロックのオフセットも含まれています (それぞれのステートメントに表示されるオフセットを、ブロックまたはモジュールのオフセットに変換できるようにするため)。

BRACKETS

BRACKETS オプションは、SQL 配列参照において左右の括弧として SQL プリプロセッサーが受け入れる記号を指定します。



symbol_1
SQL 配列参照において左括弧として認識される記号を指定します。

symbol_2
SQL 配列参照において右括弧として認識される記号を指定します。

注: 指定された 2 つの値は相互に異なるものでなければならず、PL/I 文字セットや他の PL/I オプション (NAMES、NOT、OR など) で使用されている文字であってはなりません。

デフォルトは BRACKETS('[]') です。

関連情報:

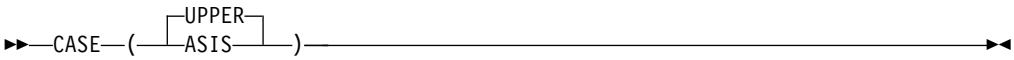
62 ページの『NAMES』
NAMES オプションでは、ID に使用できる特別言語文字 を 指定します。

63 ページの『NOT』
NOT オプションでは、論理否定演算子として使用できる代替記号を 最大 7 つ指定します。

68 ページの『OR』
OR オプションでは、論理 OR 演算子として最大 7 つの代替記号を 指定します。これらの記号は連結演算子としても使用されます。連結演算子は 2 つの 連続した論理和記号と定義されます。

CASE

CASE オプションは、一部の名前が大文字で表示されるか、またはソース・プログラムでの表示と同じフォーマットで表示されるかを制御します。



デフォルトは CASE(UPPER) です。

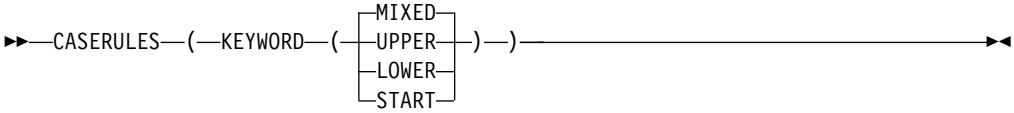
UPPER
すべての名前が大文字で表示されることを指定します。

ASIS
以下の対象において、ソースにおける大/小文字で名前が表示されることを指定します。

- リスト AGGREGATE、ATTRIBUTES、および XREF
- 組み込み関数 PACKAGENAME、PROCNAME、および ORDINALNAME によって返される値
- PUT DATA 出力 (TEST オプションが使用されていない場合、および GET DATA ステートメントで名前が使用されていない場合)

CASERULES

CASERULES オプションは、キーワードの大/小文字に関する規則の適用を制御します。



LOWER
小文字になっていないキーワードにフラグを立てるようにコンパイラーに指示します。

MIXED
すべてのキーワードをコーディングされたままの状態で受け入れるようにコンパイラーに指示します。 MIXED がデフォルトです。

START
先頭文字が大文字になっていなかったり後続の文字が小文字になっていなかったりするキーワードにフラグを立てるようにコンパイラーに指示します。

UPPER
大文字になっていないキーワードにフラグを立てるようにコンパイラーに指示します。

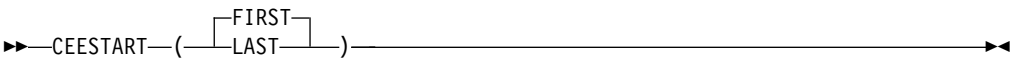
注:

1. CASERULES オプションは、OPTIONS 属性と ENVIRONMENT 属性のエレメントには適用されません。
2. CASERULES オプションは、プリプロセッサーには適用されません。

CEESTART

CEESTART オプションは、コンパイラーが CEESTART csect を配置する場所を、他のすべての生成済みオブジェクト・コードの前にするか後にするかを指定します。

注: LP(64) オプションでは、CEESTART オプションは無視されます。



CEESTART(FIRST) オプションを使用すると、コンパイラーは CEESTART csect を、他のすべての生成済みオブジェクト・コードの前に配置します。 CEESTART(LAST) オプションを使用すると、他のすべての生成済みオブジェクト・コードの後に配置します。

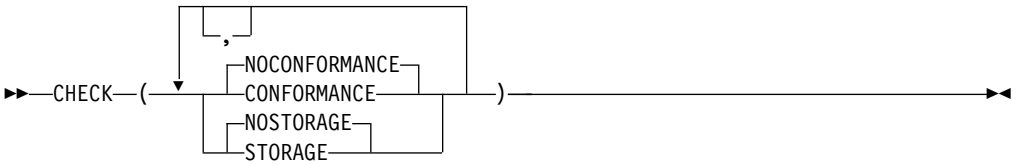
CEESTART(FIRST) を使用すると、バインド・ステップで ENTRY カードが指定されていなければ、バインダーはモジュールのエントリー・ポイントとして CEESTART を選択することになります。

リンカー CHANGE カードを使用する場合は、CEESTART(LAST) オプションを使用する必要があります。

なお、MAIN ルーチンは、ENTRY CEESTART リンケージ・エディター・カードとリンクする必要があります。 ただし CEESTART(LAST) オプションを使用する場合は、MAIN ルーチンをリンクするときに ENTRY CEESTART カードをインクルードする必要があります。

CHECK

CHECK オプションは、コンパイラーが、さまざまなプログラミング・エラーを検出するための特殊なコードを生成するかどうかを指定します。



省略形: STG、NSTG

CONFORMANCE | NOCONFORMANCE

CHECK(CONFORMANCE) を指定すると、プロシージャーに渡された引数の属性が、宣言されたパラメーターの属性と一致する場合に、以下の状況で、実行時に検査するコードをコンパイラーが生成します。

- パラメーターが固定長で宣言されたストリング (またはストリングの配列) である場合、渡された引数が一致する長さでないと、STRINGSIZE 条件が発生します。
- パラメーターがストリング (またはストリングの配列) である場合、引数が同じ長さタイプ (VARYING、NONVARYING、または VARYINGZ) でないと、STRINGSIZE 条件が発生します。
- パラメーターが (スカラーまたは構造体の) 配列である場合、渡された引数の定数境界と一致しない定数境界があると、SUBSCRIPTRANGE 条件が発生します。 すべてのエクステントが定数で、引数に含まれる配列エレメントのサイズとスペーシングがパラメーターのものと一致しない場合も、SUBSCRIPTRANGE 条件が発生します。 構造体内部の配列はチェックされません。
- パラメーターが定数エクステントを持つ構造体または共用体である場合、最後のエレメントのオフセットが、渡された引数のオフセットと一致しないと、SUBSCRIPTRANGE 条件が発生します。
- プロシージャーに RETURNS BYADDR 属性があり、その属性がストリング・タイプを指定している場合、RETURNS 値に対して渡されたストリングが一致する長さでないと、STRINGSIZE 条件が発生します。

この追加のコードは、以下のいずれかの条件が当てはまる場合は生成されません。

- プロシージャーに NODESCRIPTOR オプションが適用される。
- ブロックに ENTRY ステートメントが含まれる。
- CMPAT(LE) オプションが有効である。

STORAGE | NOSTORAGE

CHECK(STORAGE) を指定すると、コンパイラーは ALLOCATE ステートメントと FREE ステートメントに対して多少異なるライブラリー・ルーチンを呼び出します (これらのステートメントが AREA 内で使用される場合を除く)。

注: LP(64) オプションでは STORAGE サブオプションはサポートされていません。

「*PL/I* 言語解説書」で説明されている次の組み込み関数は、CHECK(STORAGE) を指定した場合だけ使用できます。

- ALLOCSIZE
- CHECKSTG

- UNALLOCATED

Enterprise PL/I アプリケーションでは、AMODE(24) は非推奨です。CHECK(STORAGE) オプションを指定してコンパイルされたコードで、AMODE(24) を使用する必要がある場合は、HEAP(,,BELOW) ランタイム・オプションも指定する必要があります。

CMPAT

CMPAT オプションは、ストリング、AREA、配列、または構造体を共用するプログラムに対して OS PL/I バージョン 1、OS PL/I バージョン 2、PL/I for MVS & VM、または Enterprise PL/I for z/OS とのオブジェクト互換性を維持するかどうかを指定します。

注: LP(64) オプションでは、CMPAT オプションは無視されます。実際上は、CMPAT(V3) が常にオンになります。

省略形: CMP

LE CMPAT(LE) を指定すると、ユーザー・プログラムは、VisualAge PL/I for OS/390 または Enterprise PL/I for z/OS を使用してコンパイルし、そのコンパイル時に CMPAT(V1) および CMPAT(V2) オプションを使用しなかったプログラムに限り、ストリング、AREA、配列、または構造体を共用できます。DB2 ストアード・プロシージャーは、CMPAT(LE) を指定してコンパイルしてはいけません。

V1 CMPAT(V1) では、CMPAT(V1) オプションが使用されている間は、OS PL/I コンパイラーでコンパイルされたプログラムや、それ以降の PL/I コンパイラーでコンパイルされたプログラムと、ストリング、AREA、配列、または構造体を共用できます。CMPAT(V1) では、JSON 組み込み関数はサポートされません。

V2 CMPAT(V2) では、CMPAT(V2) オプションが使用されている間は、OS PL/I コンパイラーでコンパイルされたプログラムや、それ以降の PL/I コンパイラーでコンパイルされたプログラムと、ストリング、AREA、配列、または構造体を共用できます。

V3 CMPAT(V3) では、いずれかの CMPAT(V*) オプションが使用されている間は、OS PL/I コンパイラーでコンパイルされたプログラムや、それ以降の PL/I コンパイラーでコンパイルされたプログラムと、ストリングを共用できます。ただし、CMPAT(V3) でコンパイルされなかったコードとは、AREA、配列、構造体を共用できません。

1 つのアプリケーション内のモジュールはすべて、同じ CMPAT オプションを指定してコンパイルする必要があります。

新旧のコードを混合する場合は、次の制限があります。 次の制限については、「*Enterprise PL/I for z/OS* コンパイラーおよびランタイム 移行ガイド」を参照してください。

DFT(DESCLIST) オプションは CMPAT(V*) オプションと対立します。 DFT(DESCLIST) オプションが CMPAT(V*) オプションとともに指定された場合は、メッセージが発行されて DFT(DESCLOCATOR) オプションが想定されます。

CMPAT(V3) では、配列は、8 バイトの整数としてとれるすべての値で宣言できます。ただし、LP(64) オプションを使用しない限りは、配列の合計サイズには今のところまだ、 CMPAT(V2) で宣言された配列と同じ制限があります。

CMPAT(V3) では、以下の組み込み関数は必ず FIXED BIN(63) の結果を返します。

- CURRENTSIZE/CSTG
- DIMENSION
- HBOUND
- LBOUND
- LOCATION
- SIZE/STG

これらの関数は 8 バイトの整数値を返すため、CMPAT(V3) では、LIMITS オプションの FIXEDBIN サブオプションにおける 2 番目のオプションは 63 でなければなりません。

ただし、CMPAT(V3) でも、ステートメントおよびフォーマット・ラベル定数は、4 バイト整数を使用して指定する必要があります。

CODEPAGE

CODEPAGE オプションは、CHARACTER と WIDECHAR の間の変換に使用されるコード・ページを指定します。また、このオプションは、PLISAX 組み込みサブルーチンで使用されるデフォルト・コード・ページを指定します。

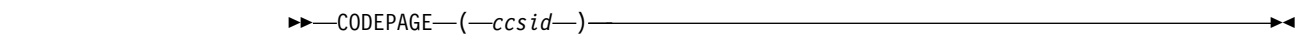


表 *4.* サポート対象の *CCSID*

01047	01145	00273	00297
01140	01146	00277	00500
01141	01147	00278	00871
01142	01148	00280	00819
01143	01149	00284	00813
01144	00037	00285	00920
01025	01155		

デフォルト CCSID 1140 は、CCSID 37 (EBCDIC Latin 1、米国) と同等ですが、ユーロ記号を含んでいます。

COMMON

COMMON オプションは、EXTERNAL STATIC 変数用の CM リンケージ・レコードを生成するようコンパイラーに指示します。

注: LP(64) オプションでは、COMMON オプションは無視されます。

COMMON オプションが指定されると、NORENT オプションが適用される場合に、INITIAL 値を含まない RESERVED 以外の EXTERNAL STATIC 変数に対して CM リンケージ・レコードが生成されます。これは、OS PL/I コンパイラーが行う処理と一致します。

NOCOMMON オプションを指定すると、Enterprise PL/I の初期のリリースではそうであったように、SD レコードが書き込まれます。

COMMON オプションは、RENT オプションと一緒に、または $n > 7$ の場合の LIMITS(EXTNAME(*n*)) と一緒に使用してはなりません。

COMPILE

COMPILE オプションは、プリプロセス中またはセマンティック検査中に指定されている重大度のメッセージが生成された場合に、ソース・プログラムのすべてのセマンティック検査のあとでコンパイラーを停止させます。

コンパイラーが処理を続行するかどうかは、下記のリスト内の NOCOMPILE オプションで指定されたとおりの、検出したエラーの重大度で決まります。
NOCOMPILE オプションを指定すると、セマンティック検査の後、処理は無条件に停止されます。

省略形: C、NC

COMPILE
重大エラーまたは回復不能エラーが検出されない限り、コードを生成します。このサブオプションは NOCOMPILE(S) と同等です。

NOCOMPILE
コンパイルはセマンティック検査後に停止されます。

NOCOMPILE(W)
警告、エラー、重大エラー、または回復不能エラーが検出された場合、コードは生成されません。

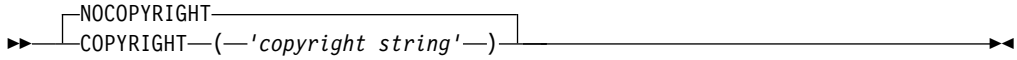
NOCOMPILE(E)
エラー、重大エラー、または回復不能エラーが検出された場合は、コードを生成しません。

NOCOMPILE(S)
重大エラーまたは回復不能エラーが検出された場合は、コードを生成しません。

コンパイルが NOCOMPILE オプションによって終了された場合、相互参照リストおよび属性リストを作成することができます。ソース・プログラムに続く他のリストは作成されません。

COPYRIGHT

COPYRIGHT オプションは、オブジェクト・モジュール内にストリングを配置します。このストリングは、オブジェクトのリンク先であるロード・モジュールとともにメモリーにロードされます。



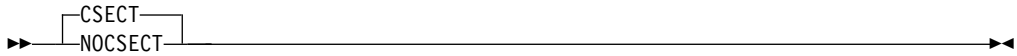
ストリングの長さは 1000 文字に制限されます。ただし、ストリングは、100 文字を超える場合、オプション・リストに表示されません。

ロケールが異なっていてもストリングを読み取ることができるように、インバリアント文字セットからの文字だけを使用する必要があります。

CSECT

CSECT オプションを使用すると、名前付き CSECT がオブジェクト・モジュール (生成された場合) に組み込まれます。

このオプションは、SMP/E によって、ご使用の製品を使用可能にしたり、プログラムのデバッグを支援したりする場合に使用します。



省略形: CSE、NOCSE

NOCSECT オプションを使用すると、ユーザーのオブジェクト・モジュールのコードおよび静的セクションにはデフォルトの名前が指定されます。

CSECT オプションを使用すると、ユーザーのオブジェクト・モジュールのコードおよび静的セクションには、次のように定義された「パッケージ名」によって名前が指定されます。

- パッケージ・ステートメントが使用された場合は、「パッケージ名」はパッケージ・ステートメントの一番左のラベルになります。
- そうでない場合は、最初のプロシージャー・ステートメントにある一番左のラベルが「パッケージ名」になります。

長さ 7 の「変更パッケージ名」は、次のように形成されます。

- パッケージ名が 7 文字より短い場合は、アスタリスク (*) が前に追加されて、7 文字の長さの変更パッケージ名が作成されます。
- パッケージ名が 7 文字を超える場合は、最初の *n* 文字と最後の *7 - n* 文字 (7 から n を減算した値の文字数の末尾文字) からなるパッケージ名に変更されます (*n* は CSECTCUT オプションで設定されます)。
- それ以外の場合は、パッケージ名がそのまま変更後のパッケージ名になります。

コード csect 名は、変更パッケージ名に 1 を追加して作成されます。

静的 csect 名は、変更パッケージ名に 2 を追加して作成されます。

そのため、SAMPLE というパッケージの場合、コード csect 名は *SAMPLE1 となり、静的 csect 名は *SAMPLE2 となります。

CSECTCUT

CSECTCUT オプションは、CSECT オプションの処理のときに、コンパイラーが、どのようにロング・ネームを処理するかを制御します。

CSECTCUT オプションは、CSECT オプションを指定しない限り影響しません。CSECT オプションで使用される「パッケージ名」が 7 文字以下の場合も影響しません。

CSECTCUT オプションの中の値 *n* は、0 および 7 の間でなければいけません。

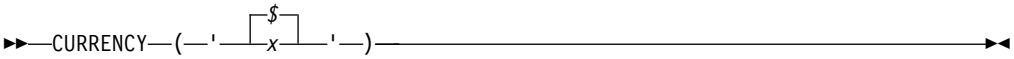
CSECT オプションで使用される「パッケージ名」が 7 文字を超える場合、コンパイラーはその名前を 7 文字に (最初の *n* 文字と最後の *7 - n* 文字を使用することによって) 縮小します。

例えば、BEISPIEL という名前を持つ 1 つのプロシージャーから成るコンパイルについて考えてみてください。

- CSECTCUT(3) では、コンパイラーはその名前を BEIPIEL に縮小します。
- CSECTCUT(4) では、コンパイラーはその名前を BEISIEL に縮小します。

CURRENCY

CURRENCY オプションを指定すると、ドル記号の代わりにピクチャー・ストリングで代替文字を指定できます。



省略形: CURR

x コンパイラーおよびランタイムがピクチャー・ストリング内でドル記号として認識し受け入れる必要がある文字

DBCS

DBCS オプションを指定すると、GRAPHIC オプションが指定されていなくても、リスト (生成された場合) は DBCS の存在を識別します。

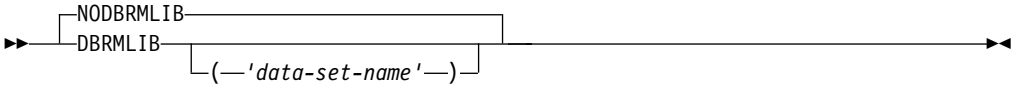
z/OS UNIX システム・サービスでは、NODBCS オプションを指定すると、リストが生成される場合はすべての DBCS シフト・コードが「.」と示されます。

バッチおよび z/OS UNIX システム・サービスではいずれも、NODBCS オプションを指定すると、リスト・ページのヘッダー・テキストに、一致しないシフト・コードは含まれません。

GRAPHIC も指定されている場合は、NODBCS オプションを指定してはなりません。

DBRMLIB

DBRMLIB オプションは、SQL プリプロセッサーが z/OS UNIX システム・サービスで動作しているときにその SQL プリプロセッサーの DBRM データ・セットを作成する場所を指定します。



DBRMLIB オプションがデータ・セット名なしで指定された場合、DBRM データ・セットはソース .pli ファイルと同じ名前 (ただし、拡張子は .dbrm) で現行 UNIX システム・サービス・ディレクトリーに作成されます。例えば、次のコマンドを実行すると、

```
pli -c -qpp=sql -qdbrmlib sample.pli
```

DBRM データ・セットは sample.dbrm で作成されます。

DBRMLIB オプションがデータ・セット名と一緒に指定された場合、そのデータ・セット名は、DB2 で必要となる属性を持つ PDS(E) の名前を指定していなければなりません。 DBRM データ・セットは、ソース .pli ファイルと同じ名前を持つメンバーとしてのデータ・セットで作成されます。例えば、次のコマンドを実行すると、

```
pli -c -qpp=sql -qdbrmlib="USER.TEST.DBRM" sample.pli
```

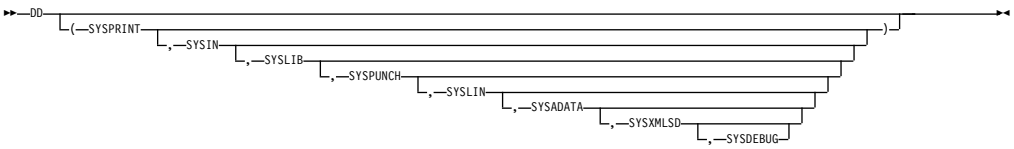
DBRM データ・セットは USER.TEST.DBRM(SAMPLE) で作成されます。

コンパイラーが UNIX システム・サービスのもとで SQL プリプロセッサーを起動する場合、DBRMLIB オプションを指定する必要があります。このオプションは、コンパイラーが z/OS UNIX システム・サービスの下で SQL プリプロセッサーを呼び出す場合にのみ有効です。

デフォルトは NODBRMLIB です。

DD

DD オプションを使用すれば、コンパイラーが使用する各種データ・セットに対して代替 DD 名を指定できます。



DD 名は 8 つまで指定できます。その DD 名は順番に以下のデータ・セットに対して代替 DD 名を指定します。

- SYSPRINT
- SYSIN
- SYSLIB
- SYSPUNCH
- SYSLIN
- SYSADATA
- SYSXMLSD
- SYSDEBUG

1 次コンパイラー・ソース・ファイルの DD 名として ALTIN を使用する場合は、DD(SYSPRINT,ALTIN) を指定する必要があります。 DD(ALTIN) を指定した場合は、SYSIN が 1 次コンパイラー・ソース・ファイルの DD 名として使用され、ALTIN がコンパイラー・リストの DD 名として使用されます。

アスタリスク (*) を使用して、デフォルト DD 名が使用されるように指示することもできます。したがって、DD(*,ALTIN) は DD(SYSPRINT,ALTIN) と同等です。

DDSQL

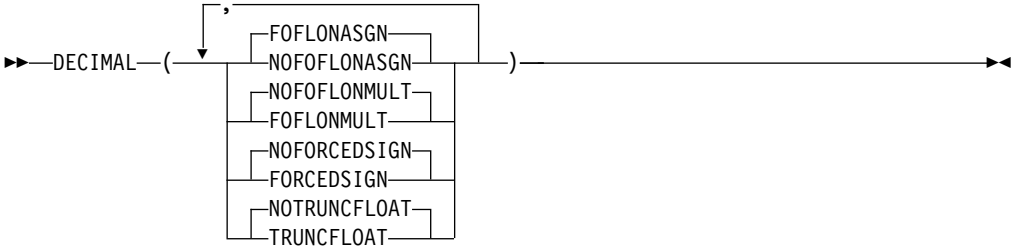
DDSQL オプションを使用すれば、SQL プリプロセッサーが EXEC SQL INCLUDE ステートメントを解決するときに使用するデータ・セットに対して代替 DD 名を指定できます。

DDSQL('') オプションでは、EXEC SQL INCLUDE ステートメントの解決に使用される DD 名は、DD コンパイラー・オプションの SYSLIB の DD 名です。

このオプションは、SQL プリコンパイラーから統合 SQL プリプロセッサーに移行する場合に役に立つ可能性があります。

DECIMAL

DECIMAL オプションは、特定の FIXED DECIMAL 演算および代入をコンパイラーが処理する方法を指定します。



FOFLONASGN

FOFLONASGN オプションが有効になっていて SIZE 条件が無効になっている場合に、FIXED DECIMAL 式が FIXED DECIMAL ターゲットに代入されて有効数字が失われるときは必ず FIXEDOVERFLOW 条件を発生させるコードをコンパイラーが生成することを FOFLONASGN オプションは要求します。

反対に、NOFOFLONASGN オプションを指定すると、コンパイラーは、このような代入で有効数字が失われたときに FIXEDOVERFLOW 条件を発生しないコードを生成します。

例えば、FIXED DEC(5) として宣言された変数 A があるとします。その場合、代入 A = A + 1 で、FOFLONASGN オプションでは FOFL が発生しますが、NOFOFLONASGN オプションで FOFL は発生しません。

ただし、NOFOFLONASGN オプションを指定すると、LIMITS オプションの FIXEDDEC サブオプションで許可されるよりも桁数が多くなる結果を生成する演算によって、FIXEDOVERFLOW 条件が発生する可能性があることに注意してください。 例えば、値 999_999_999_999_999 を持つ FIXED DEC(15) として宣言された変数 B があり、LIMITS の FIXEDDEC サブオプションで最大精度が 15 として指定されているとします。その場合、代入 B = B + 1 では FIXEDOVERFLOW 条件が発生します (FOFL が有効な場合)。

FOFLONMULT

MULTIPLY 組み込み関数が使用されていて、その組み込み関数で生成される FIXED DEC 結果が、MULTIPLY 組み込み関数で指定された精度には大きすぎる場合に必ず FIXEDOVERFLOW 条件を発生させるコードをコンパイラーが生成することを FOFLONMULT オプションは要求します。

逆に、NOFOFLONMULT オプションでは、コンパイラーは、そのような MULTIPLY 組み込み関数が使用されている場合に切り捨てた結果を生成するコードを生成します。

なお、FOFLONMULT オプションを使用すると、デフォルト言語のセマンティクスが変更されます (このため、SIZE 条件が有効にされていない限り、FIXED DEC に適用された MULTIPLY 組み込み関数の大きすぎる結果は切り捨てられます)。

FORCEDSIGN

FORCEDSIGN オプションが指定された場合、コンパイラーは強制的に追加コードを生成して、値ゼロの FIXED DECIMAL 結果が生成されるときは必ず結果の符号ニブルの値が 'C'X となるようにします。このオプションを指定すると、コンパイラーが生成するコードは、NOFORCEDSIGN サブオプションで生成されるコードよりもパフォーマンスが大幅に低下します。

また、このオプションが有効であると、コードを実行するときに、データ例外がより発生しやすくなります。例えば、1 つの FIXED DEC(5) 変数を別の FIXED DEC(5) 変数に代入する場合、通常、コンパイラーは移動を実行するために MVC 命令を生成します。ただし、FORCEDSIGN オプションが有効になっている場合は、結果の符号が望ましいものになるように、コンパイラーは移動を実行する ZAP 命令を生成します。ソースに無効なパック 10 進データが含まれていると、MVC 命令ではなく、ZAP 命令が 10 進データ例外を発生させます。

このオプションでは、ある PICTURE 変数が別の PICTURE 変数に代入される場合も、データ例外が発生することがあります。通常、その際の変換では、FIXED DEC への暗黙的変換が行われるためです (これにより、このオプションでは、無効なデータがソースに含まれている場合にデータ例外を発生させる ZAP 命令が生成されます)。

DECIMAL(NOFORCEDSIGN) を指定すると、ある種の計算では「負のゼロ」が生成されることがあります。ただし、プログラマーが負のゼロの取得に依存するのは、それほど小さい絶対値の値を保持できない FIXED DEC に負のリテラルを割り当てる (例えば、FIXED DEC(5,2) に -.001 を割り当てる) ときだけにしてください。

TRUNCFLOAT

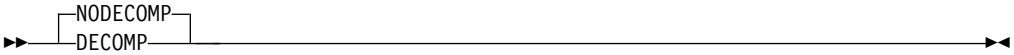
このサブオプションは、切り捨てが行われる可能性があるときに固定小数点への浮動小数点の代入を処理する方法をコンパイラーに指示するものです。

TRUNCFLOAT では、16 進浮動小数点値が FIXED DEC(*p*,*q*) に変換される場合 (*p* <=18 および abs(*q*) <= *p*) およびソース値がターゲットには大きすぎる場合、ソース値は切り捨てられて、オーバーフローは発生しません。

デフォルトは NOTRUNCFLOAT です。

DECOMP

DECOMP オプションは、コンパイルで使用される式の分解を示すリスト・セクションを生成するようにコンパイラーに指示します。



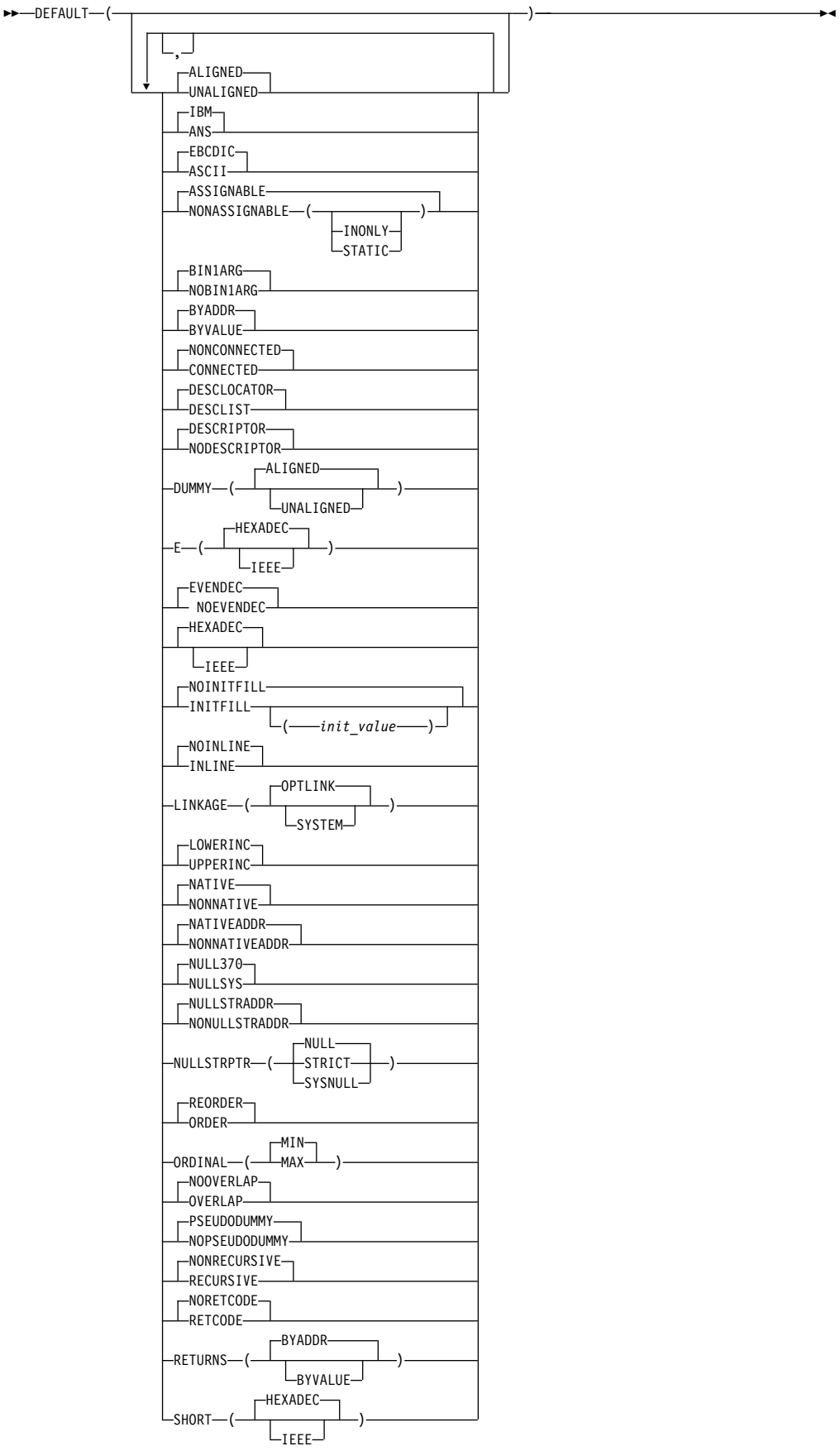
DECOMP オプションが指定された場合、コンパイラーは、ソース・プログラムで使用されているすべての式に関してすべての中間式とその属性を示すリスト・セクションを生成します。

NODECOMP オプションが指定された場合、コンパイラーはこのリスト・セクションを生成しません。

デフォルトは NODECOMP です。

DEFAULT

DEFAULT オプションは、属性およびオプションのデフォルトを指定します。これらのデフォルトは、属性またはオプションがソース・コードで明示的または暗黙に指定されていない場合だけ適用されます。


DEFAULT
(
,
ALIGNED
UNALIGNED
IBM
ANS
EBCDIC
ASCII
ASSIGNABLE
NONASSIGNABLE
(
INONLY
STATIC
)
BIN1ARG
NOBIN1ARG
BYADDR
BYVALUE
NONCONNECTED
CONNECTED
DESCLOCATOR
DESCLIST
DESCRIPTOR
NODESCRIPTOR
DUMMY
(
ALIGNED
UNALIGNED
)
E
(
HEXADEC
IEEE
)
EVENDEC
NOEVENDEC
HEXADEC
IEEE
NOINITFILL
INITFILL
(
init_value
)
NOINLINE
INLINE
LINKAGE
(
OPTLINK
SYSTEM
)
LOWERINC
UPPERINC
NATIVE
NONNATIVE
NATIVEADDR
NONNATIVEADDR
NULL370
NULLSYS
NULLSTRADDR
NONULLSTRADDR
NULLSTRPTR
(
NULL
STRICT
SYSNULL
)
REORDER
ORDER
ORDINAL
(
MIN
MAX
)
NOOVERLAP
OVERLAP
PSEUDODUMMY
NOPSEUDODUMMY
NONRECURSIVE
RECURSIVE
NORETCODE
RETCODE
RETURNS
(
BYADDR
BYVALUE
)
SHORT
(
HEXADEC
IEEE
)

省略形: DFT、ASGN、NONASGN、NONCONN、CONN、INL、NOINL

ALIGNED | UNALIGNED

このサブオプションは、ご使用のすべての変数に対してバイト位置合わせを強制します。

ALIGNED を指定すると、明示的に (おそらく親構造体で) または DEFAULT ステートメントにより暗黙に UNALIGNED 属性が指定されていない限り、文字、ビット、グラフィック、およびピクチャーを除くすべての変数に ALIGNED 属性が与えられます。

UNALIGNED を指定すると、明示的に (おそらく親構造体で) または DEFAULT ステートメントにより暗黙に ALIGNED 属性が指定されていない限り、すべての変数に UNALIGNED 属性が与えられます。

ALIGNED がデフォルトです。

IBM | ANS

このサブオプションは、IBM のデフォルトを使用するのか ANS SYSTEM のデフォルトを使用するのかを指定します。 IBM および ANS の演算デフォルトを次の表に示します。

属性	**DEFAULT(IBM)**	**DEFAULT(ANS)**
FIXED DECIMAL	(5,0)	(10,0)
FIXED BINARY	(15,0)	(31,0)
FLOAT DECIMAL	(6)	(6)
FLOAT BINARY	(21)	(21)

IBM サブオプションでは、名前が I から N までの文字で始まる変数のデフォルトは FIXED BINARY であり、それ以外の変数のデフォルトは FLOAT DECIMAL です。 ANS サブオプションを選択した場合は、すべての変数のデフォルトは FIXED BINARY です。

IBM がデフォルトです。

ASCII | EBCDIC

このサブオプションは、問題プログラム文字データの内部表現に使用される文字セットのデフォルトを設定します。

ASCII は、ASCII 文字セット照合順序に依存するプログラムをコンパイルするときにだけ使用します。 例えば、プログラムが数字のソート・シーケンスまたは小文字および大文字のアルファベット順を使用している場合に、このような依存関係が存在します。 また、高位ビットの状態を変えて大文字の英字を作成するプログラムにも、このような依存関係が存在します。

注: コンパイラーは、A および E を文字ストリングの接尾部としてサポートします。 A 接尾部は、EBCDIC コンパイラー・オプションが効力を持っている場合でも、ストリングが ASCII データを表現していることを示します。 同様に、E 接尾部は、DEFAULT(ASCII) を選択した場合でもストリングが EBCDIC であることを示します。

```
'123'A is the same as '313233'X
'123'E is the same as 'F1F2F3'X
```

EBCDIC がデフォルトです。

ASSIGNABLE | NONASSIGNABLE

このオプションを指定すると、コンパイラーは、ASSIGNABLE 属性または NONASSIGNABLE 属性を持つものとして宣言されていないすべての静的変数に、指定の属性を適用します。 コンパイラーは、NONASSIGNABLE 変数が割り当てのターゲットであるステートメントにフラグを付けます。

ASSIGNABLE がデフォルトです。

INONLY

NONASSIGNABLE(INONLY) が指定されると、INONLY 属性で宣言されたパラメーターに NONASSIGNABLE 属性が付与されます。

STATIC

NONASSIGNABLE(STATIC) が指定されると、STATIC 変数に NONASSIGNABLE 属性が付与されます。

サブオプション INONLY および STATIC は、属性 ASSIGNABLE または NONASSIGNABLE を持つ変数に対して効果がありませんし、親から属性 ASSIGNABLE または NONASSIGNABLE を継承する構造体メンバーに対しても効果がありません。

BYVALUE パラメーターに INONLY 属性が付与されるのは (NON)ASSIGNABLE 属性が解決された後です。そのため、NONASSIGNABLE(INONLY) サブオプションは BYVALUE パラメーターに対して効果がありません。

STATIC 変数と INONLY 変数の両方に NONASSIGNABLE 属性を指定するには、サブオプション NONASSIGNABLE(STATIC INONLY) を指定する必要があります。

NONASSIGNABLE 属性はサブオプションなしで指定できます。その場合は、NONASSIGNABLE(STATIC) を指定したことになります。

BIN1ARG | NOBIN1ARG

このサブオプションは、プロトタイプ化されていない関数に渡される 1 バイトの REAL FIXED BIN 引数をコンパイラーが処理する方法を制御します。

BIN1ARG が指定された場合、コンパイラーは、プロトタイプ化されていない関数に FIXED BIN 引数をそのまま渡します。

ただし、NOBIN1ARG の場合、コンパイラーは、プロトタイプ化されていない関数に渡された 1 バイトの REAL FIXED BIN 引数を 2 バイトの一時的な FIXED BIN 引数に代入して、代わりにその一時的な引数を渡します。

次の例を考えてください。

```
dcl f1 ext entry;
dcl f2 ext entry( fixed bin(15) );

call f1( 1b );
call f2( 1b );
```

DEFAULT(BIN1ARG) を指定した場合、コンパイラーは 1 バイトの FIXED BIN(1) 引数のアドレスをルーチン `f1` に渡し、2 バイトの FIXED BIN(15) 引数のアドレスをルーチン `f2` に渡します。ただし、DEFAULT(NOBIN1ARG) を指定した場合、コンパイラーは 2 バイトの FIXED BIN(15) 引数のアドレスを両方のルーチンに渡します。

ルーチン f1 が COBOL ルーチンの場合、COBOL では 1 バイトの整数をサポートしていないため、このルーチンに 1 バイトの整数の引数を渡すと問題が発生する可能性があることに注意してください。この場合、DEFAULT(NOBIN1ARG) の使用が有用となることもありますが、エントリー宣言ステートメントで引数属性を指定する方がより良いと考えられます。

BIN1ARG がデフォルトです。

BYADDR | BYVALUE

このサブオプションは、引数やパラメーターを参照で渡すのか値で渡すのかについてのデフォルトを設定します。BYVALUE は、いくつかの特定の引数およびパラメーターだけに適用されます。 詳しくは、「*PL/I* 言語解説書」を参照してください。

BYADDR はデフォルトです。

CONNECTED | NONCONNECTED

このサブオプションは、パラメーターの接続または非接続に関するデフォルトを設定します。CONNECTED を指定すると、パラメーターをレコード単位の入出力のターゲットまたはソースとして、あるいはストリング・オーバーレイ定義の基礎として使用できます。

NONCONNECTED がデフォルトです。

DESCLIST | DESCLOCATOR

DEFAULT(DESCLIST) を指定すると、コンパイラーはリストの中のすべての記述子を「隠れた」最後のパラメーターとして渡します。

DEFAULT(DESCLOCATOR) を指定すると、以前のリリースの PL/I の場合と同様に、記述子を必要とするパラメーターが記述子またはロケーターを使用して渡されます。 これにより、古いコードは、あるルーチンから、ポインターを受け取ることを想定しているルーチンに構造体を渡す場合でも、引き続き機能できます。

DFT(DESCLIST) オプションは、CMPAT(V*) オプションと対立します。DFT(DESCLIST) がいずれかの CMPAT(V*) オプションと一緒に指定されると、メッセージが発行されて DFT(DESCLOCATOR) オプションが想定されます。

DESCLOCATOR がデフォルトです。

DESCRIPTOR | NODESCRIPTOR

PROCEDURE ステートメントで DESCRIPTOR を使用すると、記述子リストが渡されたことを示します。ENTRY ステートメントで DESCRIPTOR を使用すると、記述子リストを渡す必要があることを示します。 NODESCRIPTOR を使用すると、より効率的なコードになりますが、次の制限があります。

- PROCEDURE ステートメントに関して、いずれかのパラメーターに次の項目が含まれている場合、NODESCRIPTOR は無効です。
  - 配列の境界、ストリングの長さ、または配列のサイズ (NONASSIGNABLE 属性を持つ VARYING ストリングまたは VARYINGZ ストリングの場合を除く) に指定されたアスタリスク (*)
  - NONCONNECTED 属性
  - UNALIGNED BIT 属性

- ENTRY 宣言では、ENTRY 記述リストの中で配列の境界、ストリングの長さ、またはエリアのサイズにアスタリスク (*) が指定されている場合は、NODESCRIPTOR は無効です。

DESCRIPTOR がデフォルトです。

DUMMY(ALIGNED | UNALIGNED)

このサブオプションは、仮引数が作成される状態の数を減らします。

DUMMY(ALIGNED) は、引数が位置合わせにおいてのみパラメーターと相違している場合にも、仮引数を作成するべきであることを示します。
DUMMY(UNALIGNED) は、スカラー (不変ビットを除く) またはスカラーの配列が位置合わせにおいてのみパラメーターと相違している場合に、そのスカラーまたはスカラーの配列には仮引数を作成してはならないことを示します。

次の例を考えてみてください。

```
dcl
  1 a1 unaligned,
    2 b1   fixed bin(31),
    2 b2   fixed bin(15),
    2 b3   fixed bin(31),
    2 b4   fixed bin(15);

dcl x entry( fixed bin(31) );

call x( b3 );
```

DEFAULT(DUMMY(ALIGNED)) を指定すると仮引数が作成されますが、DEFAULT(DUMMY(UNALIGNED)) を指定すると仮引数は作成されません。

DUMMY(ALIGNED) がデフォルトです。

E (HEXADEC | IEEE)

E サブオプションは、E フォーマット項目の指数として使用する数字の桁数を指定します。

E(IEEE) を指定すると、E フォーマット項目の指数として 4 桁の数字が使用されます。

E(HEXADEC) を指定すると、E フォーマット項目の指数として 2 桁の数字が使用されます。

DFT(E(HEXADEC)) を指定した場合に、99 より大きい絶対値を持つ指数が含まれる式を使用しようとすると、SIZE 条件が発生します。

コンパイラー・オプション DFT(IEEE) が有効になっている場合、通常はオプション DFT(E(IEEE)) も使用する必要があります。ただし、このオプションを指定すると、DFT(E(HEXADEC)) では有効になる E フォーマット項目のいくつかが無効になります。例えば、DFT(E(IEEE)) を指定すると、E フォーマット項目が無効であるため、ステートメント `put skip edit(x) (e(15,8));` にフラグが立てられます。

E(HEXADEC) がデフォルトです。

EVENDEC | NOEVENDEC

このサブオプションは、偶数精度を指定して宣言された固定小数点変数に関するコンパイラーの許容度を制御します。

NOEVENDEC のもとでは、固定小数点変数の精度は次の最大の奇数に切り上げられます。

EVENDEC を指定して FIXED DEC(2) 変数に 123 を割り当てると、SIZE 条件が発生します。 NOEVENDEC を指定した場合は、SIZE 条件は発生しません。

EVENDEC がデフォルトです。

HEXADEC | IEEE

このサブオプションは、すべての FLOAT 変数およびすべての浮動小数点中間結果を保持するために使用されるデフォルト表現を指定します。また、このサブオプションは、コンパイラーが浮動小数点の式を評価するときに、16 進の命令と数学ルーチンを使用するか、IEEE 浮動小数点の命令と数学ルーチンを使用するかを決定します。

JAVA とやり取りするプログラムに加え、浮動小数点データのデフォルト表現として IEEE を使用するプラットフォームとデータの受け渡しをするプログラムにも、IEEE オプションを使用することをお勧めします。

HEXADEC がデフォルトです。

INITFILL | NOINITFILL

このサブオプションは、自動変数のデフォルト初期化を制御します。

16 進値 (nn) を使用して INITFILL を指定した場合は、ブロックに入るたびに、ブロック内のすべての自動変数によって使用されるストレージを初期化するために、その値が使用されます。 16 進値を入力しない場合、デフォルトは '00' です。

16 進値は引用符を付けても付けなくても指定できますが、引用符を付けて 16 進値を指定する場合は、そのストリングの末尾に X を付けてはならないことに注意してください。

NOINITFILL を指定した場合は、変数が明示的に初期化されない限り、自動変数によって使用されるストレージには任意のビット・パターンを保持できます。

INITFILL は、プログラムの実行速度を低下させる可能性があるため、実動プログラムでは指定しないでください。ただし、INITFILL オプションの生成するコードは、LE STORAGE オプションよりも高速に実行されます。 また、このオプションは、プログラム開発時に、未初期化自動変数を検出するために役立ちます。 DFT(INITFILL('00')) および DFT(INITFILL('ff')) を指定してプログラムが正しく実行されれば、未初期化自動変数は存在しないと考えられます。

NOINITFILL がデフォルトです。

INLINE | NOINLINE

このオプションは、インライン・プロシージャー・オプションのデフォルトを設定します。

INLINE を指定すると、コードの実行が高速になりますが、場合によっては実行可能ファイルも大きくなります。 インライン化によるパフォーマンスの改善方法の詳細については、 373 ページの『第 15 章 パフォーマンスの向上』を参照してください。

NOINLINE がデフォルトです。

LINKAGE

プロシージャー呼び出しのためのリンケージ規則は次のとおりです。

OPTLINK

Enterprise PL/I のデフォルトのリンケージ規則。 このリンケージにより最良のパフォーマンスが得られます。

SYSTEM

システム API の標準リンケージ規則。

LINKAGE(OPTLINK) は、JAVA によって呼び出されるすべてのルーチンや、JAVA に対する呼び出しを行うすべてのルーチンに使用します。また、LINKAGE(OPTLINK) は、C によって呼び出されるすべてのルーチンや、C に対して呼び出しを行うすべてのルーチンにも使用します (C コードがデフォルト以外のリンケージを使用してコンパイルされた場合を除く)。

LINKAGE(SYSTEM) は、最後 (および最後のみ) のパラメーターのアドレスにおいて高位ビットがオンになっていることを予期するすべての非 PL/I ルーチンに使用します。

PROCEDURE または ENTRY に OPTIONS(ASSEMBLER) を指定すると、このオプションの設定値に関係なく LINKAGE(SYSTEM) が強制されるので、注意してください。

LINKAGE(OPTLINK) がデフォルトです。

注: LINKAGE サブオプションは、LP(64) オプションでは無視されます。

LOWERINC | UPPERINC

LOWERINC を指定した場合は、コンパイラーは INCLUDE ファイルの実際のファイル名が小文字であることを要求します。 UPPERINC を指定した場合は、コンパイラーはこの名前が大文字であることを要求します。

注: このサブオプションは、z/OS UNIX 環境でのコンパイルにだけ適用されます。

z/OS UNIX では、組み込み名が作成されると拡張子 `.inc` が付けられます。例えば、DFT(LOWERINC) オプションでは、ステートメント `%INCLUDE STANDARD;` が指定されると、コンパイラーは `standard.inc` を組み込もうとします。しかし、DFT(UPPERINC) オプションでは、ステートメント `%INCLUDE STANDARD;` が指定されると、コンパイラーは `STANDARD.INC` を組み込もうとします。

LOWERINC はデフォルト値です。

NATIVE | NONNATIVE

このサブオプションは、固定 2 進数、序数、オフセット、エリア、および可変ストリング・データの内部表現だけに影響します。NONNATIVE サブオプションが有効な場合、NONNATIVE 属性は、 NATIVE 属性を指定して宣言されていないこの種のすべての変数に適用されます。

NONNATIVE は、非ネイティブ・フォーマットに依存してこの種の変数を保持するプログラムをコンパイルする場合だけ指定してください。

ご使用のプログラムで固定 2 進変数がポインター変数またはオフセット変数に基づいている (あるいは、それとは反対にポインター変数またはオフセット変数が固定 2 進変数に基づいている) 場合は、次のサブオプションの組み合わせをどちらか指定します。

- NATIVE サブオプションと NATIVEADDR サブオプションの両方
- NONNATIVE サブオプションと NONNATIVEADDR サブオプションの両方

それ以外の組み合わせを指定すると、結果は予測できません。

NATIVE がデフォルトです。

NATIVEADDR | NONNATIVEADDR

このサブオプションが影響するのはポインターの内部表現のみです。NONNATIVEADDR サブオプションが有効な場合、NONNATIVE 属性は、NATIVE 属性を指定して宣言されていないすべてのポインター変数に適用されます。

ご使用のプログラムで固定 2 進変数がポインター変数またはオフセット変数に基づいている (あるいは、それとは反対にポインター変数またはオフセット変数が固定 2 進変数に基づいている) 場合は、次のサブオプションの組み合わせをどちらか指定します。

- NATIVE サブオプションと NATIVEADDR サブオプションの両方
- NONNATIVE サブオプションと NONNATIVEADDR サブオプションの両方

それ以外の組み合わせを指定すると、結果は予測できません。

NATIVEADDR がデフォルトです。

NULLSYS | NULL370

このサブオプションは、NULL 組み込み関数によって戻される値を決定します。 NULLSYS を指定すると、binvalue(null()) は 0 に等しくなります。
| NULL370 を指定すると、binvalue(null()) は LP(32) では 'FF_00_00_00'xn と
| 等しくなり、LP(64) では '00000000_7FFFFBAD'xn と等しくなります。

NULL370 がデフォルトです。

NULLSTRADDR | NONULLSTRADDR

このサブオプションは、引数として渡されたヌル・ストリングをコンパイラーがどのように処理するのかを制御します。

NULLSTRADDR では、ヌル・ストリングが入り口呼び出しで引数として指定されていると、コンパイラーは自動ストレージの初期化部分のアドレスを渡します。 これは、OS PL/I および PL/I for MVS コンパイラーの動作と互換性があります。

ただし、NONULLSTRADDR を指定した場合は、入り口の呼び出し時に引数としてヌル・ストリングが指定されると、コンパイラーによってヌル・ポインターが引数のアドレスとして渡されます。これは、Enterprise PL/I コンパイラーの早期リリースの動作と互換性があります。

NULLSTRADDR がデフォルトです。

NULLSTRPTR

このサブオプションは、POINTER に割り当てられたヌル・ストリングをコンパイラーがどのように処理するのかを制御します。

NULLSTRPTR(SYSNULL) を指定すると、POINTER に '' を割り当てた結果はポインターに SYSNULL() を割り当てたときと同じになります。

NULLSTRPTR(NULL) を指定すると、POINTER に '' を割り当てた結果はポインターに NULL() を割り当てたときと同じになります。

NULLSTRPTR(STRICT) を指定すると、POINTER に '' を割り当てたり比較したりすると、無効のフラグが立てられます。

NULLSTRPTR(NULL) がデフォルトです。

ORDER | REORDER

このサブオプションは、オブジェクト・コードの最適化に影響します。REORDER を指定すると、コードをより最適化できます。詳しくは、373 ページの『第 15 章 パフォーマンスの向上』を参照してください。

REORDER がデフォルトです。

ORDINAL(MIN | MAX)

ORDINAL(MAX) を指定すると、定義に PRECISION 属性が含まれていないすべての順序数に、属性 PREC(31) が与えられます。 これを指定しない場合は、これらの順序数には、その値の範囲をカバーする最小の精度が与えられます。

ORDINAL(MIN) がデフォルトです。

OVERLAP | NOOVERLAP

OVERLAP を指定すると、コンパイラーは、割り当てでソースとターゲットのオーバーラップが起こることがあると想定し、割り当ての結果が正しくなるように、必要に応じて追加のコードを生成します。

OVERLAP サブオプションは、ストリング変数のみに適用されます。これは FIXED DECIMAL またはその他の変数タイプの割り当てには効果がありません。これらの割り当てでは、ソースとターゲットはオーバーラップしてはいけません。

NOOVERLAP を使用すると、パフォーマンスの優れたコードが生成されます。ただし、NOOVERLAP を使用する場合は、ソースとターゲットがオーバーラップしないようにする必要があります。

NOOVERLAP がデフォルトです。

PSEUDODUMMY | NOPSEUDODUMMY

このサブオプションは、プロトタイプ化されていない関数に対する引数として SUBSTR 参照が指定されたときに仮引数を作成するのかどうかを決定します。

PSEUDODUMMY を指定すると、プロトタイプ化されていない関数に対する引数として SUBSTR 参照が指定されたときに仮引数が作成されます。

NOPSEUDODUMMY を指定すると、プロトタイプ化されていない関数に対する引数として SUBSTR 参照が指定されたときに仮引数は作成されません。

PSEUDODUMMY がデフォルトです。

RECURSIVE | NONRECURSIVE

DEFAULT(RECURSIVE) を指定すると、コンパイラーはすべてのプロシージャ

ーに RECURSIVE 属性を適用します。 DEFAULT(NONRECURSIVE) を指定すると、RECURSIVE 属性を持つプロシージャー以外のすべてのプロシージャーが非再帰的になります。

NONRECURSIVE がデフォルトです。

RETCODE | NORETCODE
RETCODE が指定されると、コンパイラーは追加のコードを生成して、RETURNS 属性を持たない外部プロシージャーが PLIRETV 組み込み関数を呼び出すことにより取得した整数値を返してから、プログラムがそのプロシージャーから戻るようにします。

NORETCODE を指定すると、RETURNS 属性を持たないプロシージャーに対して特別なコードは生成されません。

NORETCODE がデフォルトです。

RETURNS (BYVALUE | BYADDR)
このサブオプションは、関数が値を返す方法に関するデフォルトを設定します。詳しくは、「*PL/I* 言語解説書」を参照してください。

アプリケーションに ENTRY ステートメントが含まれていて、その ENTRY ステートメントまたはそのステートメントを含むプロシージャー・ステートメントに RETURNS オプションが指定されている場合は、RETURNS(BYADDR) を指定してください。また、それらのエントリーのエントリー宣言でも、RETURNS(BYADDR) を指定する必要があります。

RETURNS(BYADDR) がデフォルトです。

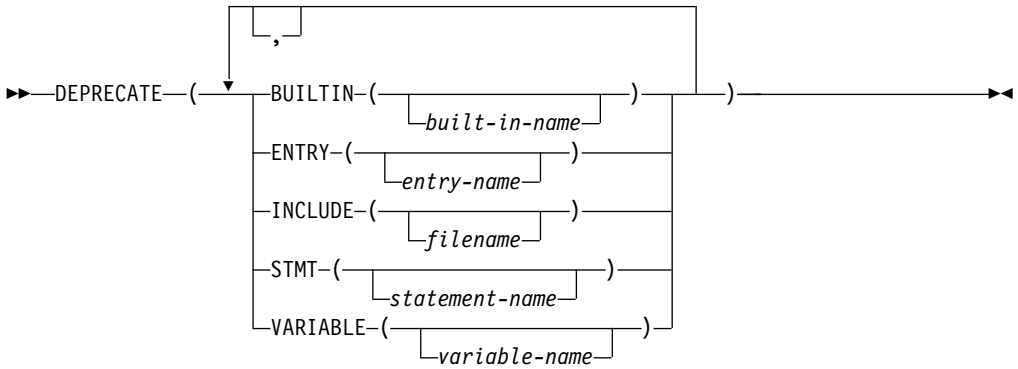
SHORT (HEXADEC | IEEE)
このサブオプションは、IBM 以外の UNIX コンパイラーとの互換性を向上させます。 SHORT (HEXADEC) は、*p* <= 21 の場合に FLOAT BIN (*p*) を短い (4 バイトの) 浮動小数点数にマップします。 SHORT (IEEE) は、*p* <= 24 の場合に FLOAT BIN (*p*) を短い (4 バイトの) 浮動小数点数にマップします。

SHORT (HEXADEC) がデフォルトです。

デフォルト : DEFAULT(ALIGNED IBM EBCDIC ASSIGNABLE BIN1ARG BYADDR NONCONNECTED DESCLOCATOR DESCRIPTOR DUMMY(ALIGNED) E(HEXADEC) EVENDEC HEXADEC NOINITFILL NOINLINE LINKAGE(OPTLINK) LOWERINC NATIVE NATIVEADDR NULL370 NULLSTRPTR(NULL) NULLSTRADDR REORDER ORDINAL(MIN) NOOVERLAP PSEUDODUMMY NONRECURSIVE NORETCODE RETURNS(BYADDR) SHORT(HEXADEC))

DEPRECATE

このオプションは、非推奨の変数名、組み込みファイル名、およびステートメント名にフラグを立てて、エラー・メッセージを出力します。



BUILTIN
BUILTIN 属性を持つすべての *built-in-name* 宣言にフラグを立てます。

built-in-name
BUILTIN 変数の名前

ENTRY
ENTRY 属性を持つすべての *entry-name* 宣言にフラグを立てます。

entry-name
レベル 1 名

INCLUDE
filename を含むすべての %INCLUDE ステートメントにフラグを立てます。

filename
ファイルの名前

STMT
名前が *statement-name* であるすべてのステートメントにフラグを立てます。

statement-name
ステートメントの名前

この名前は、PL/I ステートメントの最初のキーワードによって識別されます。 STMT オプションは以下のキーワードを受け入れます。

allocate	assert	attach	begin	call	close	delay	delete
detach	display	exit	fetch	flush	free	get	goto
iterate	leave	locate	on	open	put	read	release
resignal	revert	rewrite	signal	stop	wait	write	

VARIABLE
BUILTIN 属性も ENTRY 属性も持たない *variable name* のすべての宣言にフラグを立てます。

variable-name
レベル 1 名

DEPRECATE オプションを指定するには、空のサブオプション・リストを使用してこれらのサブオプションを 1 つ以上指定する必要があります。 例えば、次の 2 つの指定はどちらも無効です。

- DEPRECATE

- DEPRECATE(BUILTIN)

サブオプションの 1 つを指定しても、以前に指定された他のどのサブオプションの設定も変更されません。

サブオプションを複数回指定すると、前の指定が置き換えられます。

いずれの場合も、サブオプション・リストの検査は行われません。

デフォルト: DEPRECATE(BUILTIN() ENTRY() INCLUDE() STMT() VARIABLE())

例

- 以下の指定は同等です。

```
DEPRECATE(ENTRY(old)) DEPRECATE(BUILTIN(acos))
DEPRECATE(ENTRY(old) BUILTIN(acos))
```

- 次の例では、最初の指定の x は y に置き換えられます。

```
DEPRECATE(BUILTIN(x)) DEPRECATE(BUILTIN(y))
DEPRECATE(BUILTIN(y))
```

DEPRECATENEXT

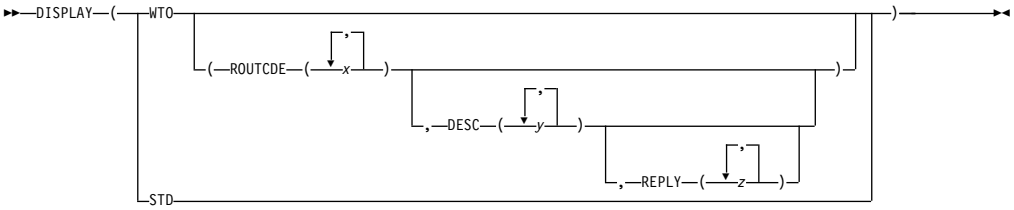
DEPRECATENEXT オプションの目的および使用方法は DEPRECATE オプションと同じです。ただし、将来の開発フェーズで非推奨とされる項目に対してコンパイラーが発行するメッセージは、エラー・メッセージではなく警告メッセージです。

デフォルト: DEPRECATENEXT(BUILTIN() ENTRY() INCLUDE() STMT() VARIABLE())

詳しくは、 34 ページの『DEPRECATE』を参照してください。

DISPLAY

DISPLAY オプションは、DISPLAY ステートメントが入出力を実行する方法を決定します。



STD

DISPLAY ステートメントは、すべてテキストを stdout に書き出し、REPLY テキストを stdin から読み込んで完了します。

WTO

REPLY が指定されない DISPLAY ステートメントはすべて WTO を使用して完了し、REPLY が指定された DISPLAY ステートメントはすべて WTOR を使用して完了します。これはデフォルトです。

次のサブオプションがサポートされています。

ROUTCDE
WTO で ROUTCDE として使用される 1 つ以上の値を指定します。 デフォルトの ROUTCDE は 2 です。

DESC
WTO で DESC として使用される 1 つ以上の値を指定します。デフォルトの DESC は 3 です。

REPLY
WTOR で DESC として使用される 1 つ以上の値を指定します。省略した場合は、DESC オプション (またはデフォルト) の値が使用されます。

ROUTCDE、DESC、および REPLY に指定される値はすべて 1 から 16 までの間の値でなければなりません。

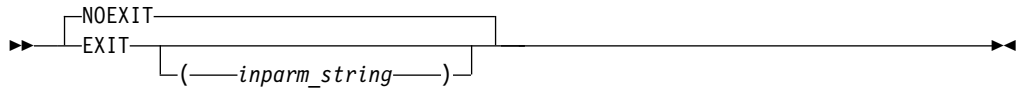
DLLINIT

DLLINIT オプションは、MAIN でないすべての外部プロシージャーに OPTIONS(FETCHABLE) を適用します。 このオプションは、1 つの外部プロシージャーを含むコンパイル単位でのみ使用します。その際、そのプロシージャーを DLL としてリンクする必要があります。

NODLLINIT はユーザーのプログラムには影響しません。

EXIT

EXIT オプションにより、コンパイラー・ユーザー出口を呼び出すことができます。

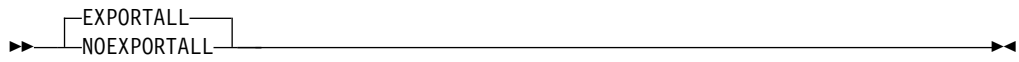


inparm_string
初期化中にコンパイラー・ユーザー出口ルーチンに渡されるストリング。 ストリングの長さは最大 31 文字です。

このオプションの使用方法について詳しくは、 541 ページの『第 24 章 ユーザー出口の用法』を参照してください。

EXPORTALL

EXPORTALL オプションは、外部で定義されたすべてのプロシージャーおよび変数を DLL アプリケーションが使用できるようにエクスポートするかどうかを制御します。



EXPORTALL では、外部で定義されたすべてのプロシージャーおよび変数 (OPTION(DLLINTERNAL) を指定するものを除く) がエクスポートされます。

NOEXPORTALL では、外部で定義されたプロシージャーおよび変数 (OPTION(DLLEXTERNAL) や OPTIONS(FETCHABLE) を指定するものを除く) はエクスポートされません。

単純にフェッチされるモジュール (DLL 関数を使用してアクセスされるモジュールではない) を DLL としてリンクする場合は、NOEXPORTALL オプションを使用する必要があります。

デフォルトは EXPORTALL ですが、NOEXPORTALL の方がパフォーマンスに優れています。

EXTRN

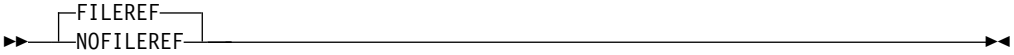
EXTRN オプションは、外部入り口定数の EXTRN を発行する時期を制御します。

FULL
宣言されたすべての外部入り口定数に対して EXTRN を発行します。これはデフォルトです。

SHORT
参照されている定数に対してのみ EXTRN を発行します。

FILEREF

(NO)FILEREF オプションは、コンパイラーがファイル参照テーブルを生成するのかどうかを制御します。 NOFILEREF オプションが指定された場合に、コンパイラーが指定の FLAG 設定に対してメッセージを発行しないと、ファイル参照テーブルはリストから除去されます。



デフォルトは FILEREF です。

FLAG

FLAG オプションは、コンパイラー・リストにメッセージをリストすることが必要になるエラーの最小重大度を指定します。

省略形: F

I すべてのメッセージをリストします。

W 通知メッセージを除くすべてのメッセージをリストします。

E 警告メッセージと通知メッセージを除くすべてのメッセージをリストします。

S 重大エラー・メッセージおよび回復不能エラー・メッセージだけをリストします。

指定した重大度を下回っているメッセージ、またはコンパイラー出口ルーチンによりフィルターに掛けられて取り除かれたメッセージは、リストされません。

FLOAT

FLOAT オプションは、追加の浮動小数点レジスターの使用、および 10 進浮動小数点をサポートするかどうかを制御します。

DFP
DFP 機能が使用されます。「*z/Architecture* 解説書」資料の説明にあるとおり、DECIMAL FLOAT データはすべて DFP フォーマットで保持されます。DECIMAL FLOAT を使用する操作は、その資料に記載されている DFP ハードウェア命令を使用して実行されます。

NODFP
DFP 機能は使用されません。

FLOAT(DFP) では以下の事項が適用されます。

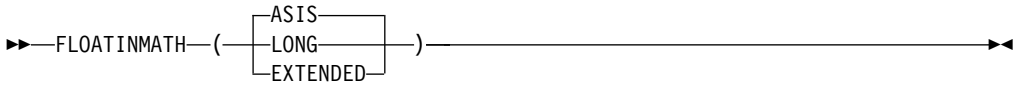
- 拡張 DECIMAL FLOAT の最大精度は 34 (16 進浮動小数点と同様に 33 ではない) になります。
- 短精度 DECIMAL FLOAT の最大精度は 7 (16 進浮動小数点と同様に 6 ではない) になります。
- 次の組み込み関数での DECIMAL FLOAT の値は、すべて適切に変更されます。
  - EPSILON
  - HUGE
  - MAXEXP
  - MINEXP
  - PLACES
  - RADIX
  - TINY
- 次の組み込み関数はすべて DECIMAL FLOAT の適切な値を返します (また、人間がはるかに容易に理解できる値を返します。例えば、SUCC(1D0) は 1.000_000_000_000_001 になり、ROUND 関数は小数点以下の桁で丸めます)。
  - EXPONENT
  - PRED
  - ROUND

  - SCALE
  - SUCC
- 10 進浮動小数点リテラルは、ゼロ以外の数字が小数点の後に続かないように、必要に応じて「右側ユニット表示」に変換された場合、つまり、指数が調整された場合 (例えば、3.1415E0 を 31415E-4 として表示する場合に行われるように)、当該リテラルの精度に対する正常な数値の範囲内にある指数を持つ必要があります。この範囲は、MINEXP-1 および MAXEXP-1 の値によって定められます。 特に、次が保持される必要があります。
  - 短精度の浮動小数点の場合、 -95 <= 指数 <= 90
  - 長精度の浮動小数点の場合、 -383 <= 指数 <= 369
  - 拡張精度の浮動小数点の場合、-6143 <= 指数 <= 6111
- DECIMAL FLOAT が CHARACTER に変換される場合、ストリングは、指数に 4 桁を保持します (対して 16 進浮動小数点の場合は 2 桁使用されます)。
- IEEE および HEXADEC 属性は、FLOAT BIN に適用される場合のみ受け入れられ、DEFAULT(IEEE/HEXADEC) オプションは FLOAT BIN にのみ適用されます。
- 数学組み込み関数 (ACOS、COS、SQRT など) は、DECIMAL FLOAT 引数を受け入れ、対応する言語環境プログラム機能を使用してこの引数を評価します。DECIMAL FLOAT 指数も同様に処理されます。
- 演算において一方のオペランドが FLOAT DECIMAL であり他方が 2 進 (つまり、FIXED BINARY、FLOAT BINARY、または BIT) である場合に DFP を使用するのであれば、その演算で行われる変換に留意する必要があります。そうした演算では、PL/I 言語の規則により、FLOAT DECIMAL オペランドは FLOAT BINARY に変換され、その変換にはライブラリー呼び出しが必要となります。 例えば、*A = A + B;* という形式の代入があるとします。 *A* は FLOAT DECIMAL であり、*B* は FIXED BINARY です。この場合は、3 つの変換が行われ、そのうち 2 つはライブラリー呼び出しとなります。
  1. *A* は、ライブラリー呼び出しにより、FLOAT DECIMAL から FLOAT BINARY に変換されます。
  2. *B* は、インライン・コードにより、FIXED BINARY から FLOAT BINARY に変換されます。
  3. *A + B* の和は、ライブラリー呼び出しにより、FLOAT BINARY から FLOAT DECIMAL に変換されます。

  DECIMAL 組み込み関数を使用することが有効であると考えられる状況: ステートメントが *A = A + DEC(B);* に変更された場合は、ライブラリー呼び出しが除去されます。また、ライブラリー呼び出しは、*B* を FLOAT DECIMAL 一時変数に代入してから、その一時変数を *A* に追加しても、除去できます。
- 組み込み関数 SQRTF は DECIMAL FLOAT 引数に対してはサポートされません (マップ先にすることのできるハードウェア命令がないため)。
- DFP は、CAST タイプ付き関数でサポートされません。

FLOATINMATH

FLOATINMATH オプションは、数学組み込み関数を呼び出すときに、コンパイラーが使用する精度を指定します。



ASIS
数学組み込み関数に対する引数が、long 型または拡張型浮動小数点精度を持つように強制されなくなります。

LONG
short 型浮動小数点精度の数学組み込み関数に対する引数が、最大 long 型浮動小数点精度に変換され、同じ最大 long 型浮動小数点精度の結果を出します。

EXTENDED
short または long 型浮動小数点精度の数学組み込み関数に対する引数が、最大拡張型浮動小数点精度に変換され、同じ最大拡張型浮動小数点精度の結果を生成します。

精度 *p* の FLOAT DEC 式で使用される精度は、*p* <= 6 の場合は short 型浮動小数点精度であり、6 < *p* <= 16 の場合は long 型浮動小数点精度であり、*p* > 16 の場合は拡張型浮動小数点精度です。

精度 *p* の FLOAT BIN 式で使用される精度は、*p* <= 21 の場合は short 型浮動小数点精度であり、21 < *p* <= 53 の場合は long 型浮動小数点精度であり、*p* > 53 の場合は拡張型浮動小数点精度です。

最大拡張型浮動小数点精度は、プラットフォームによって決まります。

GOFF

GOFF オプションは、一般オブジェクト・ファイル・フォーマット (GOFF) でオブジェクト・ファイルを生成するようにコンパイラーに指示します。

GOFF および OBJECT オプションが有効になっている場合、コンパイラーは、オブジェクト・ファイルを GOFF フォーマットで生成します。

NOGOFF および OBJECT オプションが有効になっている場合、コンパイラーは、オブジェクト・ファイルを XOBJ フォーマットで生成します。

GOFF フォーマットは、S/370 オブジェクト・モジュール・フォーマットおよび拡張オブジェクト・モジュール・フォーマットを置き換えるものです。 このフォーマットによって、以前のフォーマットのさまざまな制限 (例: 16 MB のセクション・サイズ) が除去され、ロング・ネームおよび長い属性のネイティブ z/OS サポートなど、多くの便利な拡張機能が提供されるようになりました。GOFF は、XCOFF および ELF などの業界標準の一部の側面を取り込んでいます。

GOFF オプションを指定した場合、出力オブジェクトをバインドするバインダーを使用する必要があります。

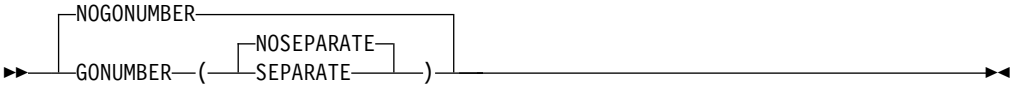
GOFF オプションでは、以下のオプションはサポートされていません。

- COMMON
- NOWRITABLE(PRV)

注: GOFF およびファイル名が重複したソース・ファイルを使用した場合、リンカーは、エラーを出してコード・セクションの 1 つを破棄する可能性があります。このような場合には、NOCSECT を指定して CSECT オプションをオフにしてください。

GONUMBER

GONUMBER オプションは、ソース・プログラムの行番号をランタイム・メッセージに含めるための追加情報をコンパイラーが生成することを指定します。



省略形: GN、NGN

SEPARATE
TEST(SEPARATE) オプションを指定すると、生成されたステートメント番号テーブルが別個のデバッグ・ファイルに入れられます。 GONUMBER (SEPARATE) を使用すると、ランタイム・メッセージにステートメント番号を組み込むことはできません。

NOSEPARATE
生成されたステートメント番号テーブルがオブジェクト・デックに入れられます。

あるいは、オフセット・アドレスを使用して行番号を派生させることもできます。オフセット・アドレスは、ランタイム・メッセージにも、OFFSET オプションで生成されるテーブルまたは LIST オプションで生成されるアセンブラー・リストにも、常に含まれています。

GONUMBER の使用は TEST オプションの ALL および STMT サブオプションにより強制されます。

GOSTMT オプションは存在しないことに注意してください。実行時にエラーの発生個所を特定するための情報を生成するオプションは、GONUMBER オプションのみです。 GONUMBER オプションを使用すると、ランタイム・エラー・メッセージに含まれる「ステートメント (statement)」という語は、STMT オプションが有効な場合でも、NUMBER コンパイラー・オプションで使用される行番号を参照します。

TEST(SEPARATE) なしで GONUMBER(SEPARATE) オプションを指定すると、このオプションは GONUMBER(NOSEPARATE) に変更されます。

TEST と NOGONUMBER の両方を指定すると、NOGONUMBER オプションは GONUMBER(NOSEPARATE) に変更されます。

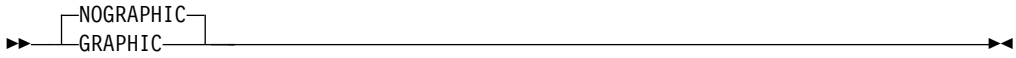
デフォルトは NOGONUMBER です。

互換性のため、GONUMBER オプションを指定した場合のデフォルト・サブオプションは SEPARATE になります。

GRAPHIC

GRAPHIC オプションを指定すると、ソース・プログラムに 2 バイト文字を入れることができます。

16 進コード「0E」はシフトアウト制御コードとして扱われ、「0F」はシフトイン制御コードとして扱われます (コメント内やストリング定数内も含めて、ソース・プログラム内のどこにこれらのコードが現れるのかは関係ありません)。



省略形: GR、NGR

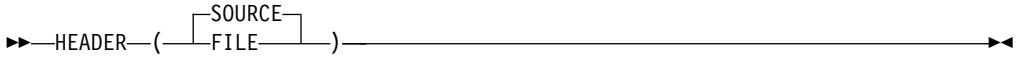
GRAPHIC オプションを指定すると、コンパイル中に使用されるどの STREAM ファイルにも GRAPHIC ENVIRONMENT オプションが適用されます。

ソース・プログラムで次のいずれかを使用する場合は、GRAPHIC オプションを指定しなければなりません。

- DBCS ID
- 漢字ストリング定数
- 混合ストリング定数
- ソース内のどこか別の場所にあるシフト・コード

HEADER

HEADER オプションを使用すれば、コンパイラー・リスト内の各ヘッダー行の中央に表示される内容を制御できます。



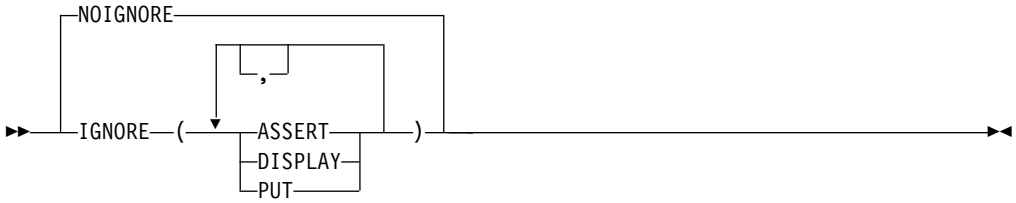
HEADER(SOURCE) が指定された場合、コンパイラーは、コンパイラー・リスト内の各ヘッダー行の中央に、*PROCESS ステートメントの後で最初に位置するソース行のテキストを使用します。

HEADER(FILE) が指定された場合、コンパイラーは、コンパイラー・リスト内の各ヘッダー行の中央にソース・ファイルの名前を使用します。

デフォルトは HEADER(SOURCE) です。

IGNORE

IGNORE オプションは、ASSERT ステートメント、DISPLAY ステートメント、および PUT ステートメントを無視するかどうかを制御します。ステートメントを無視するというのは、ステートメントをセミコロンで置き換えるようなものです。



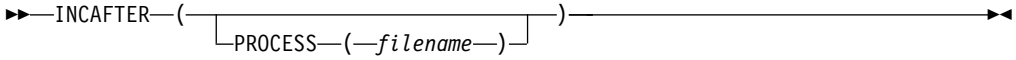
ASSERT
コンパイラーはすべての ASSERT ステートメント (そのステートメントに含まれているすべての関数参照を含む) を無視します。

DISPLAY
コンパイラーは、ステートメントに組み込まれているすべての関数参照を含め、すべての DISPLAY ステートメントを無視します。

PUT
コンパイラーはすべての PUT FILE ステートメントを無視します。

INCAFTER

INCAFTER オプションでは、ソース・プログラムの中で特定のステートメントの後に組み込むファイルを指定します。



filename
最後の PROCESS ステートメントの後に組み込むファイルの名前

現在、PROCESS は唯一のサブオプションであり、最後の PROCESS ステートメントの後に組み込むファイルの名前を指定します。

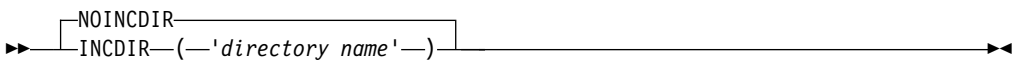
次の例について検討してみます。

```
INCAFTER(PROCESS(DFTS))
```

この例は、ソースの最後の PROCESS ステートメントの後にステートメント `%INCLUDE DFTS;` をコーディングするのと同等です。

INCDIR

INCDIR コンパイラー・オプションは、組み込みファイルの検索に使用される検索パスに追加されるディレクトリーを指定します。



directory name
組み込みファイルを検索するディレクトリーの名前。 INCDIR オプションを複数回指定できます。その場合、ディレクトリーは指定された順序で検索されます。

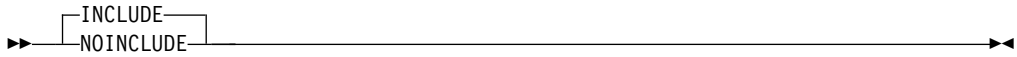
バッチの場合を除いて、コンパイラーは次の順序で INCLUDE ファイルを探します。

1. 現行ディレクトリー
2. –I フラグまたは INCDIR コンパイラー・オプションで指定されたディレクトリー
3. /usr/include ディレクトリー
4. INCPDS コンパイラー・オプションで指定された PDS

バッチ環境では、このオプションは DFT(LOWERINC) オプションと併用されることが最良と考えられ、このオプションは `%include x;` 形式の組み込みステートメントにのみ影響します。このような組み込みステートメントの場合、`x.inc` という名前の hfs ファイルは、このオプションで指定されたディレクトリーで最初に検索されます。 hfs ファイルが見つからない場合、`x` は SYSLIB DD で指定された PDS(E) のメンバーであるはずです。 `%include dd(x);` 形式の組み込みステートメントの場合、hfs ファイルは組み込まれません。メンバー `x` は常に、指定の DD で指定された PDS のメンバーでなければなりません。

INCLUDE

INCLUDE コンパイラー・オプションは、コンパイラーの最終パスが %INCLUDE ステートメントと %XINCLUDE ステートメントを処理するかどうかを制御します。



INCLUDE
マクロ・プリプロセッサーと、コンパイラーの最終パスの両方が、%INCLUDE ステートメントと %XINCLUDE ステートメントを処理します。

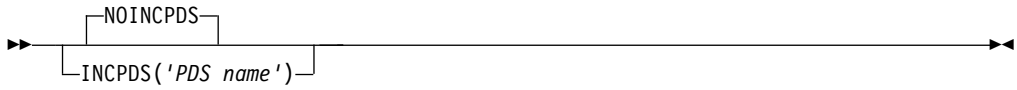
NOINCLUDE
マクロ・プリプロセッサーのみが %INCLUDE ステートメントと %XINCLUDE ステートメントを処理します。

INCLUDE がデフォルトです。

INCPDS

INCPDS オプションは、z/OS UNIX においてプログラムをコンパイルするときにコンパイラーが組み込むファイルがある PDS を指定します。

注: このオプションは、z/OS UNIX 環境でのコンパイルにだけ適用されます。



PDS name
組み込まれるファイルがある PDS の名前。

例えば、SOURCE.PLI という PDS からプログラム TEST をコンパイルする場合に、PDS SOURCE.INC に含まれる組み込みファイルを使用するときは、次のコマンドを指定します。

```
pli -c -qincpds="SOURCE.INC" "//'SOURCE.PLI(TEST)'"
```

コンパイラーは次の順序で組み込みファイルを検索します。

1. 現行ディレクトリー
2. –I フラグまたは INCDIR コンパイラー・オプションで指定されたディレクトリー
3. /usr/include ディレクトリー
4. INCPDS コンパイラー・オプションで指定された PDS

INITAUTO

INITAUTO オプションは、INITIAL 属性を指定せずに宣言されたすべての AUTOMATIC 変数に、INITIAL 属性を追加するようコンパイラーに指示します。

INITAUTO(FULL) を指定すると、コンパイラーは、INITIAL 属性を持たないすべての AUTOMATIC 変数に、そのデータ属性に従って以下の INITIAL 属性を追加します。

- INIT((*) 0) - データ属性が FIXED または FLOAT の場合
- INIT((*) '') - データ属性が PICTURE、CHAR、BIT、GRAPHIC、または WIDECHAR の場合
- INIT((*) SYSNULL()) - データ属性が POINTER または OFFSET の場合
- INIT((*) NULLENTRY()) - データ属性が ENTRY の場合

コンパイラーは、その他の属性を持つ変数には INITIAL 属性を追加しません。

完全に初期化されていない AUTOMATIC 変数 (ただし、DFT(INITFILL) オプションと違って、これらの変数は意味のある初期値を持つようになりました) が入っている各ブロックごとに、プロローグの中により多くのコードが INITAUTO により生成されるようになり、パフォーマンスに悪い影響を与えることになります。

INITAUTO オプションは、NOINIT 属性を指定して宣言された変数に対しては、INITIAL 属性を適用しません。

INITAUTO(SHORT) が指定されると、コンパイラーは、INITIAL 属性を持たない AUTOMATIC 変数に INITIAL 属性を追加します (ただし、この変数もスカラーであり、次のいずれかの属性を持つ場合に限られます)。

- POINTER
- OFFSET
- FIXED BIN
- FLOAT

- NONVARYING BIT
- NONVARYING CHAR(1)
- NONVARYING WCHAR(1)

INITAUTO(FULL) の場合と同様、追加された INITIAL 属性はデータ・タイプに適合します。

ランタイム STORAGE オプションを使用してすべてのストレージをゼロ設定しても、最適化プログラムが未初期化の AUTOMATIC 変数、特に、レジスターに合せて最適化されている同変数に対して不要なコードを生成することがあります。上にリストしたデータ・タイプを持つスカラー変数はレジスターに最適化される可能性がある変数です。したがって、INITAUTO(SHORT) を使用した方が DFT(INITFILL) を使用するよりパフォーマンスへの影響が少ない場合があります。ただし、後者は最適化プログラムにあいまいなコードを残します。さらに、コードが正しいのは、すべての変数が使用前に明示的に初期化された場合に限られます。

INITBASED

INITBASED オプションは、INITIAL 属性を指定せずに宣言されたすべての BASED 変数に、INITIAL 属性を追加するようコンパイラーに指示します。

このオプションは、BASED 変数に対してであることを除き、INITAUTO と同じ機能を実行します。

INITBASED オプションにより、完全に初期化されていない BASED 変数の ALLOCATE に対して、より多くのコードが生成されるようになり、パフォーマンスに悪い影響を与えることになります。

INITBASED オプションは、NOINIT 属性を指定して宣言された変数に対しては、INITIAL 属性を適用しません。

INITCTL

INITCTL オプションは、INITIAL 属性を指定せずに宣言されたすべての CONTROLLED 変数に、INITIAL 属性を追加するようコンパイラーに指示します。

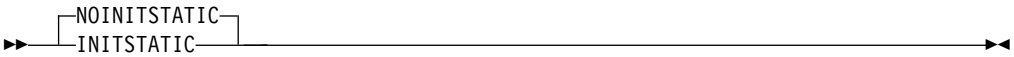
このオプションは、CONTROLLED 変数に対してであることを除き、INITAUTO と同じ機能を実行します。

INITCTL オプションにより、完全に初期化されていない CONTROLLED 変数の ALLOCATE に対して、より多くのコードが生成されるようになり、パフォーマンスに悪い影響を与えることになります。

INITCTL オプションは、NOINIT 属性を指定して宣言された変数に対しては、INITIAL 属性を適用しません。

INITSTATIC

INITSTATIC オプションは、INITIAL 属性を指定せずに宣言されたすべての STATIC 変数に、INITIAL 属性を追加するようコンパイラーに指示します。



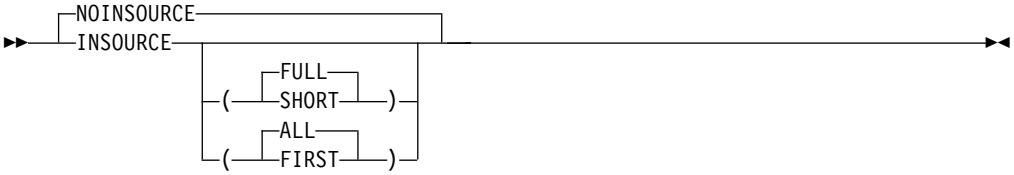
このオプションは、STATIC 変数に対してであることを除き、INITAUTO と同じ機能を実行します。

INITSTATIC オプションでは、一部に大きいオブジェクトを作成したり、長いコンパイルを作成する可能性はありますが、それ以外はパフォーマンスに影響を与えることはありません。

INITSTATIC オプションは、NOINIT 属性を指定して宣言された変数に対しては、INITIAL 属性を適用しません。

INSOURCE

INSOURCE オプションは、PL/I マクロ、CICS、または SQL のプリプロセッサーが変換できるように、コンパイラーがソース・プログラムのリストを組み込むことを指定します。



省略形: IS、NIS

FULL
INSOURCE リストは %NOPRINT ステートメントを無視し、プリプロセッサーがソースを変換する前にすべてのソースがリストに組み込まれます。

FULL がデフォルトです。

SHORT
INSOURCE リストは %PRINT ステートメントと %NOPRINT ステートメントを区別します。

ALL
INSOURCE リストには、各プリプロセッサーとコンパイラー自体によって生成されるソース・リストが含まれます。ALL はデフォルトです。

FIRST
INSOURCE リストには、最初のプリプロセッサーによって生成されたソース・リストのみが含まれます。

INSOURCE オプションを指定すると、プログラムのロジックとは関係なく、各ファイルの読み取り順にテキストがリストに入れられます。 例えば、次のような単純なプログラムについて検討してみます。このプログラムでは、PROC ステートメントと END ステートメントの間に %INCLUDE ステートメントがあります。

```
insource: proc options(main);
  %include member;
end;
```

INSOURCE リストには、メインプログラムが取り込むすべてのテキストが含まれ、その後に、ファイル member 内のテキストが組み込まれます (さらに、そのファイル内のテキストがすべて含まれた後で、次のファイルのテキストが組み込まれ、それが繰り返されます)。

INSOURCE(SHORT) オプションを指定した場合、%INCLUDE ステートメントによってインクルードされるテキストは、%INCLUDE ステートメントの実行時に有効だった print/noprint 状況を継承しますが、その print/noprint 状況はインクルードされるテキストの終わりで復元されます (ただし SOURCE リスト内では、インクルードされるテキストの終わりで print/noprint 状況は復元されません)。

INTERRUPT

INTERRUPT オプションを指定すると、コンパイル済みプログラムがアテンション要求 (割り込み) に応答します。

省略形: INT、NINT

このオプションにより、コンパイル済み PL/I プログラムが対話式システムの下で実行されるときのアテンション割り込みの効果が決まります。 このオプションは、TSO の下で実行されるプログラムに対してだけ効果があります。 ATTENTION 条件の発生に依存するプログラムを作成した場合、そのプログラムは INTERRUPT オプションを使ってコンパイルしなければなりません。 このオプションを使用すると、アテンション割り込みをプログラミングの不可欠な部分にすることができます。 この方法により、プログラムを対話式で大幅に制御できます。

INTERRUPT オプションを指定すると、アテンション割り込みが生じた場合、確立されている ATTENTION ON ユニットが制御を得ます。 ATTENTION ON ユニットの実行が完了すると、GOTO ステートメントで別の場所が指定されていない限り、制御は割り込み点に戻ります。ATTENTION ON ユニットを確立していないと、アテンション割り込みは無視されます。

NOINTERRUPT を指定すると、プログラムの実行時のアテンション割り込みで、ATTENTION ON ユニットに制御が渡ることはありません。

テストの目的だけにアテンション割り込み機能が必要な場合は、INTERRUPT オプションの代わりに TEST オプションを使用してください。

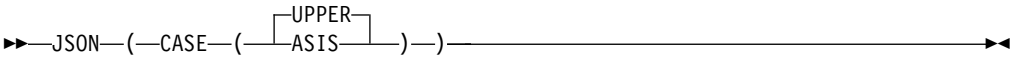
関連情報:

101 ページの『TEST』
TEST オプションは、コンパイラーがオブジェクト・コードの一部 として生成する検査機能のレベルを指定します。 このオプションを使用すれば、テスト・フックの位置を制御したり、記号テーブルを生成するかどうかを制御したりできます。

533 ページの『第 22 章 割り込みとアテンションの処理』

JSON

JSON オプションを使用すれば、JSONPUT 組み込み関数によって生成されたり JSONGET 組み込み関数によって予期されたりする JSON テキスト内の名前の大/小文字を選択できます。



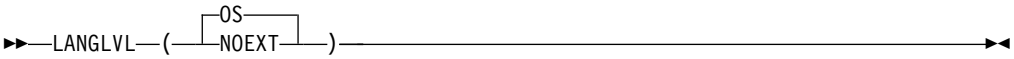
CASE(UPPER | ASIS)

CASE(UPPER) サブオプションが指定された場合は、JSONPUT 組み込み関数によって生成されたり JSONGET 組み込み関数によって期待されたりする JSON テキスト内の名前がすべて大文字になります。

CASE(ASIS) サブオプションが指定された場合は、JSONPUT 組み込み関数によって生成されたり JSONGET 組み込み関数によって期待されたりする JSON テキスト内の名前は、宣言で使用されている大/小文字になります。マクロ・プリプロセッサー・オプション CASE(ASIS) を使用せずに MACRO プリプロセッサーを使用すると、コンパイラーによって参照されるソースでは名前がすべて大文字になるため、JSON(CASE(ASIS)) オプションを指定しても無意味です。

LANGLVL

LANGLVL オプションでは、コンパイラーに受け入れさせたい PL/I 言語定義のレベルを指定します。



NOEXT

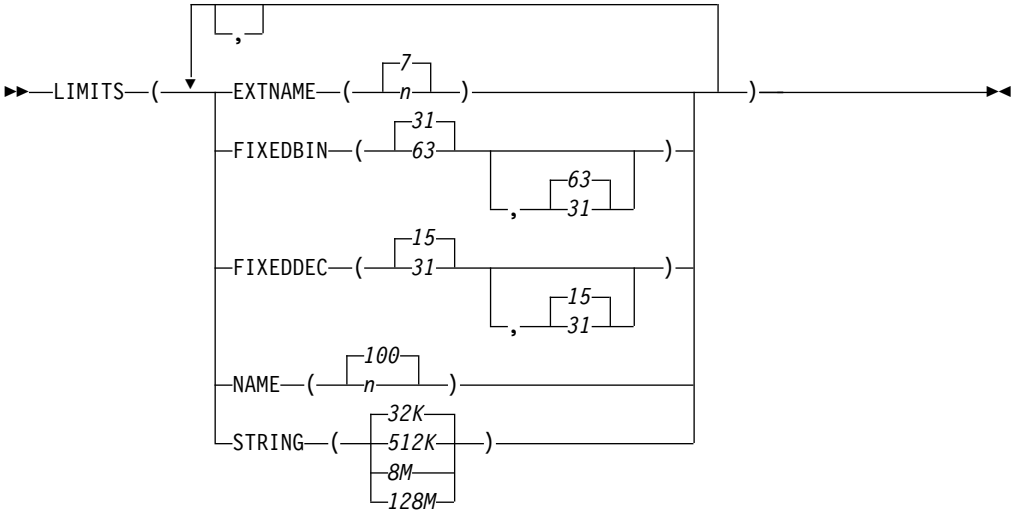
以下の ENVIRONMENT オプションのみが受け入れられます。

Bkwd	Genkey	Keyloc	Relative
Consecutive	Graphic	Organization	Scalarvarying
Ctlasa	Indexed	Recsize	Vsam
Deblock	Keylength	Regional	

OS すべての ENVIRONMENT オプションが使用できます。 すべての ENVIRONMENT オプションが 260 ページの表 15 にリストされています。

LIMITS

LIMITS オプションでは、各種のインプリメンテーションの制限を指定します。



EXTNAME

EXTERNAL 名の最大長を指定します。*n* の最大値は 100、最小値は 7 です。

FIXEDDEC

FIXED DECIMAL の最大精度として 15 または 31 のいずれかを指定します。デフォルトは FIXEDDEC(15,31) です。

FIXEDDEC(15,31) を指定した場合は 15 桁を超える桁数で FIXED DECIMAL 変数を宣言できますが、15 桁を超える桁数のオペランドが式に含まれていない限り、コンパイラーはすべての算術演算に対して最大精度として 15 を使用します。

FIXEDDEC(15,31) が指定された場合は、FIXEDDEC(31) が指定された場合よりもパフォーマンスがはるかに向上します。

FIXEDDEC(15) と FIXEDDEC(15,15) は等価であり、同様に FIXEDDEC(31) と FIXEDDEC(31,31) も等価です。

FIXEDDEC(31,15) は指定できません。

FIXEDBIN

SIGNED FIXED BINARY の最大精度として 31 または 63 を指定します。 デフォルトは (31,63) です。

FIXEDBIN(31,63) を指定した場合は 8 バイト整数を宣言できますが、式に 8 バイト整数が含まれていない限り、コンパイラーはすべての整数算術演算に対して 4 バイト整数を使用します。

ただし、FIXEDBIN(31,63) オプションまたは FIXEDBIN(63) オプションが指定された場合、コンパイラーは、データ型が混在する式に対して 8 バイト整数算術演算を使用することがあります。例えば FIXED BIN(31) の値が FIXED DEC(13) の値に加算される場合、コンパイラーは FIXED BIN の結果を生成し、LIMITS(FIXEDBIN(31,63)) が指定されていると、その結果の精度は 31 よ

り大きくなります (FIXED DEC の精度が 9 より大きいため)。この状況が発生すると、コンパイラーは通知メッセージ IBM2809 を発行します。

FIXEDBIN(31,63) は、FIXEDBIN(63) よりもかなり良いパフォーマンスを提供します。

FIXEDBIN(31) と FIXEDBIN(31,63) は等価です。同様に、FIXEDBIN(63) と FIXEDBIN(63,63) も等価です。

FIXEDBIN(63,31) や FIXEDBIN(31,31) は使用できません。

UNSIGNED FIXED BINARY の最大精度は、1 を加えた数、つまり 32 と 64 です。

NAME

プログラムの中の変数名の最大長を指定します。 *n* の最大値は 100、最小値は 31 です。

STRING

これは、BIT 変数、CHARACTER 変数、または WIDECHAR 変数の長さのしきい値として値 32K、512K、8M、および 128M を受け入れます。つまり、長さはしきい値未満でなければならないため、対応する制限は 32767、524287、8388607、および 134217727 ということになります。

- デフォルト値は 32K です。これより大きい値は、CMPAT(V3) オプションと BIFPREC(31) オプションも指定される場合に限り受け入れられます。
- この制限は NONVARYING、VARYINGZ、および VARYING4 に適用されますが、VARYING には適用されません。 VARYING に指定できる最大値は 32K です。

LINECOUNT

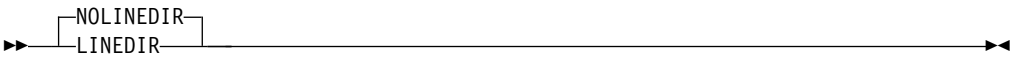
LINECOUNT オプションは、コンパイラー・リストのページ当たりの行数 (ブランク行と見出し行を含む) を指定します。

省略形: LC

n リストの 1 ページの行数。 値の範囲は 10 から 32767 までです。

LINEDIR

LINEDIR オプションは、%LINE ディレクティブをコンパイラーが受け入れるようにするかどうかを指定します。



LINEDIR オプションが指定されると、コンパイラーはすべての %INCLUDE ステートメントを拒否します。 LINEDIR オプションが指定されると、コンパイラーは TEST オプションの SEPARATE サブオプションの使用も拒否します。

LIST

LIST オプションは、コンパイラーが疑似アセンブラー・リストを生成するように指定します。

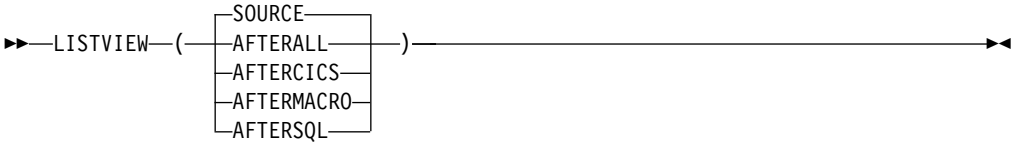
LIST オプションを指定すると、コンパイル時に必要な時間と領域が増加します。OFFSET と MAP オプションは、ごくわずかのコストで必要な情報を提供します。

各ブロックごとに、疑似アセンブラー・リストも、リストの終わりに、全コンパイル単位の開始位置からそのブロック内の最初の命令までのオフセットで組み込まれます。

LISTVIEW

LISTVIEW オプションは、コンパイラーがソース・リストでソースを表示するかどうか、または 1 つ以上のプリプロセッサーで処理された後にソースを表示するかどうかを指定します。

LISTVIEW オプションは、NOSOURCE オプションが有効になっていると無視されます。



SOURCE
ソース・リストで生のソースを表示します。また、さらに重要なこととして、IBM Debug Tool でこれをソース・ビューとして表示します。

AFTERALL
最後のプリプロセッサーの最後の呼び出し (存在する場合) の MDECK 由来のソースとしてソース・リストで表示します。また、さらに重要なこととして、TEST コンパイラー・オプションの SEPARATE サブオプションも指定されている場合は、Debug Tool でこれをソース・ビューとして立ち上げます。

AFTERALL の省略形として AALL を使用できます。

AFTERCICS
CICS プリプロセッサーの最後の呼び出し (存在する場合) の MDECK 由来のソースとしてソース・リストで表示します。また、さらに重要なこととして、TEST コンパイラー・オプションの SEPARATE サブオプションも指定されている場合は、Debug Tool でこれをソース・ビューとして立ち上げます。

ACICS は、AFTERCICS の省略形として使用されることがあります。

AFTERMACRO
MACRO プリプロセッサーの最後の呼び出し (存在する場合) の MDECK 由来のソースとしてソース・リストで表示します。また、さらに重要なこととして、

TEST コンパイラー・オプションの SEPARATE サブオプションも指定されている場合は、Debug Tool でこれをソース・ビューとして立ち上げます。

AMACRO は、AFTERMACRO の省略形として使用されることがあります。

AFTERSQL
SQL プリプロセッサーの最後の呼び出し (存在する場合) の MDECK 由来のソースとしてソース・リストで表示します。また、さらに重要なこととして、TEST コンパイラー・オプションの SEPARATE サブオプションも指定されている場合は、Debug Tool でこれをソース・ビューとして立ち上げます。

ASQL は、AFTERSQL の省略形として使用されることがあります。

TEST オプションを指定して、LISTVIEW に SOURCE 以外のサブオプションを指定した場合には、TEST オプションに SEPARATE サブオプションも指定する必要があります。

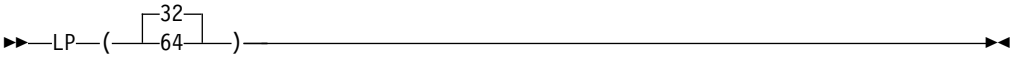
次の例は、AFTERMACRO サブオプション、AFTERSQL サブオプション、および AFTERALL サブオプションの各種効果を示しています。

PP オプションが PP(MACRO('INCONLY'), SQL, MACRO) であるとします。

- LISTVIEW(AFTERMACRO) では、TEST(SEP) が指定された場合にリストおよび Debug Tool ソース・ウィンドウに示される「ソース」は、MACRO プリプロセッサーの 2 番目の呼び出しによって生成された MDECK から得られたかのように表示されます。
- LISTVIEW(AFTERSQL) では、TEST(SEP) が指定された場合にリストおよび Debug Tool ソース・ウィンドウに示される「ソース」は、SQL プリプロセッサーの呼び出しによって生成された MDECK から得られたかのように表示されます (したがって、%DCL およびその他のマクロ・ステートメントは依然として表示されます)。
- LISTVIEW(AFTERALL) では、マクロ・プリプロセッサーが PP オプションの最後であるため、「ソース」は LISTVIEW(AFTERMACRO) オプションの下にきます。

LP

LP オプションは、コンパイラーが 31 ビット・コードを生成するのか 64 ビット・コードを生成するのかを指定します。また、このオプションは、POINTER、HANDLE、および関連変数のデフォルト・サイズも決定します。



32 LP(32) では、コンパイラーは 31 ビット・コードを生成します。さらに、タイプ *size_t* は FIXED BIN(31) に解決されます。 POINTER および HANDLE のデフォルト・サイズは 4 バイトです。

64 LP(64) では、コンパイラーは 64 ビット・コードを生成します。さらに、タイプ *size_t* は FIXED BIN(63) に解決されます。 POINTER および HANDLE のデフォルト・サイズは 8 バイトです。

注: LP(64) では、一部のコンパイラー・オプションは適用されません。詳しくは、 221 ページの『コンパイラー・オプションを使用して 64 ビット・アプリケーションをビルド』を参照してください。

デフォルトは LP(32) です。

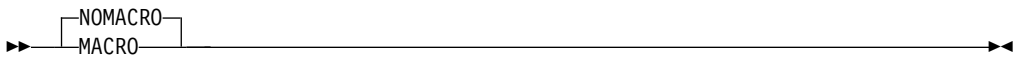
関連情報:

221 ページの『第 7 章 64 ビット・アプリケーションを開発する場合の考慮事項』
Enterprise PL/I を使用すれば、31 ビット/64 ビット・アプリケーションを開発できます。 ご使用のアプリケーションで 64 ビット環境がサポートされるようにするには、必要に応じてコードを調整しなければならない場合があります。 このセクションでは、開発時およびコンパイル時に考慮すべき事項について説明します。

215 ページの『第 6 章 64 ビット・プログラムに対するリンク・エディットおよび実行』
LP(64) でのコンパイルが終わった 64 ビット・プログラムは、未解決の相互参照、および言語環境プログラム・ランタイム・ライブラリーに対する参照を含む 1 つ以上のオブジェクト・モジュールで構成されています。 これらの参照は、リンク・エディット時 (静的) または実行時 (動的) に 解決されます。

MACRO

MACRO オプションはマクロ・プリプロセッサーを呼び出します。



省略形: M、NM

PP(MACRO) オプションを使用して MACRO プリプロセッサーを呼び出すこともできます。ただし、同じコンパイルで MACRO オプションと PP(MACRO) オプションを両方とも使用することは勧められません。

関連情報:

68 ページの『PP』
PP オプションは、コンパイル前に呼び出すプリプロセッサー (およびそれらの順序) を指定します。

132 ページの『マクロ・プリプロセッサー』
マクロを使用すれば、インプリメンテーションの詳細と処理対象のデータを隠して演算のみを表す方法で、通常使用される PL/I コードを作成できます。 汎用のサブルーチンとは対照的に、マクロでは、個別用途ごとに必要となるコードのみを生成できます。 マクロ・プリプロセッサーは、MACRO オプションまたは PP(MACRO) オプションを指定することによって呼び出すことができます。

MAP

MAP オプションを指定すると、ダンプ内の静的変数と自動変数を見つけるために使用できる追加情報が、コンパイラーによって生成されます。

MARGINI

MARGINI オプションでは、INSOURCE オプションおよび SOURCE オプションで生成されたリストの左側のマージンの前の桁と、右側のマージンの後の桁に、コンパイラーが置く文字を指定します。

省略形: MI、NMI

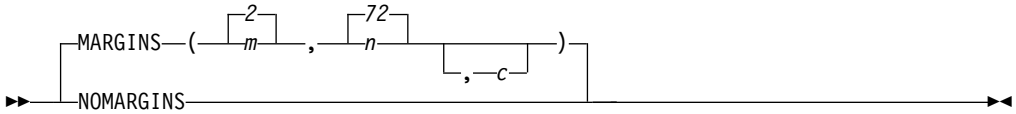
c マージン標識として印刷される文字。

注**:** NOMARGINI は MARGINI(' ') と同等です。

MARGINS

MARGINS オプションは、各コンパイラー入力レコードのどの部分に PL/I ステートメントが含まれるのかを指定します。また、MARGINS オプションは、SOURCE オプションと INSOURCE オプションの一方または両方が適用される場合に、リストをフォーマット設定する ANS 制御文字の位置も指定します。コンパイラーは、これらの限界外にあるデータは処理しませんが、ソース・リストには入れます。

PL/I ソースがソース入力レコードから取り出される際、レコードの最初のデータ・バイトが、その直前のレコードの最後のデータ・バイトのすぐ後にくるように取り出されます。 変数レコードの場合、ブランクが必要であれば、必ず各レコードのマージン間に明示的にブランクを挿入するようにしなければなりません。



省略形: MAR

m コンパイラーによって処理される左端の文字 (最初のデータ・バイト) の桁番号。 これは、100 を超えてはなりません。

n コンパイラーによって処理される右端の文字 (最後のデータ・バイト) の桁番号。 *m* より大きくしますが、必ず、200 以下にします。

可変長レコードは、最大レコード長になるように効果的にブランクが埋め込まれます。

c ANS プリンター制御文字の桁番号。 200 以下でなければならず、また *m* と *n* に指定された値の範囲外でなければなりません。 *c* の値として 0 を指定すると、ANS 制御文字がないことを示します。次の制御文字だけを使用できます。

(ブランク**)**
1 行スキップしてから印刷する。

| **0** 2 行スキップしてから印刷する。

| **–** 3 行スキップしてから印刷する。

| **+** スキップしないで印刷する。

| **1** 改ページする

| これ以外の文字を使用するとエラーになり、ブランクに置き換えられます。

| 最大ソース・レコード長を超える値 *c* は使用しないでください。リストのフォーマットが予測できないものになるためです。この問題を避けるには、可変長レコードのソース・マージンの左側に紙送り制御文字を置きます。

| %PAGE ステートメントや %SKIP ステートメントを使用する代わりに MARGINS(,,c) を指定できます (「*PL/I* 言語解説書」を参照してください)。

固定長レコードの IBM 提供のデフォルトは MARGINS(2,72) です。可変長レコードと不定長レコードの IBM 提供のデフォルトは MARGINS(10,100) です。 このデフォルトは、プリンター制御文字がないことを指定します。

プログラム内の 1 次入力のデフォルトを指定変更するには、MARGINS オプションを使用します。 2 次入力のマージンは 1 次入力の場合と同じでなければなりません。

NOMARGINS オプションは、前に出現した MARGINS オプションのインスタンスを抑制します。 このオプションの目的は、ご使用のシステムのコンパイル時間オプションをデフォルト設定にできるようにすることで、このコンパイル時間オプションは、変数ソース・フォーマット・ファイルが使用可能になっている間、固定フォーマット・ソース設定用に調整された MARGINS オプションを使用します。

コンパイラーに渡されたパラメーター・ストリングの一部として使用する場合は、通常、NOMARGINS オプションを指定します。 このコンパイラーは、%PROCESS の中にオプションを見付けると、NOMARGINS を無視します。

MAXBRANCH

MAXBRANCH オプションは、分岐の数が多すぎるブロックにフラグを立てます。分岐には、すべての条件付きジャンプ、および分岐テーブルに取り込むことができる SELECT ステートメント内の各 WHEN が含まれます。

max
ブロックのサイクロマティックな複雑さや条件付きの複雑さを測定する限度。デフォルトは 2000 です。

「`if a then ...; else ...`」という形式のステートメントの場合は、そのステートメントを含むブロック内の分岐の総数に 1 が加算されます。「`if a = 0 | b = 0 then ...`」という形式のステートメントの場合は、2 が加算されます。

MAXGEN

MAXGEN オプションは、任意のユーザー・ステートメントに対して生成される中間言語ステートメントの最大数を指定します。このオプションが指定されると、この最大数を超えたすべてのステートメントにコンパイラーがフラグを立てます。

任意のユーザー・ステートメントに対して生成される中間言語ステートメントの数は、コンパイラー・リリース、コンパイラー保守レベル、および有効になっているコンパイラー・オプションによって異なる可能性があります。このオプションの唯一の目的は、過剰な量のコードが生成される (つまり、コーディングが正常に行われていない可能性がある) ステートメントを検出できるように支援することです。

ただし、プリプロセッサーが使用されると、一部のステートメントに対して生成される中間言語ステートメントの数が非常に多くなる可能性があることに注意してください。このような状態では、MAXGEN しきい値を大きくするか、または LISTVIEW(AFTERALL) オプションを使用する方がよい場合もあります。

デフォルトは MAXGEN(100000) です。

MAXMEM

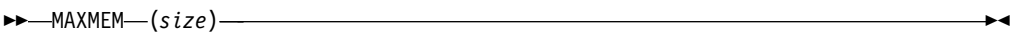
OPTIMIZE とともにコンパイルを行う場合、MAXMEM オプションは、メモリーを多く消費する特定の最適化のローカル・テーブル用に使用されるメモリーの量を指定のキロバイト数に制限します。

MAXMEM に対して指定できるメモリーの範囲は 1 から 2097152 までです。デフォルトは 1048576 です。

最大値の 2097152 を指定した場合、コンパイラーは無制限のメモリーが使用可能であることを想定します。 最大値より小さい値を MAXMEM に指定した場合 (特に OPT(2) オプションが有効の場合) は、コンパイラーがメッセージを出すことがあります。このメッセージは、最適化が禁止されていることを示し、MAXMEM により大きな値を使用するようにユーザーに促します。

MAXMEM オプションは、使用可能なメモリーの量がデフォルト値によって暗黙指定される量より少ない (または多い) ことが分かっている場合に使用してください。

MAXMEM オプションに指定したメモリーが最適化に十分でない場合は、最適化の品質が低下した状態でコンパイルが完了し、警告メッセージが出されます。



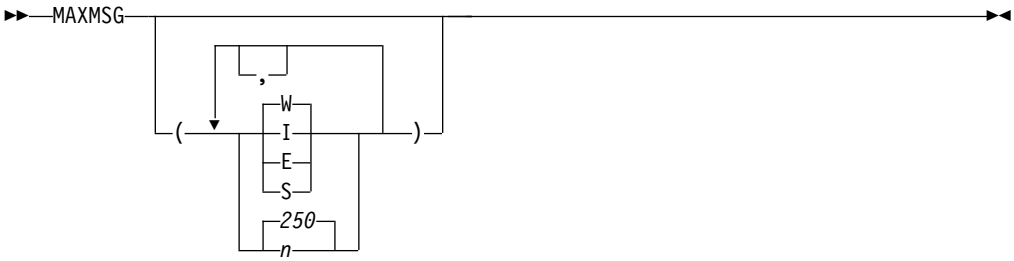
省略形: MAXM

MAXMEM に大きなサイズを指定した場合は、コンパイルされるソース・ファイル、ソース内のサブプログラムのサイズ、およびコンパイル用に使用できる仮想記憶域のサイズによっては、仮想記憶域の不足が原因でコンパイルが停止する場合があります。

MAXMEM オプションを使用する利点は、大規模で複雑なアプリケーションをコンパイルする場合に、コンパイラーが「**仮想記憶域の不足**」を示すエラー・メッセージを出してコンパイルを終了するのでなく、最適化品質の少し低下したオブジェクト・モジュールを作成して、警告メッセージを生成することです。

MAXMSG

MAXMSG オプションは、コンパイル時に生成されるはずの指定された重大度 (またはそれ以上) を持つメッセージの最大数を指定します。



I すべてのメッセージを数えます。

W 情報メッセージを除くすべてのメッセージを数えます。

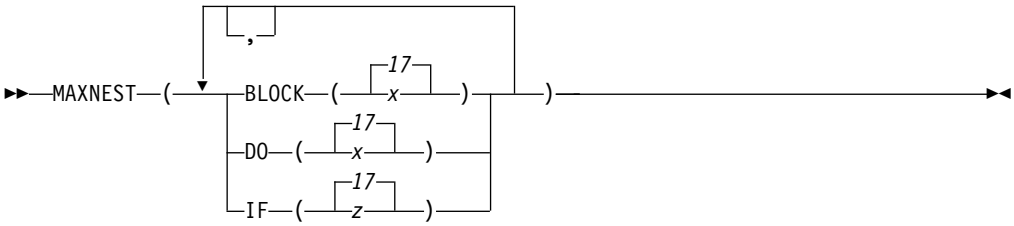
E 警告メッセージと通知メッセージを除くすべてのメッセージを数えます。

S 重大エラー・メッセージおよび回復不能エラー・メッセージだけを数えます。

n メッセージの数がこの値を超えた場合、コンパイルを終了します。 指定の重大度より低いメッセージ、またはコンパイラー出口ルーチンによりフィルターに掛けられて取り除かれたメッセージは、カウントされません。値は 0 から 32767 までの範囲で入力します。 0 を指定した場合、指定された重大度の最初のエラーが検出されるとコンパイルは終了します。

MAXNEST

MAXNEST オプションは、複雑すぎるプログラムというフラグをコンパイラーがプログラムに立てるまでに許可される各種ステートメントの最大ネストを指定します。



BLOCK
BEGIN および PROCEDURE ステートメントの最大ネストを指定します。

DO DO ステートメントの最大ネストを指定します。

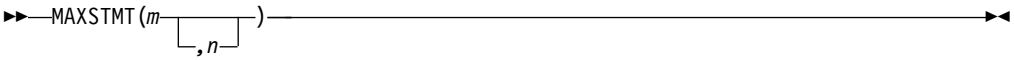
IF IF ステートメントの最大ネストを指定します。

ネスト限度の値範囲は 1 から 50 までです。

デフォルトは、MAXNEST(BLOCK(17) DO(17) IF(17)) です。

MAXSTMT

MAXSTMT オプションが指定された場合、コンパイラーは、指定された数よりも多くのステートメントを持つブロックにフラグを立てます。また、最適化に関しては、コンパイラーは、指定された数を超えるステートメントを持つすべてのブロックの最適化をオフにします。



m これは、OPT(2) のカットオフ値を指定します。デフォルトは 4096 です。

n オプション。これは、OPT(3) のカットオフ値を指定します。デフォルトは 8192 です。

MAXSTMT(*m*) が指定されている場合、*n* は *m* に設定されます。そのため、MAXSTMT(4096) と指定することは MAXSTMT(4096,4096) と指定することと同じです。

MAXSTMT に対して大きな *m* が指定されているときに、多くのステートメントを含むブロックがいくつかある場合は、使用できる仮想記憶域が十分にないと、コンパイルが打ち切られることがあります。

MAXTEMP

MAXTEMP オプションは、コンパイラー生成一時ステートメント用のストレージの量を非常に多く使用しているステートメントについて、コンパイラーがいつフラグを立てるかを判断します。

max

コンパイラー生成一時ステートメントに使用できるバイト数の限度。 *max* で指定されたバイト数より多くのバイトを使用するすべてのステートメントにコンパイラーがフラグを立てます。*max* のデフォルトは 50000 です。

このオプションでフラグが立てられたステートメントを調べてください。そのステートメントを別の方法でコーディングすれば、コードで必要となるスタック・ストレージの量を減らすことができる可能性があります。

MDECK

MDECK オプションを指定すると、プリプロセッサーは、z/OS の場合は SYSPUNCH DD ステートメントで定義されたファイルに、z/OS UNIX の場合は .dek ファイルに、プリプロセッサーの出力のコピーを作成します。

省略形: MD、NMD

MDECK オプションを使用すると、プリプロセッサーの出力を 80 桁のレコードのファイルとして保持できます。 このオプションは、MACRO オプションの使用中にだけ使用することができます。

AFTERALL
最後のプリプロセッサーが呼び出された後でファイルが生成されます。

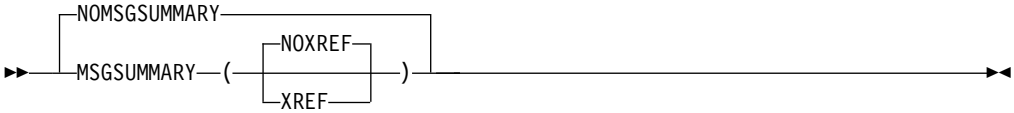
AFTERMACRO
マクロ・プリプロセッサーが最後に呼び出された後でファイルが生成されます (呼び出しがあった場合)。

関連情報:

55 ページの『MACRO』
MACRO オプションはマクロ・プリプロセッサーを呼び出します。

MSGSUMMARY

MSGSUMMARY オプションは、コンパイル中に発行された全メッセージの要約をコンパイラーがリストに追加するかどうかを決定します。



MSGSUMMARY(NOXREF)
コンパイラーがメッセージの要約をリストに追加します。 この要約はリスト内のファイル参照テーブルの後に配置されます。 この要約はコンパイラー・コンポーネント別にソートされ、さらに各コンポーネント内で重大度別にソートされ、続いてメッセージ番号でソートされます。

この要約には以下の情報が含まれます。

- コンパイルで生成された各メッセージ (同じメッセージが複数ある場合でも示されるメッセージは 1 つのみ)
- 各メッセージが生成された回数

MSGSUMMARY(XREF)
コンパイラーがメッセージの要約をリストに追加します。 この要約は、MSGSUMMARY(NOXREF) が指定されたときに追加されるものと同じです。ただし、この要約では、各メッセージの後ろに、メッセージが発行された行またはステートメントの番号がすべてリストされるという点が異なります。

NOMSGSUMMARY
メッセージの要約は生成されません。

NOMSGSUMMARY はデフォルトです。 MSGSUMMARY が指定されているときは、MSGSUMMARY(NOXREF) がデフォルトです。

MSGSUMMARY を使用して生成されたメッセージ要約を含むコンパイラー・リスト例については、 127 ページの図 4 を参照してください。

NAME

NAME オプションは、コンパイラーによって作成される TEXT ファイルに NAME レコードを入れるように指定します。

省略形: N

NAME オプションのサブオプションとして name が指定されない場合に使用される名前は次のように決定されます。

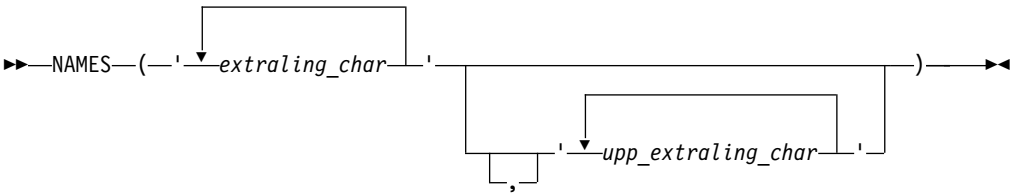
- PACKAGE ステートメントがある場合は、そのステートメントの左端にある名前が使用される。
- そうでない場合には、最初の PROCEDURE ステートメントの左端にある名前が使用される。

LIMITS(EXTNAME(*n*)) オプション (*n* <= 8) が使用されている場合、名前の長さは 8 文字を超えてはなりません。

NAMES

NAMES オプションでは、ID に使用できる特別言語文字 を指定します。

特別言語文字とは、「*PL/I* 言語解説書」で定義されている特殊文字、26 個の英字、および 10 個の数字以外の文字です。



extralingual_char
特別言語文字。

upp_extraling_char
最初のサブオプションで指定した文字に対応する大文字として解釈させる特別言語文字

2 番目のサブオプションを省略すると、PL/I は最初のサブオプションで指定された文字を小文字と大文字の両方として使用します。 2 番目のサブオプションを指定する場合は、最初のサブオプションで指定したのと同じ数の文字を指定しなければなりません。

デフォルトは NAMES('#@$' '#@$') です。

NATLANG

NATLANG オプションは、コンパイラーのメッセージやヘッダーなどの言語を指定します。

ENU
コンパイラーのメッセージやヘッダーなどはすべて大/小文字混合の英語になります。

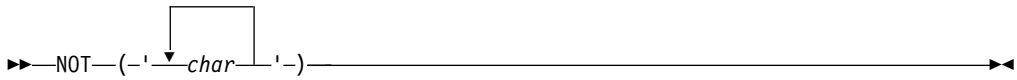
UEN
コンパイラーのメッセージやヘッダーなどはすべて大文字の英語になります。

NEST

NEST オプションを指定すると、SOURCE オプションの実行結果のリストに、各ステートメントごとのブロック・レベルと do グループ・レベルが示されます。

NOT

NOT オプションでは、論理否定演算子として使用できる代替記号を最大 7 つ指定します。



char
単一の SBCS 文字

標準の論理 NOT 記号 (¬) を除き、英字、数字、および「*PL/I* 言語解説書」に定義されている特殊文字はどれも指定できません。少なくとも有効な文字を 1 文字指定する必要があります。

NOT オプションを指定すると、標準 NOT 記号は、文字ストリング内の 1 つの文字として指定しない限り認識されなくなります。

例えば、NOT('~') を指定すると、波形記号 'A1'X が論理 NOT 演算子として認識され、標準 NOT 記号 ('¬') '5F'X は認識されません。同様に、NOT('~¬') は波形記号または標準 NOT 記号のどちらかが論理否定演算子として認識されることを意味します。

NOT 記号用の IBM 提供のデフォルト・コード・ポイントは、'5F'X です。論理否定記号は、キーボード上では論理否定記号 (¬) または脱字記号 (^) として表記されていることがあります。

NULLDATE

NULLDATE オプションは、一部の日時処理組み込み関数で SQL ヌル日付を有効な日付として受け入れるようにコンパイラーに指示します。

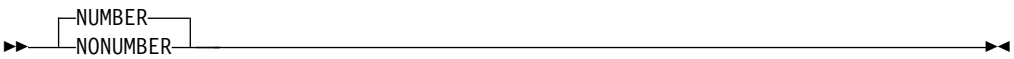
NULLDATE オプションが指定された場合、VALIDDATE 組み込み関数および REPATTERN 組み込み関数は SQL ヌル日付 (*year*、*day*、および *month* はすべて 1) を有効な日付として受け入れます。

デフォルトは NONULLDATE です。

NUMBER

NUMBER オプションは、ソース・プログラム内のステートメントを、そのステートメントの取得元であるファイルの行番号とファイル番号で識別すること、およびこの番号ペアを使用して、
AGGREGATE、ATTRIBUTES、LIST、MAP、OFFSET、SOURCE、および XREF オプションから作成されるコンパイラー・リストにおいてステートメントを識別することを指定します。

リストの終わりのファイル参照テーブルでは、コンパイル時に読み取られるそれぞれの入力ファイルに割り当てられた番号を示します。



プリプロセッサーを使用している場合は、ソース・リストの複数行が同じ行番号およびファイル番号で識別されることがあります。例えば、ほとんどすべての EXEC CICS ステートメントがソース・リストに複数のコード行を生成しますが、そのコードはすべて 1 つの行番号とファイル番号で識別されます。

LIST オプションで生成される疑似アセンブラー・リストでは、最初のファイルのファイル番号がブランクのままにしておかれます。

NUMBER と STMT は相互に排他的であり、一方を指定すると他方が無効になります。

デフォルトは NUMBER です。

OBJECT

OBJECT オプションは、コンパイラーがオブジェクト・モジュールを作成することを指定します。 バッチ z/OS 環境では、コンパイラーはオブジェクトを、SYSLIN DD によって定義されたデータ・セットに格納します。 z/OS UNIX では、コンパイラーは、.o ファイルを作成します。

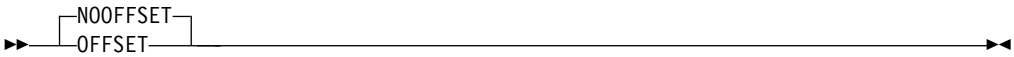
省略形: OBJ、NOBJ

NOOBJECT オプションを指定すると、コンパイラーはオブジェクト・モジュールを作成しません。 ただし、NOOBJECT オプションが指定されると、コンパイラーはすべての未初期化変数の検出を行うだけでなく、すべての構文セマンティック解析フェーズも実行するため、NOCOMPILE オプション、NOSEMANTIC オプション、または NOSYNTAX オプションが指定された場合よりも多くのメッセージが生成される可能性があります。

NOOBJECT オプションを指定すると、LIST、MAP、OFFSET、および STORAGE オプションは無視されます。

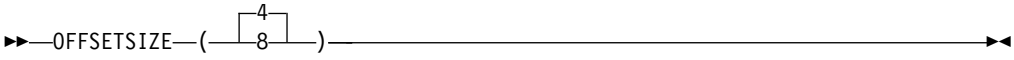
OFFSET

OFFSET オプションは、コンパイラーがそれぞれのプロシージャーと BEGIN ブロックについて、プロシージャーの 1 次エントリー・ポイントから相対的なオフセット・アドレスを付けて、行番号のテーブルを表示することを指定します。 このテーブルは、GONUMBER オプションが使用されていない時に、ランタイム・エラー・メッセージからステートメントを識別するために使用できます。



OFFSETSIZE

OFFSETSIZE オプションは 64 ビット・アプリケーションにおける OFFSET 変数のサイズを決定します。



4 OFFSETSIZE(4) では、すべての OFFSET 変数のサイズが 4 バイトになります。これはデフォルトです。

8 OFFSETSIZE(8) では、すべての OFFSET 変数のサイズが 8 バイトになります。

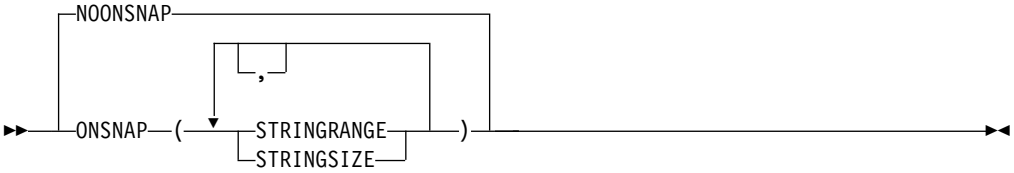
OFFSET 変数または AREA 変数のいずれかを共用するすべてのコードを、OFFSETSIZE オプションに対して同じ値を設定してコンパイルする必要があります。

OFFSETSIZE オプションは、LP(32) オプションが有効になっていると無視されます。

ONSNAP

OPTIONS(MAIN) 属性または OPTIONS(FROMALIEN) 属性を持つ PROCEDURE の場合、ONSNAP オプションは、コンパイラーが `ON STRINGRANGE SNAP;` ステー

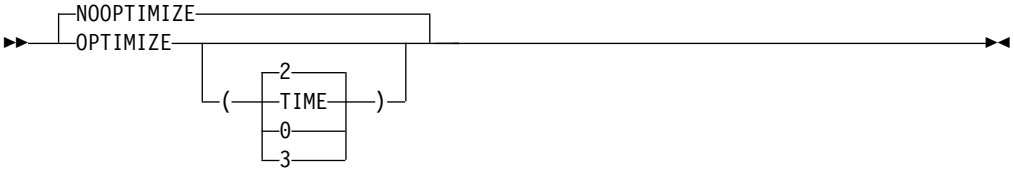
トメントと ON STRINGSIZE SNAP; ステートメントの一方または両方をその PROCEDURE のプロローグ・コードに挿入するように指定します。これにより、このような PROCEDURE から呼び出された他のルーチンで対応する条件が発生した場合に呼び出しチェーンを突き止めやすくなります。



ONSNAP オプションは、これらの属性のいずれかを持たない PROCEDURE には何の影響もありません。

OPTIMIZE

OPTIMIZE オプションでは、必要な最適化のタイプを指定します。



省略形: OPT、NOPT

OPTIMIZE(0)
高速コンパイルを指定しますが、最適化は禁止します。

OPTIMIZE(2)
さらに効率のよいオブジェクト・プログラムが作成されるために、生成された機械命令を最適化します。 このタイプの最適化により、オブジェクト・モジュールに必要な主記憶域の大きさを削減することもできます。

OPTIMIZE(3)
OPTIMIZE(2) のもとで行われたすべての最適化に加えて、追加の最適化を実行します。 OPTIMIZE(3) では、コンパイラーは通常、特に大容量ブロックで多くの変数を持つプログラムに対し、より小さく効率のよいオブジェクト・コードを生成します。 しかし、OPTIMIZE(3) を使用してコンパイルを行うことは、OPTIMIZE(2) を使用した場合よりもかなり多くの時間および領域を必要とすることにもなります。

OPTIMIZE オプションと一緒に、DFT(REORDER) オプションを使用することを強くお勧めします。 実際、以下の条件がすべて当てはまる場合、PROCEDURE ブロックまたは BEGIN ブロックに対する OPTIMIZE の効果は大幅に限定されます。

- ORDER オプションがブロックに適用されている。
- ハードウェアによって検出される条件 (ZERODIVIDE など) に対応する ON ユニットがブロックに含まれている。
- ブロックに、これらの ON ユニットからの分岐のターゲットになる (可能性がある) ラベルがある。

OPTIMIZE(2) を指定すると、NOOPTIMIZE の場合よりコンパイル時間が大幅に増えることがあり、所要スペースが大幅に増えることがあります。 例えば、OPTIMIZE(2) を指定して大規模なプログラムをコンパイルするには数分かかる場合があり、100M 以上の領域が必要になる可能性があります。

OPTIMIZE(3) を使用すると、OPTIMIZE(2) を使用する場合よりもコンパイルに必要な時間と領域が増加します。大規模なプログラムの場合、OPTIMIZE(3) でのプログラムのコンパイル時間は、OPTIMIZE(2) で必要とされる時間の 2 倍を超える可能性があります。

最適化中、コンパイラーは実行時効率を高めるために、コードを移動することができます。 その結果、プログラム・リスト中のステートメント番号が、ランタイム・メッセージで使用されるステートメント番号と対応しなくなることがあります。

NOOPTIMIZE は OPTIMIZE(0) と同等です。

OPTIMIZE(TIME) は、OPTIMIZE(2) と同等です。

OPTIMIZE(2) または OPTIMIZE(3) を使用すると、以下のように TEST オプションの機能が大きく制限されることに注意してください。

- TEST の HOOK サブオプションが有効になっている場合は、ブロック・フックのみが生成されます。
- TEST の NOHOOK サブオプションが有効になっている場合は、変数をリストしたり変更したりしようとすると失敗する可能性があります (変数が最適化されてレジスターに登録されている可能性があるため) し、特定のステートメントで停止しようとするとデバッガーが何度も停止する可能性があります (そのステートメントが複数の部分に分割されている可能性があるため)。

PREFIX オプションを 1 つ以上のチェックアウト条件 (SIZE、STRINGRANGE、STRINGSIZE、および SUBSCRIPTRANGE) と一緒に使用すると、コンパイルに必要とされる時間とスペースが大幅に増える可能性があります。

関連情報:

373 ページの『第 15 章 パフォーマンスの向上』

ユーザーのプログラムの速度を向上することに関する多数の考慮事項は、 使用するコンパイラーとそれを実行するプラットフォームには関係ありません。 しかし、この章では、考慮事項の中でも PL/I コンパイラーとそれによって生成される コードに特有な考慮事項を識別して説明します。

OPTIONS

OPTIONS オプションは、このコンパイル中に使用されるコンパイラー・オプションを示したリストを、コンパイル・リスト内にコンパイラーが組み込むことを指定します。

省略形: OP、NOP

このリストには、デフォルトで適用されたすべてのオプション、EXEC ステートメントの PARM パラメーターまたは呼び出しコマンド (pli) で指定されたオプション、%PROCESS ステートメントで指定されたオプション、z/OS の下で IBM_OPTIONS 環境変数で指定されたオプション、および任意のオプション・ファイルから取り込まれたすべてのオプションが含まれます。

OPTIONS(DOC) を指定すると、OPTIONS リストには、コンパイラーがリリースされた時点でこの文書に記述されたオプション (およびサブオプション) のみが組み込まれます。

OPTIONS(ALL) を指定すると、OPTIONS リストには、コンパイラーがリリースされた後、PTF によって追加されたオプションも組み込まれます。

OR

OR オプションでは、論理 OR 演算子として最大 7 つの代替記号を指定します。これらの記号は連結演算子としても使用されます。連結演算子は 2 つの連続した論理和記号と定義されます。

注: 引用符と引用符の間にブランクをコーディングしないでください。

OR 記号 (|) の IBM 提供のデフォルト・コード・ポイントは '4F'X です。

char
単一の SBCS 文字

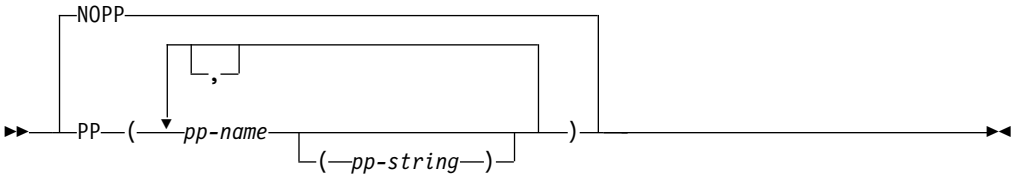
標準の論理 OR 記号 (|) を除き、英字、数字、および「*PL/I* 言語解説書」に定義されている特殊文字はどれも指定できません。 少なくとも有効な文字を 1 文字指定する必要があります。

OR オプションを指定すると、標準 OR 記号は、文字ストリング内の 1 つの文字として指定しない限り認識されなくなります。

例えば、OR('¥') を指定すると、円記号 'E0'X が論理 OR 演算子として認識され、2 つの連続した円記号は連結演算子として認識されます。 標準 OR 記号 '|'、'4F'X は、どちらの演算子としても認識されません。 同様に OR('¥|') を指定すると、円記号または標準 OR 記号のどちらかが論理 OR 演算子として認識され、一方または両方の記号を使用して連結演算子を作成できます。

PP

PP オプションは、コンパイル前に呼び出すプリプロセッサー (およびそれらの順序) を指定します。



pp-name
特定のプリプロセッサーに与えられた名前。 現在サポートされているプリプロセッサーは、CICS、INCLUDE、MACRO、および SQL だけです。 未定義の名前を使用すると診断エラーの原因となります。

pp-string
対応するプリプロセッサーのオプションを表す、引用符で区切られた 100 文字以内のストリング。 例えば、PP(MACRO('CASE(ASIS)')) は、オプション CASE(ASIS) を指定してマクロ・プリプロセッサーを呼び出します。

プリプロセッサー・オプションは左から右へ処理されます。2 つのオプションが対立する場合は、最後の (右端の) オプションが使用されます。 例えば、オプション・ストリング 'CASE(ASIS) CASE(UPPER)' を指定してマクロ・プリプロセッサーを呼び出した場合は、オプション CASE(UPPER) が使用されます。

最大 31 のプリプロセッサー・ステップを指定でき、同じプリプロセッサーを複数回指定できます (CICS および SQL のプリプロセッサーを除く)。CICS プリプロセッサーを呼び出すことができるのは 1 回まで、SQL プリプロセッサーは 2 回までです。 SQL プリプロセッサーは、最初の指定で INCONLY がオプションとして指定された場合にのみ、2 回呼び出すことができます。

PP オプションとともに MACRO オプションが指定された場合は、MACRO プリプロセッサーが PP オプションにおけるプリプロセッサーのリストの先頭に追加されます (まだリストの先頭に置かれていない場合)。そのため、MACRO と PP(SQL MACRO) が指定されている場合は、PP オプションが PP(+(MACRO SQL MACRO) となり、MACRO プリプロセッサーが 2 回呼び出されます。ただし、MACRO と PP(MACRO SQL) が指定された場合は、PP オプションは変更されずに、MACRO プリプロセッサーが 1 回のみ呼び出されます。ただし、同じコンパイルで MACRO オプションと PP(MACRO) オプションを両方とも使用することは勧められません。

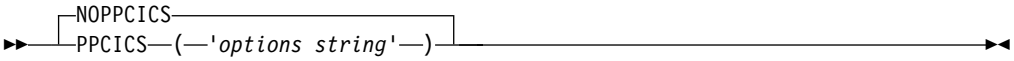
PP オプションを複数回指定した場合、コンパイラーはそれらを実質的に連結します。 したがって、PP(SQL) PP(CICS) を指定することは、PP(SQL CICS) を指定するのと同じことになります。これはまた、PP(MACRO SQL('CCSID0')) と PP(MACRO SQL('CCSID0 DATE(ISO)')) を指定すると、PP オプションは結果として PP(MACRO SQL('CCSID0') MACRO SQL('CCSID0 DATE(ISO)')) となり、MACRO プリプロセッサーと SQL プリプロセッサーの両方が 2 回呼び出され、SQL プリプロセッサーの 2 回目の呼び出しがエラーになることも意味しています。前の SQL オプションを指定変更するためにこれを行うのであれば、プリプロセッサー・オプションを PP オプションで指定するのではなく PPSQL オプションで指定する (つまり、PP(MACRO SQL) PPSQL('CCSID0 DATE(ISO)') を指定する) 方がよい場合があります。

関連情報:

131 ページの『第 2 章 PL/I プリプロセッサー』
PL/I コンパイラーを使用するときは、ご使用のプログラムにおいて組み込みプリプロセッサーを 1 つ以上指定できます。組み込みプリプロセッサー、マクロ・プリプロセッサー、SQL プリプロセッサー、または CICS プリプロセッサーを指定できます。 また、これらのプリプロセッサーを呼び出す順序を指定できます。

PPCICS

PPCICS オプションは、CICS プリプロセッサーが呼び出される場合にそれに渡すオプションを指定します。



PPCICS('EDF') PP(CICS) と指定することは、PP(CICS('EDF')) と指定することと同じです。

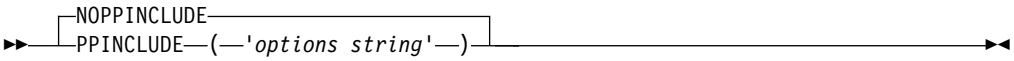
このオプションは、PP(CICS) オプションが指定されなければ有効ではありません。ただし、CICS プリプロセッサーが呼び出されるときに使用される CICS プリプロセッサー・オプションのセットを指定する場合は、インストール・オプションの出口でこのオプションを指定することができます。その結果、PP(CICS) を指定すると必ず、PPCICS オプションで指定されたオプションのセットが使用されます。

また、PPCICS オプションで指定されたオプションは、プリプロセッサーが呼び出されるときに指定されるオプションの形に変更されます。 したがって、PPCICS('EDF') PP(CICS('NOEDF')) を指定することは、PP(CICS('EDF NOEDF'))、あるいはより単純な PP(CICS('NOEDF')) を指定するのと同じことになります。

オプション・ストリングの長さは 1000 文字に制限されます。 ただし、ストリングは、100 文字を超える場合、オプション・リストに表示されません。

PPINCLUDE

PPINCLUDE オプションは、INCLUDE プリプロセッサーが呼び出される場合にそれに渡すオプションを指定します。



PPINCLUDE('ID(-inc)') PP(INCLUDE) と指定することは、PP(INCLUDE('ID(-inc)')) と指定することと同じです。

このオプションは、PP(INCLUDE) オプションが指定されなければ有効ではありません。ただし、INCLUDE プリプロセッサーが呼び出された場合に使用されるべき INCLUDE プリプロセッサー・オプションのセットを指定したい場合は、このオプションをインストール・オプション出口で指定します。その結果、PP(INCLUDE) を指定すると必ず、PPINCLUDE オプションで指定されたオプションのセットが使用されます。

また、PPINCLUDE オプションで指定されたオプションは、プリプロセッサーが呼び出されるときに指定されるオプションの形に変更されます。 したがって、PPINCLUDE('ID(-inc)') PP(INCLUDE('ID(+include)')) を指定することは、PP(INCLUDE('ID(-inc) ID(+include)'))、あるいはより単純な PP(INCLUDE('ID (+include)')) を指定するのと同じことになります。

オプション・ストリングの長さは 1000 文字に制限されます。 ただし、ストリングは、100 文字を超える場合、オプション・リストに表示されません。

PPLIST

PPLIST オプションは、各プリプロセッサー・フェーズで生成されるリスト部分をコンパイラーが保持するのか削除するのかを制御します。

PPLIST(KEEP) が指定されている場合、コンパイラーは各プリプロセッサー・フェーズで生成されるリスト部分を保持します。

PPLIST(ERASE) が指定されている場合、コンパイラーは、メッセージを出力しないプリプロセッサー・フェーズで生成されるリスト部分を削除します。

コンパイラーは、EXIT オプションおよび FLAG オプションで抑制されているメッセージを対象としません。 そのため、FLAG(W) と PPLIST(ERASE) の両方が指定されている場合、コンパイラーは、警告メッセージ、エラー・メッセージ、重大メッセージのいずれも生成しないプリプロセッサーからの出力をすべて抑制します。

PPLIST(KEEP) がデフォルトです。

PPMACRO

PPMACRO オプションは、マクロ・プリプロセッサーが呼び出される場合にそれに渡すオプションを指定します。

PPMACRO('CASE(ASIS)') PP(MACRO) を指定することは、PP(MACRO('CASE (ASIS)')) と指定することと同じです。

このオプションは、PP(MACRO) オプションが指定されなければ有効ではありません。ただし、マクロ・プリプロセッサーが呼び出されるときに使用されるマクロ・プリプロセッサー・オプションのセットを指定する場合は、インストール・オプションの出口でこのオプションを指定することができます。その結果、MACRO オプションまたは PP(MACRO) オプションを指定すると必ず、PPMACRO オプションで指定されたオプションのセットが使用されます。

また、PPMACRO オプションで指定されたオプションは、プリプロセッサーが呼び出されるときに指定されるオプションの形に変更されます。 したがって、

PPMACRO('CASE(ASIS)') PP(MACRO('CASE(UPPER)')) を指定することは、PP(MACRO('CASE(ASIS) CASE(UPPER)'))、あるいはより単純な PP(MACRO('CASE(UPPER)'))を指定するのと同じことになります。

オプション・ストリングの長さは 1000 文字に制限されます。 ただし、ストリングは、100 文字を超える場合、オプション・リストに表示されません。

PPSQL

PPSQL オプションは、その SQL プリプロセッサーに渡されるオプションを指定します。

PPSQL('APOSTSQL') PP(SQL) を指定することは、PP(SQL('APOSTSQL')) を指定するのと同じことになります。

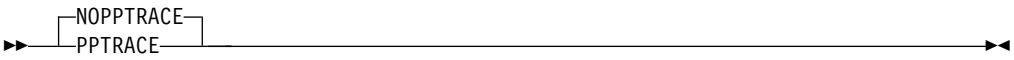
このオプションは、PP(SQL) オプションが指定されなければ有効ではありません。ただし、SQL プリプロセッサーが呼び出される場合に使用されるオプションのセットを指定する場合は、インストール・オプションの出口でこのオプションを指定することができます。こうすると、PP(SQL) を指定した時点で、 PPSQL オプションでのオプションのセットが使用されるようになります。

また、PPSQL オプションで指定されたオプションよりも、プリプロセッサーが呼び出されるときに指定されるオプションの方が優先されます。したがって、PPSQL('APOSTSQL') PP(SQL('QUOTESQL')) を指定することは、PP(SQL('APOSTSQL QUOTESQL'))、またはさらに単純な PP(SQL('QUOTESQL')) を指定するのと同じことになります。

オプション・ストリングの長さは 1000 文字に制限されます。ただし、ストリングは、100 文字を超える場合、オプション・リストに表示されません。

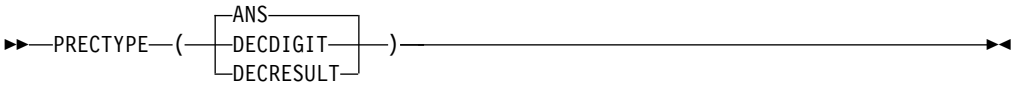
PPTRACE

PPTRACE オプションを指定すると、プリプロセッサー用にデック・ファイルが書き出されるとき、そのファイルの中の各非ブランク行の前に %LINE ディレクティブの行が追加されます。このディレクティブは、その非ブランク行が帰属するオリジナルのソース・ファイルと行を示します。



PRECTYPE

PRECTYPE オプションは、オペランドが FIXED であり、少なくとも 1 つが FIXED BIN である場合に、MULTIPLY、DIVIDE、ADD、および SUBTRACT 組み込み関数の属性をコンパイラーが取り出す方法を決定します。



ANS

PRECTYPE(ANS) では、BIF(*x,y,p*) および BIF(*x,y,p,*0) の値 *p* は、2 進数字を指定するものとして解釈されます。演算は 2 項演算として行われます。その結果には属性 FIXED BIN(*p,*0) が付与されます。

ただし、BIF(*x,y,p,q*) で *q* がゼロではない場合、演算は 10 進演算として行われ、その結果には属性 FIXED DEC(*t,u*) が付与されます。 *t* および *u* は *p* および *q* の 10 進数に相当するもの (すなわち、*t* = 1 + ceil(*p* / 3.32) および *u* = ceil(*q* / 3.32)) です。この場合、*x*、*y*、*p*、および *q* は、実際にすべて 10 進数に変換されます (これとは対照的に、DECDIGIT サブオプションでは、*x* と *y* のみが 10 進数に変換されます (*q* がゼロであっても 10 進数に変換されます))。この状態では、コンパイラーは通知メッセージ 1BM1053 を出します。

DECDIGIT

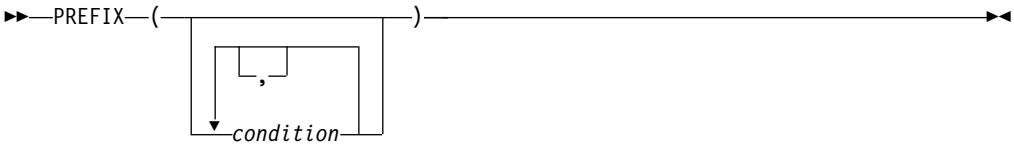
PRECTYPE(DECDIGIT) では、BIF(*x,y,p*) および BIF(*x,y,p,*0) の値 *p* は、10 進数字を指定するものとして解釈されます。演算は 2 項演算として行われます。その結果には属性 FIXED BIN(*s*) が付与されます。 *s* は *p* に対応する 2 進数に相当します (すなわち *s* = ceil(3.32**p*))。 BIF(*x,y,p,q*) の場合 (*q* はゼロ以外)、PRECTYPE(DECDIGIT) での結果は、下で説明する PRECTYPE (DECRESULT) の結果と同じです。

DECRESULT

PRECTYPE(DECRESULT) では、BIF(*x,y,p*) の値 *p*、および BIF(*x,y,p,q*) の値 *p* と *q* は 10 進数字を指定するものとして解釈されます。演算は 10 進演算として行われます。その結果には属性 FIXED DEC(*p,*0) または FIXED DEC(*p,q*) がそれぞれ付与されます。結果は、DECIMAL 組み込み関数が *x* と *y* に適用された場合に生成されるものと同じになります。

PREFIX

PREFIX オプションを指定すると、ソース・プログラムの変更を必要とせずに、指定した PL/I 条件をコンパイル中のコンパイル単位の中で使用可能にしたり使用不可にしたりできます。指定した条件接頭語は、最初の PACKAGE ステートメントまたは PROCEDURE ステートメントの先頭に付けられます。



condition

PL/I プログラムにおいて有効/無効にできる任意の条件 (「*PL/I* 言語解説書」を参照)。

PREFIX オプションを 1 つ以上のチェックアウト条件
(SIZE、STRINGRANGE、STRINGSIZE、および SUBSCRIPTRANGE) と一緒に使用すると、コンパイルに必要とされる時間とスペースが大幅に増える可能性があります。

デフォルト: PREFIX(CONVERSION FIXEDOVERFLOW INVALIDOP OVERFLOW NOSIZE NOSTRINGRANGE NOSTRINGSIZE NOSUBSCRIPTRANGE UNDERFLOW ZERODIVIDE)

PROCEED

PROCEED オプションを指定すると、前にプリプロセッサーが発行したメッセージの重大度に応じて、プリプロセッサーによる処理の完了後にコンパイラーが停止します。

省略形: PRO、NPRO

PROCEED
NOPROCEED(S) と同等です。

NOPROCEED
プリプロセッサーがコンパイルを終えた後、処理を停止します。

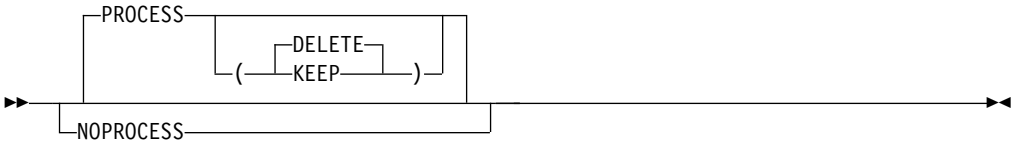
NOPROCEED(S)
このプリプロセスの段階で重大エラーまたは回復不能エラーが検出された場合は、プリプロセッサーおよびコンパイラーの呼び出しは継続されません。

NOPROCEED(E)
このプリプロセスの段階でエラー、重大エラー、または回復不能エラーが検出された場合は、プリプロセッサーおよびコンパイラーの呼び出しは継続されません。

NOPROCEED(W)
このプリプロセスの段階で警告、エラー、重大エラー、または回復不能エラーが検出された場合は、プリプロセッサーやコンパイラーの呼び出しは継続されません。

PROCESS

PROCESS オプションは、*PROCESS ステートメントが許可されるかどうか、および許可される場合に、それらのステートメントが MDECK ファイルに書き込まれるかどうかを決定します。



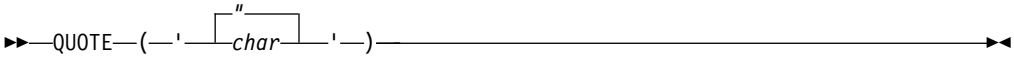
NOPROCESS オプションを指定すると、コンパイラーは、すべての *PROCESS ステートメントに E レベル・メッセージのフラグを立てます。

PROCESS(KEEP) オプションを指定すると、コンパイラーは *PROCESS ステートメントにフラグを立てることをせず、すべての *PROCESS ステートメントがコンパイラーによって MDECK 出力に保存されます。

PROCESS(DELETE) オプションを指定すると、コンパイラーは *PROCESS ステートメントにフラグを立てることをせず、どの *PROCESS ステートメントもコンパイラーによって MDECK 出力に保存されません。

QUOTE

QUOTE オプションは、引用符文字として使用可能な代替記号を 1 つ指定します。



注: 引用符と引用符の間にブランクをコーディングしないでください。

QUOTE 記号用に IBM が提供するデフォルト・コード・ポイントは '"' です。

char
単一の SBCS 文字

標準の QUOTE 記号 (") を除き、「*PL/I* 言語解説書」に定義されている特殊文字、数字、および英字はいずれも指定できません。

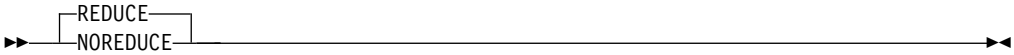
有効な文字を 1 つ指定する必要があります。

QUOTE オプションは、GRAPHIC も指定されている場合は無視されます。

REDUCE

REDUCE オプションは、埋め込みバイトに上書きすることになったとしても、構造体へのヌル・ストリングの割り当てを減らしてより単純な操作にすることをコンパイラーに許可します。

また REDUCE オプションは、構造体に POINTER フィールドが含まれていても、一致する構造体の割り当てを減らして単純な集合移動にすることをコンパイラーに許可します。



NOREDUCE オプションを指定した場合、コンパイラーは構造体に対するヌル・ストリングの割り当てを分解して、構造体の基本メンバーに対してヌル・ストリングを連続して割り当てるようになります。

NOREDUCE オプションが指定された場合に、一緒に移動されるエレメントの属性が AREA または VARYING(Z) であると、集合移動に減らすことができる BY NAME 割り当てが減らされません。

REDUCE オプションを使用すると、ヌル・ストリングを構造体に割り当てるために生成されるコードの行数が少なくなり、その結果として通常はコンパイルが高速になり、コードの実行速度が大きく向上します。 しかし、埋め込みバイトはゼロにリセットされることがあります。

例えば次の構造体では、*field12* と *field13* の間に 1 バイトの埋め込みがあります。

```
dcl
 1 sample ext,
   5  field10          bin fixed(31),
   5  field11          bin fixed(15),
   5  field12          bit(8),
   5  field13          bin fixed(31);
```

ここで、割り当て `sample = '';` について考えてみます。

NOREDUCE オプションを指定した場合、4 つの割り当てが生成され、埋め込みバイトは変更されません。

ただし、REDUCE では、割り当ては 3 つの演算に減らされますが、埋め込みバイトはゼロにリセットされます。

NOREDUCE オプションが指定された場合、コンパイラーは OS PL/I コンパイラーや PL/I for MVS コンパイラーに近い動作をします。これらのコンパイラーは、構造体に POINTER フィールドが含まれていなければ、一致する構造体の割り当てを減らして単純な集合移動にします。 NOREDUCE オプションを指定すると、このコンパイラーは同じように動作するようになります。

RENT

コードが静的変数を変更しなければ、そのコードは「本質的に再入可能」です。RENT オプションを指定すると、コンパイラーは本質的に再入可能でないコードを検出して、再入可能にします。

再入可能性について詳しくは、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。 RENT オプションを使用する場合は、生成されたオブジェクト・モジュールをリンケージ・エディターで直接処理することはできないため、PDSE を使用する必要があります。

注**:** LP(64) オプションでは、RENT オプションは無視されます。実際上は、RENT が常にオンになります。

NORENT オプションを指定すると、コンパイラーは再入不可コードから再入可能コードを特に生成しません。本質的に再入可能なコードは、そのまま再入可能です。

RENT オプションを指定してコンパイルされたプログラムを 1 つ以上含むモジュール (MAIN または FETCHABLE のどちらか) をリンクする場合は、リンク・ステップで DYNAM=DLL および REUS=RENT を指定する必要があります。

NORENT および LIMITS(EXTNAME(*n*)) (*n* <= 7 を指定) オプションを指定した場合、コンパイラーによって生成されるテキスト・デックのフォーマットは、従来の PL/I コンパイラーによって生成されるものと同じです。他のオプションを使用する場合は、PDSE を使用する必要があります。

NORENT オプションで生成されたコードは、NOWRITABLE オプションも指定していなければ再入可能にはなりません。

NORENT を使用すると、コンパイラーの機能の一部が使用できなくなります。 特に、以下の考慮事項に注意してください。

- DLL は構築できません。
- 再入可能な書き込み可能静的変数はサポートされません。
- STATIC ENTRY VARIABLE は INITIAL 値を持つことはできません。

次の制約事項に従う RENT と NORENT コードは、混在できます。

- RENT を指定してコンパイルされたコードは、EXTERNAL STATIC 変数を共用している場合、NORENT でコンパイルされたコードと混在できません。
- RENT を指定してコンパイルされたコードは、NORENT でコンパイルされたコード内の ENTRY VARIABLE セットを呼び出すことができません。
- RENT を指定してコンパイルされたコードは、NORENT でコンパイルされたコードでフェッチされた ENTRY CONSTANT を呼び出すことができません。
- RENT を指定してコンパイルされたコードは、以下のいずれかの条件が当てはまれば、NORENT を指定してコンパイルされたコードを含むモジュールをフェッチできます。
  - フェッチされたモジュールのすべてのコードが NORENT でコンパイルされている。
  - モジュールへのエントリー・ポイントを含んでいるコードが RENT でコンパイルされている。
- NORENT コードを指定してコンパイルされたコードは、RENT でコンパイルされた任意のコードを含むモジュールをフェッチできません。
- NORENT WRITABLE を指定してコンパイルされたコードは、任意の外部 CONTROLLED 変数または任意の外部 FILE を共用している場合、NORENT NOWRITABLE でコンパイルされたコードと混在できません。

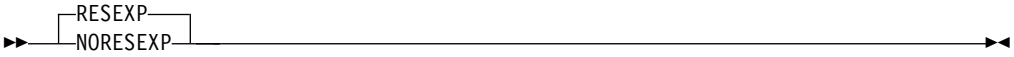
上記の制約事項に従えば、以下を行うこともできます。

- 例えば `mnorent` という NORENT ルーチンと、`mrent` という RENT ルーチンを静的にリンクして呼び出します。

- そうすると、RENT ルーチン mrent が、別にリンクされたモジュール (RENT を指定してコンパイルされたエントリー・ポイントを持つ) をフェッチして呼び出します。

RESEXP

RESEXP オプションは、これによってある条件が発生してコンパイルが S レベル・メッセージで終了するとしても、コンパイラーがコンパイル時にすべての制限つき式を評価できるように指定します。



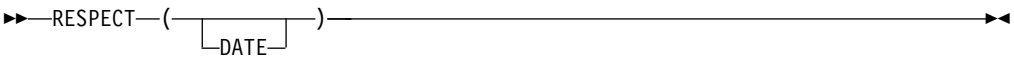
NORESEXP コンパイラー・オプションでは、コンパイラーは、INITIAL 値文節の式も含めて宣言されたすべての制限付き式を評価します。

例えば、NORESEXP オプションでは、コンパイラーは次のステートメントにフラグを立てません (ZERODIVIDE 例外が実行時に発生します)。

```
if preconditions_not_met then
   x = 1 / 0;
```

RESPECT

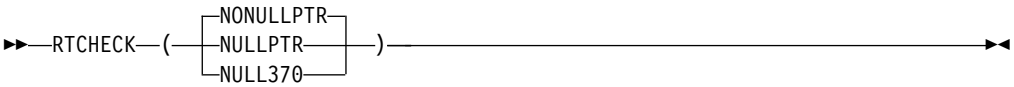
RESPECT オプションを指定すると、コンパイラーは DATE 属性のすべての指定を受け入れ、DATE 組み込み関数の結果に DATE 属性を適用します。



デフォルトの RESPECT() を使用すると、コンパイラーは DATE 属性の任意の指定を無視し、DATE 組み込み関数の結果に DATE 属性が適用されなくなります。

RTCHECK

RTCHECK オプションは、ヌル・ポインターが逆参照された場合 (つまり、そのポインターが変数の値を変更または取得するために使用された場合) に強制的に ERROR 条件を発生させるための追加コードが生成されるように指定します。



NULLPTR

NULL ポインター (つまり SYSNULL() に相当するポインター) が逆参照された場合に、強制的に ERROR 条件を発生させるための追加コードが生成されます。

NULL370

DEFAULT(NULL370) 値に相当するポインターが逆参照された場合に、強制的に ERROR 条件を発生させるための追加コードが生成されます。この値は、

LP(32) では 'FF000000'x の 16 進値であり、LP(64) では '00000000_7FFFFBAD'x の 16 進値です。

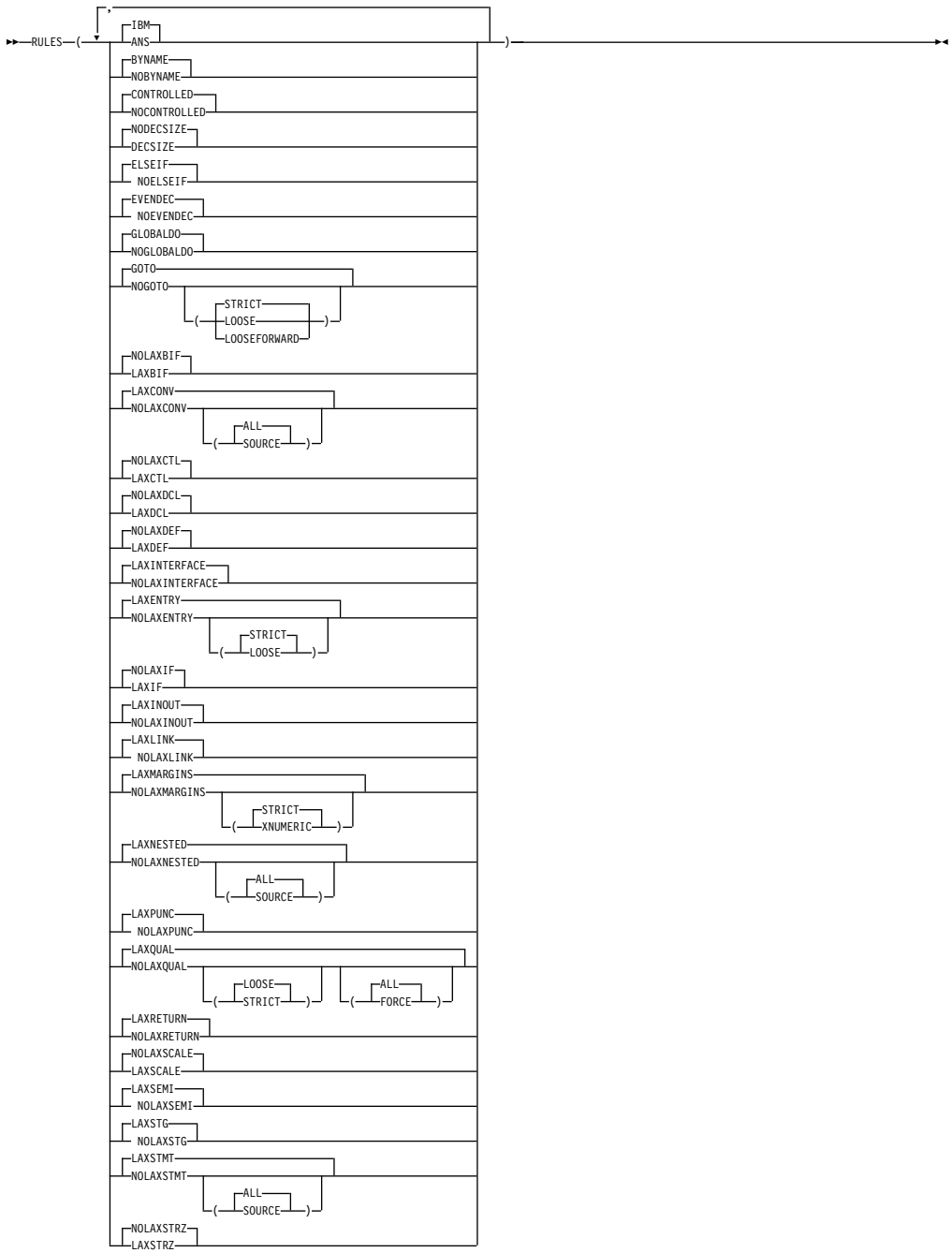
NONULLPTR
NULL ポインターが逆参照された場合に強制的に ERROR 条件を発生させるための追加コードは生成されません。

デフォルトは RTCHECK(NONULLPTR) です。

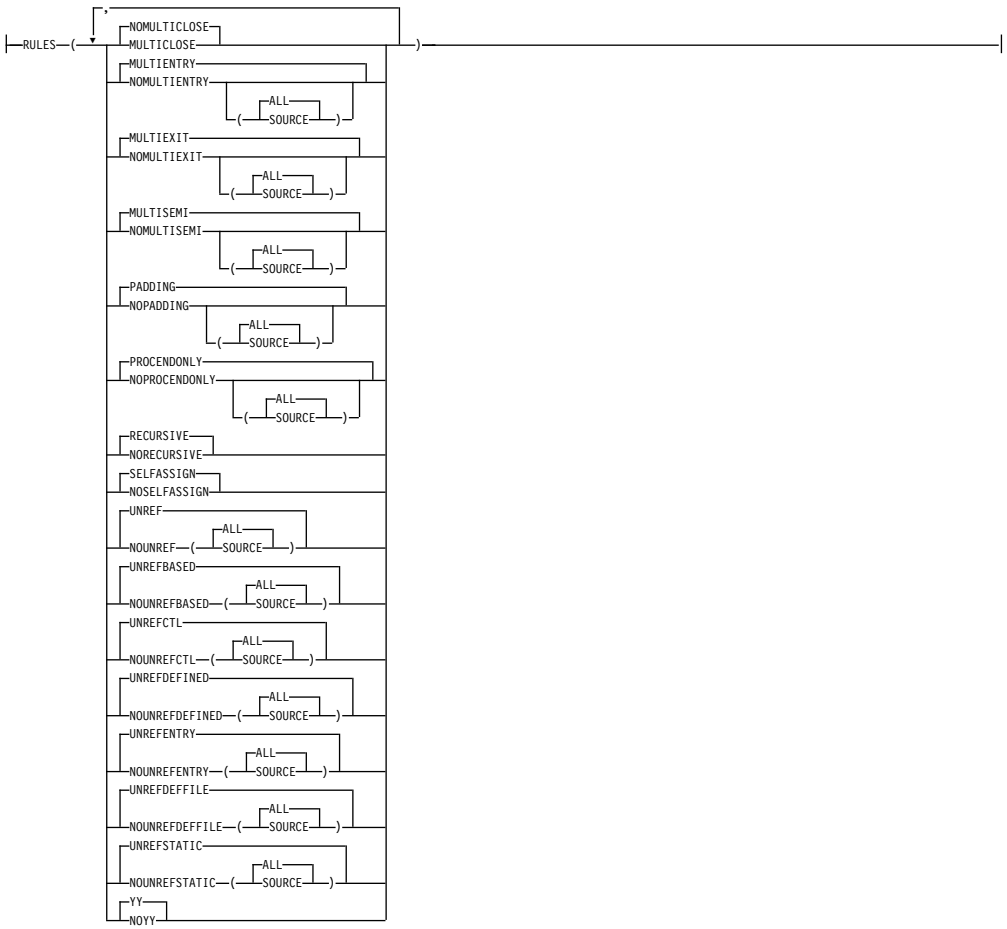
注: NULL ポインターが逆参照された場合に比較トラップ・データ例外が発生します。

RULES

RULES オプションを指定すると、ある種の言語機能を使用可能または使用禁止にすることができ、代替の選択肢があればセマンティクスを選択できます。 これは一般プログラミング・エラーの診断に役立ちます。


RULES
IBM
ANS
BYNAME
NOBYNAME
CONTROLLED
NOCONTROLLED
NODECSIZE
DECSIZE
ELSEIF
NOELSEIF
EVENDEC
NOEVENDEC
GLOBALDO
NOGLOBALDO
GOTO
NOGOTO
STRICT
LOOSE
LOOSEFORWARD
NOLAXBIF
LAXBIF
LAXCONV
NOLAXCONV
ALL
SOURCE
NOLAXCTL
LAXCTL
NOLAXDCL
LAXDCL
NOLAXDEF
LAXDEF
LAXINTERFACE
NOLAXINTERFACE
LAXENTRY
NOLAXENTRY
STRICT
LOOSE
NOLAXIF
LAXIF
LAXINOUT
NOLAXINOUT
LAXLINK
NOLAXLINK
LAXMARGINS
NOLAXMARGINS
STRICT
XNUMERIC
LAXNESTED
NOLAXNESTED
ALL
SOURCE
LAXPUNC
NOLAXPUNC
LAXQUAL
NOLAXQUAL
LOOSE
STRICT
ALL
FORCE
LAXRETURN
NOLAXRETURN
NOLAXSCALE
LAXSCALE
LAXSEMI
NOLAXSEMI
LAXSTG
NOLAXSTG
LAXSTMT
NOLAXSTMT
ALL
SOURCE
NOLAXSTRZ
LAXSTRZ

RULES:



IBM | ANS

IBM サブオプションの場合:

- ストリング・データを必要とする演算では、BINARY 属性を持つデータは BIT に変換されます。
- 算術演算または比較での変換は、「*PL/I* 言語解説書」で説明されているように行われます。
- ADD、DIVIDE、MULTIPLY、および SUBTRACT 組み込み関数の場合の変換は、スケールされた固定 2 進数として指定された演算がスケールされた固定 10 進数として評価されることを除けば、「*PL/I* 言語解説書」で説明されているように行われます。
- FIXED BIN 宣言では、ゼロ以外のスケール因数が使用できます。
- 精度処理組み込み関数 (ADD や BINARY など) の結果として FIXED BIN 属性が付与される場合は、ゼロ以外のスケール因数を指定または暗黙指定できます。
- MAX または MIN 組み込み関数への引数がすべて UNSIGNED FIXED BIN の場合でも、すべての引数が SIGNED に変換され、その関数はこれらの変換された引数で評価され、結果は常に SIGNED になります。
- 2 つの UNSIGNED FIXED BIN オペランドを使用して加算、乗算、または除算を行ったときでも、すべてのオペランドが SIGNED に変換され、その演算はこれらの変換された引数で評価され、結果は SIGNED 属性を持ちます。

- 2 つの UNSIGNED FIXED BIN オペランドに MOD または REM 組み込み関数を適用したときでも、すべての引数が SIGNED に変換され、その関数はこれらの変換された引数で評価され、結果は SIGNED 属性を持ちます。
- OPTIONS 属性を持つ変数を宣言すると、ENTRY 属性が暗黙指定されます。

ANS サブオプションの場合:

- ストリング・データを必要とする演算では、BINARY 属性を持つデータは CHARACTER に変換されます。
- 算術演算または比較での変換は、「*PL/I* 言語解説書」で説明されているように行われます。
- ADD、DIVIDE、MULTIPLY、および SUBTRACT 組み込み関数の場合の変換は、「*PL/I* 言語解説書」で説明されているように行われます。
- ゼロ以外のスケール因数は、FIXED BIN 宣言では使用できません。
- 精度処理組み込み関数 (ADD や BINARY など) の結果として FIXED BIN 属性が付与される場合は、指定または暗黙指定するスケール因数はゼロでなければなりません。
- MAX または MIN 組み込み関数の引数がすべて UNSIGNED FIXED BIN の場合でも、結果は常に UNSIGNED になります。
- 2 つの UNSIGNED FIXED BIN オペランドを用いて加算、乗算、または除算を行った場合でも、結果は UNSIGNED 属性を持ちます。
- 2 つの UNSIGNED FIXED BIN オペランドに MOD または REM 組み込み関数を適用した場合でも、結果は UNSIGNED 属性を持ちます。
- OPTIONS 属性を持つ変数を宣言すると、ENTRY 属性は暗黙指定されません。

また、RULES(ANS) では、旧コンパイラーでは無視されていた以下のエラーにより E レベル・メッセージが生成されます。

- ストリング定数を STRING 組み込み関数の引数として指定している。
- 配列参照内の添え字として指定しているアスタリスクが多すぎる。
- CONTROLLED 変数を POINTER 変数で修飾している (CONTROLLED 変数が BASED であったかのように)。

デフォルトは RULES(IBM) です。

BYNAME | NOBYNAME

NOBYNAME を指定すると、コンパイラーは E レベル・メッセージを伴うすべての BYNAME 割り当てにフラグを付けます。

デフォルトは RULES(BYNAME) です。

CONTROLLED | NOCONTROLLED

NOCONTROLLED を指定した場合、コンパイラーが CONTROLLED 属性のすべての使用に対してフラグを立てます。

CONTROLLED を指定した場合、コンパイラーは CONTROLLED 属性の使用に対してフラグを立てません。

デフォルトは RULES(CONTROLLED) です。

DECSIZE | NODECSIZE

DECSIZE を指定すると、代入によって SIZE 条件が発生した場合に SIZE 条件が使用不可であると、コンパイラーは FIXED DECIMAL 変数に対する FIXED DECIMAL 式のすべての代入にフラグを立てます。

RULES(DECSIZE) が指定されると、コンパイラーは多くのメッセージを出力することがあります。これは、SIZE が無効になっていると X = X + 1 形式のすべてのステートメントにフラグが立てられるためです (X が FIXED DECIMAL の場合)。

デフォルトは RULES(NODECSIZE) です。

ELSEIF | NOELSEIF

NOELSEIF を指定すると、直後に IF ステートメントが続く ELSE ステートメントに対しコンパイラーがフラグを立て、SELECT ステートメントとして書き直すように提案します。

一連のネストされた IF-THEN-ELSE ステートメントではなく SELECT ステートメントの使用を実施する場合に、このオプションを使用すると便利です。

また RULES(NOELSEIF) コンパイラー・オプションは、単純な DO-END で囲まれた IF ステートメントの直後に続く ELSE ステートメントにフラグを立てるようになりました。

デフォルトは RULES(ELSEIF) です。

EVENDEC | NOEVENDEC

NOEVENDEC を指定すると、コンパイラーは、偶数精度を指定するすべての FIXED DECIMAL 宣言にフラグを立てます。

デフォルトは RULES(EVENDEC) です。

GLOBALDO | NOGLOBALDO

NOGLOBALDO の指定は、コンパイラーに、親ブロックで宣言された制御変数を持つすべての DO ループにフラグを立てるように指示します。

デフォルトは RULES(GLOBALDO) です。

GOTO | NOGOTO

NOGOTO(STRICT) を指定すると、コンパイラーはラベル定数への GOTO ステートメントにフラグを立てます。ただし、その GOTO が ON ユニットを終了する場合を除きます。

NOGOTO(LOOSE) を指定すると、コンパイラーはラベル定数への GOTO ステートメントにフラグを立てます。ただし、その GOTO が ON ユニットを終了する場合や、ターゲットのラベル定数がその GOTO ステートメントと同じブロック内にある場合を除きます。

NOGOTO(LOOSEFORWARD) を指定すると、コンパイラーはラベル定数への GOTO ステートメントにフラグを立てます。ただし、その GOTO が ON ユニットを終了する場合や、ターゲットのラベル定数がその GOTO ステートメントと同じブロック内にあり、GOTO ステートメントの後ろに配置されている場合を除きます。

デフォルトは RULES(GOTO) です。RULES(NOGOTO) を指定した場合のデフォルトは STRICT です。

LAXBIF | NOLAXBIF

LAXBIF を指定すると、空のパラメーター・リストを使用しない場合でも、コンパイラーは NULL のようなコンテキスト宣言を組み込み関数のために作成します。

デフォルトは RULES(NOLAXBIF) です。

LAXCONV | NOLAXCONV

RULES(LAXCONV) を指定した場合、コンパイラーはオペランドに算術型がない演算式にフラグを立てません。

RULES(NOLAXCONV) を指定した場合、コンパイラーはオペランドに算術型がない演算式にフラグを立てます。

ALL

ALL が指定された場合は、RULES(NOLAXCONV) のすべての違反にフラグが立てられます。ALL はデフォルトです。

SOURCE

SOURCE が指定された場合は、1 次ソース・ファイルで発生した違反にのみフラグが立てられます。

デフォルトは RULES(LAXCONV) です。RULES(NOLAXCONV) を指定した場合のデフォルトは ALL です。

LAXCTL | NOLAXCTL

LAXCTL を指定すると、固定エクステントを使用して CONTROLLED 変数を宣言しても、CONTROLLED 変数を異なるエクステントに割り当てることができます。 NOLAXCTL を指定すると、CONTROLLED 変数を異なるエクステントに割り当てる場合に、そのエクステントをアスタリスクとして指定するか、非定数式として指定する必要があります。

NOLAXCTL が指定されていると、次のコードは不正になります。

```
dcl a bit(8) ctl;
alloc a;
alloc a bit(16);
```

ただし、次のコードは NOLAXCTL が指定されていても有効です。

```
dcl b bit(n) ctl;
dcl n fixed bin(31) init(8);
alloc b;
alloc b bit(16);
```

デフォルトは RULES(NOLAXCTL) です。

LAXDCL | NOLAXDCL

LAXDCL を指定すると、暗黙宣言が可能になります。 NOLAXDCL を指定すると、BUILTIN の場合と SYSIN および SYSPRINT ファイルの場合を除いて、暗黙宣言およびコンテキスト宣言はすべて禁止になります。

デフォルトは RULES(NOLAXDCL) です。

LAXDEF | NOLAXDEF

LAXDEF を指定すると、いわゆる無許可の定義が、コンパイラーのメッセージなしに受け入れられます (コンパイラーが通常作成する E レベル・メッセージも出ません)。

デフォルトは RULES(NOLAXDEF) です。

LAXENTRY | NOLAXENTRY

LAXENTRY を指定すると、非プロトタイプ・エントリー宣言が許可されます。NOLAXENTRY を指定すると、コンパイラーは、プロトタイプ化されていないすべてのエントリー宣言 (つまり、パラメーター・リストを指定していないすべての ENTRY 宣言) にフラグを立てます。

注: ENTRY にパラメーターを指定しない場合は、単に ENTRY と宣言するのではなく ENTRY() と宣言する必要があることを意味していることに注意してください。

STRICT

RULES(NOLAXENTRY(STRICT)) が指定されていると、コンパイラーは、OPTIONS(ASM) 属性を持っていてプロトタイプ化されていないエントリー宣言にフラグを立てます。

LOOSE

RULES(NOLAXENTRY(LOOSE)) が指定されていると、コンパイラーは、OPTIONS(ASM) 属性を持っていてプロトタイプ化されていないエントリー宣言にフラグを立てません。

デフォルトは RULES(LAXENTRY) です。RULES(NOLAXENTRY) を指定した場合のデフォルトは STRICT です。

LAXIF | NOLAXIF

RULES(NOLAXIF) を指定すると、コンパイラーは、BIT(1) NONVARYING 属性を持たないすべての IF、WHILE、UNTIL、および WHEN 文節にフラグを立てます。 また、この指定によって、コンパイラーは `x=y=z` 形式の割り当てに対してフラグを立てますが、`x=(y=z)` 形式の割り当てに対してはフラグを立てません。

NOLAXIF の場合、以下のすべてのコードにフラグが立てられます。

```
dcl i fixed bin;
dcl b bit(8);
   .
   .
   .
if i then ...
if b then ...
```

デフォルトは RULES(NOLAXIF) です。

LAXINOUT | NOLAXINOUT

NOLAXINOUT が指定された場合、コンパイラーは、すべての ASSIGNABLE BYADDR パラメーターが入力 (および場合によっては出力) パラメーターであると想定するため、そのようなパラメーターが初期化されていないと判断すると警告を発行します。

デフォルトは RULES(LAXINOUT) です。

LAXINTERFACE | NOLAXINTERFACE

NOLAXINTERFACE が指定された場合、コンパイラーは、外部 PROCEDURE のそれぞれに有効な明示宣言を含まないコードにフラグを立てます。

- NOLAXINTERFACE では、コンパイラーは MAIN ルーチンに対してフラグを立てません。

- NOLAXINTERFACE では、コンパイラーは、PACKAGE レベルでの明示宣言を持たない PROCEDURE (そのため PACKAGE の中にないコード) に対してフラグを立てます。
- NOLAXINTERFACE では、コンパイラーは、暗黙宣言と明示宣言が一致していない PROCEDURE に対してもフラグを立てます。

デフォルトは RULES(LAXINTERFACE) です。

LAXLINK | NOLAXLINK

NOLAXLINK を指定すると、以下のいずれかが一致しない場合に、コンパイラーは、2 つの ENTRY 変数または定数の代入または比較のすべてにフラグを立てます。

- パラメーター記述リスト

  例えば、A1 が ENTRY(CHAR(8)) と宣言され、A2 が ENTRY(POINTER) VARIABLE と宣言されている場合に、RULES(NOLAXLINK) が指定されると、コンパイラーは A1 を A2 に代入しようとする処理にフラグを立てます。

- RETURNS 属性

  例えば、A3 が ENTRY RETURNS(FIXED BIN(31)) と宣言されていて、A4 が RETURNS 属性なしで ENTRY VARIABLE と宣言されている場合に、RULES(NOLAXLINK) が指定されると、コンパイラーは A3 を A4 に代入しようとする処理にフラグを立てます。

- LINKAGE および他の OPTIONS サブオプション

  例えば、A5 が ENTRY OPTIONS(ASM) と宣言されていて、A6 が OPTIONS 属性なしで ENTRY VARIABLE と宣言されている場合に、RULES(NOLAXLINK) が指定されると、コンパイラーは A5 を A6 に代入しようとする処理にフラグを立てます。これは、A5 の宣言における OPTIONS(ASM) では、A5 が LINKAGE(SYSTEM)) を持つことが暗黙指定されるのに対して、A6 には OPTIONS 属性がないため、デフォルトで A6 が LINKAGE(OPTLINK) を持つことになるためです。

デフォルトは RULES(LAXLINK) です。

LAXMARGINS | NOLAXMARGINS

NOLAXMARGINS を指定すると、STRICT および XNUMERIC サブオプションの設定に応じて、右マージンの後に非ブランク文字がある行にコンパイラーがフラグを立てます。このオプションは、誤って右マージンへ押し出された終了コメントなどのコードの検出に役立ちます。

いずれかのプリプロセッサーと共に NOLAXMARGINS および STMT オプションを使用すると、NOLAXMARGINS オプションによってフラグが立てられるステートメントは、ステートメント番号ゼロとして報告されます (ステートメントの番号付けはすべてのプリプロセッサーが終了した後でのみ行われますが、マージンの外側のテキストの検出はソースが読み取られるとすぐに行われるためです)。

STRICT

STRICT が指定された場合、コンパイラーは、右マージンの後に非ブランク文字を含むすべての行にフラグを立てます。

XNUMERIC
XNUMERIC が指定された場合、コンパイラーは、右マージンの後に非ブランク文字を含むすべての行にフラグを立てます (ただし、右マージンが 72 桁目であり、73 桁目から 80 桁目までのすべてに数字が含まれる場合を除く)。

デフォルトは RULES(LAXMARGINS) です。RULES(NOLAXMARGINS) を指定した場合のデフォルトは ALL です。

LAXNESTED | NOLAXNESTED
RULES(LAXNESTED) を指定した場合、サブプロシージャーの後にあるプロシージャー内の実行可能コードにコンパイラーはフラグを立てません。

RULES(NOLAXNESTED) を指定した場合、プロシージャー内のサブプロシージャーの後にあるすべての実行可能コードにコンパイラーがフラグを立てます。

ALL
ALL が指定された場合は、RULES(NOLAXNESTED) のすべての違反にフラグが立てられます。 ALL はデフォルトです。

SOURCE
SOURCE が指定された場合は、1 次ソース・ファイルで発生した違反にのみフラグが立てられます。

デフォルトは RULES(LAXNESTED) です。RULES(NOLAXNESTED) を指定した場合のデフォルトは ALL です。

LAXPUNC | NOLAXPUNC
NOLAXPUNC を指定すると、コンパイラーは想定される句読点が欠落している場所に E レベル・メッセージのフラグを立てます。

例えば、ステートメント `I = (1 * (2);` があるとします。この場合、コンパイラーはセミコロンの前に右括弧があるものと想定します。 RULES (NOLAXPUNC) が指定されると、このステートメントに E レベル・メッセージのフラグが立てられます。それ以外の場合は W レベル・メッセージのフラグが立てられます。

デフォルトは RULES(LAXPUNC) です。

LAXQUAL | NOLAXQUAL
NOLAXQUAL(LOOSE) を指定すると、コンパイラーはレベル 1 以外で、かつドット修飾のない構造体メンバーへのすべての参照にフラグを立てます。次の例を考えてみてください。

```
dcl
  1 a,
    2 b,
      3 b fixed bin,
      3 c fixed bin;

c   = 11;   /* would be flagged */
b.c = 13;   /* would not be flagged */
a.c = 17;   /* would not be flagged */
```

NOLAXQUAL(STRICT) を指定すると、コンパイラーはラベル 1 名を含まない構造体メンバーへのすべての参照にフラグを立てます。 次の例を考えてみてください。

```
dcl
  1 a,
    2 b,
      3 b fixed bin,
      3 c fixed bin;

c   = 11;   /* would be flagged */
b.c = 13;   /* would be flagged */
a.c = 17;   /* would not be flagged */
```

ALL
ALL が指定された場合は、RULES(NOLAXQUAL) のすべての違反にフラグが立てられます。 ALL はデフォルトです。

FORCE
FORCE が指定された場合は、FORCE(NOLAXQUAL) 属性を持つ構造体で発生した違反にのみフラグが立てられます。

デフォルトは RULES(LAXQUAL) です。RULES(NOLAXQUAL) を指定した場合のデフォルトは LOOSE および ALL です。

LAXRETURN | NOLAXRETURN
NOLAXRETURN が指定されている場合、RETURN ステートメントが以下のいずれかのように使用されているときに、エラー状態を引き起こすコードをコンパイラーが生成します。

- RETURNS オプションを指定せずにコーディングされたプロシージャー内に式を持つ場合
- RETURNS オプションを指定せずにコーディングされたプロシージャー内に式を持たない場合

コードがプロシージャーにおいて RETURNS 属性を伴って END ステートメントに到達した場合にエラーが発生します。

デフォルトは RULES(LAXRETURN) です。

LAXSCALE | NOLAXSCALE
NOLAXSCALE が指定された場合、コンパイラーはすべての FIXED BIN(*p,q*) 宣言または FIXED DEC(*p,q*) 宣言にフラグを立てます ($q < 0$ または $p < q$ の場合)。

また、コンパイラーは ROUND(*x,p*) にフラグを立てます ($p < 0$ の場合)。

コンパイラーが ROUND(*x,p*) にフラグを立てるときに発行されるメッセージは、FIXED BIN(*p,q*) 宣言や FIXED DEC(*p,q*) 宣言にコンパイラーがフラグを立てるときに発行されるメッセージと異なります。このため、EXIT オプションを使用すれば、ROUND(*x,p*) にフラグが立てられているときに発行されるメッセージを抑止して、他の疑わしい宣言に対するメッセージを維持できます。

デフォルトは RULES(NOLAXSCALE) です。

LAXSEMI | NOLAXSEMI
NOLAXSEMI を指定すると、コンパイラーは、コメント内に現れるすべてのセミコロンにフラグを立てます。

デフォルトは RULES(LAXSEMI) です。

LAXSTG | NOLAXSTG
NOLAXSTG を使用すると、コンパイラーは、B がパラメーターであったとし

ても (これが中核です)、変数 A が ADDR(B) および STG(A) > STG(B) の BASED として宣言されている場所にフラグを立てます。

NOLAXSTG が指定されているときでも、B に添え字があれば、IBM2402I E レベル・メッセージは生成されません。

B が AUTOMATIC ストレージまたは STATIC ストレージにあるとしたらコンパイラーはこの種の問題に既にフラグを立てているはずですが、B がパラメーターの場合、コンパイラーはデフォルトではこれにフラグを立てません (プログラマーによっては、実引数を記述しないでプレースホルダー属性で B を宣言するためです)。パラメーターと引数の宣言が合致する (または合致しなければならない) 場合に RULES(NOLAXSTG) を指定すると、より多くのストレージ・オーバーレイ問題を検出できるようになります。

デフォルトは RULES(LAXSTG) です。

LAXSTMT | NOLAXSTMT

NOLAXSTMT が指定された場合、コンパイラーは、複数のステートメントを持つすべての行にフラグを立てます。

ALL

RULES(NOLAXSTMT(ALL)) が指定された場合、コンパイラーはすべての NOLAXSTMT 違反にフラグを立てます。ALL はデフォルトです。

SOURCE

RULES(SOURCE) が指定された場合、コンパイラーは 1 次ソース・ファイルに含まれる違反にのみフラグを立てます。

また、NOLAXSTMT は、キーワードのリスト (空の場合もある) が指定された EXCEPT を受け入れます。そのキーワードは、行の 2 番目のステートメントがそのキーワードのいずれかで始まる場合にフラグが立てられないものです。これにより、例えば、`IF ... THEN` ステートメントと同じ行で `DO;` を使用できるようになります。

EXCEPT では以下のキーワードを使用できます。

allocate	halt
assert	if
attach	iterate
begin	leave
call	locate
cancel	on
close	open
declare	otherwise
define	put
delay	read
delete	reinit
detach	release
display	resignal
do	return
else	revert
end	rewrite
exit	select
fetch	signal
flush	stop
free	unlock
get	wait
go	when
goto	write

デフォルトは RULES(LAXSTMT) です。RULES(NOLAXSTMT) を指定した場合のデフォルトは ALL です。

LAXSTRZ | NOLAXSTRZ

LAXSTRZ を指定すると、コンパイラーは、余分のビットがすべてゼロである (または、余分の文字がすべてブランクである) 場合に、長すぎる定数値に初期化された、または割り当てられたビット変数または文字変数にフラグを立てません。

デフォルトは RULES(NOLAXSTRZ) です。

MULTICLOSE | NOMULTICLOSE

NOMULTICLOSE を指定すると、コンパイラーは、E レベル・メッセージを伴うステートメントの複数のグループを強制的に閉じるステートメントすべてにフラグを立てます。

デフォルトは RULES(NOMULTICLOSE) です。

MULTIENTRY | NOMULTIENTRY

NOMULTIENTRY が指定された場合、コンパイラーは、複数の ENTRY ステートメントを含むコードにフラグを立てます。

ALL

NOMULTIENTRY(ALL) が指定された場合、コンパイラーは RULES(NOMULTIENTRY) のすべての違反にフラグを立てます。

SOURCE

NOMULTIENTRY(SOURCE) が指定された場合、コンパイラーは 1 次ソース・ファイルで発生した違反にのみフラグを立てます。

デフォルトは RULES(MULTIENTRY) です。RULES(NOMULTIENTRY) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

MULTIEXIT | NOMULTIEXIT
NOMULTIEXIT が指定された場合、コンパイラーは、複数の RETURN ステートメントを含むコードにフラグを立てます。

ALL
NOMULTIEXIT(ALL) が指定された場合、コンパイラーは RULES(NOMULTIEXIT) のすべての違反にフラグを立てます。

SOURCE
NOMULTIEXIT(SOURCE) が指定された場合、コンパイラーは 1 次ソース・ファイルで発生した違反にのみフラグを立てます。

デフォルトは RULES(MULTIEXIT) です。RULES(NOMULTIEXIT) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

MULTISEMI | NOMULTISEMI
NOMULTISEMI が指定された場合、コンパイラーはコメントやストリング以外での複数のセミコロンを含むすべての行にフラグを立てます。

ALL
NOMULTISEMI(ALL) が指定された場合、コンパイラーは RULES(NOMULTISEMI) のすべての違反にフラグを立てます。

SOURCE
NOMULTISEMI(SOURCE) が指定された場合、コンパイラーは 1 次ソース・ファイルで発生した違反にのみフラグを立てます。

デフォルトは RULES(MULTISEMI) です。RULES(NOMULTISEMI) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

PADDING | NOPADDING
NOPADDING を指定すると、コンパイラーは、埋め込みを含むすべての構造体にフラグを立てます。

ALL
NOPADDING(ALL) が指定された場合、コンパイラーは RULES(NOPADDING) のすべての違反にフラグを立てます。

SOURCE
NOPADDING(SOURCE) が指定された場合、コンパイラーは 1 次ソース・ファイルで発生した違反にのみフラグを立てます。

デフォルトは RULES(PADDING) です。RULES(NOPADDING) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

PROCENDONLY | NOPROCENDONLY
NOPROCENDONLY を指定すると、PROCEDURE を終了するすべての END ステートメントにフラグが立てられます。ただし、その END ステートメントが PROCEDURE を指定していない場合、つまり END キーワードの直後にセミコロンがある場合に限られます。

ALL
RULES(NOPROCENDONLY(ALL)) が指定された場合、コンパイラーはすべての NOPROCENDONLY 違反にフラグを立てます。ALL はデフォルトです。

SOURCE
RULES(NOPROCENDONLY(SOURCE)) が指定された場合、コンパイラーは 1 次ソース・ファイルに含まれる違反にのみフラグを立てます。

デフォルトは RULES(PROCENDONLY) です。RULES(NOPROCENDONLY) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

RECURSIVE | NORECURSIVE
NORECURSIVE を指定した場合、RECURSIVE 属性のすべての使用に対して、または自分自身を直接呼び出すプロシージャーの使用に対して、コンパイラーがフラグを立てます。

RECURSIVE を指定した場合、RECURSIVE 属性の使用に対しても、自分自身を直接呼び出すプロシージャーの使用に対しても、コンパイラーはフラグを立てません。

注: RULES(NORECURSIVE) と DFT(RECURSIVE) を同時に使用しないでください。

デフォルトは RULES(RECURSIVE) です。

SELFASSIGN | NOSELFASSIGN
NOSELFASSIGN が指定されている場合、コンパイラーは、ソースとターゲットが同じになっているすべての割り当てにフラグを立てます。

デフォルトは RULES(SELFASSIGN) です。

UNREF | NOUNREF
NOUNREF を指定すると、参照されないレベル 1 AUTOMATIC 変数、および参照されるサブエレメントを含まない構造体または共用体であるレベル 1 AUTOMATIC 変数に、コンパイラーがフラグを立てます。NOUNREF が指定された場合は、接頭部 DSN、DFH、EYU、および SQL のいずれかで始まる名前を持つ変数は無視されます。

ALL
RULES(NOUNREF(ALL)) を指定した場合、すべての非参照変数にコンパイラーがフラグを立てます。 NOUNREF が指定されるときは、ALL がデフォルトです。

SOURCE
RULES(NOUNREF(SOURCE)) を指定した場合、INCLUDE ファイル内で宣言されていない非参照変数にコンパイラーがフラグを立てます。

デフォルトは RULES(UNREF) です。RULES(NOUNREF) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFBASED | NOUNREFBASED
NOUNREFBASED が指定された場合、コンパイラーは BASED ストレージにある非参照 BASED 変数にフラグを立てます。

ALL
RULES(NOUNREFBASED(ALL)) が指定された場合、コンパイラーはすべての非参照 BASED 変数にフラグを立てます。

SOURCE
RULES(NOUNREFBASED(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 BASED 変数にフラグを立てます。

デフォルトは RULES(UNREFBASED) です。RULES(NOUNREFBASED) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFCTL | NOUNREFCTL
NOUNREFCTL が指定された場合、コンパイラーは非参照 CTL 変数にフラグを立てます。

ALL
RULES(NOUNREFCTL(ALL)) が指定された場合、コンパイラーはすべての非参照 CTL 変数にフラグを立てます。

SOURCE
RULES(NOUNREFCTL(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 CTL 変数にフラグを立てます。

デフォルトは RULES(UNREFCTL) です。RULES(NOUNREFCTL) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFDEFINED | NOUNREFDEFINED
NOUNREFDEFINED が指定された場合、コンパイラーは非参照 DEFINED 変数にフラグを立てます。

ALL
RULES(NOUNREFDEFINED(ALL)) が指定された場合、コンパイラーはすべての非参照 DEFINED 変数にフラグを立てます。

SOURCE
RULES(NOUNREFDEFINED(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 DEFINED 変数にフラグを立てます。

デフォルトは RULES(UNREFDEFINED) です。RULES(UNREFDEFINED) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFENTRY | NOUNREFENTRY
NOUNREFENTRY が指定された場合、コンパイラーは非参照 ENTRY 定数にフラグを立てます。

ALL
RULES(NOUNREFENTRY(ALL)) が指定された場合、コンパイラーはすべての非参照 ENTRY 定数にフラグを立てます。

SOURCE
RULES(NOUNREFENTRY(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 ENTRY 定数にフラグを立てます。

デフォルトは RULES(UNREFENTRY) です。RULES(NOUNREFENTRY) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFFILE | NOUNREFFILE
NOUNREFFILE が指定された場合、コンパイラーは非参照 FILE 定数にフラグを立てます。

ALL
RULES(NOUNREFFILE(ALL)) が指定された場合、コンパイラーはすべての非参照 FILE 定数にフラグを立てます。

SOURCE
RULES(NOUNREFFILE(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 FILE 定数にフラグを立てます。

デフォルトは RULES(UNREFFILE) です。RULES(NOUNREFFILE) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

UNREFSTATIC | NOUNREFSTATIC
NOUNREFSTATIC が指定された場合、コンパイラーは非参照 STATIC 変数にフラグを立てます。

ALL
RULES(NOUNREFSTATIC(ALL)) が指定された場合、コンパイラーはすべての非参照 STATIC 変数にフラグを立てます。

SOURCE
RULES(NOUNREFSTATIC(SOURCE)) が指定された場合、コンパイラーは INCLUDE ファイル内で宣言されていない非参照 STATIC 変数にフラグを立てます。

デフォルトは RULES(UNREFSTATIC) です。RULES(NOUNREFSTATIC) を指定した場合のデフォルトのサブオプションのサブオプションは ALL です。

YY | NOYY
NOYY を指定した場合、コンパイラーは以下のような 2 桁の年の使用に対してフラグを立てます。

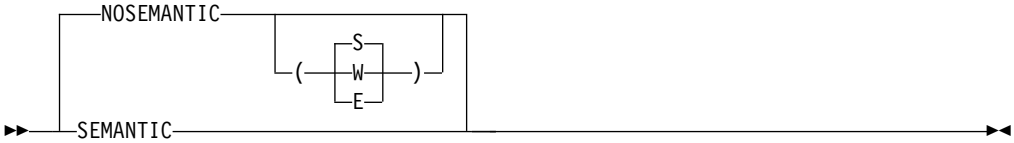
- 年が YY または ZY で指定された日時パターン
- 日時パターンを持たない DATE 属性 (パターン YYMMDD が暗黙指定されるため)
- 組み込み関数 Y4DATE、Y4JULIAN、および Y4YEAR
- DATE 組み込み関数
- 時間帯引数を持つ日時関数

デフォルトは RULES(YY) です。

デフォルト: RULES (IBM BYNAME CONTROLLED NODECSIZE EVENDEC
| ELSEIF GLOBALDO GOTO NOLAXBIF LAXCONV NOLAXCTL NOLAXDCL
| NOLAXDEF LAXENTRY NOLAXIF LAXINOUT LAXINTERFACE LAXLINK
LAXNESTED LAXPUNC LAXMARGINS(STRICT) LAXQUAL LAXRETURN
NOLAXSCALE LAXSEMI LAXSTG LAXSTMT NOLAXSTRZ NOMULTICLOSE
| MULTIENTRY MULTIEXIT MULTISEMI PADDING PROCENDONLY
| RECURSIVE SELFASSIGN UNREF UNREFBASED UNREFCTL UNREFDEFINED
| UNREFENTRY UNREFFILE UNREFSTATIC YY)

SEMANTIC

SEMANTIC オプションを指定すると、セマンティック検査段階の実行は、処理のこの段階の前に出されたメッセージの重大度に依存します。



省略形: SEM、NSEM

SEMANTIC
NOSEMANTIC(S) と同等。

NOSEMANTIC
処理は構文検査の後で停止されます。 セマンティック検査は実行されません。

NOSEMANTIC(S)
重大エラーまたは回復不能エラーが検出された場合は、セマンティック検査は行われません。

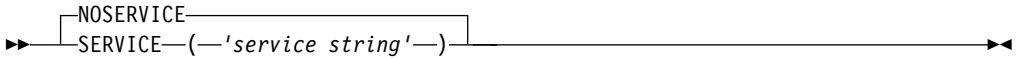
NOSEMANTIC (E)
エラー、重大エラー、または回復不能エラーが検出された場合は、セマンティック検査は行われません。

NOSEMANTIC (W)
警告、エラー、重大エラー、または回復不能エラーが検出された場合は、セマンティック検査は行われません。

ある種の重大エラーが見つかった場合は、セマンティック検査は実行されません。すべての参照が正しく解決されることをコンパイラーが確認できない場合 (例えば、組み込み関数またはエントリーの参照に引数が少なすぎる場合)、組み込み関数またはエントリーの参照の中の引数の妥当性は検査されません。

SERVICE

SERVICE オプションは、オブジェクト・モジュール内にストリングを配置します。このストリングは、オブジェクトのリンク先であるロード・モジュールとともにメモリーにロードされ、また LE ダンプにトレースバックが組み込まれている場合、このストリングはそのトレースバックにも組み込まれます。



省略形: SERV、NOSERV

ストリングの長さは 64 文字に制限されます。

ロケールが異なっていてもストリングを読み取ることができるように、インバリアント文字セットからの文字だけを使用する必要があります。

SOURCE

SOURCE オプションは、コンパイラーが、コンパイラー・リスト内にソース・プログラムのリストを組み込むことを指定します。 リストされるソース・プログラムは、オリジナルのソース入力か、あるいは (任意のプリプロセッサーが使用される場合は) プリプロセッサーの出力です。

省略形: S、NS

SPILL

SPILL オプションは、コンパイルに使用する予備域のサイズを指定します。 一度に使用されるレジスターの数が多すぎる場合、コンパイラーはレジスターの一部を予備域と呼ばれる一時記憶域にダンプします。

省略形: SP

予備域を拡張する必要がある場合、どのサイズまで増やす必要があるかを示すコンパイラー・メッセージが出されます。 ソース・プログラムに必要な予備域が分かったら、必要なサイズ (バイト) を上記の構文図のとおりに指定できます。最大予備域サイズは 3900 です。 通常、このオプションを指定する必要があるのは、OPTIMIZE を指定して非常に大規模なプログラムをコンパイルする場合だけです。

STATIC

STATIC オプションは、INTERNAL STATIC 変数を (その変数が参照されないとしても) オブジェクト・モジュールに保持するかどうかを制御します。

SHORT
INTERNAL STATIC は、使用される場合のみオブジェクト・モジュールに保管されます。

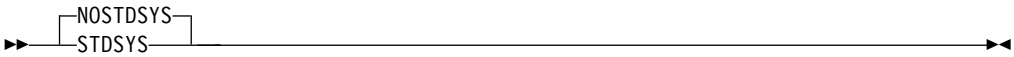
FULL
INITIAL の All INTERNAL STATIC は、オブジェクト・モジュールに保管されます。

INTERNAL STATIC 変数が「eyecatcher」として使用される場合は、STATIC(FULL) オプションを指定して、生成されたオブジェクト・モジュールにその変数を含める必要があります。

STDSYS

STDSYS オプションは、コンパイラーが SYSPRINT ファイルと C stdout ファイルを等価に、また、SYSIN ファイルを C stdin ファイルと等価にするように指定します。

注: LP(64) オプションでは、STDSYS オプションは無視されます。実際上は、STDSYS が常にオンになります。



STDSYS オプションを使用すると、PL/I および C の混在しているアプリケーションの開発とデバッグが容易になります。

SYSPRINT が stdout と等価である場合は、その LINESIZE を 132 (C が許可する最大値) より大きくすることはできません。

STMT

STMT オプションは、ソース・プログラム内のステートメントをカウントし、このステートメント番号を使用して、AGGREGATE、ATTRIBUTES、SOURCE、および XREF オプションを用いて作成されたコンパイラー・リスト内のステートメントを識別することを指定します。

デフォルトは NOSTMT です。

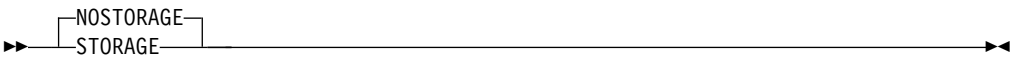
STMT オプションを指定すると、ソース・リストおよびメッセージ・リストには論理ステートメント番号とソース・ファイル番号の両方が入ります。

GOSTMT オプションは存在しないことに注意してください。実行時にエラーの発生個所を特定するための情報を生成するオプションは、GONUMBER オプションのみです。 GONUMBER オプションが使用されるときは、STMT オプションが有効になっている場合であっても、ランタイム・エラー・メッセージに含まれる「ステートメント」 という語は、NUMBER コンパイラー・オプションで使用される行番号を指します。

NUMBER と STMT は相互に排他的であり、一方を指定すると他方が無効になります。

STORAGE

STORAGE オプションは、各プロシージャーおよび各開始ブロックによって使用されるストレージの要約をリストの一部として生成するようにコンパイラーに指示します。

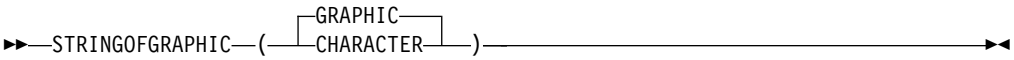


省略形: STG、NSTG

STORAGE 出力には、コンパイルの内部統計に使用されたストレージの量も含まれます。

STRINGOFGRAPHIC

STRINGOFGRAPHIC オプションは、GRAPHIC 集合に適用されたときの STRING 組み込み関数の結果が、CHARACTER または GRAPHIC のいずれの属性を持つかを決定します。



省略形: CHAR、 G

CHARACTER
STRINGOFGRAPHIC(CHAR) を指定すると、STRING 組み込み関数が UNALIGNED NONVARYING GRAPHIC 変数の配列または構造体に適用される場合、結果は CHARACTER 属性を持ちます。

GRAPHIC
STRINGOFGRAPHIC(GRAPHIC) を指定すると、STRING 組み込み関数が GRAPHIC 変数の配列または構造体に適用される場合、結果は GRAPHIC 属性を持ちます。

SYNTAX

SYNTAX オプションは、回復不能エラーが生じない限り、MACRO オプションを指定した場合、プリプロセスの後でコンパイラーが、引き続き構文検査に移ることを指定します。 コンパイラーが引き続きコンパイルを行うかどうかは、NOSYNTAX オプションで指定されたエラーの重大度によって決まります。

省略形: SYN、NSYN

SYNTAX
重大エラーまたは回復不能エラーが起こらない限り、プリプロセス後に構文検査が継続されます。 SYNTAX は NOSYNTAX(S) と同等です。

NOSYNTAX
プリプロセス後、処理は無条件に停止されます。

NOSYNTAX(W)
警告、エラー、重大エラー、または回復不能エラーが検出された場合は、構文検査は行われません。

NOSYNTAX(E)
コンパイラーはエラー、重大エラー、または回復不能エラーを検出すると、構文検査を行いません。

NOSYNTAX(S)
コンパイラーは重大エラーまたは回復不能エラーを検出すると、構文検査を行いません。

NOSYNTAX オプションでコンパイルが終了すると、相互参照リスト、属性リスト、およびソース・プログラムの後に続くその他のリストは作成されません。

このオプションを使用すると、プリプロセッサーを使用する PL/I プログラムをデバッグするときに無駄な操作を省略することができます。

NOSYNTAX オプションが有効な場合は、CICS、XOPT または XOPTS オプションを介した CICS プリプロセッサーの指定がすべて無視されます。これにより、コンパイラーが CICS 変換プログラムを呼び出す前に MACRO プリプロセッサーを呼び出すことができます。

SYSPARM

SYSPARM オプションは、マクロ機能組み込み関数 SYSPARM から返されるストリングの値を指定します。

ストリング
長さは最大 64 文字です。 デフォルトはヌル・ストリングです。

マクロ機能の詳細については、「*PL/I* 言語解説書」を参照してください。

SYSTEM

SYSTEM オプションは、MAIN PL/I プロシージャーにパラメーターを渡すのに使用するフォーマットを指定し、また一般的に、プログラムが実行されているホスト・システムを示します。

表 5 は、使用できるパラメーター・リストのタイプと、指定したホスト・システムの下でどのようにプログラムが実行されるかを示しています。 また、NOEXECOPS の暗黙の設定値も示しています。 MAIN プロシージャーは、このテーブル内で有効とされているパラメーター・リストのタイプのみを受信しなければなりません。 SYSTEM オプションの追加ランタイム情報は、「*z/OS Language Environment* プログラミング・ガイド」に記載されています。

表 *5. SYSTEM* オプション・テーブル

SYSTEM オプション	パラメーター・リストのタイプ	プログラムの稼働タイプ	暗黙の **NOEXECOPS**
SYSTEM(MVS)	1 つの CHARACTER ストリング、またはパラメーターなし	z/OS アプリケーション・プログラム	NO
	または、任意のパラメーター・リスト		YES
SYSTEM(CICS)	ポインター	CICS トランザクション	YES
SYSTEM(IMS™)	ポインター	IMS アプリケーション・プログラム	YES
SYSTEM(OS)	z/OS UNIXパラメーター・リスト (parameter list)	z/OS UNIX アプリケーション・プログラム	YES
SYSTEM(TSO)	CPPL を指すポインター[1]	TSO コマンド・プロセッサー	YES

注:

1. TSO 下で MAIN プロシージャーを呼び出す方法について詳しくは、 211 ページの『TSO/E のもとでの MAIN の呼び出し』を参照してください。

SYSTEM(IMS) では、すべてのポインターが値 (BYVALUE) で渡されると想定されますが、SYSTEM(MVS) では、すべてのポインターがアドレス (BYADDR) で渡されると想定されます。

CICS 環境で実行される MAIN プロシージャーは、SYSTEM(CICS) または SYSTEM(MVS) を指定してコンパイルする必要があります。

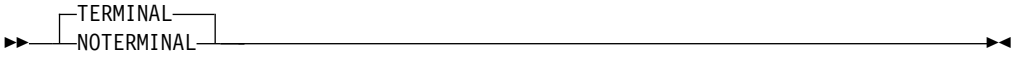
SYSTEM(MVS) を指定してコンパイルされ、ランタイム・オプションが渡されずに実行される、DB2 ストアード・プロシージャーなどのコードの MAIN プロシージャーの OPTIONS オプションには、NOEXECOPS を指定することを強くお勧めします。

コンパイラーは、複数のパラメーターを持つか、CHARACTER VARYING 以外のパラメーターを 1 つだけ持つ、SYSTEM(MVS) でコンパイルされたすべての MAIN プログラムにフラグを立てます。ライブラリーは単にパラメーター・リストを MAIN に渡すだけでオプションのスキャンやその他のパラメーター・リストのメッセージングを行わないため、このような MAIN プログラムは SYSTEM(OS) でコンパイルした方が恐らくよいと思われます。

TERMINAL

TERMINAL オプションは、コンパイル時に作成された診断メッセージおよび通知メッセージを端末に表示するかどうかを決めます。

注**:** このオプションは、z/OS UNIX 環境でのコンパイルにだけ適用されます。



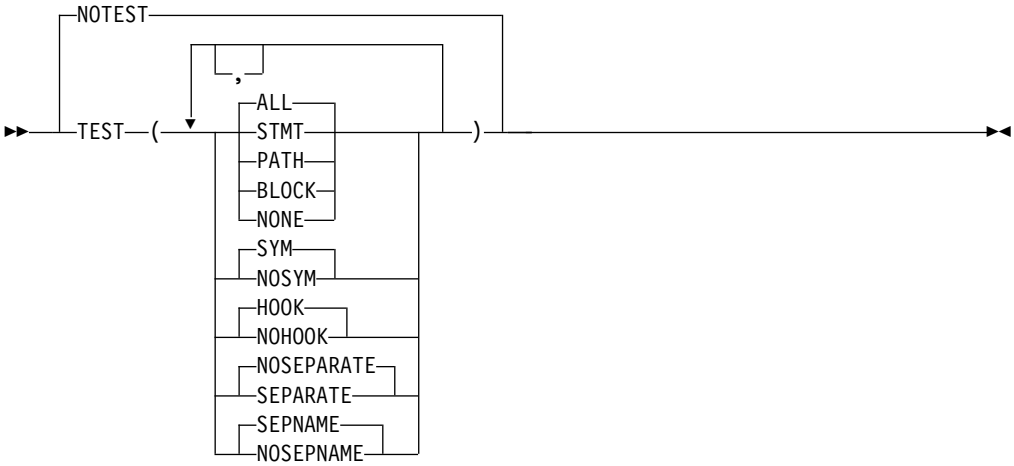
省略形: TERM、NTERM

TERMINAL
メッセージは端末に表示されます。

NOTERMINAL
通知コンパイラー・メッセージまたは診断コンパイラー・メッセージは端末に表示されません。

TEST

TEST オプションは、コンパイラーがオブジェクト・コードの一部として生成する検査機能のレベルを指定します。 このオプションを使用すれば、テスト・フックの位置を制御したり、記号テーブルを生成するかどうかを制御したりできます。



省略形: AALL、ACICS、AMACRO、ASQL

STMT
コンパイラーはステートメント・テーブルを生成します。HOOK サブオプションが有効になっている場合は、ステートメント境界とブロック境界にフックを挿入します。

PATH
コンパイラーはステートメント・テーブルを生成します。 HOOK サブオプションが有効になっている場合、コンパイラーは以下の場所にフックを挿入します。

- 反復 DO ステートメントに囲まれた最初のステートメントの前
- IF ステートメントの真の部分に含まれる最初のステートメントの前
- IF ステートメントの偽の部分に含まれる最初のステートメントの前
- SELECT グループの真の WHEN または OTHERWISE ステートメントに含まれる最初のステートメントの前

- ユーザー・ラベルに続くステートメントの前 (ラベルの付いた FORMAT ステートメントを除く)

  ステートメントに複数のラベルがある場合は、フックが 1 つのみ挿入されます。
- CALL または関数参照 (ルーチンに制御が渡される前と後の両方)
- ブロック境界

BLOCK
コンパイラーはステートメント・テーブルを生成します。HOOK サブオプションが有効になっている場合は、ブロック境界 (ブロック入り口とブロック出口) にフックを挿入します。

ALL
コンパイラーはステートメント・テーブルを生成します。HOOK サブオプションが有効になっている場合は、可能なすべての場所にフックを挿入してステートメント・テーブルを生成します。

注: opt(2) と opt(3) を指定すると、フックはブロック境界にのみ設定されます。

NONE
プログラムにフックを入れません。

SYM
変数を名前で検査するためのシンボル・テーブルを作成します。

NOSYM
シンボル・テーブルは生成されません。

NOTEST
すべてのテスト情報の生成を抑止します。

HOOK
TEST サブオプション ALL、STMT、BLOCK、または PATH のいずれかが有効である場合に、コンパイラーによって、生成されたコードにフックが挿入されます。

NOHOOK
コンパイラーによって、生成されたコードにフックは挿入されません。

IBM Debug Tool がオーバーレイ・フックを生成する場合、サブオプション ALL、PATH、STMT、または BLOCK の 1 つを指定しなければなりませんが、HOOK は指定する必要はなく、事実上 NOHOOK を指定することをお勧めします。

NOHOOK を指定すると、ENTRY および EXIT のブレークポイントは、Debug Tool が停止する PATH ブレークポイントにすぎなくなります。

SEPARATE
コンパイラーは、生成するデバッグ情報のほとんどを別個のデバッグ・ファイルに書き込みます。 このオプションを使用する場合、TEST オプションを有効にすると、コンパイラーが作成するオブジェクト・デックのサイズがかなり小さくなります。

プログラムに GET または PUT DATA ステートメントが含まれる場合、別個のデバッグ・ファイルに含まれるデバッグ情報の量が少なくなります。これらのステートメントについては、シンボル・テーブル情報をオブジェクト・デックに書き込む必要があるためです。

生成されるデバッグ情報には、コンパイラーに渡されたソースの圧縮版が常に含まれています。 つまり、ソースは SYSIN DD * を使用して指定されていたり、前のジョブ・ステップで作成された一時データ・セットであったりする可能性があります (例えば、ソースは古い SQL/CICS プリコンパイラーの出力である可能性があります)。 LISTVIEW オプションに指定するサブオプションは、ソースの内容を制御します。

SEPARATE がバッチ・コンパイルで使用される場合、そのコンパイルの JCL には SYSDEBUG の DD カードが含まれていなければなりません。その DD カードでは、RECFM=FB および 80 <= LRECL <= 1024 のデータ・セットが指定されなければなりません。

このサブオプションは LINEDIR コンパイラー・オプションと併用できません。

NOSEPARATE

コンパイラーは、生成するデバッグ情報のすべてをオブジェクト・デックに書き込みます。

このオプションを指定すると、生成されるデバッグ情報には、コンパイラーに渡されたソースの圧縮版は含まれません。 これは、プログラムをデバッグしようとするときに Debug Tool で見つけることができるデータ・セットにソースがなければならないことを意味します。

SEPNAME

コンパイラーは、別個のデバッグ・ファイルの名前をオブジェクト・デックに書き込みます。

SEPARATE オプションが有効になっていない場合、このオプションは無視されます。

NOSEPNAME

コンパイラーは、別個のデバッグ・ファイルの名前をオブジェクト・デックに書き込みません。

SEPARATE オプションが有効になっていない場合、このオプションは無視されます。

注:

- LP(64) では、フックは生成されず、コンパイラー・オプション SEP および NOSEP は無視されます。
- opt(2) または opt(3) を指定すると、フックはブロック境界にのみ設定されます。つまり、最適化されたコードのデバッグは効率的に PROCEDURE および BEGIN ブロックの出入り口のトレースに制限されます。
- SEPARATE コンパイラー・オプションを使用してコンパイルしたコードをデバッグするには、Debug Tool バージョン 6 以降を使用する必要があります。
- 範囲が連結データ・セットに及ぶ入力ファイルは、サポートされていません。

TEST(NONE,NOSYM) を指定すると、コンパイラーはこのオプションを NOTEST に設定します。

TEST(NONE,SYM) は使用しないでください。この設定を指定する意図が不明です。 TEST(ALL,SYM,NOHOOK) または TEST(STMT,SYM,NOHOOK) を指定するとよいでしょう。

NOTEST と TEST(NONE,NOSYM) 以外の TEST オプションは、いずれもプログラム・テストのためのアテンション割り込み機能を提供します。

呼び出したい ATTENTION ON ユニットがプログラムにある場合は、次のいずれかのオプションを使用してプログラムをコンパイルしなければなりません。
- INTERRUPT オプション
- NOTEST または TEST(NONE,NOSYM) 以外の TEST オプション

注: ATTENTION は TSO の下でだけサポートされます。

TEST オプションでは GONUMBER が暗黙指定されます。

TEST オプションでは、オブジェクト・コードのサイズが増大してパフォーマンスに影響が出ることがあるため、フックの数と位置に限度を設けなければならないことがあります。

TEST オプションを指定した場合、インライン化は行われません。

REFER の構造体はシンボル・テーブルでサポートされています。

TEST(SYM) が有効な場合は、Debug Tool の自動モニター機能を有効にするために、コンパイラーはテーブルを生成します。TEST(SEPARATE) オプションが有効になっていなければ、これらのテーブルのために、オブジェクト・モジュールのサイズがかなり大きくなる場合があります。Debug Tool の自動モニター機能を活動化すると、これらのテーブルを使用して、ステートメントで使用される変数の値がステートメントの実行前に表示されます。これは、変数が計算可能なタイプであるか、POINTER、OFFSET、または HANDLE の属性を持っていることが条件となります。ステートメントが割り当てステートメントである場合は、ターゲットの値も表示されます。ただし、ターゲットの初期設定や割り当てがあらかじめ行われていない場合は、その値に意味がありません。

名前に * を使用して宣言された変数はすべて、Debug Tool の使用時には表示されません。また、* が親構造体または副構造体の名前として使用されている場合は、そのいずれの子も表示されません。したがって、「名前を付けない」でおく構造体エレメントの名前には、単一の下線を使用することをお勧めします。

UNROLL

UNROLL オプションは、最適化でのループのアンロールを制御します。 ループのアンロールとは最適化の一種で、ループ本体を複数回複製し、それに応じてループ制御コードを調整します。

AUTO
コンパイラーは、アンロールの適切な対象になると判断したループをアンロールすることを許可されています。

UNROLL オプションを指定した場合、生成されるオブジェクト・コードのサイズが大きくなることがあります。

NO コンパイラーはループのアンロールを許可されていません。

UNROLL オプションは、NOOPTIMIZE オプションが有効になっていると無視されます。

ループのアンロールは、命令のスケジューリングとソフトウェア・パイプラインに対して命令レベルの並列処理を行うことにより、プログラムのパフォーマンスを改善します。 また、これにより新しいループ本体の中のコードが大きくなり、それがレジスター割り振りへのプレッシャー増大につながるために、パフォーマンスが低下することがあります。

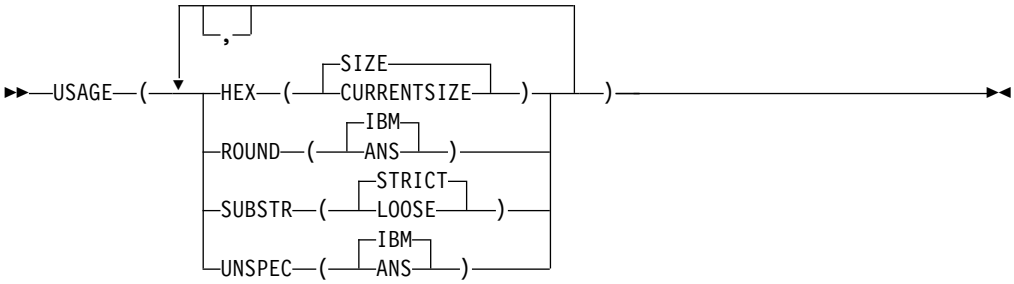
したがって、ループをアンロールする前に以下の手順を実行して、 UNROLL オプションによって特定アプリケーションのパフォーマンスが改善されるかどうかを確認してください。

1. 通常のオプションを指定してプログラムをコンパイルします。
2. 標準的なワークロードでプログラムを実行します。
3. UNROLL オプションを指定してプログラムを再コンパイルします。
4. 同じ条件でプログラムを再実行します。

UNROLL(AUTO) がデフォルトです。

USAGE

USAGE オプションを使用すれば、選択した複数の組み込み関数に対して別々のセマンティクスを選択できます。



HEX(SIZE | CURRENTSIZE)
HEX(SIZE) サブオプションで、HEX が VARYING または VARYINGZ ストリングに適用された場合、ストリングで使用されているストレージの最大容量を示す 16 進ストリングが返されます。

HEX(CURRENTSIZE) サブオプションで、HEX が VARYING または VARYINGZ ストリングに適用された場合、ストリングで使用されているストレージの現在容量を示す 16 進ストリングが返されます。

ROUND(IBM | ANS)

ROUND(IBM) サブオプションを指定すると、ROUND 組み込み関数の最初の引数が FLOAT 属性を持っている場合、2 番目の引数は無視されます。

ROUND(ANS) サブオプションを指定すると、ROUND 組み込み関数は、「*PL/I* 言語解説書」で説明されているようにインプリメントされます。

SUBSTR(STRICT | LOOSE)

SUBSTR(STRICT) サブオプションでは、x が CHARACTER タイプである場合は、長さ MIN(z, MAXLENGTH(x)) または長さ z を SUBSTR(x,y,z) 組み込み関数参照に割り当てる許可をコンパイラーに付与します。

SUBSTR(LOOSE) サブオプションでは、同じ参照で、長さが z であるストリングが返されます。

SUBSTR(LOOSE) サブオプションは、SUBSTR(x,y,z) 参照 (x は CHAR(1) BASED 変数) が指定されている場合に役立つ可能性があります。

STRINGRANGE が有効になっている場合に、z > MAXLENGTH(x) であれば、このオプションのいずれかの設定で STRINGRANGE が適用されます。

UNSPEC(IBM | ANS)

UNSPEC(IBM) サブオプションを指定すると、UNSPEC を構造体に適用することはできません。配列に適用すると、ビット・ストリングの配列が戻されます。

UNSPEC(ANS) サブオプションを指定すると、UNSPEC を構造体に適用できます。構造体または配列に適用すると、UNSPEC はシングル・ビット・ストリングを戻します。

デフォルト: USAGE(HEX(SIZE) ROUND(IBM) SUBSTR(STRICT) UNSPEC(IBM))

WIDECHAR

WIDECHAR オプションは、WIDECHAR データが保管されるフォーマットを指定します。

BIGENDIAN

WIDECHAR データをビッグ・エンディアン・フォーマットで保管するように指示します。 例えば、UTF-16 文字 1 の WIDECHAR 値は '0031'x として保管されます。

LITTLEENDIAN

WIDECHAR データをリトル・エンディアン・フォーマットで保管するように指示します。 例えば、UTF-16 文字 1 の WIDECHAR 値は '3100'x として保管されます。

WX 定数は、常にビッグ・エンディアン・フォーマットで指定する必要があります。 したがって、WIDECHAR(LITTLEENDIAN) オプションを指定した場合、値 '1' は '3100'x として保管されますが、この値は常に '0031'wx として指定する必要があります。

WINDOW

WINDOW オプションは、各種の日付関連の組み込み関数で使用される w ウィンドウ引数を設定します。

w 固定ウィンドウの開始を表す符号なし整数、または「スライディング」ウィンドウを指定する負の整数。例えば、WINDOW(-20) は、プログラムを実行する 20 年前に開始されるウィンドウを示します。

WRITABLE

WRITABLE オプションは、コンパイラーが静的ストレージを書き込み可能なストレージとして扱うことができることを指定します (また、そうする場合は、その結果のコードが再入不可になります)。

このオプションは、RENT オプションを指定してコンパイルされたプログラムには効果がありません。

注: LP(64) オプションでは、WRITABLE オプションは無視されます。

NORENT WRITABLE オプションを使用すると、コンパイラーは、以下の定数または変数を実装するために静的ストレージ上で書き込みできるようになります。

- CONTROLLED 変数
- FETCHABLE ENTRY 定数
- FILE 定数

NORENT WRITABLE オプションを指定すると、CONTROLLED 変数を使用するモジュール、入出力を実行するモジュール、または FETCH を使用するモジュールは、再入可能にはなりません。

NORENT NOWRITABLE オプションでは、コンパイラーが、以下の定数または変数の静的ストレージ上に書き込まないようにする必要があります。

- CONTROLLED 変数
- FETCHABLE ENTRY 定数
- FILE 定数

NORENT NOWRITABLE オプションを指定すると、 CONTROLLED 変数を使用するモジュール、入出力を実行するモジュール、または FETCH を使用するモジュールは、再入可能になります。

FWS および PRV サブオプションは、以下のように、コンパイラーが CONTROLLED 変数を扱う方法を決定します。

FWS

EXTERNAL プロシージャーに入るときに、コンパイラーはライブラリー呼び出しを行って、プロシージャー (およびすべてのサブプロシージャー) 内で CONTROLLED 変数をアドレス指定するために使用できるストレージを見つけます。

PRV

コンパイラーは、CONTROLLED 変数をアドレス指定するために、旧 OS PL/I コンパイラーで使用していたものと同じ、疑似レジスター変数メカニズムを使用します。

このため、NORENT NOWRITABLE(PRV) オプションを指定すると、旧コードと新規コードで CONTROLLED 変数を共用することができます。

ただし、NORENT NOWRITABLE(PRV) オプションを指定すると、CONTROLLED 変数の使用が、旧コンパイラーと同じすべての制約を受けることも意味しています。

NORENT NOWRITABLE(FWS) オプションを指定すると、以下のアプリケーションは、RENT または WRITABLE オプションを指定してコンパイルされた場合と同じようには実行されないことがあります。

- CONTROLLED 変数を使用するアプリケーション
- FILE CONSTANT を FILE VARIABLE に割り当てるアプリケーション

NORENT NOWRITABLE(FWS) を指定した場合のアプリケーションのパフォーマンスは、多くの CONTROLLED 変数を多くの PROCEDURE で使用していると特に悪くなる場合があります。

NOWRITABLE オプションを指定すると、PROCEDURE の外側で PACKAGE において以下の変数や定数を宣言できません。

- CONTROLLED 変数
- FETCHABLE ENTRY 定数
- FILE 定数

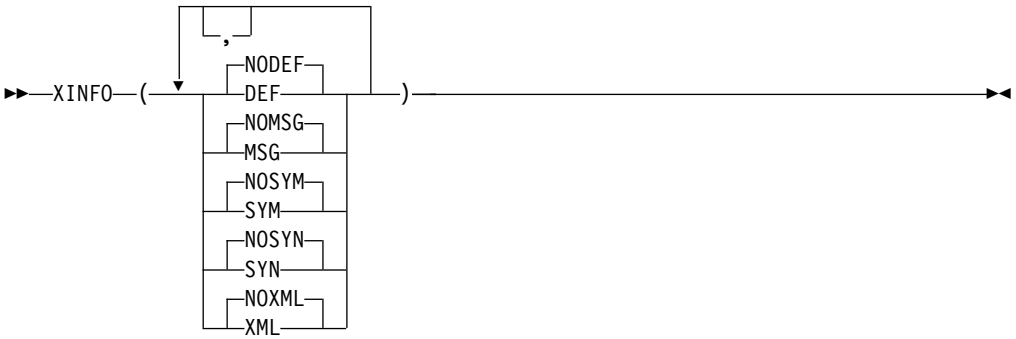
NORENT WRITABLE を指定してコンパイルされたコードは、任意の外部 CONTROLLED 変数を共用する、NORENT NOWRITABLE を指定してコンパイルされたコードと混在できません。一般に、WRITABLE でコンパイルされたコードと NOWRITABLE でコンパイルされたコードを混在しないでください。

関連情報:

76 ページの『RENT』
コードが静的変数を変更しなければ、そのコードは「本質的に再入可能」です。RENT オプションを指定すると、コンパイラーは本質的に再入可能でない コードを検出して、再入可能にします。

XINFO

XINFO オプションは、現行コンパイル単位に関する追加の情報が入ったファイルを追加して生成するようにコンパイラーに指定します。



DEF

定義 SIDEDECK ファイルが作成されます。 このファイルには、コンパイル単位に関する以下の情報がリストされます。

- 定義されているすべての EXTERNAL プロシージャー
- 定義されているすべての EXTERNAL 変数
- 静的に参照されるすべての EXTERNAL ルーチンおよび変数
- 動的に呼び出されるフェッチ済みモジュール

バッチ環境では、このファイルは SYSDEFSD DD ステートメントによって指定されたファイルに書き込まれます。 z/OS UNIX システム・サービスでは、このファイルはオブジェクト・デックと同じディレクトリーに作成されて拡張子 def が付けられます。

例えば、次のプログラムがあるとします。

```
defs: proc;
   dcl (b,c) ext entry;
   dcl x ext fixed bin(31) init(1729);
   dcl y ext fixed bin(31) reserved;
   call b(y);
   fetch c;
   call c;
end;
```

この場合、次の def ファイルが生成されます。

```
EXPORTS CODE
  DEFS
EXPORTS DATA
  X
IMPORTS
  B
  Y
FETCH
  C
```

def ファイルを使用して、アプリケーションの依存性グラフを作成したり、相互参照分析を行ったりすることができます。

NODEF

定義 SIDEDECK ファイルは作成されません。

MSG

メッセージ情報が ADATA ファイルに生成されます。

バッチでは、ADATA ファイルは、SYSADATA DD ステートメントによって指定されたファイルに生成されます。 z/OS UNIX では、ADATA はオブジェクト・ファイルと同じディレクトリーに生成されて、拡張子 adt が付けられます。

NOMSG
メッセージ情報は ADATA ファイルに生成されません。 MSG も SYM も指定しないと、ADATA ファイルは生成されません。

SYM
シンボル情報が ADATA ファイルに生成されます。

バッチでは、ADATA ファイルは、SYSADATA DD ステートメントによって指定されたファイルに生成されます。 z/OS UNIX では、ADATA ファイルはオブジェクト・ファイルと同じディレクトリーに生成されて、拡張子 adt が付けられます。

NOSYM
シンボル情報は ADATA ファイルに生成されません。

SYN
構文情報が ADATA ファイルに生成されます。 XINFO(SYN) オプションを指定すると、コンパイラーが必要とするストレージ量 (メモリーと生成されるファイルの両方) が大幅に増えることがあります。

バッチでは、ADATA ファイルは、SYSADATA DD ステートメントによって指定されたファイルに生成されます。 z/OS UNIX では、ADATA ファイルはオブジェクト・ファイルと同じディレクトリーに生成されて、拡張子 adt が付けられます。

NOSYN
構文情報は ADATA ファイルに生成されません。

XML
XML サイド・ファイルが作成されます。 この XML ファイルには以下の項目が含まれます。

- コンパイル用のファイル参照テーブル
- コンパイルされたプログラムのブロック構造
- コンパイル時に作成されたメッセージ

バッチ環境では、このファイルは SYSXMLSD DD ステートメントによって指定されるファイルに書き込まれます。 z/OS UNIX システム・サービスでは、このファイルはオブジェクト・デックと同じディレクトリーに作成されて拡張子 xml が付けられます。

作成された XML の DTD ファイルは次のようになります。

```
<?xml encoding="UTF-8"?>

<!ELEMENT PACKAGE ((PROCEDURE)*,(MESSAGE)*,FILEREFERNCETABLE)>
<!ELEMENT PROCEDURE (BLOCKFILE,BLOCKLINE,(PROCEDURE)*,(BEGINBLOCK)*)>
<!ELEMENT BEGINBLOCK (BLOCKFILE,BLOCKLINE,(PROCEDURE)*,(BEGINBLOCK)*)>
<!ELEMENT MESSAGE (MSGNUMBER,MSGLINE?,MSGFILE?,MSGTEXT)>
<!ELEMENT FILE (FILENUMBER,INCLUDEDFROMFILE?,INCLUDEDONLINE?,FILENAME)>
<!ELEMENT FILEREFERENCETABLE (FILECOUNT,FILE+)>

<!ELEMENT BLOCKFILE (#PCDATA)>
<!ELEMENT BLOCKLINE (#PCDATA)>
```

```
<!ELEMENT MSGNUMBER (#PCDATA)>
<!ELEMENT MSGLINE (#PCDATA)>
<!ELEMENT MSGFILE (#PCDATA)>
<!ELEMENT MSGTEXT (#PCDATA)>
<!ELEMENT FILECOUNT (#PCDATA)>
<!ELEMENT FILENUMBER (#PCDATA)>
<!ELEMENT FILENAME (#PCDATA)>
<!ELEMENT INCLUDEFROMFILE (#PCDATA)>
<!ELEMENT INCLUDEDONLINE (#PCDATA)>
```

NOXML

XML サイド・ファイルは作成されません。

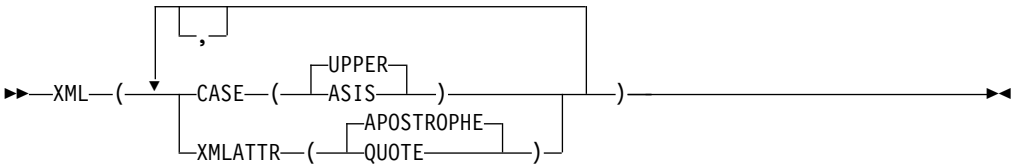
関連情報:

565 ページの『SYSADATA メッセージ情報』

XINFO コンパイル時オプションの MSG サブオプションを指定すると、コンパイラーは SYSADATA ファイルを生成します。

XML

XML オプションは、XMLCHAR 組み込み関数によって生成される XML での名前の大/小文字を指定します。



CASE(UPPER | ASIS)

CASE(UPPER) サブオプションを指定した場合、XMLCHAR 組み込み関数によって生成された XML 内の名前がすべて大文字になります。

CASE(ASIS) サブオプションを指定した場合、XMLCHAR 組み込み関数によって生成される XML 内の名前の大/小文字は、宣言で使用したとおりになります。なお、マクロ・プリプロセッサー・オプション CASE(ASIS) を使用しないでマクロ・プリプロセッサーを使用した場合は、コンパイラーが表示するソースの名前はすべて大文字になり、XML(CASE(ASIS)) オプションを指定しても機能しません。

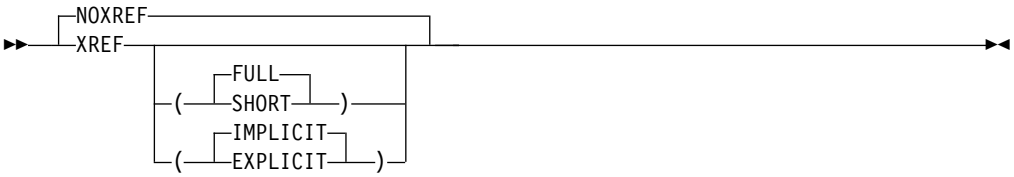
XMLATTR(APOSTROPHE | QUOTE)

XMLATTR(APOSTROPHE) サブオプションが指定された場合は、XMLCHAR 組み込み関数によって生成される XML 属性はすべてアポストロフィで囲まれます。

XMLATTR(QUOTE) サブオプションが指定された場合は、XMLCHAR 組み込み関数によって生成される XML 属性はすべて引用符で囲まれます。
CODEPAGE(1026) および CODEPAGE(1155) が指定された場合は、引用文字の 16 進値は 'FC'x であり、それ以外のすべてのサポート対象 EBCDIC コード・ページでは、引用文字の 16 進値は '7F'x です。

XREF

XREF オプションは、プログラム内で使用する名前の相互参照テーブル、ならびにその名前が宣言または参照されるステートメントの数をコンパイラー・リストに入れることを指定します。



省略形: X、NX

FULL
XREF(FULL) では、すべての ID と属性がコンパイラー・リストに組み込まれます。FULL がデフォルトです。

SHORT
XREF(SHORT) では、非参照 ID はコンパイラー・リストから除外されます。

EXPLICIT
XREF(EXPLICIT) では、構造が参照されると、その構造のみがコンパイラー・リストに組み込まれます。

IMPLICIT
XREF(IMPLICIT) の場合、構造が参照されると、その構造と、そのすべてのメンバーがコンパイラー・リストに組み込まれます。IMPLICIT がデフォルトです。

XREF オプションを使用して作成された相互参照リストに入れられない名前は、END ステートメントのラベル参照だけです。例えば、プロシージャー PROC1 のステートメント番号 20 が `END PROC1;` であると想定します。この場合、ステートメント番号 20 は PROC1 用の相互参照リストには入りません。

XREF オプションと ATTRIBUTES オプションを両方とも指定すると、2 つのリストが組み合わされます。 SHORT と FULL が対立する場合は、最後に指定したオプションで使用法が決まります。 例えば ATTRIBUTES (SHORT) XREF (FULL) を使用するとその結果は、リストを組み合わせる FULL オプションになります。

関連情報:

120 ページの『相互参照テーブル』
ATTRIBUTES と XREF を指定すると、相互参照テーブルと属性テーブルが結合されます。 名前に対する属性のリストは、ファイル番号と行番号で識別されます。

オプションの中のブランク、コメント、およびストリング

オプションを指定するときにブランクを使用できる場所では常に、必要な数のブランクやコメントも指定できます。ただし、いくつかの規則に注意する必要があります。

%PROCESS の行またはオプション・ファイルの行にコメントを指定する場合、そのコメントは先頭から末尾まで同じ行になければなりません。

同様に、コマンド行や PARM= に指定されるコメントも先頭から末尾まで同じ行になければなりません。

同じ規則がストリングに適用されます。 %PROCESS 行またはオプション・ファイルの行にストリングを指定する場合、そのストリングは先頭から末尾まで同じ行になければなりません。同様に、コマンド行や PARM= に指定されるストリングも先頭から末尾まで同じ行になければなりません。

デフォルト・オプションの変更

デフォルト設定のコンパイラー・オプションを変更する場合は、コンパイラーのインストール時に、サンプル・ジョブ IBMZWIOP を編集して実行依頼する必要があります。

このジョブで、あらかじめ適用しておくオプションを指定し、これら以外のオプションは後で追加するという形にすることで、デフォルト・オプションを実質的に変更することができます。またこのジョブでは、後から追加して最終的に適用するオプションを指定することで、デフォルト・オプションを実質的に変更することができます。この方法を使用すると、このジョブで指定したオプションが指定変更されることがありません。

マクロ・プリプロセッサーのデフォルト・オプションを変更する場合は、インストール時にこのジョブの一部として、該当する PPMACRO オプションを指定することによって行うこともできます。 PPCICS オプションと PPSQL オプションを使用すれば、それぞれ CICS プリプロセッサーと SQL プリプロセッサーについて変更することができます。

詳しくは、サンプル・ジョブに記述されている説明を参照してください。

%PROCESS ステートメントまたは *PROCESS ステートメントでのオプションの指定

%PROCESS ステートメントまたは *PROCESS ステートメントは各外部プロシージャーの開始を識別します。これらのステートメントを使用すれば、コンパイルごとにコンパイラー・オプションを指定できます。 %PROCESS または *PROCESS をプログラムで使用でき、どちらも等しく受け入れられます。

注: 整合性をとるため、および読みやすくするために、%PROCESS ステートメントも *PROCESS ステートメントも常に %PROCESS とします。

隣接する %PROCESS ステートメント内で指定するオプションは、ソース・ステートメントの入力の終わりまでのコンパイルに対してか、または次の %PROCESS ステートメントまでのコンパイルに対して用いられます。

%PROCESS ステートメントでオプションを指定するには、次のようにコーディングします。

```
%PROCESS options;
```

ここで *options* はコンパイラー・オプションのリストです。

オプション・リストの終わりにはセミコロンを入れなければならず、また、オプション・リストはデフォルトの右側ソース・マージンを超えてはなりません。 パーセント記号 (%) またはアスタリスク (*) がレコードの最初の列になければなりません。 キーワード PROCESS は、次のバイト (桁) 内か、任意の数のブランクの後に置くことができます。 オプション・キーワードは、コンマまたは少なくとも 1 つのブランクを使って区切らなければなりません。

文字数を制限するものはレコードの長さだけです。オプションをどれも指定したくない場合は、次のようにコーディングします。

```
%PROCESS;
```

%PROCESS ステートメントを次のレコードまで続けなければならない場合は、リストの前半部分を任意の区切り文字の後で終了してから、次のレコードへ移ります。キーワードやキーワード引数を複数のレコードに分割することはできません。%PROCESS ステートメントを複数行にまたがって続けることも、または、新たに %PROCESS ステートメントを開始することもできます。複数の隣接する %PROCESS ステートメントを以下に例示します。

```
%PROCESS INT F(I) AG A(F) OP STG NEST X(F) SOURCE ;
%PROCESS LIST TEST ;
```

コンパイル時オプション、その省略形構文、および IBM 提供のデフォルトは、 4 ページの表 3 に示してあります。

%PROCESS ステートメントがあるかどうかをコンパイラーが判別する方法は、初期ソース・ファイルのフォーマット設定によって決まります。

F または **FB** フォーマット
: レコードの先頭文字がアスタリスク (*) またはパーセント記号 (%) の場合に、コンパイラーは、次の非ブランク文字が PROCESS かどうかを検査します。

V または **VB** フォーマット
: レコードの先頭文字が数字の場合に、コンパイラーは最初の 8 文字がシーケンス番号であると想定し、9 番目の文字がアスタリスク (*) またはパーセント記号 (%) であれば、次の非ブランク文字が PROCESS であるかどうかを検査します。ただし、先頭文字が数字ではなく、アスタリスク (*) またはパーセント記号 (%) の場合、コンパイラーは次の非ブランク文字が PROCESS かどうかを検査します。

U フォーマット
: レコードの先頭文字がアスタリスク (*) またはパーセント記号 (%) の場合に、コンパイラーは次の非ブランク文字が PROCESS かどうかを検査します。

% ステートメントの使用

コンパイラーの操作を指示するステートメントは、パーセント (%) 記号で始まります。 % ステートメントを使用すれば、ソース・プログラム・リストを制御してソース・プログラムに外部ストリングを組み込むことができます。 % ステートメントにはラベルや条件接頭語が付いてはなりません。また、複合ステートメントの単位にすることもできません。 % ステートメントはおのおの、1 行内に単独で入っていなければなりません。

各 % 制御ステートメントの使用法を以下に示します。これらのステートメントについて詳しくは、「*PL/I* 言語解説書」を参照してください。

%INCLUDE
外部テキストをソース・プログラムに組み込むようコンパイラーに指示します。

%XINCLUDE
外部テキストがまだ組み込まれていない場合、外部テキストをソース・プログラムに組み込むようコンパイラーに指示します。

%PRINT
SOURCE および INSOURCE のリストの印刷を再開するよう、コンパイラーに指示します。

%NOPRINT
SOURCE および INSOURCE のリストの印刷を、 %PRINT ステートメントが見つかるまで延期するようコンパイラーに指示します。

%PAGE
プログラム・リスト内の %PAGE ステートメントの直後のステートメントを、次ページの最初の行に印刷するようコンパイラーに指示します。

%POP
最新の %PUSH により保存された %PRINT および %NOPRINT の状況を復元するように、コンパイラーに指示します。

%PUSH
%PRINT および %NOPRINT の現行状況を後入れ先出し方式でプッシュダウン ・スタックに保存します。

%SKIP
スキップしたい行数を指定します。

%INCLUDE ステートメントの使用

%INCLUDE ステートメントは、コンパイル単位内の指定された点に追加の PL/I ファイルを組み込むために使用します。

%INCLUDE ステートメントを使用して、ライブラリーにあるソース・テキストを PL/I プログラムに組み込む方法については、「*PL/I* 言語解説書」を参照してください。

バッチ・コンパイルの場合
ライブラリー とは、メンバーと呼ばれるその他のデータ・セットを保管するのに使用できる z/OS 区分データ・セットです。%INCLUDE ステート

メントを使って PL/I プログラムに挿入しようとするソース・テキストは、ライブラリー内のメンバーとして存在していなければなりません。 ソース・ステートメント・ライブラリーをコンパイラーに対して定義するプロセスについて詳しくは、 190 ページの『ソース・ステートメント・ライブラリー (SYSLIB)』を参照してください。

ステートメント `%INCLUDE DD1 (INVERT);` は、DD1 という名前の DD ステートメントで定義されたライブラリーのメンバー INVERT 内のソース・ステートメントを、ソース・プログラムに連続して挿入することを指定します。コンパイル・ジョブ・ステップには、適切な DD ステートメントが入っていなければなりません。

dd 名を省略すると、SYSLIB という dd 名がとられます。その場合、SYSLIB という名前で DD ステートメントを組み込む必要があります。(IBM 提供のカタログ式プロシージャーの場合は、コンパイル・プロシージャー・ステップにこの名前の DD ステートメントは組み込まれていません。)

z/OS UNIX コンパイルの場合

実際の組み込みファイルの名前は、UPPERINC を指定しない限り、小文字でなければなりません。 例えば、組み込みステートメント `%include sample` を使用すると、コンパイラーはファイル `sample.inc` を検出できますが、ファイル `SAMPLE.inc` は検出できません。組み込みステートメント `%include SAMPLE` を使用しても、コンパイラーは `sample.inc` を検索します。

コンパイラーは次の順序で組み込みファイルを検索します。

1. 現行ディレクトリー
2. –I フラグまたは INCDIR コンパイラー・オプションで指定されたディレクトリー
3. `/usr/include` ディレクトリー
4. INCPDS コンパイラー・オプションで指定された PDS

コンパイラーが検出した最初のファイルがソースに組み込まれます。

%INCLUDE ステートメントによってソース・テキスト内に %PROCESS ステートメントが組み込まれると、コンパイル・エラーになります。

117 ページの図 1 は、%INCLUDE ステートメントを使って、プロシージャー TEST 内に FUN 用のソース・ステートメントを組み込む方法を示しています。 ライブラリー HPU8.NEWLIB が修飾名 PLI.SYSLIB を持つ DD ステートメントに定義されていますが、これはこのジョブ用のカタログ式プロシージャーのステートメントに付け加えられます。 ソース・ステートメント・ライブラリーが SYSLIB という名前の DD ステートメントで定義されているため、%INCLUDE ステートメントに DD 名を入れる必要はありません。

ソース・プログラムや、入れようとするテキストにマクロ・ステートメントがまったく入っていなければ、プリプロセッサーを呼び出す必要はありません。

```
//OPT4#9     JOB
//STEP3      EXEC IBMZCBG,PARM.PLI='INC,S,A,X,NEST'
//PLI.SYSLIB DD DSN=HPU8.NEWLIB,DISP=OLD
//PLI.SYSIN  DD *
   TEST: PROC OPTIONS(MAIN) REORDER;
     DCL ZIP PIC '99999';           /* ZIP CODE                      */
     DCL EOF BIT INIT('0'B);
     ON ENDFILE(SYSIN) EOF = '1'B;
     GET EDIT(ZIP) (COL(1), P'99999');
     DO WHILE(¬EOF);
       PUT SKIP EDIT(ZIP, CITYFUN(ZIP)) (P'99999', A(16));
       GET EDIT(ZIP) (COL(1), P'99999');
     END;
     %PAGE;
     %INCLUDE FUN;
   END;                             /* TEST                          */
//GO.SYSIN DD *
95141
95030
94101
//
```

図 *1.* ライブラリーからのソース・ステートメントの組み込み

コンパイラー・リストの使用

コンパイルのときにコンパイラーは、大半がオプショナルのリストを生成しますが、そのリストには、ソース・プログラム、コンパイル、およびオブジェクト・モジュールに関する情報が入っています。

次のリストの説明は、印刷ページ上の外観について述べています。

注: コンパイラー・リストは利用可能ですが、これはプログラミング・インターフェースではないため、変更される可能性があります。

特定の処理段階に達する前にコンパイルが終了する場合、対応するリストは表示されません。

見出し情報

リストの最初のページは、製品番号、コンパイラー・バージョン番号、コンパイラー・ビルド日時を指定するストリング、およびコンパイル開始日時によって識別されます。このページおよび以降のページには番号が付けられます。

次にリストには、このコンパイルに指定されたすべてのオプションが表示されます。 これらのオプションは、NOOPTIONS オプションが指定されていても表示されます。以下のオプションが以下の順序で示されます。

- 初期インストール・オプション (インストール時のオプション・セットで、あらかじめ適用されるオプションです。これら以外のオプションは、後で追加されます)。
- z/OS UNIX では、IBM_OPTIONS 環境変数で指定されたオプション。

- コンパイラーに渡されるパラメーター・ストリング (すなわち、z/OS UNIX におけるコマンド行、またはバッチにおける PARM=) で指定されたオプション。
- コンパイラー・パラメーター・ストリングで指定されたオプション・ファイルで指定されたオプション。

  これには、各オプション・ファイルの名前とその内容が、コンパイラーが読み取ったそのままの形式で含まれます。
- ソース内の *PROCESS または %PROCESS 行で指定されたオプション。
- 最終インストール・オプション (インストールで設定されたオプション、および他の何らかのオプションの後に適用されるオプション)。

リストの終わり付近には、コンパイル時にエラーや警告状態がなにも検出されなかった旨のステートメントか、または 1 つ以上のエラーが検出された旨のメッセージが入ります。 メッセージのフォーマットについては、 125 ページの『メッセージと戻りコード』を参照してください。リストの最後から 2 番目の行は、コンパイルに要した時間を示します。リストの最後の行は `END OF COMPILATION OF xxxx` です (*xxxx* は外部プロシージャー名です)。 NOSYNTAX コンパイラー・オプションを指定した場合、またはコンパイルの初期段階でコンパイラーが異常終了した場合、外部プロシージャー名 *xxxx* は組み込まれずに切り捨てられて、この行は `END OF COMPILATION` となります。

以下のトピックでは、リストのオプション部分を指定順に説明します。

コンパイルに使用するオプション

OPTIONS オプションを指定すると、コンパイルに指定されたオプション (デフォルト・オプションを含む) の完全なリストが、次のページから示されます。

コンパイル時に最終的に有効になるすべてのオプションの設定値がリストに示されます。オプションの設定値が初期インストール・オプションの適用後にデフォルト設定値と異なる場合は、その行に正符号 (+) のマークが付けられます。

プリプロセッサー入力

MACRO オプションと INSOURCE オプションを両方とも指定すると、コンパイラーは、各行の左側に順次番号を付けてから、プリプロセッサーへの入力を 1 行に 1 レコードずつリストします。

プリプロセッサーがエラーまたはエラーの可能性を検出すると、そのページまたは入力リストに続くページにメッセージが印刷されます。このメッセージのフォーマットは、 125 ページの『メッセージと戻りコード』の説明にあるコンパイラー・メッセージのフォーマットと同じです。

SOURCE プログラム

SOURCE オプションを指定すると、コンパイラーは 1 行に 1 レコードずつリストします。 これらのレコードには、ソース行番号とソース・ファイル番号が必ず含まれます。ただし、ファイルに含まれる行の数が 999999 行以上になる場合、コンパイラーは、ファイルが大きすぎることを示すフラグをファイルに立て、そのファイルのソース行番号に含まれる下位 6 桁のみをリストします。

入力レコードにプリンター制御文字、%SKIP ステートメント、または %PAGE ステートメントが入っている場合は、それらの指定にしたがって行のスペーシングが行われます。

リストの印刷を停止するには、%NOPRINT ステートメントを使用します。

リストの印刷を再開するには、%PRINT ステートメントを使用します。

MACRO オプションを指定すると、ソース・リストでは、1 次入力データ・セットの %INCLUDE ステートメントの代わりに組み込まれたテキストが印刷されます。

ステートメントのネスト・レベル

NEST オプションを指定すると、見出し LEV と NT の下にあるステートメントまたは行番号の右側にブロック・レベルと DO レベルがそれぞれ出力されます。

次の例を参照してください。

```
Line.File LV NT
   1.0             A: PROC OPTIONS(MAIN);
   2.0     1       B: PROC;
   3.0     2          DCL K(10,10) FIXED BIN (15);
   4.0     2          DCL Y FIXED BIN (15) INIT (6);
   5.0     2          DO I=1 TO 10;
   6.0     2  1          DO J=1 TO 10;
   7.0     2  2             K(I,J) = N;
   8.0     2  2          END;
   9.0     2  1       BEGIN;
  10.0     3  1          K(1,1)=Y;
  11.0     3  1       END;
  12.0     2  1    END B;
  13.0     1       END A;
```

ATTRIBUTE と相互参照テーブル

ATTRIBUTES オプションを指定すると、コンパイラーは、ソース・プログラム内の ID リストの入った属性テーブルを、それぞれの宣言属性とデフォルト属性を付けて印刷します。XREF オプションを使用すると、コンパイラーは、ID が入っているステートメントのファイル番号と行番号を併記したうえで、ソース・プログラム内の ID のリストの入った相互参照テーブルを印刷します。

ATTRIBUTES と XREF を両方指定すると、2 つのテーブルが組み合わされます。これらのテーブル内で ID を明示的に宣言すると、コンパイラーはその DECLARE のファイル番号と行番号をリストします。 コンテキストで宣言された変数は +++++ とマークされ、それ以外の暗黙的に宣言された変数は ***** とマークされます。

属性テーブル

属性テーブルには、ソース・プログラムにおける ID のリストが、宣言された属性およびデフォルト属性とともに含まれます。

コンパイラーは、属性 INTERNAL および REAL を決して組み込みません。 それぞれと相対立する属性の EXTERNAL と COMPLEX が現れない限り、これらの属性を想定することができます。

ファイル ID に関しては、属性 FILE が常に現れ、属性 EXTERNAL は適用時に現れます。それ以外の場合は、コンパイラーは明示的に宣言された属性のみをリストします。

OPTIONS 属性は ENTRY 属性が適用されなければ表示されず、次のオプションのみが適宜表示されます。

- ASSEMBLER
- COBOL
- FETCHABLE
- FORTRAN
- NODESCRIPTOR
- RETCODE

コンパイラーは、配列用の次元属性をまず印刷します。境界は配列宣言のとおりに印刷されますが、式はアスタリスクに置き換えられます。ただし、式がコンパイラーにより定数に還元された場合は、定数の値が印刷されます。

文字ストリング、ビット・ストリング、漢字ストリング、または区域変数の場合、コンパイラーは宣言のとおりに長さを印刷しますが、式はアスタリスクに置き換えられます。ただし、式がコンパイラーにより定数に還元された場合は、定数の値が印刷されます。

相互参照テーブル

ATTRIBUTES と XREF を指定すると、相互参照テーブルと属性テーブルが結合されます。名前に対する属性のリストは、ファイル番号と行番号で識別されます。

以下の条件下では、相互参照テーブルの `Sets:` 部分に ID が表示されます。

- その ID が代入ステートメントのターゲットである。
- その ID が DO ループでループ制御変数として使用される。
- その ID が ALLOCATE ステートメントまたは LOCATE ステートメントの SET オプションで使用される。
- その ID が DISPLAY ステートメントの REPLY オプションで使用される。

未参照の ID がある場合、それらは別個のテーブルに示されます。

集合長さテーブル

集合長さテーブルは AGGREGATE オプションで取得されます。このテーブルには、配列ではなく構造体が組み込まれています。この構造体は非固定エクステントを保持し、構造体内部のエレメントのサイズとオフセットは、不正確であるか、または * として指定されます。

リストされた集合の場合、テーブルには次の情報が入っています。

- 宣言された集合のロケーション
- 集合名と集合内の各エレメント
- 集合の先頭からの各エレメントのバイト・オフセット
- 各エレメントの長さ
- 各集合、構造体、および副構造体の全長

- 各エレメントの次元の合計数

データ長テーブルに示されているデータ・オフセットを解釈する場合は注意が必要です。 奇数オフセットは、ハーフワード位置合わせ、フルワード位置合わせ、またはダブルワード位置合わせが行われていないデータ・エレメントを必ずしも表しません。ある構造体またはそのエレメント用に位置合わせ済みの属性を指定したり暗示したりすると、テーブルの初めに対して適切な位置合わせが行われていることをテーブルが示さなくても、適切な位置合わせ要件はその構造体内の他のエレメントに合致します。

2 つの構造体エレメント間に埋め込みがある場合は、/*PADDING*/ というコメントと、適切な診断情報が表示されます。

ステートメント・オフセット・アドレス

LIST コンパイル・オプションを使用すると、コンパイラーはコンパイラー・リストに疑似アセンブラー・リストを組み込みます。 このリストには、BLKOFF コンパイラー・オプションの設定によって異なる意味を持つオフセットが命令ごとに含まれています。

- BLKOFF オプションを指定した場合、このオフセットは、命令が属している関数またはサブルーチン用の 1 次エントリー・ポイントからの命令のオフセットになります。 つまり、このオプションのもとでは、オフセットはそれぞれの新規ブロックごとに再設定されます。
- NOBLKOFF オプションを指定した場合、このオフセットは、コンパイル単位の開始位置からの命令のオフセットになります。 つまり、このオプションのもとではオフセットは累積されます。

疑似アセンブラー・リストには、各ブロックのコードの終わりに現行モジュールの開始位置からのブロックのオフセットも含まれています (それぞれのステートメントに表示されるオフセットを、ブロックまたはモジュールのオフセットに変換できるようにするため)。

これらのオフセットを、ランタイム・エラー・メッセージに示されたオフセットと共に使用して、このメッセージが該当するステートメントを判別できます。

OFFSET オプションは、それぞれのステートメントに対して、そのステートメントに属する最初の命令のオフセットを提供するテーブルを作成します。

122 ページの図 2 に示す例では、メッセージは SUB1 の入り口からオフセット +98 で条件が生じたことを示しています。 このオフセットは、コンパイラー・リストの抜粋では行番号 8 と関連して示されています。このエラーがあることを示すステートメントからの実行時出力は、 122 ページの図 3 のように示されます。

```
Compiler Source
    Line.File
       1.0
       2.0      TheMain: proc options( main );
       3.0        call sub1();
       4.0        Sub1: proc;
       5.0          dcl (i, j) fixed bin(31);
       6.0
       7.0          i = 0; j = 0;
       8.0          j = j / i;
       9.0          put skip data( j );
      10.0        end Sub1;
      11.0      end TheMain;

 . . .

OFFSET OBJECT CODE       LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

000000                   000002 |        THEMAIN  DS       0D
000000  47F0  F024       000002 |                 B        36(,r15)
000004  01C3C5C5                                          CEE eyecatcher
000008  000000B0                                          DSA size
00000C  000001F8                                          =A(PPA1-THEMAIN)
000010  47F0  F001       000002 |                 B        1(,r15)

 . . .

000000                   000004 |        SUB1     DS       0D
000000  47F0  F024       000004 |                 B        36(,r15)
000004  01C3C5C5                                          CEE eyecatcher
000008  00000140                                          DSA size
00000C  00000190                                          =A(PPA1-SUB1)
000010  47F0  F001       000004 |                 B        1(,r15)

...

000086  5020  D0B8       000007 |                 ST       r2,I(,r13,184)
00008A  1842             000007 |                 LR       r4,r2
00008C  5040  D0BC       000007 |                 ST       r4,J(,r13,188)
000090  5800  D0B8       000008 |                 L        r0,I(,r13,184)
000094  8E40  0020       000008 |                 SRDA     r4,32
000098  1D40             000008 |                 DR       r4,r0
00009A  1805             000008 |                 LR       r0,r5
00009C  5000  D0BC       000008 |                 ST       r0,J(,r13,188)
0000A0  4100  D0C0       000009 |                 LA       r0,_temp1(,r13,192)
0000A4  5000  D130       000009 |                 ST       r0,_temp2(,r13,304)
0000A8  A708  5A88       000009 |                 LHI      r0,H'23176'
0000AC  4000  D0EC       000009 |                 STH      r0,_temp1(,r13,236)
0000B0  5800  6004       000009 |                 L        r0,SYSPRINT(,r6,4)
0000B4  5000  D12C       000009 |                 ST       r0,_temp2(,r13,300)
0000B8  4100  0001       000009 |                 LA       r0,1
0000BC  5000  D0C0       000009 |                 ST       r0,_temp1(,r13,192)
0000C0  4100  D128       000009 |                 LA       r0,_temp2(,r13,296)

...
```

図 2. ステートメント番号の検索 (コンパイラー・リストの例)

```
Message :

IBM0301S ONCODE=320  The ZERODIVIDE condition was raised.
         From entry point SUB1 at compile unit offset +00000098
         at entry offset +00000098 at address 0EB00938.
```

図 3. ステートメント番号の検索 (ランタイム・メッセージの例)

ダンプと ON ユニット SNAP エラー・メッセージ内に示された項目オフセットをこのテーブルと対比すれば、誤りのあるステートメントを見つけ出すことができます。このステートメントを識別するには、メッセージ内で指名されたブロックに関連したテーブルのセクションを探し出してから、メッセージ内のオフセット以下かまたはそれと等しい最大オフセットを見つけ出します。 このオフセットに関連したステートメント番号が求める番号です。

ストレージ・オフセット・リスト

MAP コンパイル・オプションを使用すると、コンパイラーはコンパイラー・リストにストレージ・オフセット・リストを組み込みます。

ストレージ・オフセット・リストは、以下のレベル 1 変数がプログラムで使用された場合に、これらのストレージ内のロケーションを示します。

- AUTOMATIC
- CONTROLLED (PARAMETER を除く)
- FETCHABLE でない STATIC (ENTRY CONSTANT を除く)

このリストには、コンパイラーが生成した一時データの一部も組み込まれます。

調節可能エクステントを持つ AUTOMATIC 変数が使用された場合は、このテーブルに、以下の 2 つの項目があります。

- 変数名の前に `_addr` が付いた項目 - 変数のアドレスのロケーションを示します
- 変数名の前に `_desc` が付いた項目 - 変数の記述子のアドレスのロケーションを示します

STATIC 変数および CONTROLLED 変数を使用した場合、ストレージ・ロケーションは RENT/NORENT コンパイラー・オプションに依存し、NORENT オプション指定した場合は、CONTROLLED 変数のロケーションも WRITABLE/NOWRITABLE コンパイラー・オプションに依存します。

ストレージ・オフセット・リストの最初の列には IDENTIFIER というラベルが付けられています。この列には、4 列目に示されているロケーションが代入される変数の名前が含まれています。

ストレージ・オフセット・リストの 2 番目の列は、DEFINITION というラベルが付けられ、*"B-F:N"* という形式のストリングを含んでいます。

- ここで、*B* は変数が宣言されたブロックの番号です。

  このブロック番号に応じてブロック名をブロック名リストで検索できます。このブロック名リストはストレージ・オフセット・リスト (および、存在する場合は、疑似アセンブリー・リスト) の前にあります。
- ここで、*F* は変数が宣言されたソース・ファイルの番号です。

  このファイル番号に対応するファイル名を、ファイル参照テーブル (コンパイル・リスト全体の終わりのあたりにある) で検索できます。
- ここで、*N* は変数がソース・ファイル内で宣言されたソース行の番号です。

ストレージ・オフセット・リストの 3 番目の列は、ATTRIBUTES というラベルが付けられ、変数のストレージ・クラスを示します。

ストレージ・オフセット・リストの 4 番目の列は、ラベルがなく、変数のロケーションを検索する方法を示します。

このストレージ・オフセット・リストはブロック別および変数名別にソートされ、ユーザー変数のみも含まれます。 また、MAP オプションを指定すると、コンパイラーは以下のマップも生成します。

- すべての STATIC 変数をリストした「静的マップ」(16 進オフセット別にソート)。
- ブロックごとにすべての AUTOMATIC 変数をリストした「自動マップ」(16 進オフセット別にソート)。

PL/I 言語のマッピング規則で、構造体を、構造体が開始していると思われる場所から最大で 8 バイト分オフセットする必要がある場合があります。例えば、次のように宣言された AUTOMATIC 構造体 A を考えてみます。

```
dcl
  1 A,
    2 B char(2),
    2 C fixed bin(31);
```

C は、4 バイト境界に位置合わせする必要があり、この構造体には 2 バイトの埋め込みが必要です。ただし、PL/I はその 2 バイトを、B の後ではなく B の前に配置します。構造体の先頭の前にあるこの 2 バイトの「埋め込み」は、構造体のハング・バイト と呼ばれます。

これらのハング・バイトは、コンパイラーによって生成された「自動マップ」にも反映されます。 「ストレージ・オフセット・リスト」には、ハング・バイトが含まれない A のオフセットと長さが示されます。

```
A     Class = automatic,   Location = 186 : 0xBA(r13),     Length = 6
```

一方、「自動マップ」には、ハング・バイトが含まれた A のオフセットと長さが示されます。

```
OFFSET (HEX)   LENGTH (HEX)   NAME

     98              8        #MX_TEMP1
     A0             18        _Sfi
     B8              8        A
```

式および属性のリスト

DECOMP コンパイラー・オプションを使用する場合、コンパイラーは、ソース・プログラムで使用されるすべての式に対するすべての中間式とその属性を示すセクションをコンパイラー・リストに組み込みます。

関連情報:

24 ページの『DECOMP』
DECOMP オプションは、コンパイルで使用される式の分解を示すリスト・セクションを生成するようにコンパイラーに指示します。

ファイル参照テーブル

ファイル参照テーブルには、コンパイル時に読み取られるファイルに関する情報が登録されています。

ファイル参照テーブルは、以下のファイル情報がリストされた 3 つの列で構成されます。

- コンパイラーによってファイルに割り当てられた番号
- ファイルの組み込み元データ
- ファイルの名前

最初にリストされるファイルはソース・ファイルなので、組み込み元の列にある最初の項目はブランクです。 この列にある後続のエントリーには組み込みステートメントの行番号が示され、その後にピリオドと、組み込みステートメントが含まれるソース・ファイルのファイル番号が続きます。

ファイルが PDS または PDSE のメンバーである場合、ファイル名は完全修飾データ・セット名とメンバー名を示します。

ファイルがサブシステム (ライブラリアンなど) を介して組み込まれた場合、ファイル名の形式は `DD:ddname(member)` になります。ただし、

- *ddname* は、%INCLUDE ステートメントに指定された DD 名 (あるいは、DD 名が指定されなかった場合は SYSLIB)
- *member* は %INCLUDE ステートメントで指定されるメンバー名

メッセージと戻りコード

プリプロセッサーまたはコンパイラーがエラーまたはエラーの可能性を検出すると、メッセージが生成されます。 コンパイラーは、コンパイル・ジョブまたはジョブ・ステップごとに、成功または失敗の程度を示す戻りコードを生成します。

メッセージ

プリプロセッサーが生成したメッセージは、プリプロセッサーが処理したステートメントのリストの直後のリストに印刷されます。 %NOTE ステートメントを使用すれば、プリプロセス段階で独自のメッセージを生成できます。このようなメッセージは、特定の置換が何回行われたかを示すのに使用できます。 コンパイラーが生成したメッセージはリストの最後に印刷されます。

メッセージが生成されないコンパイルの場合、コンパイラーは、コンパイラー・メッセージがリストされるはずの行に「**コンパイラー・メッセージはありません** (no compiler messages)」というメッセージを組み込みます。

メッセージは次のフォーマットで示されます。

```
PPPnnnnI X
```

PPP
メッセージの発信元を識別する接頭部です (例えば、IBM は PL/I コンパイラーを示します)。

nnnn
4 桁のメッセージ番号です。

X 重大度コードを識別するものです。

メッセージはすべて重大度により格付けされます。重大度コードは、I、W、E、S、および U です。

コンパイラーは、表 6 に示すように、FLAG オプションで指定されたメッセージの重大度に等しいかより大きい重大度を持つメッセージだけをリストします。

表 *6.* リストされたメッセージの最低重大度を選択するための *FLAG* オプションの使用

メッセージのタイプ	オプション
通知	FLAG(I)
警告	FLAG(W)
エラー	FLAG(E)
重大エラー	FLAG(S)
回復不能エラー	常にリストされる

各メッセージのテキスト、説明、および推奨プログラマー応答については、「*Enterprise PL/I* メッセージおよびコード」を参照してください。

戻りコード

各コンパイル・ジョブまたはジョブ・ステップごとにコンパイラーは、操作がどの程度成功または失敗したかをオペレーティング・システムに示すための戻りコードを生成します。 z/OS の場合、このコードはステップの終わり メッセージに現れますが、その前には各ステップ別のジョブ制御ステートメントとジョブ・スケジューラー・メッセージのリストが入っています。

各重大度コードとそれに対応する戻りコードについて、表 7 で説明します。

表 *7. PL/I* エラー・コードと戻りコードの説明

重大度コード	戻りコード	メッセージ・タイプ	説明
I	0000	通知	コンパイルされたプログラムは正常に実行されます。非効率になる可能性のあるコードや、その他の注意すべき条件があると、コンパイラーはユーザーに通知します。
W	0004	警告	構文的には有効でも、ステートメントにエラー (警告対象) がある場合があります。 コンパイルされたプログラムは正常に実行されても、予期に反する結果になったり、著しく非効率になったりする場合があります。
E	0008	エラー	コンパイラーにより修正されたエラー。コンパイルされたプログラムは正常に実行されても、予想とは異なる結果になる場合があります。
S	0012	重大	コンパイラーにより修正されないエラー。プログラムがコンパイルされ、オブジェクト・モジュールが生成されても、そのモジュールを使用してはなりません。
U	0016	回復不能	コンパイルを強制終了させるエラー。 オブジェクト・モジュールは正常には作成されません。

注: コンパイラー・メッセージはこれらの重大度レベルによりグループ別に印刷されます。

例

下記のコンパイラー・リスト例は、コンパイラーが次の `msgsumm` プログラムをオプション PP(SQL,MACRO,CICS)、SOURCE、FLAG(I)
、INSOURCE、MSGSUMMARY(XREF) でコンパイルしたときに生成されます。

```
msgsumm: proc;

  exec sql include sqlca;

  exec cics what now;

  exec cics not this;

  %dcl z0 fixed bin;
  %dcl z1 fixed dec;
end;
```

注: このプログラムには意図的な誤りがあります。 MSGSUMMARY オプションが指定されているため、コンパイラーはリストの最後に「Summary of Messages」セクションを組み込んでいます。MSGSUMMARY オプションの XREF サブオプションが指定されているため、このセクションには、要約に示された各メッセージに関連する行番号も含まれています。

```
| 5655-PL5  IBM(R) Enterprise PL/I for z/OS       V5.R2.M0 (Built:20170804)                    2017.08.14 16:39:08   Page     1
                    Options Specified
  Install:
  Command: +DD:OPTIONS
  File: DD:OPTIONS
    PP(SQL,MACRO,CICS),S,F(I),IS,MSGSUMMARY(XREF)
  Install:
| 5655-PL5  IBM(R) Enterprise PL/I for z/OS                                                   2017.08.14 16:39:08   Page     2
 SQL (Built:20170602) Preprocessor Source
    Line.File
       1.0
       2.0      msgsumm: proc;
       3.0
       4.0        exec sql include sqlca;
       5.0
       6.0        exec cics what now;
       7.0        exec cics not this;
       8.0
       9.0        %dcl z0 fixed bin;
      10.0        %dcl z1 fixed dec;
      11.0      end;
| 5655-PL5  IBM(R) Enterprise PL/I for z/OS                                                   2017.08.14 16:39:08   Page     3
 SQL Preprocessor Options Used
     CCSID0
   NOCODEPAGE
     DEPRECATE( STMT() )
   NOEMPTYDBRM
   NOINCONLY
   NOWARNDECP
 DB2 for z/OS Coprocessor Options Used
   APOST
   APOSTSQL
   ATTACH(TSO)
   CCSID(500)
   CONNECT(2)
   DEC(15)
   DECP(DSNHDECP)
   FLOAT(S390)
   NEWFUN(V11)
   TWOPASS
   PERIOD
   STDSQL(NO)
   SQL(DB2)
   NOXREF
   NO SOURCE
   DSNHDECP LOADED FROM - (DSNB10.SDSNLOAD(DSNHDECP))
| 5655-PL5  IBM(R) Enterprise PL/I for z/OS                                                   2017.08.14 16:39:08   Page     4
 SQL Preprocessor Messages
 Message       Line.File Message Description
 IBM3250I W       4.0    DSNH053I DSNHPSRV  NO SQL STATEMENTS WERE FOUND
 IBM3000I I              DSNH4790I DSNHPSRV  DSNHDECP HAS CCSID 500 IN EFFECT
```

| 図 4. コンパイラー・リスト例

```
5655-PL5  IBM(R) Enterprise PL/I for z/OS                          2017.08.14 16:39:08   Page     5
 MACRO (Built:20170501) Preprocessor Source
    Line.File
       1.0
       2.0      msgsumm: proc;
       2.0
       3.0
       4.0
       4.0      /*$*$*$
       4.0        exec sql include sqlca
       4.0       $*$*$*/
       4.0      DCL
       4.0        1 SQLCA  ,
       4.0          2 SQLCAID     CHAR(8),
       4.0          2 SQLCABC     FIXED BIN(31),
       4.0          2 SQLCODE     FIXED BIN(31),
       4.0          2 SQLERRM     CHAR(70) VAR,
       4.0          2 SQLERRP     CHAR(8),
       4.0          2 SQLERRD(6)  FIXED BIN(31),
       4.0          2 SQLWARN,
       4.0            3 SQLWARN0  CHAR(1),
       4.0            3 SQLWARN1  CHAR(1),
       4.0            3 SQLWARN2  CHAR(1),
       4.0            3 SQLWARN3  CHAR(1),
       4.0            3 SQLWARN4  CHAR(1),
       4.0            3 SQLWARN5  CHAR(1),
       4.0            3 SQLWARN6  CHAR(1),
       4.0            3 SQLWARN7  CHAR(1),
       4.0          2 SQLEXT,
       4.0            3 SQLWARN8  CHAR(1),
       4.0            3 SQLWARN9  CHAR(1),
       4.0            3 SQLWARNA  CHAR(1),
       4.0            3 SQLSTATE  CHAR(5);
       4.0
       5.0
       6.0        exec cics what now;
       7.0        exec cics not this;
       8.0
       9.0        %dcl z0 fixed bin;
      10.0        %dcl z1 fixed dec;
      11.0      end;
```

コンパイラー・リスト例 *(続き)*

```
5655-PL5  IBM(R) Enterprise PL/I for z/OS                                  2017.08.14 16:39:08   Page     6
 MACRO Messages
 Message       Line.File Message Description
 IBM3552I E       9.0    The statement element BIN is invalid. The statement
                         will be ignored.
 IBM3552I E      10.0    The statement element DEC is invalid. The statement
                         will be ignored.
 IBM3258I W       9.0    Missing ; assumed before BIN.
 IBM3258I W      10.0    Missing ; assumed before DEC.
5655-PL5  IBM(R) Enterprise PL/I for z/OS                                  2017.08.14 16:39:08   Page     7
 CICS (Built:20170504) Preprocessor Source
    Line.File
       1.0
       2.0      MSGSUMM: PROC;
       2.0
       3.0
       4.0
       4.0      /*$*$*$
       4.0        exec sql include sqlca
       4.0       $*$*$*/
       4.0      DCL
       4.0        1 SQLCA  ,
       4.0          2 SQLCAID     CHAR(8),
       4.0          2 SQLCABC     FIXED BIN(31),
       4.0          2 SQLCODE     FIXED BIN(31),
       4.0          2 SQLERRM     CHAR(70) VAR,
       4.0          2 SQLERRP     CHAR(8),
       4.0          2 SQLERRD(6)  FIXED BIN(31),
       4.0          2 SQLWARN,
       4.0            3 SQLWARN0  CHAR(1),
       4.0            3 SQLWARN1  CHAR(1),
       4.0            3 SQLWARN2  CHAR(1),
       4.0            3 SQLWARN3  CHAR(1),
       4.0            3 SQLWARN4  CHAR(1),
       4.0            3 SQLWARN5  CHAR(1),
       4.0            3 SQLWARN6  CHAR(1),
       4.0            3 SQLWARN7  CHAR(1),
       4.0          2 SQLEXT,
       4.0            3 SQLWARN8  CHAR(1),
       4.0            3 SQLWARN9  CHAR(1),
       4.0            3 SQLWARNA  CHAR(1),
       4.0            3 SQLSTATE  CHAR(5);
       4.0
       5.0
       6.0        EXEC CICS WHAT NOW;
       7.0        EXEC CICS NOT THIS;
       8.0
      11.0      END;
```

コンパイラー・リスト例 *(続き)*

```
5655-PL5  IBM(R) Enterprise PL/I for z/OS                                  2017.08.14 16:39:08   Page     8
 CICS Messages
 Message       Line.File Message Description
 IBM3750I S       6.0    DFH7059I S  WHAT COMMAND IS NOT VALID AND IS NOT
                         TRANSLATED.
 IBM3750I S       7.0    DFH7059I S  NOT COMMAND IS NOT VALID AND IS NOT
                         TRANSLATED.
5655-PL5  IBM(R) Enterprise PL/I for z/OS                                  2017.08.14 16:39:08   Page     9
 No Compiler Messages
 File Reference Table
   File    Included From  Name
      0                   DD:SYSIN
5655-PL5  IBM(R) Enterprise PL/I for z/OS                                  2017.08.14 16:39:08   Page    10
 Summary of Messages
 Component Message     Total  Default Message Description
 SQL       IBM3250I W      1  DSNH053I DSNHPSRV  NO SQL STATEMENTS WERE FOUND
                              Refs: 4.0
 SQL       IBM3000I I      1  DSNH4790I DSNHPSRV  DSNHDECP HAS CCSID 500 IN EFFECT
                              Refs: 0.0
 MACRO     IBM3552I E      2  The statement element %1 is invalid. The statement will be ignored.
                              Refs: 9.0 10.0
 MACRO     IBM3258I W      2  Missing %1 assumed before %2.
                              Refs: 9.0 10.0
 CICS      IBM3750I S      2  DFH7059I S  WHAT COMMAND IS NOT VALID AND IS NOT TRANSLATED.
                              Refs: 6.0 7.0
 Compiler  <none>

 Component    Return Code    Messages (Total/Suppressed)    Time
 SQL             4               2  /  0                   0 secs
 MACRO           8               4  /  0                   0 secs
 CICS           12               2  /  0                   0 secs
 Compiler        0               0  /  0                   0 secs
 End of compilation
```

コンパイラー・リスト例 *(続き)*

第 2 章 PL/I プリプロセッサー

PL/I コンパイラーを使用するときは、ご使用のプログラムにおいて組み込みプリプロセッサーを 1 つ以上指定できます。組み込みプリプロセッサー、マクロ・プリプロセッサー、SQL プリプロセッサー、または CICS プリプロセッサーを指定できます。また、これらのプリプロセッサーを呼び出す順序を指定できます。

- インクルード・プリプロセッサーは、特殊なインクルード・ディレクティブを処理し、外部ソース・ファイルを取り込みます。
- マクロ・プリプロセッサーは、% ステートメントとマクロに基づいていて、ソース・プログラムを変更します。
- SQL プリプロセッサーは、ソース・プログラムを変更し、EXEC SQL ステートメントを PL/I ステートメントに変換します。
- CICS プリプロセッサーは、ソース・プログラムを変更し、EXEC CICS ステートメントを PL/I ステートメントに変換します。

各プリプロセッサーは、ニーズに合わせて処理を調整するために使用できるオプションをいくつかサポートしています。

コンパイル時オプション MDECK、INSOURCE、および SYNTAX は、PP オプションも指定される場合にのみ有効です。

関連情報:

60 ページの『MDECK』
MDECK オプションを指定すると、プリプロセッサーは、z/OS の場合は SYSPUNCH DD ステートメントで定義されたファイルに、z/OS UNIX の場合は .dek ファイルに、プリプロセッサーの出力のコピーを作成します。

48 ページの『INSOURCE』
INSOURCE オプションは、PL/I マクロ、CICS、または SQL のプリプロセッサーが変換できるように、コンパイラーがソース・プログラムのリストを組み込むことを指定します。

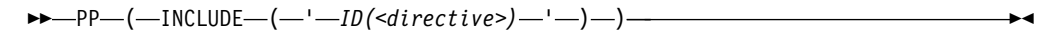
98 ページの『SYNTAX』
SYNTAX オプションは、回復不能エラーが生じない限り、MACRO オプションを指定した場合、プリプロセスの後でコンパイラーが、 引き続き構文検査に移ることを指定します。 コンパイラーが引き続きコンパイルを行うかどうかは、NOSYNTAX オプションで指定されたエラーの重大度によって 決まります。

インクルード・プリプロセッサー

インクルード・プリプロセッサーを使用すると、PL/I ディレクティブ %INCLUDE 以外のインクルード・ディレクティブを使用して、外部ソース・ファイルをプログラムに取り込むことができます。

次の構文図は、INCLUDE プリプロセッサーによってサポートされるオプションを示しています。

© Copyright IBM Corp. 1999, 2017



ID インクルード・ディレクティブの名前を指定します。 最初の一続きの非ブランク文字としてのこのディレクティブで始まる行は、インクルード・ディレクティブとして扱われます。

指定するディレクティブの後に、1 つ以上のブランク、およびインクルード・メンバー名が必要で、最後にオプションでセミコロンを付けることができます。ddname(membername) の構文はサポートされません。

次の例では、1 つ目のインクルード・ディレクティブは有効で、2 つ目のものは無効です。

```
++include payroll
++include syslib(payroll)
```

例 1

次の例では、-INC (および場合によっては先行ブランク) から始まる行がすべて、インクルード・ディレクティブとして扱われます。

```
pp( include( 'id(-inc)'))
```

例 2

次の例では、++INCLUDE (および場合によっては先行ブランク) から始まる行がインクルード・ディレクティブとして扱われます。

```
pp( include( 'id(++include)'))
```

マクロ・プリプロセッサー

マクロを使用すれば、インプリメンテーションの詳細と処理対象のデータを隠して演算のみを表す方法で、通常使用される PL/I コードを作成できます。汎用のサブルーチンとは対照的に、マクロでは、個別用途ごとに必要となるコードのみを生成できます。マクロ・プリプロセッサーは、MACRO オプションまたは PP(MACRO) オプションを指定することによって呼び出すことができます。

PP(MACRO) はオプションなしで指定することも、 133 ページの『マクロ・プリプロセッサーのオプション』で説明されているオプションとともに指定することもできます。

これらすべてのオプションに対してデフォルトが指定された場合のマクロ・プリプロセッサーの動作は OS PL/I V2R3 マクロ・プリプロセッサーの動作と同じです。

オプションを指定する場合、リストは引用符 (左右の引用符が一致すれば、単一引用符でも二重引用符でもかまいません) で囲まなければなりません。例えば、FIXED(BINARY) オプションを指定するには、PP(MACRO('FIXED(BINARY)')) と指定します。

複数のオプションを指定する場合は、コンマまたは 1 つ以上のブランクでオプションを分離する必要があります。例えば、CASE(ASIS) および RESCAN(UPPER) オ

プションを指定するには、PP(MACRO('CASE(ASIS) RESCAN(UPPER)')) または PP(MACRO("CASE(ASIS),RESCAN(UPPER)")) と指定することができます。 オプションは任意の順序で指定できます。

コンパイラーのマクロ・プリプロセス機能については、「*PL/I* 言語解説書」を参照してください。

マクロ・プリプロセッサーのオプション

このセクションでは、マクロ・プリプロセッサーがサポートするオプションについて説明します。

CASE

このオプションは、プリプロセッサーが入力テキストを大文字に変換する必要があるかどうかを指定します。

UPPER
入力テキストを大文字に変換します。

ASIS
入力テキストは「現状のまま」です。

GRAPHIC オプションでは、CASE(ASIS) が常に有効です。

DBCS

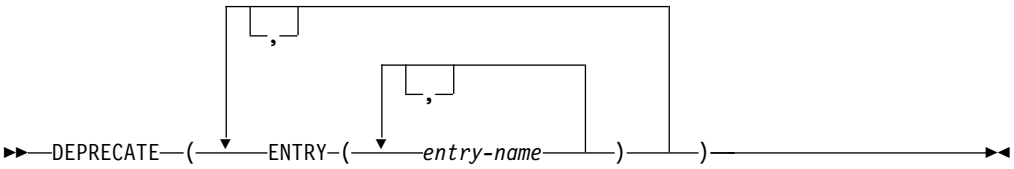
このオプションは、テキスト置換時にプリプロセッサーが DBCS を正規化するかどうかを指定します。

EXACT
入力テキストは「現状のまま」です。プリプロセッサーは <kk.B> と <kk>B を別の名前として扱います。

INEXACT
入力テキストは「正規化」されます。プリプロセッサーは <kk.B> と <kk>B を同じ名前の 2 つのバージョンとして扱います。

DEPRECATE

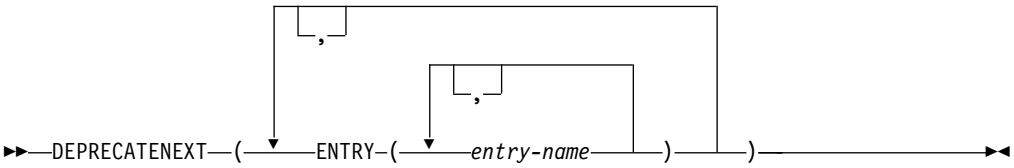
このオプションは、エラー・メッセージを発行して非推奨にしたいマクロ・プロシージャーの使用箇所にフラグを立てます。



ENTRY
これは、*entry-name* という名前のマクロ・プロシージャーのすべての使用箇所にフラグを立てます。

DEPRECATENEXT

このオプションは、警告メッセージを発行して非推奨にしたいマクロ・プロシージャーの使用箇所にフラグを立てます。



ENTRY
これは、*entry-name* という名前のマクロ・プロシージャーのすべての使用箇所にフラグを立てます。

FIXED

このオプションは、FIXED 変数のデフォルト基数を指定します。

DECIMAL
FIXED 変数の属性は REAL FIXED DEC(5) になります。

BINARY
FIXED 変数の属性は REAL SIGNED FIXED BIN(31) になります。

INCONLY

INCONLY オプションは、プリプロセッサーで %INCLUDE および %XINCLUDE ステートメントのみを処理する必要があることを指定します。

NOINCONLY オプションは、プリプロセッサーですべてのプリプロセッサー・ステートメントを処理する必要があり、%INCLUDE および %XINCLUDE ステートメントのみではないことを指定します。

INCONLY オプションが有効な場合は、マクロとして INCLUDE も XINCLUDE も使用できません。

- プロシージャー名
- ステートメント・ラベル
- 変数名

INCONLY オプションおよび NOINCONLY オプションは同時に指定できません。

また、互換性のため、NOINCONLY がデフォルトです。

NAMEPREFIX

NAMEPREFIX オプションは、プリプロセッサー・プロシージャーおよび変数の名前の先頭文字が指定された文字でなければならないことを指定します。

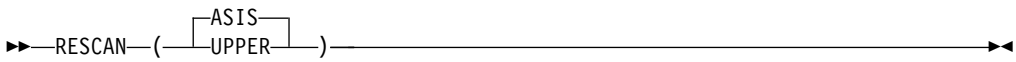
NONAMEPREFIX オプションは、プリプロセッサー・プロシージャーおよび変数の名前が特定の 1 文字で始まる必要がないことを指定します。

文字は「現状のまま」で指定し、引用符で囲まないようにしてください。

デフォルトは NONAMEPREFIX です。

RESCAN

このオプションは、テキストの再スキャンのとき、プリプロセッサーが ID の大/小文字をどのように処理するかを指定します。



UPPER
再スキャンは大文字小文字を区別しません。

ASIS
再スキャンは大文字小文字を区別します。

このオプションの影響を見るため、次のコード・フラグメントについて考えてみましょう。

```
%dcl eins char ext;
%dcl text char ext;

%eins = 'zwei';

%text = 'EINS';
display( text );

%text = 'eins';
display( text );
```

PP(MACRO('RESCAN(ASIS)')) を指定してコンパイルを行った場合、2 番目の表示ステートメントで、値 `text` は `eins` で置き換えられますが、それ以上の置き換え

は行われません。これは、RESCAN(ASIS) では eins とマクロ変数 eins が一致しないためです (前者は現状のままであるのに対して後者は大文字になる)。そのため、次のテキストが生成されます。

```
DISPLAY( zwei );

DISPLAY( eins );
```

しかし、PP(MACRO('RESCAN(UPPER)')) を指定してコンパイルを行った場合、2番目の表示ステートメントで、text の値は eins で置き換えられますが、RESCAN(UPPER) では eins とマクロ変数 eins が一致するため (どちらも大文字)、さらに置き換えが行われます。そのため、次のテキストが生成されます。

```
DISPLAY( zwei );

DISPLAY( zwei );
```

つまり、RESCAN(UPPER) は大/小文字の区別を無視し、RESCAN(ASIS) は大/小文字を区別します。

マクロ・プリプロセッサーの例

プリプロセッサーを使用してソース・デックを作成する方法を以下に例示します。

137 ページの図 5 に示されている例で、ソース・ステートメントは、プリプロセッサー変数 USE に割り当てられている値に応じてサブルーチン (CITYSUB) または関数 (CITYFUN) のいずれかを表します。

SYSPUNCH に使用する DSNAME には、プリプロセッサー出力が入るソース・プログラム・ライブラリーを指定します。 通常、コンパイルが続行され、プリプロセッサー出力がコンパイルされます。

```
//OPT4#8  JOB
//STEP2 EXEC  IBMZC,PARM.PLI='MACRO,MDECK,NOCOMPILE,NOSYNTAX'
//PLI.SYSPUNCH DD  DSNAME=HPU8.NEWLIB(FUN),DISP=(NEW,CATLG),UNIT=SYSDA,
//          SPACE=(TRK,(1,1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//PLI.SYSIN  DD *
 /* GIVEN ZIP CODE, FINDS CITY                                      */
 %DCL USE CHAR;
 %USE = 'FUN'                  /* FOR SUBROUTINE, %USE = 'SUB' */ ;
 %IF USE = 'FUN' %THEN %DO;
 CITYFUN: PROC(ZIPIN) RETURNS(CHAR(16)) REORDER; /* FUNCTION        */
                       %END;
                %ELSE %DO;
 CITYSUB: PROC(ZIPIN, CITYOUT) REORDER;     /* SUBROUTINE           */
   DCL CITYOUT CHAR(16);          /* CITY NAME                       */
                       %END;
   DCL (LBOUND, HBOUND) BUILTIN;
   DCL ZIPIN PIC '99999';        /* ZIP CODE                         */
   DCL 1 ZIP_CITY(7) STATIC,     /* ZIP CODE - CITY NAME TABLE       */
         2 ZIP PIC '99999' INIT(
                     95141, 95014, 95030,
                     95051, 95070, 95008,
                     0),           /* WILL NOT LOOK AT LAST ONE      */
         2 CITY CHAR(16) INIT(
                     'SAN JOSE', 'CUPERTINO', 'LOS GATOS',
                     'SANTA CLARA', 'SARATOGA', 'CAMPBELL',
                     'UNKNOWN CITY');  /* WILL NOT LOOK AT LAST ONE  */
   DCL I FIXED BIN(31);
   DO I = LBOUND(ZIP,1) TO             /* SEARCH FOR ZIP IN TABLE    */
          HBOUND(ZIP,1)-1              /* DON'T LOOK AT LAST ELEMENT */
          WHILE(ZIPIN ¬= ZIP(I));
   END;
 %IF USE = 'FUN' %THEN %DO;
   RETURN(CITY(I));                    /* RETURN CITY NAME           */
                       %END;
                %ELSE %DO;
   CITYOUT=CITY(I);                    /* RETURN CITY NAME           */
                       %END;
 END;
```

図 *5.* ソース・デックを作成するためのマクロ・プリプロセッサーの使用

SQL プリプロセッサー

通常、PL/I プログラムのコーディングは、プログラムが DB2 データベースにアクセスする場合でもアクセスしない場合でも同じです。ただし、DB2 データの検索、更新、挿入、および削除を行ったり、他の DB2 サービスを使用したりするには、SQL ステートメントを使用する必要があります。 PL/I アプリケーションでは、動的および静的の EXEC SQL ステートメントを使用できます。

DB2 とやり取りするには、次の作業を行う必要があります。

- 必要な SQL ステートメントをコーディングし、**EXEC SQL** で区切る。
- DB2 プリコンパイラーを使用する。または、DB2 for z/OS バージョン 9 リリース 1 以降を使用している場合は、PL/I PP(SQL()) コンパイラー・オプションを指定してコンパイルを行う。

EXEC SQL サポートを利用するためには、まず DB2 システムへのアクセス権限が必要です。権限については、担当の DB2 データベース管理者にお問い合わせください。

プログラミングとコンパイルに関する考慮事項

PL/I SQL プリプロセッサーを使用すると、組み込み SQL ステートメントを含むソース・プログラムはコンパイル時に PL/I コンパイラーによって処理され、ユーザーは別個のプリコンパイル・ステップを使用する必要がありません。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、PL/I SQL プリプロセッサーを使用することをお勧めします。

このプリプロセッサーを使用すると、デバッグ中に SQL ステートメントのみが表示される (生成された PL/I ソースは表示されない) ため、IBM Debug Tool による対話式デバッグが強化されます。ただし、SQL プリプロセッサーを使用するには、DB2 for z/OS バージョン 9 リリース 1 以降が必要です。

このプリプロセッサーを使用すると、SQL プログラム上の DB2 プリコンパイラーの制限の一部が解除されます。 このプリプロセッサーを使用して SQL ステートメントを処理すると、次のことが可能になります。

- ネストされた SQL INCLUDE ステートメントを使用する。
- 構造化ホスト変数の完全修飾名を使用する。
- トップレベルのソース・ファイル内だけでなく、ネストされた PL/I プログラムの任意のレベルで SQL ステートメントを組み込む。
- PL/I データ・タイプを指定できる任意の場所で、SQL TYPE 属性を使用する。そのような属性はすべて、構造体エレメント、配列、および BASED など任意のストレージ・クラスで分配でき、使用できます。
- LIKE 属性を使用して宣言された変数をホスト変数として使用する。

SQL プリプロセッサーはソースをスキャンして EXEC SQL ステートメント、DECLARE ステートメント、および宣言のブロックを区切るステートメントを探すため、すべての PL/I ステートメントが構文的に正しくなければなりません。 ステートメントが正しくコーディングされていないと、プリプロセッサーは BEGIN、DO、PACKAGE、PROCEDURE、または SELECT ステートメントに対応する END ステートメントを探すときに誤動作し、一部のホスト変数参照を正しく解決することができなくなる可能性があります。このような間違ったコードを識別しやすくするために、SQL プリプロセッサーは以下のようなエラーにフラグを立てます。

- 対応する右括弧がない左括弧
- 末尾にセミコロンがない SELECT ステートメント
- THEN キーワードがない IF ステートメント
- 無効な記号で始まるステートメント

ソース・コードに MACRO ステートメントと SQL ステートメントがどちらも含まれている場合:

- SQL プリプロセッサーの INCONLY オプションを使用せずに SQL プリプロセッサーや MACRO プリプロセッサーを呼び出す場合は、MACRO プリプロセッサーを最初に呼び出す必要があります。その場合、実行可能 PL/I ステートメントが許可されるすべての場所で、EXEC SQL ステートメントが許可されます。
- SQL プリプロセッサーの INCONLY オプションを使用して SQL プリプロセッサーや MACRO プリプロセッサーを呼び出す場合は、MACRO プリプロセッサーの前に SQL プリプロセッサーを呼び出すことができます。その場合、EXEC SQL ステートメントは、その直前にセミコロンを指定しさえすれば使用できます (間に空白やコメントを含めてもかまいません)。

SQL プリプロセッサーは、 PL/I コンパイラーと同じ方法で DBCS をサポートします。 GRAPHIC PL/I コンパイラー・オプションが有効になっている場合、一部のソース言語エレメントは DBCS 文字でも SBCS 文字でも記述できます。特に、ソース・プログラムの以下の場所で DBCS 文字を使用できます。

- コメント内
- ステートメント・ラベルおよび ID の一部として
- G または M リテラルで

以下の制約事項は、SQL ステートメントをプログラミングおよびコンパイルする際に、 PL/I 組み込み関数、コンパイラー・オプション、およびステートメントを使用する場合に適用されます。

- EXEC SQL ステートメントの PL/I への変換時に、以下の組み込み関数が、生成されるコードに含まれる場合があります。 以下の組み込み関数のいずれかを構造体でエレメント名として使用する場合は、これらを BUILTIN として明示的に宣言することも必要です。
  - ADDR
  - LENGTH
  - MAXLENGTH
  - PTRVALUE
  - SYSNULL
- EXEC SQL ステートメントでは、BIND(: *t, p* :) などの PL/I タイプ付き関数は使用できません。
- プリプロセッサーを使用してコンパイルを行う場合、以下のコンパイラー・オプションは使用できません。
  - DFT(ASCII)
  - DFT(IEEE)
- SQL 照会では DECLARE STATEMENT ステートメントは使用しないでください。 PL/I プリプロセッサーは常にこれらのステートメントを無視します。

SQL プリプロセッサー・オプションを使用してコンパイルを行うと、オブジェクト・モジュールやリストなど通常の PL/I コンパイラー出力とともに、DB2 データベース要求モジュール (DBRM) が生成されます。DB2 バインド・プロセスへの入力になる DBRM データ・セットには、プログラム内の SQL ステートメントとホ

スト変数に関する情報が入っています。 ただし、バインドまたは実行時の処理の場合、DBRM の中のすべての情報が重要であるというわけではありません。 例えば、DBRM の中の HOST 値は、PL/I 以外の言語を指定するものであり、気にする理由は何もありません。 これは、HOST 値のインストール・デフォルトとして他の言語が選択されていることを意味するだけであって、このことがプログラムのバインド処理やランタイム処理に影響することはありません。

EMPTYDBRM オプションが有効で、ソースが以下のいずれかの条件を満たしている場合、ステートメントには変換が不要であったことを示すメッセージがプリプロセッサーから発行され、DBRM は作成されません。

- ソースに EXEC SQL ステートメントが含まれていない。
- ソースに EXEC SQL INCLUDE SQLCA および EXEC SQL INCLUDE SQLDA 以外の EXEC SQL INCLUDE ステートメントのみが含まれている。

PL/I コンパイラー・リストには、プリプロセッサーが生成したエラー診断情報 (SQL ステートメントの構文エラーなど) が含まれています。 EXEC SQL ステートメントのリストが、元のソースによく似た読みやすいフォーマットで表示されます。

プリプロセッサーを使用するには、以下の操作を行う必要があります。

- プログラムのコンパイル時に次のオプションを指定する。

  ```
  PP(SQL('options'))
  ```

  このコンパイラー・オプションは、組み込み SQL プリプロセッサーを起動するようにコンパイラーに指示します。 SQL キーワードの後に、SQL 処理オプションのリストを括弧で囲んで指定します。 これらのオプションはコンマまたはスペースで区切ることができ、オプションのリストは引用符で囲む必要があります (単一引用符か二重引用符を使用し、同じ種類の引用符で囲む必要があります)。

  例えば `PP(SQL('DATE(USA),TIME(USA)')` は、 DATE および TIME の両データ・タイプに対して USA フォーマットを使用するようにプリプロセッサーに指示します。

  また、LOB サポートを使用するには次のオプションを指定する必要があります。

  ```
  LIMITS( FIXEDBIN(31,63)  FIXEDDEC(15,31) )
  ```

  SQL プリプロセッサー・オプションを指定するには、PPSQL コンパイラー・オプションを使用する方法もあります。 この使用方法について詳しくは、 72 ページの『PPSQL』を参照してください。
- コンパイル・ステップ用の JCL に、次のデータ・セットに対する DD ステートメントを組み込む。
  - DB2 ロード・ライブラリー (*prefix*.SDSNLOAD)

    SQL プリプロセッサーは、SQL ステートメントの処理を行うために DB2 モジュールを呼び出します。 このため、DB2 ロード・ライブラリーのデータ・セット名を、コンパイル・ステップ用の STEPLIB 連結に組み込む必要があります。
  - SQL INCLUDE ステートメント用のライブラリー

ソース・プログラムへの 2 次入力を指定する SQL INCLUDE *member-name* ステートメントがプログラムにある場合は、*member-name* を含むデータ・セットの名前を、コンパイル・ステップ用の SYSLIB 連結に組み込む必要があります。

– DBRM ライブラリー

PL/I プログラムをコンパイルすると DB2 データベース要求モジュール (DBRM) が生成されるため、DBRM の書き込み先データ・セットを指定するために DBRMLIB DD ステートメントが必要です。

DBRMLIB DD ステートメントは、コンパイル中に複数回開いたり、閉じたりすることができるデータ・セットを指定する必要があります。

例えば、JCL には次のような行を指定します。

```
//STEPLIB  DD  DSN=DSNA10.SDSNLOAD,DISP=SHR
//SYSLIB   DD  DSN=PAYROLL.MONTHLY.INCLUDE,DISP=SHR
//DBRMLIB  DD  DSN=PAYROLL.MONTHLY.DBRMLIB.DATA(MASTER),DISP=SHR
```

SQL プリプロセッサー・オプション

このセクションでは、SQL プリプロセッサーがサポートするオプションについて説明します。

SQL プリプロセッサーには 2 つのグループのオプション (PL/I SQL プリプロセッサーによって処理されるオプションと、DB2 コプロセッサーによって処理されるオプション) を渡すことができます。これらのオプションは、PP(SQL('*option-list*')) オプションのオプション・ストリングで指定する必要があります。これらのオプションは、オプション・ストリング内で混在させることができます。

SQL プリプロセッサー・オプションを指定するときは、オプションのリストを 1 対の引用符で囲む必要があります。例えば、CCSID0 オプションを指定する場合は、PP(SQL('CCSID0')) のように指定する必要があります。

表 8 は、PL/I SQL プリプロセッサー・オプション、その省略形 (存在する場合)、および IBM 提供のデフォルト値をリストしたものです。この表では、相互排他的なオプションは縦棒 (|) によって分離されています。また括弧 ([ ]) は、囲まれたオプションが省略可能であることを示しています。

DB2 コプロセッサー・オプションについて詳しくは、「*DB2 for z/OS* アプリケーション・プログラミングおよび *SQL* 解説書」を参照してください。

表 *8. SQL* プリプロセッサー・オプション、および *IBM* 提供のデフォルト値

SQL プリプロセッサー・オプション	省略名	**z/OS** デフォルト
CCSID0 | NOCCSID0	-	CCSID0
CODEPAGE | NOCODEPAGE	-	NOCODEPAGE
DEPRECATE(STMT([EXPLAIN | GRANT | REVOKE | SET_CURRENT_SQLID]))	-	DEPRECATE(STMT())
EMPTYDBRM | NOENTRYDBRM	-	NOENTRYDBRM
HOSTCOPY | NOHOSTCOPY	-	HOSTCOPY

表 8. SQL プリプロセッサー・オプション、および IBM 提供のデフォルト値 (続き)

SQL プリプロセッサー・オプション	省略名	z/OS デフォルト
INCONLY | NOINCONLY	-	NOINCONLY
WARNDECP | NOWARNDECP	-	NOWARNDECP

CCSID0

CCSID0 オプションは、PL/I SQL プリプロセッサーによって、WIDECHAR 以外のホスト変数に CCSID 値が割り当てられないことを指定します。

NOCCSID0 オプションは、PL/I SQL プリプロセッサーによってホスト変数に CCSID 値を割り当てることを許可します。

ご使用のプログラムが FOR BIT DATA 列を、BIT データではないデータ・タイプで更新する場合は、CCSID0 を選択します。 CCSID0 は、ホスト変数が CCSID に関連付けられていないことを DB2 に通知して、割り当てを行えるようにします。そうでなければ、BIT データではない CCSID に関連したホスト変数が FOR BIT DATA 列に割り当てられ、DB2 にエラーが発生します。

WIDECHAR には、1200 という CCSID 値が常に割り当てられています。

DB2 プリコンパイラーを使用した古い PL/I プログラムとの互換性を維持するには、CCSID0 を有効にします。

CCSID0 と NOCCSID0 は相互排他的なオプションです。

デフォルトは CCSID0 です。

CODEPAGE

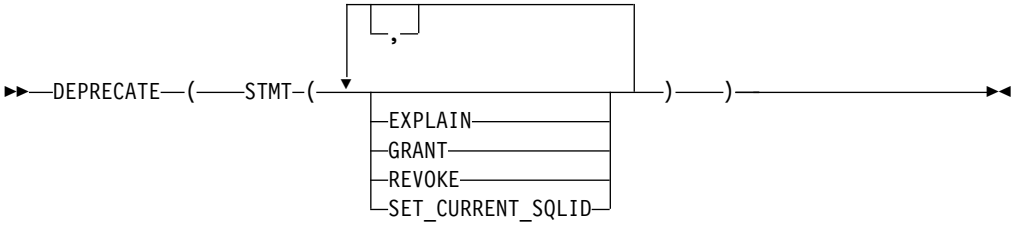
CODEPAGE オプションが有効になっている場合、コンパイラー・オプション CODEPAGE は必ず文字タイプの SQL ホスト変数の CCSID として使用されます。

NOCODEPAGE オプションが有効になっているときは、SQL プリプロセッサー・オプション NOCCSID0 も有効になっている場合に限り、コンパイラー・オプション CODEPAGE が文字タイプの SQL ホスト変数の CCSID として使用されます。

デフォルトは NOCODEPAGE です。

DEPRECATE

DEPRECATE オプションは、指定されたステートメントにプリプロセッサーによって非推奨のフラグが立てられるように指示します。



STMT
プリプロセッサーによって非推奨のフラグが立てられるステートメントのリストを指定します。 このリストは空でもかまいません。

EXPLAIN
EXPLAIN SQL ステートメント。

GRANT
GRANT SQL ステートメント。

REVOKE
REVOKE SQL ステートメント。

SET_CURRENT_SQLID
SET CURRENT SQLID SQL ステートメント。

デフォルトは DEPRECATE(STMT()) です。

EMPTYDBRM

EMPTYDBRM オプションは、EXEC SQL ステートメントがコードに含まれていない場合でも、SQL プリプロセッサーが常に DBRM を作成することを指定します。ただし、INCONLY オプションが呼び出された場合、SQL プリプロセッサーは DBRM を作成しません。

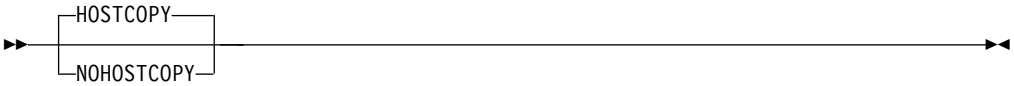
NOENTRYDBRM オプションは、SQL プリプロセッサーが DBRM を作成しないことを指定します。

デフォルトは NOENTRYDBRM です。

HOSTCOPY

HOSTCOPY オプションは、SQL プリプロセッサーが LP(64) での実行時に、各 EXEC SQL ステートメントの前後で、2 GB 境界より下のストレージとの間でホスト変数をコピーするコードを生成するかどうかを決定します。

NOHOSTCOPY が指定された場合、SQL プリプロセッサーはこのコードを生成しません。ただし、すべてのホスト変数が 2 GB 境界より下のストレージにあることを (例えば、ホスト変数を、ALLOC31 組み込み関数によって取得されるベース・ポインターを持つ BASED 変数にすることによって) 確認することはユーザーの責任です。



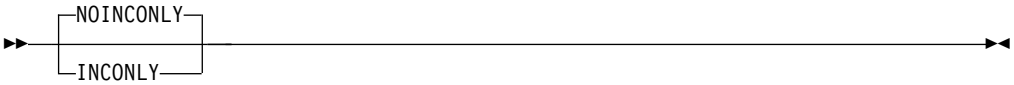
デフォルトは HOSTCOPY です。

LP(32) では HOSTCOPY オプションは無視されます。

INCONLY

INCONLY オプションは、SQL プリプロセッサーが EXEC SQL INCLUDE ステートメントのみを処理するように指定します。ただし、SQL 連絡域 (SQLCA) および SQL 記述子域 (SQLDA) の includes を除きます。このオプションが有効になっている場合は、SQL プリプロセッサーによってコードは生成されません。

NOINCONLY オプションは、SQL プリプロセッサーが EXEC SQL INCLUDE ステートメントだけではなく、すべてのステートメントを処理することを指定します。



INCONLY オプションが指定されている場合、コンパイラーは SQL オプション・リストを生成しません。 INCONLY では他のすべてのオプションが無視されるためです。

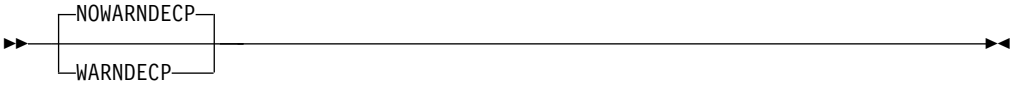
INCONLY オプションおよび NOINCONLY オプションは同時に指定できません。

また、互換性のため、NOINCONLY がデフォルトです。

WARNDECP

WARNDECP オプションが有効になっているときに、コンパイルで DB2 に付属の DSNHDECP モジュールが使用されると、プリプロセッサーは警告メッセージを発行します。

NOWARNDECP オプションが有効になっているときは、警告メッセージは発行されません。



デフォルトは NOWARNDECP です。

SQL プロセッサー・オプションに関する PL/I 固有の注意事項

このトピックでは、SQL プロセッサー・オプション FLOAT、ONEPASS、および STDSQL を指定するときに従う必要のある規則についていくつか説明します。

PL/I コンパイラーに対して以下の SQL プロセッサー・オプションを指定するときは、以下の規則が適用されます。

FLOAT
FLOAT オプションが PL/I の DEFAULT(HEXADEC|IEEE) オプションと異なる場合は、エラー・メッセージが発行されます。

ONEPASS | TWOPASS
オプション ONEPASS を指定するときは、すべてのホスト変数を SQL ステートメントでの使用前に宣言する必要があります。

STDSQL
オプション STDSQL(YES) を指定する場合は、SQL BEGIN DECLARE SECTION ステートメントと SQL END DECLARE SECTION ステートメントの間ですべてのホスト変数を宣言する必要があります。

PL/I アプリケーション内での SQL ステートメントのコーディング

「*DB2 for z/OS SQL* 解説書」 で定義されている言語を使用すれば、PL/I アプリケーションにおいて SQL ステートメントをコーディングできます。このセクションでは、SQL コードの特定要件について説明します。

SQL 連絡域の定義

SQL ステートメントを含む PL/I プログラムには、SQLCODE 変数 (STDSQL(86) プリプロセッサー・オプションを使用する場合)、または SQL 連絡域 (SQLCA) を組み込む必要があります。

146 ページの図 6 に示すように、SQLCA の一部は SQLCODE 変数と SQLSTATE 変数です。

- SQLCODE の値は、各 SQL ステートメントの実行後にデータベース・サービスによって設定されます。 アプリケーションは、SQLCODE 値を検査して、最後の SQL ステートメントが正常に実行されたかどうかを判別できます。
- SQLSTATE 変数は、コンパイラーが SQL ステートメントの結果を分析するときに SQLCODE 変数の代替として使用できます。SQLCODE 変数と同様に、SQLSTATE 変数は各 SQL ステートメントの実行後にデータベース・サービスによって設定されます。

SQLCA 宣言を組み込むには、EXEC SQL INCLUDE ステートメントを使用します。

```
exec sql include sqlca;
```

SQLCA 構造体は、SQL 宣言セクション内で定義してはなりません。 SQLCODE 宣言と SQLSTATE 宣言の有効範囲には、プログラム内のすべての SQL ステートメントの有効範囲が含まれていなければなりません。

```
Dcl
  1 Sqlca,
    2 sqlcaid      char(8),           /* Eyecatcher = 'SQLCA   '   */
    2 sqlcabc      fixed binary(31),  /* SQLCA size in bytes = 136 */
    2 sqlcode      fixed binary(31),  /* SQL return code           */
    2 sqlerrmc     char(70) var,      /* Error message tokens      */
    2 sqlerrp      char(8),           /* Diagnostic information    */
    2 sqlerrd(0:5) fixed binary(31),  /* Diagnostic information    */
    2 sqlwarn,                        /* Warning flags             */
      3 sqlwarn0   char(1),
      3 sqlwarn1   char(1),
      3 sqlwarn2   char(1),
      3 sqlwarn3   char(1),
      3 sqlwarn4   char(1),
      3 sqlwarn5   char(1),
      3 sqlwarn6   char(1),
      3 sqlwarn7   char(1),
    2 sqlext,
      3 sqlwarn8   char(1),
      3 sqlwarn9   char(1),
      3 sqlwarna   char(1),
      3 sqlstate   char(5);       /* State corresponding to SQLCODE */
```

図 *6. SQLCA の PL/I 宣言*

SQL 記述子域の定義

SQLCA とは異なり、1 つのプログラム内に複数の SQL 記述子域 (SQLDA) を指定したり、SQLDA に、任意の有効な名前を付与したりできます。

以下のステートメントには SQLDA が必要です。

PREPARE *statement-name* INTO *descriptor-name* FROM *host-variable*
EXECUTE...USING DESCRIPTOR *descriptor-name*
FETCH...USING DESCRIPTOR *descriptor-name*
OPEN...USING DESCRIPTOR *descriptor-name*
DESCRIBE *statement-name* INTO *descriptor-name*

SQLDA を組み込むには、EXEC SQL INCLUDE ステートメントを使用します。

```
exec sql include sqlda;
```

SQLDA は SQL 宣言セクション内で定義してはなりません。

```
Dcl
  1 Sqlda based(Sqldaptr),
    2 sqldaid     char(8),          /* Eye catcher = 'SQLDA   ' */
    2 sqldabc     fixed binary(31), /* SQLDA size in bytes=16+44*SQLN*/
    2 sqln        fixed binary(15), /* Number of SQLVAR elements*/
    2 sqld        fixed binary(15), /* # of used SQLVAR elements*/
    2 sqlvar(Sqlsize refer(sqln)),  /* Variable Description     */
      3 sqltype   fixed binary(15), /* Variable data type       */
      3 sqllen    fixed binary(15), /* Variable data length     */
      3 sqldata   pointer,          /* Pointer to variable data value*/
      3 sqlind    pointer,          /* Pointer to Null indicator*/
      3 sqlname   char(30) var ;    /* Variable Name            */

Dcl
  1 Sqlda2 based(Sqldaptr),
    2 sqldaid2     char(8),          /* Eye catcher = 'SQLDA   ' */
    2 sqldabc2     fixed binary(31), /* SQLDA size in bytes=16+44*SQLN*/
    2 sqln2        fixed binary(15), /* Number of SQLVAR elements*/
    2 sqld2        fixed binary(15), /* # of used SQLVAR elements*/
    2 sqlvar2(Sqlsize refer(sqln2)), /* Variable Description     */
      3 sqlbiglen,
        4 sqllongl fixed binary(31),
        4 sqlrsvdl fixed binary(31),
      3 sqldatal   pointer,
      3 sqltname   char(30) var;

dcl Sqlsize    fixed binary(15);     /* number of sqlvars (sqln) */
dcl Sqldaptr   pointer;
dcl Sqltripled char(1) value('3');
dcl Sqldoubled char(1) value('2');
dcl Sqlsingled char(1) value(' ');
```

図 *7. SQL* 記述子域の *PL/I* 宣言

SQL ステートメントの組み込み

プログラムの最初のステートメントは、PROCEDURE または PACKAGE ステートメントでなければなりません。実行可能ステートメントを使用できる任意の場所で、任意の SQL ステートメントをプログラムに追加できます。

また、以下の SQL ステートメントを PACKAGE 内、および任意のプロシージャーの外部に追加できます。

- EXEC SQL BEGIN DECLARE SECTION
- EXEC SQL END DECLARE SECTION
- EXEC SQL DECLARE (ステートメントに対して実行可能コードを生成する必要がない場合)
- EXEC SQL INCLUDE

それぞれの SQL ステートメントは EXEC (または EXECUTE) SQL で始まり、セミコロン (;) で終わる必要があります。

例えば、UPDATE ステートメントは次のようにコーディングされます。

```
exec sql update DSN8A10.DEPT
set   Mgrno  = :Mgr_Num
where Deptno = :Int_Dept;
```

コメント:

SQL ステートメントのほかに、ブランクを入力できる場所では組み込み SQL ステートメントにコメントを組み込むことができます。

コメントが SQL ステートメント内にある場合は、コメントを閉じるスラッシュ (/) がリスト内で「より大」記号 (>) として示されます。コンパイラーが特定の SQL ステートメントをソース・リストにどのように表示するかを以下に例示します。

次のサンプル SQL ステートメントにはコメントが含まれています。

```
exec sql insert into table /* some text */ values(:data);
```

コンパイラーは、このステートメントをソース・リストにおいて次のように表示します。

```
/*$*$*$
 exec sql insert into table /* some text *> values(:data)
 $*$*$*/
```

SQL ステートメントに組み込まれると、 SQL スタイルのコメント ('--') がサポートされます。

SQL ステートメントの継続:

SQL ステートメントの行継続規則は、他の PL/I ステートメントと同じです。

コードの組み込み: SQL ステートメントまたは PL/I ホスト変数宣言ステートメントは、ソース・コードに次の SQL ステートメントを入れることで組み込むことができます。これを、ステートメントを組み込む場所に置いてください。

```
exec sql include member;
```

マージン: SQL ステートメントは、列 *m* から *n* までででコーディングする必要があります。 *m* と *n* は、MARGINS(*m*,*n*) コンパイラー・オプションで指定されます。

名前: ホスト変数には、任意の有効な PL/I 変数名を使用できます。ホスト変数名の長さは、LIMITS(NAME(*n*)) コンパイラー・オプションで指定された値 *n* を超えてはなりません。

ステートメント・ラベル: END DECLARE SECTION ステートメント、および INCLUDE text-file-name ステートメントは例外ですが、実行可能 SQL ステートメントには PL/I ステートメントと同様にラベル接頭部を付けることができます。

WHENEVER ステートメント: SQL WHENEVER ステートメントの GOTO 文節のターゲットは、 PL/I ソース・コード内のラベルでなければならず、 WHENEVER ステートメントによって影響を受ける任意の SQL ステートメントのスコープ内に存在する必要があります。

ホスト変数の使用

SQL ステートメントで使用されるホスト変数はすべて、明示的に宣言する必要があり、 SQL ステートメント内では、すべてのホスト変数の前にコロン (:) を付ける必要があります。

ホスト変数参照では添え字は使用できません。

以下のトピックでは、ホスト変数の使用方法について詳しく説明します。

- 『配列をホスト変数として使用』
- 『ホスト変数の宣言』
- 150 ページの『スカラー・ホスト変数の宣言』
- 152 ページの『SQL および PL/I の同等なデータ・タイプの判別』
- 155 ページの『SQL データ・タイプと PL/I データ・タイプの互換性の判別』

配列をホスト変数として使用**:** 次の 2 つの方法でのみ、配列をホスト変数として使用できます。

- ホスト構造の標識変数の配列として
- 次のいずれかのステートメントで使用される場合、ホスト変数の配列として
  - 複数行のフェッチに対する FETCH ステートメント
  - 複数行の挿入での INSERT ステートメント
  - 複数行の MERGE ステートメント

これらの配列は、すべて 1 次元でなければなりません。また、CONNECTED 属性を有し、定数境界を有している必要があります。

配列をホスト変数として使用するこれ以外の方法は、すべて無効です。

ホスト変数の宣言**:**

ホスト変数の宣言は、通常の PL/I 変数宣言と同じ場所で行うことができます。

有効な PL/I 宣言のサブセットだけが、有効なホスト変数宣言として認識されます。

SQL プリプロセッサーは、DEFINE ALIAS ステートメント、DEFINE ORDINAL ステートメント、および DEFINE STRUCTURE ステートメントを構文解析します。つまり、SQL ステートメントで使用できる PL/I タイプが DEFINE ALIAS ステートメントで定義される場合、そのタイプで宣言された変数も SQL ステートメントで使用できます。

プリプロセッサーは、PL/I DEFAULT ステートメントに指定されたデータ属性デフォルトを使用しません。 変数の宣言が認識されない場合は、ステートメントがその変数を参照すると、次のようなメッセージが出されることがあります。

```
'The host variable token ID is not valid'
```

LIKE を使用して宣言された構造体、またはこの構造体の 1 つのエレメントを、ホスト変数として使用する場合は、LIKE オブジェクトの宣言が SQL プリプロセッサーによって認識されなければなりません。 例えば、インクルードされていない %INCLUDE ファイルに LIKE オブジェクトが含まれていてはなりません。

ホスト変数宣言内で、制限付きの式を使用すれば、配列の境界や、ストリングの長さを定義できます。ただし、この式には、以下のいずれかの形式が必要です。

- 式に適用された接頭演算子 (この式が整数に評価できる場合)

- 2 つの式に適用された加算演算子または減算演算子 (いずれの式も整数に評価できる場合)
- 2 つの式に適用された乗算演算子 (いずれの式も整数に評価できる場合)
- 名前付き定数に対する参照 (この参照が整数に評価できる場合)
- 組み込み関数
  INDICATORS、HBOUND、HBOUNDACROSS、LENGTH、MAXLENGTH のいずれか
- 整数の数値

名前付き定数を使用してホスト変数の境界や長さを定義できますが、名前付き定数自体をホスト変数として使用することはできません (ただし、以下の両方の条件が適用される場合を除く)。

- DB2 が EXEC SQL ステートメントにおける対象の位置で、名前の付いていない単純な定数を許可している。
- 名前付き定数に以下のいずれかの属性がある。
  - CHARACTER: この場合、名前付き定数の VALUE 属性は文字ストリングを指定しなければなりません。
  - FIXED: この場合、名前付き定数の VALUE 属性は 10 進数を指定するか、または上述と同じ制限を使用して整定数に変換できる式を指定しなければなりません。

変数の名前とデータ属性だけがプリプロセッサーによって使用され、位置合わせ、スコープ、およびストレージの属性は無視されます。

スカラー・ホスト変数の宣言**:** 以下のデータ属性のいずれかを使用して、スカラー・ホスト変数を宣言する必要があります。

CHARACTER、**GRAPHIC**、または **WIDECHAR**
CHARACTER、GRAPHIC、または WIDECHAR 属性を使用して宣言されるホスト変数は、ストリング・ホスト変数と呼ばれます。 ストリング・ホスト変数には以下の制限が適用されます。

- NONVARYING 属性または VARYING 属性のどちらかが必要である。
- VARYING 属性を持っている場合は、NATIVE 属性が必要である。

FIXED BINARY、**FIXED DECIMAL**、または **FLOAT**
FIXED および BINARY、FIXED および DECIMAL、または FLOAT 属性を使用して宣言されたホスト変数は、数値ホスト変数と呼ばれます。数値ホスト変数には以下の制限が適用されます。

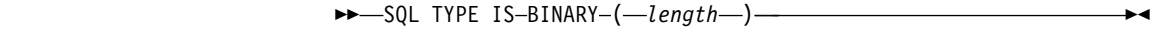
- REAL 属性が必要である。
- FIXED 属性および BINARY 属性がある場合は、SIGNED 属性、NATIVE 属性、ゼロ・スケール係数、および 7 を超える精度が必要である。
- FIXED 属性および DECIMAL 属性がある場合は、精度よりも小さく負数ではないスケール係数が必要である。
- FLOAT 属性および DECIMAL 属性がある場合は、FLOAT(DFP) オプションが有効になっていない限り、17 未満の精度が必要である。

- FLOAT 属性および BINARY 属性がある場合は、54 未満の精度が必要である。

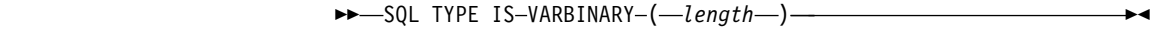
SQL TYPE

SQL TYPE 属性を使用して宣言されるホスト変数は、 SQL TYPE ホスト変数と呼ばれます。この属性の仕様は、以下の構文図のいずれかに準拠している必要があります。

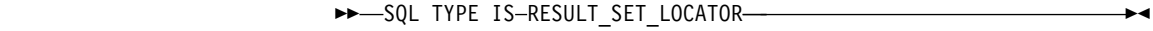
BINARY



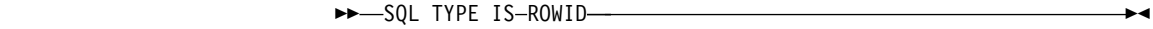
VARBINARY



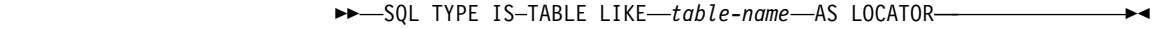
結果セット・ロケーター



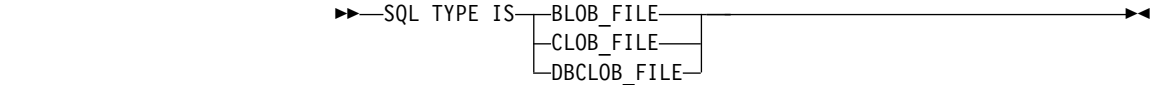
ROWID



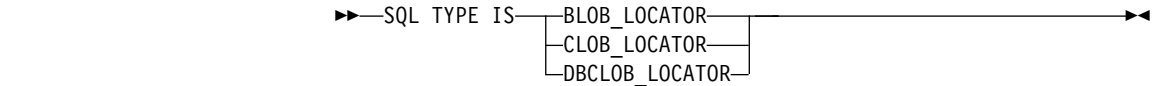
テーブル・ロケーター



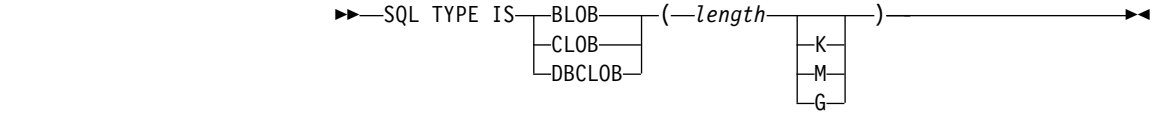
LOB ファイル参照



LOB ロケーター



LOB 変数

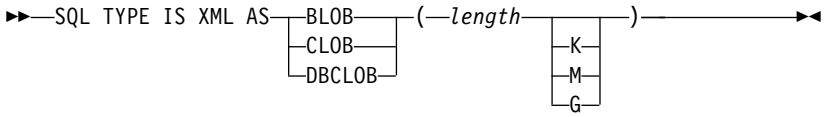


BLOB

BLOB の代わりに **BINARY LARGE OBJECT** も使用できます。

CLOB
CLOB の代わりに **CHARACTER LARGE OBJECT** や **CHAR LARGE OBJECT** も使用できます。

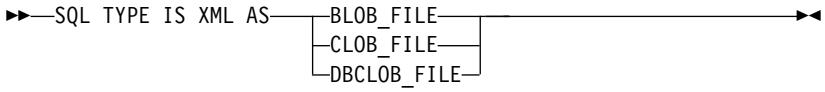
XML LOB 変数



BLOB
BLOB の代わりに **BINARY LARGE OBJECT** も使用できます。

CLOB
CLOB の代わりに **CHARACTER LARGE OBJECT** や **CHAR LARGE OBJECT** も使用できます。

XML ファイル参照



以下の定数宣言が SQL プリプロセッサーによって生成されます。 これらの宣言を使用すると、ファイル参照ホスト変数を使用する際にファイル・オプション変数を設定できます。

```
DCL SQL_FILE_READ      FIXED BIN(31) VALUE(2);
DCL SQL_FILE_CREATE    FIXED BIN(31) VALUE(8);
DCL SQL_FILE_OVERWRITE FIXED BIN(31) VALUE(16);
DCL SQL_FILE_APPEND    FIXED BIN(31) VALUE(32);
```

SQL および **PL/I** の同等なデータ・タイプの判別**:** ホスト変数の基本 SQLTYPE および SQLLEN は、 表 9 および 153 ページの表 10 に従って決定されます。 ホスト変数が標識変数とともに指定される場合、 SQLTYPE は基本 SQLTYPE に 1 を加えた値です。

所定の SQL データ・タイプと同等の PL/I データ・タイプを判別するには、154 ページの表 11 および 155 ページの表 12 を使用できます。

表 *9. PL/I* 宣言から生成される *SQL* データ・タイプ

PL/Iデータ・タイプ **(data type)**	ホスト変数の **SQLTYPE**	ホスト変数の **SQLLEN**	**SQL** データ・タイプ
BIN FIXED(p)、7 < p <= 15	500	2	SMALLINT
BIN FIXED(p)、15 < p <= 31	496	4	INTEGER
BIN FIXED(p)、31 < p <= 63	492	8	BIGINT
DEC FIXED(p,s)、0<=p<=15、および 0<=s<=p	484	p (byte 1) s (byte 2)	DECIMAL(p,s)
BIN FLOAT(p)、1 ≤ p ≤ 21	480	4	REAL または FLOAT(n) 1<=n<=21

表 9. *PL/I* 宣言から生成される *SQL* データ・タイプ (続き)

PL/Iデータ・タイプ **(data type)**	ホスト変数の **SQLTYPE**	ホスト変数の **SQLLEN**	**SQL** データ・タイプ
BIN FLOAT(p)、22 ≤ p ≤ 53	480	8	DOUBLE PRECISION または FLOAT(n)、22<=n<=53
FLOAT(NODFP) を指定した場合:			
DEC FLOAT(p), 1 ≤ p ≤ 6	480	4	FLOAT (単精度)
DEC FLOAT(p), 7 ≤ p ≤ 16	480	8	FLOAT (倍精度)
FLOAT(DFP) を指定した場合:			
DEC FLOAT(p), 1 ≤ p ≤ 7	996	4	DECFLOAT (単精度)
DEC FLOAT(p), 7 ≤ p ≤ 16	996	8	DECFLOAT (倍精度)
DEC FLOAT(p)、16 ≤ p ≤ 34	996	16	DECFLOAT (拡張 10 進数)
CHAR(n)	452	n	CHAR(n)
CHAR(n) VARYING	448	n	VARCHAR(n)
GRAPHIC(n)、1 ≤ n ≤ 127	468	n	GRAPHIC(n)
GRAPHIC(n) VARYING	464	n	VARGRAPHIC(n)

表 10. *SQL TYPE* 宣言から生成される *SQL* データ・タイプ

PL/Iデータ・タイプ **(data type)**	ホスト変数の **SQLTYPE**	ホスト変数の **SQLLEN**	**SQL** データ・タイプ
SQL TYPE IS BLOB(*n*) 1<*n*<2147483647	404	*n*	BLOB(*n*)
SQL TYPE IS CLOB(*n*) 1<*n*<2147483647	408	*n*	CLOB(*n*)
SQL TYPE IS DBCLOB(*n*) 1<*n*<1073741823 (2)	412	*n*	DBCLOB(*n*) (2)
SQL TYPE IS ROWID	904	40	ROWID
SQL TYPE IS VARBINARY(*n*) 1<*n*<32704	908	*n*	VARBINARY(*n*)
SQL TYPE IS BINARY(*n*) 1<*n*<255	912	*n*	BINARY(*n*)
SQL TYPE IS BLOB_FILE	916	267	BLOB ファイル参照 (1)
SQL TYPE IS CLOB_FILE	920	267	CLOB ファイル参照 (1)
SQL TYPE IS DBCLOB_FILE	924	267	DBCLOB ファイル参照 (1)
SQL TYPE IS BLOB_LOCATOR	960	4	BLOB ロケーター (1)
SQL TYPE IS CLOB_LOCATOR	964	4	CLOB ロケーター (1)
SQL TYPE IS DBCLOB_LOCATOR	968	4	DBCLOB ロケーター (1)
SQL TYPE IS RESULT_SET_LOCATOR	972	4	結果セット・ロケーター

表 10. *SQL TYPE* 宣言から生成される *SQL* データ・タイプ (続き)

PL/Iデータ・タイプ **(data type)**	ホスト変数の **SQLTYPE**	ホスト変数の **SQLLEN**	**SQL** データ・タイプ
SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	976	4	テーブル・ロケーター (1)

注:
1. このデータ・タイプを列タイプとして使用しないでください。
2. *n* は 2 バイト文字の数です。

表 11. *SQL* データ・タイプと *PL/I* 宣言の対応

SQL データ・タイプ	**PL/I** の同等のもの	注
SMALLINT	BIN FIXED(15)	
INTEGER	BIN FIXED(31)	
BIGINT	BIN FIXED(63)	
DECIMAL(p,s)	DEC FIXED(p) または DEC FIXED(p,s)	p = precision および s = scale; 1 ≤ p ≤ 31 および 0 ≤ s ≤ p
FLOAT(NODFP) を指定した場合:		
REAL または FLOAT(n)	BIN FLOAT(p) または DEC FLOAT(m)	1 ≤ p ≤ 21 および 1 ≤ m ≤ 6
DOUBLE PRECISION、DOUBLE、または FLOAT(n)	BIN FLOAT(p) または DEC FLOAT(m)	22 ≤ p ≤ 53 および 7 ≤ m ≤ 16
FLOAT(DFP) を指定した場合:		
DECFLOAT	DEC FLOAT(m)	短精度 10 進浮動小数 1 ≤ m ≤ 7
DECFLOAT	DEC FLOAT(m)	長精度 10 進浮動小数 7 ≤ m ≤ 16
DECFLOAT	DEC FLOAT(m)	拡張 10 進数浮動小数 16 ≤ m ≤ 34
CHAR(n)	CHAR(n)	1 ≤ n ≤ 32767
VARCHAR(n)	CHAR(n) VAR	
GRAPHIC(n)	GRAPHIC(n)	*n* は、2 バイト文字の数を示す (バイト数ではない) 正整数。1 ≤ n ≤ 16383
VARGRAPHIC(n)	GRAPHIC(n) VAR	*n* は、2 バイト文字の数を示す (バイト数ではない) 正整数。1 ≤ n ≤ 16383
DATE	CHAR(n)	*n* の最小値は 10。
TIME	CHAR(n)	*n* の最小値は 8。
TIMESTAMP	CHAR(n)	*n* の最小値は 26。

表 12. *SQL* データ・タイプと *SQL TYPE* 宣言の対応

SQL データ・タイプ	**PL/I** の同等のもの	注
結果セット・ロケーター	SQL TYPE IS RESULT_SET_LOCATOR	このデータ・タイプは、結果セットの受け取りにだけ使用します。 このデータ・タイプを列タイプとして使用しないでください。
テーブル・ロケーター	SQL TYPE IS TABLE LIKE *table-name* AS LOCATOR	このデータ・タイプは、ユーザー定義の関数またはストアード・プロシージャー内だけで、変位テーブルの行を受け取るために使用します。 このデータ・タイプを列タイプとして使用しないでください。
BLOB ロケーター	SQL TYPE IS BLOB_LOCATOR	このデータ・タイプは、BLOB 列のデータを操作するためにだけ使用します。 このデータ・タイプを列タイプとして使用しないでください。
CLOB ロケーター	SQL TYPE IS CLOB_LOCATOR	このデータ・タイプは、CLOB 列のデータを操作するためにだけ使用します。 このデータ・タイプを列タイプとして使用しないでください。
DBCLOB ロケーター	SQL TYPE IS DBCLOB_LOCATOR	このデータ・タイプは、DBCLOB 列のデータを操作するためにだけ使用します。 このデータ・タイプを列タイプとして使用しないでください。
BLOB ファイル参照	SQL TYPE IS BLOB_FILE	このデータ・タイプは、BLOB ファイルの参照としてのみ使用します。 このデータ・タイプを列タイプとして使用しないでください。
CLOB ファイル参照	SQL TYPE IS CLOB_FILE	このデータ・タイプは、CLOB ファイルの参照としてのみ使用します。 このデータ・タイプを列タイプとして使用しないでください。
DBCLOB ファイル参照	SQL TYPE IS DBCLOB_FILE	このデータ・タイプは、DBCLOB ファイルの参照としてのみ使用します。 このデータ・タイプを列タイプとして使用しないでください。
BLOB(*n*)	SQL TYPE IS BLOB(*n*)	1<*n*<2147483647
CLOB(*n*)	SQL TYPE IS CLOB(*n*)	1<*n*<2147483647
DBCLOB(*n*)	SQL TYPE IS DBCLOB(*n*)	*n* は 2 バイト文字の数です。 1<*n*<1073741823
ROWID	SQL TYPE IS ROWID	
XML AS	SQL TYPE IS XML AS ...	XML バージョンの BLOB、CLOB、DBCLOB、BLOB_FILE、CLOB_FILE、または DBCLOB_FILE を記述するのに使用します。

SQL データ・タイプと **PL/I** データ・タイプの互換性の判別**:** SQL ステートメント内の PL/I ホスト変数は、その変数が使用される列と互換性があるタイプでなければなりません。

- 数値データ・タイプは、相互に互換性があります。 SMALLINT 列、INTEGER 列、DECIMAL 列、または FLOAT 列は、BIN FIXED(15)、BIN FIXED(31)、DECIMAL(*p,s*)、BIN FLOAT(*n*) (*n* は 22 から 53 までの範囲にある数値)、または DEC FLOAT(*m*) (*m* は 7 から 16 までの範囲にある数値) の PL/I ホスト変数と互換性があります。
- 文字データ・タイプは、相互に互換性があります。 CHAR または VARCHAR の列は、固定長または可変長の PL/I 文字ホスト変数と互換性があります。

- Datetime データ・タイプは、文字ホスト変数と互換性があります。
  DATE、TIME、または TIMESTAMP の列は、固定長または可変長の PL/I 文字ホスト変数と互換性があります。

  必要に応じて、データベース・マネージャーは自動的に固定長文字ストリングを可変長ストリングに変換したり、可変長ストリングを固定長文字ストリングに変換したりします。

ホスト構造体の使用

構造体または共用体でないメンバーを持つ構造体の名前を、PL/I ホスト構造体の名前にすることができます。

次の例で、B はスカラー C1 と C2 からなるホスト構造体の名前です。

```
dcl 1 A,
      2 B,
        3 C1 char(...),
        3 C2 char(...);
```

ホスト構造体は 2 レベルに制限されます。 ホスト構造体は、ホスト変数の名前付き集合と考えることができます。

ホスト構造体のリーフ要素にはそれぞれ、以下の有効なホスト・データ属性のいずれかが必要です (149 ページの『ホスト変数の宣言』での説明を参照)。

- CHARACTER、GRAPHIC、または WIDECHAR
- FIXED BINARY、FIXED DECIMAL、または FLOAT
- SQL TYPE

標識変数の使用

標識変数は、2 バイトの整数 (BIN FIXED(15)) であるか、2 バイトの整数の配列であるか、または 2 バイトの整数 (もしくはその配列) のみを含む構造体です。

検索時には、関連したホスト変数にヌル値が割り当てられているかどうかを示すために、標識変数が使用されます。 列への割り当て時には、ヌル値を割り当てる必要があるかどうかを示すために、負の標識変数が使用されます。

構造体は、関連ホスト変数も構造体である場合に限り、標識変数として使用できます。

標識変数はホスト変数と同じ方法で宣言され、両変数の宣言はプログラマーの裁量でどのように組み合わせることもできます。

SQL プリプロセッサーでは、標識配列の下限が 1 である必要はありません。

標識変数には、属性 REAL NATIVE SIGNED FIXED BIN(15) が必要です。

157 ページの図 8 に示されているステートメントの変数を宣言する方法を以下に例示します。

```
exec sql fetch Cls_Cursor into :Cls_Cd,
                               :Day :Day_Ind,
                               :Bgn :Bgn_Ind,
                               :End :End_Ind;
```

図 *8.* 標識変数を含む *SQL* ステートメント

変数は以下のように宣言できます。

```
exec sql begin declare section;
dcl Cls_Cd    char(7);
dcl Day       bin fixed(15);
dcl Bgn       char(8);
dcl End       char(8);
dcl (Day_Ind, Bgn_Ind, End_Ind)   bin fixed(15);
exec sql end declare section;
```

ホスト構造体の例

以下の例では、ホスト構造体と標識配列の宣言が示されていて、その後に、ホスト構造体にデータを取り込むために使用できる SQL ステートメントが示されています。

```
dcl 1 games,
      5 sunday,
        10 opponents char(30),
        10 gtime     char(10),
        10 tv        char(6),
        10 comments  char(120) var;
dcl indicator(4) fixed bin (15);


exec sql
     fetch cursor_a
     into :games.sunday:indicator;
```

LOB データの操作

LOBS、CLOBS、および BLOBS は、最大で 2,147,483,647 バイト (2 ギガバイト) の長さにすることができます。 2 バイト CLOBS は、1,073,741,823 文字 (1 ギガバイト) の長さにすることができます。 DB2 表にあるラージ・オブジェクト (LOB) データを使用するには、データをデータベース内に置いたまま、LOB ロケーターおよび LOB ファイル参照などの技法を使用してデータを操作します。

すべての LOB データを保持するためのホスト変数を宣言する方法は非効率または非現実的です。なぜならば、ご使用のプログラムが多くのストレージを割り振らなければならなくなるし、DB2 が多くのデータを移動しなければならなくなるためです。 そのため、LOB データを扱うには以下の手法を使用することをお勧めします。

- LOB ロケーター

  LOB ロケーターを使用すれば、LOB データをホスト変数に移さずに操作できます。 LOB ロケーターを使用することにより、プログラムに必要となるメモリーの量が大幅に少なくなります。

- LOB ファイル参照

  LOB ファイル参照変数を使用すれば、DB2 システム外の外部ファイルと、LOB 列との間でデータをインポートしたりエクスポートしたりできます。

大きなデータ断片の移動を最小限にするための手法について詳しくは、「*DB2 for z/OS アプリケーション・プログラミングおよび SQL 解説書*」、または IBM Redbooks® 資料「*LOBs with DB2 for z/OS: Stronger and Faster*」を参照してください。

PL/I および DB2 環境で CLOB を扱う方法の例については、pliclob サンプル・プログラムを参照してください。

LOB ロケーター

LOB ロケーターを使用すれば、LOB データと、それに関連するすべての基礎的な活動が実体化されないようにすることができます。

LOB ロケーターを使用したときの利点を以下にリストします。

- LOB ロケーターを使用して LOB を操作するときにストレージが節約される
- データベースから取得することなくデータを操作できる
- LOB を保持するためにあまりストレージが使用されない
- 大きなデータ断片を移動するための時間やリソースが不要になるためパフォーマンスが向上する

特に、LOB ロケーターは以下の状況で役に立ちます。

- LOB の小さな部分のみを必要とする場合
- LOB 全体を保持するだけの十分なメモリーがない場合
- パフォーマンスが重要な場合
- クライアント環境またはサーバー環境において、ネットワークを使用してシステム間でデータを移動しない場合

以下のコード例は pliclob サンプル・プログラムからのものです。このサンプル・プログラムは、LOB ロケーターを使用して、dsn8a10.emp_photo_resume DB2 V10 表にある resume CLOB のセクションを識別したり操作したりします。 (各行の前にある番号はプログラムの一部ではなく、プログラムの後にある説明で使用します。)

```
1.   dcl hv_loc_resume  sql type is clob_locator;
2.    exec sql
3.       select resume into :hv_loc_resume
4.       from dsn8a10.emp_photo_resume
5.       where empno = :hv_empno;
6.
7.    exec sql
8.       set :start_resume = (posstr(:hv_loc_resume, 'Resume:'));
```

2 行目から 5 行目では、LOB ロケーター hv_loc_resume を emp_photo_resume 表内の従業員番号 hv_empno の履歴書 (resume) の位置に設定しています。 7 行目から 8 行目では、*start_resume* ホスト変数を resume の 'Resume:' セクションの先頭に設定しています。 これで、履歴書 (resume) をデータベース内に置いたまま、resume データの操作を開始できます。

LOB ファイル参照変数

LOB ファイル参照変数を使用すれば、DB2 システム外の外部ファイルと、LOB 列との間でデータをインポートしたりエクスポートしたりできます。

LOB ファイル参照変数を使用したときの利点を以下にリストします。

- ホスト変数を使用して LOB データを移動する場合よりも処理時間が短い。データの移動が DB2 処理時間やネットワーク転送時間とオーバーラップすることはありません。
- アプリケーション・ストレージの使用量が少ない。 LOB データは DB2 からファイルに直接移動されるため、アプリケーションのメモリー内で実体化されることはありません。

以下のコード例は pliclob サンプル・プログラムからのものです。このサンプル・プログラムは、LOB ファイル参照を使用して、resume の一部をトリムした新規バージョンを外部ファイル内に作成します。 (各行の前にある番号はプログラムの一部ではなく、プログラムの後にある説明で使用します。)

```
1.   dcl hv_clob_file   sql type is clob_file;
2.   name_string = '/SYSTEM/tmp/pliclob2.txt';
3.   hv_clob_file.sql_lob_file_name_len = length(name_string);
4.   hv_clob_file.sql_lob_file_name     = name_string;
5.   hv_clob_file.sql_lob_file_options  = ior(sql_file_overwrite);
6.
7.   exec sql
8.    values ( substr(:hv_loc_resume,:start_resume,
9.                    :start_pers_info-:start_resume)
10.          || substr(:hv_loc_resume,:start_work_hist,
11.                    :start_interests-:start_work_hist)
12.          )
13.    into :hv_clob_file;
```

ホスト変数 *hv_clob_file* が LOB ファイル参照として宣言されています。 2 行目から 4 行目では、LOB ファイル参照のファイル名フィールドに完全修飾ファイル名を設定し、ファイル名の長さを設定しています。 既存のファイルがすべて上書きされるように、overwrite オプションを設定しています (5 行目)。 このようなオプションや他のファイル・オプションについて詳しくは、「*DB2 for z/OS* アプリケーション・プログラミングおよび *SQL* 解説書」を参照してください。

次に、8 行目から 13 行目で、SQL VALUES ステートメントを使用して、resume の名前と resume のワーク・ヒストリー・セクションとを連結し、hv_clob_file LOB ファイル参照に直接入れています。

例: pliclob サンプル・プログラム

以下の PL/I サンプル・プログラムは、PL/I と DB2 の環境で CLOB を操作する方法を示しています。

このプログラムを正しく実行するには、DB2 に付属のサンプル・データベースがインストールされていなければなりません。 このサンプルでは、DB2 V10 および表 dsn8a10.emp_photo_resume が前提となります。別のバージョンの DB2 を使用している場合は、表の参照を変更する必要があります。

注:

- LOB ロケーターおよび LOB ファイル参照変数を使用する場合、resume CLOB はデータベース内に残り、メモリーやストレージには入りません。
- データベース内の resume CLOB のフォーマットが変更されることはありません。resume のフォーマットの変更は、書き出された 2 番目のファイルでのみ行われます。

```
pliclob: procedure options(main);
 display('begin pliclob');
 exec sql include sqlca;

 dcl hv_empno       char(06);
 dcl name_string    char(256) var;
 dcl hv_resume      sql type is clob(50k);
 dcl hv_clob_file   sql type is clob_file;
 dcl hv_loc_resume  sql type is clob_locator;

 dcl start_resume     fixed bin(31);
 dcl start_pers_info  fixed bin(31);
 dcl start_dept_info  fixed bin(31);
 dcl start_education  fixed bin(31);
 dcl start_work_hist  fixed bin(31);
 dcl start_interests  fixed bin(31);

 /* Extract resume CLOB for employee '000130' into a file in z/OS  */
 /* UNIX file system. The contents of this file shows the initial  */
 /* format of the resume CLOB in the data base.                    */
 /* Note: this program must have 'write' access to the directory   */
 /*       designated in the 'name_string' variable.                */
 name_string = '/SYSTEM/tmp/pliclob1.txt';
 hv_clob_file.sql_lob_file_name_len = length(name_string);
 hv_clob_file.sql_lob_file_name     = name_string;
 hv_clob_file.sql_lob_file_options  = ior(sql_file_overwrite);

 hv_empno = '000130';
 exec sql
    select resume into :hv_clob_file
    from dsn8a10.emp_photo_resume
    where empno = :hv_empno;
 display('file1   sqlca.sqlcode = ' || sqlca.sqlcode );

 /* Next, a CLOB locator is used to locate the resume CLOB for     */
 /* employee number '000130' in the data base.  Then a series of   */
 /* DB2 SET statements using the posstr DB2 function finds the     */
 /* beginning position of each section within the resume.          */
 exec sql
    select resume into :hv_loc_resume
    from dsn8a10.emp_photo_resume
    where empno = :hv_empno;
 display('select resume sqlcode = '|| sqlca.sqlcode);

 exec sql set :start_resume =
          (posstr(:hv_loc_resume, 'Resume:'));
 display('first set sqlcode     = '|| sqlca.sqlcode);

 exec sql set :start_pers_info =
          (posstr(:hv_loc_resume, 'Personal Information'));
 display('second set sqlcode    = '|| sqlca.sqlcode);
```

図 *9. pliclob* サンプル・プログラム

```
 exec sql set :start_dept_info =
          (posstr(:hv_loc_resume, 'Department Information'));
 display('third set sqlcode     = '|| sqlca.sqlcode);

 exec sql set :start_education =
          (posstr(:hv_loc_resume, 'Education'));
 display('fourth set sqlcode    = '|| sqlca.sqlcode);

 exec sql set :start_work_hist =
          (posstr(:hv_loc_resume, 'Work History'));
 display('fifth set sqlcode     = '|| sqlca.sqlcode);

 exec sql set :start_interests =
          (posstr(:hv_loc_resume, 'Interests'));
 display('sixth set sqlcode     = '|| sqlca.sqlcode);

 /* Finally, by using the CLOB locator and the start references    */
 /* of each section in the resume, along with the DB2 substr and   */
 /* concatenate (||) functions, the resume CLOB is written out to  */
 /* a second file in a slightly different format:                  */
 /*   1. the Personal Information section is omitted due to        */
 /*      privacy concerns.                                         */
 /*   2. the sections within the resume are written out in this    */
 /*      order: Resume, Work History, Education then Department    */
 /*      Information.                                              */
 /*                                                                */
 /* After the second file is written out, the changes to the       */
 /* resume CLOB can be verified by comparing the contents of the   */
 /* two files pliclob1.txt and pliclob2.txt.                       */
 /*                                                                */
 /* Note: this program must have 'write' access to the directory   */
 /*       designated in the 'name_string' variable.                */
 name_string = '/SYSTEM/tmp/pliclob2.txt';
 hv_clob_file.sql_lob_file_name_len = length(name_string);
 hv_clob_file.sql_lob_file_name     = name_string;
 hv_clob_file.sql_lob_file_options  = ior(sql_file_overwrite);

 exec sql
  values ( substr(:hv_loc_resume,:start_resume,
                  :start_pers_info-:start_resume)
         || substr(:hv_loc_resume,:start_work_hist,
                  :start_interests-:start_work_hist)
         || substr(:hv_loc_resume,:start_education,
                  :start_work_hist-:start_education)
         || substr(:hv_loc_resume,:start_dept_info,
                  :start_education-:start_dept_info)
        )
   into :hv_clob_file;

 display('file2   sqlca.sqlcode = ' || sqlca.sqlcode );
 display('End   pliclob');

end;
```

pliclob サンプル・プログラム *(続き)*

SQL プリプロセッサー・メッセージの抑止

IBM 提供のコンパイラー・ユーザー出口 (IBMUEXIT) を使用して、メッセージを抑止したり、メッセージの重大度を変更したりすることができます。

ユーザー出口の変更によるプリプロセッサー・メッセージの抑止例については、549 ページの『SQL メッセージの抑止例』を参照してください。

CICS プリプロセッサー

CICS 環境でトランザクションとして実行される PL/I アプリケーション内では、EXEC CICS ステートメントを使用できます。

PP(CICS) オプションを指定しない場合、EXEC CICS ステートメントが構文解析され、ステートメント内の変数参照が検証されます。変数参照が正しい場合、NOCOMPILE オプションが有効であれば、メッセージは出されません。 CICS 変換プログラムを呼び出さず、COMPILE オプションが有効になっていると、コンパイラーは S レベル・メッセージを発行します。

PP オプションの CICS サブオプションを指定すると、コンパイラーは、CICS プリプロセッサーを呼び出します。 互換性のため、CICS オプション、XOPT オプション、または XOPTS オプションのいずれかが有効な場合でも、コンパイラーは CICS プリプロセッサーを呼び出します。ただし、これらのオプションはいずれも PP(CICS) オプションと一緒に指定してはなりません。

プログラミングとコンパイルに関する考慮事項

CICS の環境で実行するプログラムを開発する場合は、次の 2 つの方法のどちらかで EXEC CICS コマンドをすべて変換する必要があります。

- PL/I コンパイルの前のジョブ・ステップで、CICS 提供のコマンド言語変換プログラムを使用する
- PL/I コンパイル時に、PL/I CICS プリプロセッサーを使用する (CICS TS 2.2 以降が必要)

CICS プリプロセッサーを使用するには、PP(CICS) および DFT(EBCDIC) コンパイル時オプションも指定する必要があります。CICS に渡されるデータはすべてネイティブ・フォーマットでなければなりません。

PP(CICS) オプションのサブオプションの 1 つとして CICS を指定しないかぎり、コンパイラーは、ソースの中に EXEC CICS ステートメントがあると、すべてフラグを立てます。同様に、EXEC CPSM または EXEC DLI ステートメントについても、PP(CICS) オプションのサブオプションとしてそれぞれ CPSM または DLI を指定しないと、これらのステートメントにフラグが立てられます。

CICS プログラムが MAIN プロシージャーである場合、SYSTEM(CICS) オプションまたは SYSTEM(MVS) オプションも指定してコンパイルする必要があります。SYSTEM(MVS) を指定してコンパイルする場合は、ランタイム APAR PQ91318 に対応する PTF を適用する必要があります。このオプションでは NOEXECOPS が暗黙指定され、MAIN プロシージャーに渡されるパラメーターはすべて POINTER でなければなりません。 SYSTEM コンパイル時オプションに関する説明については、 99 ページの『SYSTEM』を参照してください。

CICS プログラムを再入可能にしたい場合で、ご使用のプログラムが FILE または CONTROLLED 変数を使用している場合は、それも NOWRITABLE でコンパイルしなければなりません。

CICS プログラムで、EXEC CICS ステートメントが含まれたファイルがインクルードされているか、またはこのステートメントが含まれたマクロが使用されている場

合は、コードを変換 (上記のいずれかの方法で) する前にマクロ・プリプロセッサーも実行する必要があります。 CICS プリプロセッサーを使用する場合、次の例に示すような PP オプション 1 つを使用してこのプリプロセッサーを指定できます。

```
pp (macro(...) cics(...) )
```

CICS プリプロセッサーは、CICS 変数および API の宣言のセットをすべてのネストなしプロシージャーに追加します。 したがって、このプリプロセッサーは、対応する END ステートメントを必要とするすべてのステートメントを追跡し、これらのステートメントの中に欠落しているものや正しくないものがある場合は、このプリプロセッサーが誤って導かれる可能性があるため、これらの宣言を挿入しません。また、このプリプロセッサーは、このようなステートメントのネストの深さが 150 を超える場合は、重大メッセージで終了します。

最後に、CICS プリプロセッサーを使用するには、PL/I コンパイラーに対して STEPLIB DD の一部として CICS SDFHLOAD データ・セットが必要です。

CICS プリプロセッサー・オプション

CICS 変換プログラムは、数多くのオプションをサポートしています。

これらのオプションについては、「*CICS Application Programming Guide*」を参照してください。

これらのオプションは引用符 (左右の引用符が同じであれば単一引用符でも二重引用符でもかまわない) で囲む必要があることに注意してください。例えば、EDF オプションを指定して CICS プリプロセッサーを呼び出すには、オプションを PP(CICS('EDF')) と指定します。

PL/I アプリケーション内での CICS ステートメントのコーディング

「*CICS on Open Systems Application Programming Guide*」に定義されている言語を使用して PL/I アプリケーションで CICS ステートメントをコーディングできます。 CICS コード固有の要件については、以下のセクションで説明します。

CICS ステートメントの組み込み

組み込みプリプロセッサーではなく、CICS 変換プログラムを使用したい場合は、ユーザーの PL/I プログラムの最初のステートメントは PROCEDURE ステートメントでなければなりません。実行可能ステートメントを置くことができる任意の場所で、プログラムに CICS ステートメントを追加できます。 それぞれの CICS ステートメントは EXEC (または EXECUTE) CICS で始まり、セミコロン (;) で終わる必要があります。

例えば、GETMAIN ステートメントは次のようにコーディングされます。

```
EXEC CICS GETMAIN SET(BLK_PTR) LENGTH(STG(BLK));
```

コメント:

CICS ステートメントのほかに、ブランクを入力できる場所では組み込み CICS ステートメントに PL/I コメントを組み込むことができます。

CICS ステートメントの継続:

CICS ステートメントの行継続規則は、他の PL/I ステートメントと同じです。

コードの組み込み:

組み込まれるコードに EXEC CICS ステートメントが含まれている場合、または EXEC CICS ステートメントを生成する PL/I マクロがプログラムで使用される場合は、以下のいずれかのオプションを使用する必要があります。
- MACRO コンパイル時オプション
- PP オプションの MACRO オプション (PP オプションの CICS オプションの前)

マージン:

CICS ステートメントは、MARGINS コンパイル時オプションで指定された列の範囲内でコーディングする必要があります。

ステートメント・ラベル:

EXEC CICS ステートメントには、PL/I ステートメントと同様にラベル接頭部を付けることができます。

PL/I を使用した CICS トランザクションの作成

PL/I を CICS 機能と組み合わせて使用して、CICS サブシステム用のアプリケーション・プログラム (トランザクション) を作成することができます。これを行う場合、PL/I プログラムには、通常はオペレーティング・システムから直接提供される機能が CICS から提供されます。これらの機能には、ほとんどのデータ管理機能や、ジョブとタスクの管理機能すべてが含まれます。

次の PL/I CICS プログラムの制限を順守する必要があります。
- マクロ・レベル CICS はサポートされません。
- PL/I の入力および出力は、以下の場合を除いて使用できません。
  - PUT FILE(SYSPRINT)
  - DISPLAY
  - CALL PLIDUMP
- PLISRTx 組み込みサブルーチンは使用できません。
- PL/I 以外の言語で書かれたルーチンに EXEC CICS ステートメントが含まれている場合、そのルーチンを PL/I CICS プログラムから呼び出すことはできません。EXEC CICS ステートメントを含む非 PL/I プログラムとやり取りする場合は、EXEC CICS LINK または EXEC CICS XCTL を使用する必要があります。

CICS では PUT FILE(SYSPRINT) は許可されていますが、パフォーマンスが低下する恐れがあるため、通常、PUT FILE(SYSPRINT) は実動プログラムで使用しないでください。

CICS EIB アドレスは、CICS 変換プログラム、または OPTIONS(MAIN) プログラムの場合は PL/I CICS プリプロセッサーのいずれかによってのみ生成されるため、以下のいずれかの方法で OPTIONS(FETCHABLE) ルーチンの EIB にアドレッシングできるようにする必要があります。

- 次のコマンドを使用します。

  ```
  EXEC CICS ADDRESS EIB(DFHEIPTR)
  ```

- EIB アドレスを、外部プロシージャーを呼び出す CALL ステートメントに引数として渡します。

エラー処理

言語環境プログラムでは、一部の EXEC CICS コマンドは、いずれの PL/I ON ユニットでも、PL/I ON ユニットから呼び出されるいずれのコードでも使用できません。

以下の EXEC CICS コマンドは ON ユニットでは使用できません。

- EXEC CICS ABEND
- EXEC CICS HANDLE AID
- EXEC CICS HANDLE ABEND
- EXEC CICS HANDLE CONDITION
- EXEC CICS IGNORE CONDITION
- EXEC CICS POP HANDLE
- EXEC CICS PUSH HANDLE

他のすべての EXEC CICS コマンドは ON ユニット内で許可されています。 ただし、それらのコマンドは、NOHANDLE オプション、RESP オプション、または RESP2 オプションを使用してコーディングする必要があります。

第 3 章 PL/I カタログ式プロシージャーの用法

本章ではIBM Enterprise PL/I for z/OS コンパイラーで使用する IBM 提供の標準カタログ式プロシージャーについて説明します。また、プロシージャーの呼び出し方、一時的または永続的な変更方法についても説明します。

任意のカタログ式プロシージャーを使用するためには、言語環境プログラム SCEERUN データ・セットが STEPLIB に存在し、コンパイラーからアクセス可能になっていなければなりません。

カタログ式プロシージャーは、ライブラリーに保管されたジョブ制御ステートメントのセットです。カタログ式プロシージャーには、1 つ以上の EXEC ステートメントがあり、さらに 1 つ以上の DD ステートメントが続く場合があります。 ステートメントを検索するには、入力ストリーム内の EXEC ステートメントの PROC パラメーターの中で、カタログ式プロシージャーの名前を指定します。

カタログ式プロシージャーを使うと、時間を節約し、ジョブ制御言語 (JCL) エラーを減らすことができます。 カタログ式プロシージャー内のステートメントがユーザー要件と正しく一致していなくても、ジョブが終わるまでの間、簡単にステートメントを変更したり、新たにステートメントを付け加えることができます。 このプロシージャーは見直しを行って変更を加え、使用できる各種機能を最も効率よく利用できるよう、また、ユーザー固有の規則を守れるようにしなければなりません。

IBM 提供のカタログ式プロシージャー

このセクションでは、Enterprise PL/I for z/OS で使用するために提供されている PL/I カタログ式プロシージャーについて説明します。

コンパイルおよびリンク・エディットのための個々のステートメントについては、186 ページの『JCL を使用した z/OS の下でのコンパイラーの呼び出し』 および「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

以下の PL/I カタログ式プロシージャーは、Enterprise PL/I for z/OS で使用するために提供されています。

IBMZC
: コンパイルのみ

IBMZCB
: コンパイルおよびバインド

IBMZCBG
: コンパイル、バインド、および実行

IBMQC
: コンパイルのみ (64 ビット)

IBMQCB
: コンパイルおよびバインド (64 ビット)

IBMQCBG
: コンパイル、バインド、および実行 (64 ビット)

© Copyright IBM Corp. 1999, 2017

| カタログ式プロシージャー IBMZCB、IBMZCBG、IBMQCB、および IBMQCBG は、DFSMS/MVS 1.4 で導入されたプログラム管理バインダーの機能を使用します。これらのプロシージャーは、PDSE にプログラム・オブジェクトを作成します。

上記のカタログ式プロシージャーには、入力データ・セット用の DD ステートメントは入っていません。ユーザーが必ず提供しなければなりません。 図 10 の例は、PL/I プログラムのコンパイル、バインド、および実行のためにカタログ式プロシージャー IBMZCBG を呼び出す時に使用する JCL ステートメントを示しています。

Enterprise PL/I は、最小 REGION サイズ 32M を必要とします。 大きいプログラムにはより多くのストレージが必要です。 実行しようとするカタログ式プロシージャーを呼び出す EXEC ステートメント上で REGION を指定しないと、コンパイラーはユーザーのサイト用にデフォルト REGION サイズを使用します。 このデフォルト・サイズは、PL/I プログラムのサイズにより適切な場合と適切でない場合があります。

最適化をオンにしてプログラムをコンパイルする場合は、必要な REGION サイズ (および時間) がはるかに大きくなることがあります。EXEC ステートメントでの REGION の指定方法の例は、図 10 を参照してください。

例: カタログ式プロシージャーの呼び出し

```
//COLEGO    JOB
//STEP1     EXEC IBMZCBG, REGION.PLI=32M
//PLI.SYSIN DD *
                .
                .
                .
    (insert PL/I program to be compiled here)
                .
                .
                .
/*
```

図 *10.* カタログ式プロシージャーの呼び出し

コンパイルのみ (IBMZC)

170 ページの図 11 に示すカタログ式プロシージャー IBMZC には、プロシージャー・ステップが 1 つだけあります。このステップでコンパイル用に指定されているオプションは OBJECT と OPTIONS です。 (IBMZPLI は、コンパイラーのシンボル名です。) コンパイル・プロシージャー・ステップの入ったその他のカタログ式プロシージャーと同様に、IBMZC には入力データ・セット用の DD ステートメントは入っていません。ユーザーが修飾 dd 名 PLI.SYSIN を付けて適切なステートメントを必ず提供しなければなりません。

OBJECT コンパイル時オプションを指定すると、コンパイラーは、リンケージ・エディターへの入力に適した構文のオブジェクト・モジュールを、SYSLIN という名

前の DD ステートメントで定義された標準データ・セットに入れます。 このステートメントは順次装置上で &&LOADSET という名前の一時データ・セットを定義します。ジョブ終了後にオブジェクト・モジュールを保持したい場合は、&&LOADSET を永続名 (すなわち、&& で始まらない名前) で置き換え、またそのデータ・セットを使用した最後のプロシージャー・ステップについては、適切な DISP パラメーターに KEEP を指定しなければなりません。 そのためには、下記のように、ユーザー独自の SYSLIN DD ステートメントを提供します。 このステートメント上のデータ・セット名と後処理パラメーターにより、IBMZC プロシージャーの SYSLIN DD ステートメントの指定が変更されます。 この例では、コンパイル・ステップが唯一のジョブ・ステップです。

```
//PLICOMP EXEC IBMZC
//PLI.SYSLIN  DD  DSN=MYPROG,DISP=SHR
//PLI.SYSIN   DD ...
```

170 ページの図 11 の DISP パラメーターにある MOD という項により、コンパイラーはデータ・セットに複数のオブジェクト・モジュールを入れることができます。また、PASS により、対応する DD ステートメントがある限り、以降のプロシージャー・ステップでもそのデータ・セットを使用できます。

SYSLIN SPACE パラメーターを指定すると、1 シリンダーの初期割り振りを行うことができ、必要であればさらに 15 回の割り振りを行うことができます (合計 16 シリンダー)。

```
//IBMZC   PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of IBM                            *
//* 5655-PL5                                                        *
| //* Copyright IBM Corp. 1999, 2017 All Rights Reserved             *
//*                                                                 *
//* US Government Users Restricted Rights - Use, duplication or     *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*
//*                                                                 *
//*******************************************************************
//*
//* IBM Enterprise PL/I for Z/OS
//* VERSION 5 RELEASE 2 MODIFICATION 0
//*
//*  COMPILE A PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*
//*  USER MUST SUPPLY //PLI.SYSIN DD STATEMENT THAT IDENTIFIES
//*   LOCATION OF COMPILER INPUT
//*
//*******************************************************************
//* COMPILE STEP
//*******************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS'
| //STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
| //         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
```

図 *11.* カタログ式プロシージャー *IBMZC*

コンパイルおよびバインド (IBMZCB)

171 ページの図 12 の IBMZCB カタログ式プロシージャーには、2 つのプロシージャー・ステップがあります。1 つは PLI で、これはカタログ式プロシージャー IBMZC と同じです。もう 1 つは BIND で、これは最初のプロシージャー・ステップで作成されたオブジェクト・モジュールをバインドするためにプログラム管理バインダー (シンボル名 IEWBLINK) を呼び出します。

コンパイル・プロシージャー・ステップ用の入力データには、修飾 dd 名の PLI.SYSIN が必要です。 EXEC ステートメント BIND 内の COND パラメーターは、コンパイラーが生成した戻りコードが 8 より大きい場合 (つまり、コンパイル時に重大エラーや回復不能エラーが発生した場合) に当該プロシージャー・ステップをバイパスするように指定します。

```
//IBMZCB  PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200,GOPGM=GO
//*
//*****************************************************************
//*                                                               *
//* Licensed Materials - Property of IBM                          *
| //* 5655-PL5                                                     *
| //* Copyright IBM Corp. 1999, 2017 All Rights Reserved           *
//*                                                               *
//* US Government Users Restricted Rights - Use, duplication or   *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*
//*                                                               *
//*****************************************************************
//*
//* IBM Enterprise PL/I for Z/OS
//* VERSION 5 RELEASE 2 MODIFICATION 0
//*
//*  COMPILE AND BIND A PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   GOPGM     GO               MEMBER NAME FOR PROGRAM OBJECT
//*
//*********************************************************************
//* COMPILE STEP
//*********************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS'
| //STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
| //         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//*********************************************************************
//* BIND STEP
//*********************************************************************
//BIND     EXEC PGM=IEWBLINK,COND=(8,LT,PLI),
//             PARM='XREF,COMPAT=PM3'
//SYSLIB   DD  DSN=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=*.PLI.SYSLIN,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&&GOSET(&GOPGM),DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(1024,(50,20,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD  DUMMY
//SYSIN    DD  DUMMY
```

図 *12.* カタログ式プロシージャー *IBMZCB*

プログラム管理バインダーは、常に、SYSLMOD という名前の DD ステートメントで定義された標準データ・セットに、作成したプログラム・オブジェクトを入れます。 カタログ式プロシージャーの中のこのステートメントは、新しい一時ライブラリー &&GOSET を指定します。プログラム・オブジェクトはこの一時ライブラリーに入れられ、GO というメンバー名を与えられます。 一時ライブラリーを指定する際は、カタログ式プロシージャーはプログラム・オブジェクトが同じジョブ内

で実行されることを想定します。プログラム・オブジェクトを保持したい場合は、SYSLMOD という名前の DD ステートメントをユーザー独自の名前に置き換えなければなりません。

コンパイル、バインド、および実行 (IBMZCBG)

173 ページの図 13 の IBMZCBG カタログ式プロシージャーには、PLI、BIND、GO の 3 つのプロシージャー・ステップがあります。 PLI と BIND は IBMZCB の 2 つのプロシージャー・ステップと同じです。GO は BIND ステップで作成されたプログラム・オブジェクトを実行します。 GO ステップは、前のプロシージャー・ステップで重大エラーまたは回復不能エラーが発生しなかった場合にのみ実行されます。

コンパイル・プロシージャー・ステップ用の入力データは、PLI.SYSIN という名前の DD ステートメントで指定しなければならず、また、GO ステップ用のものは、GO.SYSIN という名前の DD ステートメントで指定しなければなりません。

```
//IBMZCBG  PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200,GOPGM=GO
//*
//*****************************************************************
//*                                                               *
//* Licensed Materials - Property of IBM                          *
| //* 5655-PL5                                                     *
| //* Copyright IBM Corp. 1999, 2017 All Rights Reserved           *
//*                                                               *
//* US Government Users Restricted Rights - Use, duplication or   *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*
//*                                                               *
//*****************************************************************
//*
//* IBM Enterprise PL/I for Z/OS
//* VERSION 5 RELEASE 2 MODIFICATION 0
//*
//*  COMPILE, BIND, AND RUN A PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   GOPGM     GO               MEMBER NAME FOR PROGRAM OBJECT
//*
//*****************************************************************
//* COMPILE STEP
//*****************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS'
| //STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
| //         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//*****************************************************************
//* BIND STEP
//*****************************************************************
//BIND     EXEC PGM=IEWBLINK,COND=(8,LT,PLI),
//             PARM='XREF,COMPAT=PM3'
//SYSLIB   DD  DSN=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=*.PLI.SYSLIN,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&&GOSET(&GOPGM),DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(1024,(50,20,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD  DUMMY
//SYSIN    DD  DUMMY
//*****************************************************************
//* RUN STEP
//*****************************************************************
//GO       EXEC PGM=*.BIND.SYSLMOD,COND=((8,LT,PLI),(8,LE,BIND))
//STEPLIB  DD  DSN=&LIBPRFX;.SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

図 *13.* カタログ式プロシージャー *IBMZCBG*

コンパイルのみ - 64 ビット (IBMQC)

IBMQC カタログ式プロシージャー (175 ページの図 14 を参照) には、プロシージャー・ステップが 1 つのみ含まれています。このステップでコンパイル用に指定されているオプションは OBJECT と OPTIONS です。 (IBMZPLI は、コンパイラーのシンボル名です。) コンパイル・プロシージャー・ステップが含まれるその他のカタログ式プロシージャーと同様に、IBMQC には入力データ・セット用の DD ステートメントは含まれていません。修飾 DD 名 PLI.SYSIN を持つ適切なステートメントを必ず指定する必要があります。

LP(64) コンパイル時オプションによりコンパイラーは 64 ビット・コードを生成します。 OBJECT コンパイル時オプションを指定すると、コンパイラーは、リンケージ・エディターへの入力に適した構文のオブジェクト・モジュールを、SYSLIN という名前の DD ステートメントで定義された標準データ・セットに入れます。 このステートメントは順次装置上で &&LOADSET という名前の一時データ・セットを定義します。ジョブ終了後にオブジェクト・モジュールを保持したい場合は、&&LOADSET を永続名 (すなわち、&& で始まらない名前) で置き換え、またそのデータ・セットを使用した最後のプロシージャー・ステップについては、適切な DISP パラメーターに KEEP を指定しなければなりません。 そのためには、下記のように、ユーザー独自の SYSLIN DD ステートメントを提供します。 このステートメント上のデータ・セット名と後処理パラメーターは、IBMQC プロシージャー SYSLIN DD ステートメント上のデータ・セット名と後処理パラメーターに優先します。この例では、コンパイル・ステップが唯一のジョブ・ステップです。

```
//PLICOMP EXEC IBMQC
//PLI.SYSLIN DD DSN=MYPROG,DISP=SHR
//PLI.SYSIN DD ...
```

175 ページの図 14 の DISP パラメーターにある MOD という項により、コンパイラーはデータ・セットに複数のオブジェクト・モジュールを入れることができます。また、PASS により、対応する DD ステートメントが組み込まれていれば、以降のプロシージャー・ステップでそのデータ・セットを使用できます。

SYSLIN SPACE パラメーターを指定すると、1 シリンダーの初期割り振りを行うことができ、必要であればさらに 15 回の割り振りを行うことができます (合計 16 シリンダー)。

```
//IBMQC   PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of IBM                            *
//* 5655-PL5                                                        *
//* Copyright IBM Corp. 1999, 2017 All Rights Reserved              *
//*                                                                 *
//* US Government Users Restricted Rights - Use, duplication or     *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*
//*                                                                 *
//*******************************************************************
//*
//* IBM Enterprise PL/I for Z/OS
//* VERSION 5 RELEASE 2 MODIFICATION 0
//*
//*  COMPILE A 64-BIT PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*
//*  USER MUST SUPPLY //PLI.SYSIN DD STATEMENT THAT IDENTIFIES
//*   LOCATION OF COMPILER INPUT
//*
//*******************************************************************
//* COMPILE STEP
//*******************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='LP(64),OBJECT,OPTIONS'
//STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
//         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
```

図 *14.* カタログ式プロシージャー *IBMQC (64* ビット*)*

コンパイルおよびバインド **- 64** ビット **(IBMQCB)**

IBMQCB カタログ式プロシージャー (176 ページの図 15 を参照) には、PLI と BIND という 2 つのプロシージャー・ステップが含まれています。 PLI はカタログ式プロシージャー IBMQC と同じです。 BIND は、最初のプロシージャー・ステップで作成されたオブジェクト・モジュールをバインドするためにプログラム管理バインダー (シンボル名 IEWBLINK) を呼び出します。

コンパイル・プロシージャー・ステップ用の入力データには、修飾 dd 名の PLI.SYSIN が必要です。 EXEC ステートメント BIND 内の COND パラメーターは、コンパイラーが生成した戻りコードが 8 より大きい場合 (つまり、コンパイル時に重大エラーや回復不能エラーが発生した場合) に当該プロシージャー・ステップをバイパスするように指定します。

64 ビット・プログラムをバインドする場合は、DYNAM(DLL) バインダー・オプションが必要です。また、オプション CASE(MIXED)、AMODE(64)、および

RMODE(ANY) を強くお勧めします。さらに、PL/I サイド・デック IBMPQV11 および C サイド・デック・ファイル CELQS003 をバインダーへの入力として組み込む必要があります。

```
//IBMQCB  PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200,GOPGM=GO
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of IBM                            *
//* 5655-PL5                                                        *
//* Copyright IBM Corp. 1999, 2017 All Rights Reserved              *
//*                                                                 *
//* US Government Users Restricted Rights - Use, duplication or     *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*
//*                                                                 *
//*******************************************************************
//*
//* IBM Enterprise PL/I for z/OS
//* バージョン 5 リリース 2 モディフィケーション 0
//*
//*  COMPILE AND BIND A 64-BIT PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   GOPGM     GO               MEMBER NAME FOR PROGRAM OBJECT
//*
//*******************************************************************
//* COMPILE STEP
//*******************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='LP(64),OBJECT,OPTIONS'
//STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
//         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//*******************************************************************
//* BIND STEP
//*******************************************************************
//BIND     EXEC PGM=IEWBLINK,COND=(8,LT,PLI),
// PARM='CASE(MIXED),DYNAM(DLL),AMODE=64,RMODE=ANY,LIST,MAP,XREF'
//SYSLIB   DD  DSN=&LIBPRFX..SCEEBND2,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=*.PLI.SYSLIN,DISP=(OLD,DELETE)
//         DD  DSN=&LIBPRFX..SCEELIB(CELQS003),DISP=SHR
//         DD  DSN=&LIBPRFX..SCEELIB(IBMPQV11),DISP=SHR
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&&GOSET(&GOPGM),DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(1024,(50,20,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD  DUMMY
//SYSIN    DD  DUMMY
```

図 *15.* カタログ式プロシージャー *IBMQCB (64* ビット*)*

プログラム管理バインダーは、常に、SYSLMOD という名前の DD ステートメントで定義された標準データ・セットに、作成したプログラム・オブジェクトを入れます。 カタログ式プロシージャーの中のこのステートメントは、新しい一時ライブ

ラリー &&GOSET を指定します。プログラム・オブジェクトはこの一時ライブラリーに入れられ、GO というメンバー名を与えられます。 一時ライブラリーを指定する際は、カタログ式プロシージャーはプログラム・オブジェクトが同じジョブ内で実行されることを想定します。プログラム・オブジェクトを保持したい場合は、SYSLMOD という名前の DD ステートメントをユーザー独自の名前に置き換えなければなりません。

コンパイル、バインド、および実行 - 64 ビット (IBMQCBG)

IBMQCBG カタログ式プロシージャー (178 ページの図 16 を参照) には、PLI、BIND、および GO という 3 つのプロシージャー・ステップが含まれています。 PLI と BIND は IBMQCB の 2 つのプロシージャー・ステップと同じです。GO では、BIND ステップで作成されたプログラム・オブジェクトが実行されます。 GO ステップは、前のプロシージャー・ステップで重大エラーまたは回復不能エラーが発生しなかった場合にのみ実行されます。

コンパイル・プロシージャー・ステップ用の入力データは、PLI.SYSIN という名前の DD ステートメントで指定しなければならず、また、GO ステップ用のものは、GO.SYSIN という名前の DD ステートメントで指定しなければなりません。

```
//IBMQCBG  PROC LNGPRFX='IBMZ.V5R2M0',LIBPRFX='CEE',
//             SYSLBLK=3200,GOPGM=GO
//*
//*******************************************************************
//* Licensed Materials - Property of IBM                            *
//* 5655-PL5                                                        *
//* Copyright IBM Corp. 1999, 2017 All Rights Reserved              *
//*                                                                 *
//* US Government Users Restricted Rights - Use, duplication or     *
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. *
//*                                                                 *
//*******************************************************************
//*
//*  IBM Enterprise PL/I for z/OS
//*  バージョン 5 リリース 2 モディフィケーション 0
//*
//*  COMPILE, BIND, AND RUN A 64-BIT PL/I PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IBMZ.V5R2M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*   LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
//*   SYSLBLK   3200             BLKSIZE FOR OBJECT DATA SET
//*   GOPGM     GO               MEMBER NAME FOR PROGRAM OBJECT
//*
//*******************************************************************
//* COMPILE STEP
//*******************************************************************
//PLI      EXEC PGM=IBMZPLI,PARM='LP(64),OBJECT,OPTIONS'
//STEPLIB  DD  DSN=&LNGPRFX..SIBMZCMP,DISP=SHR
//         DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=&SYSLBLK)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSALLDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//*******************************************************************
//* BIND STEP
//*******************************************************************
//BIND     EXEC PGM=IEWBLINK,COND=(8,LT,PLI),
// PARM='CASE(MIXED),DYNAM(DLL),AMODE=64,RMODE=ANY,LIST,MAP,XREF'
//SYSLIB   DD  DSN=&LIBPRFX..SCEEBND2,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=*.PLI.SYSLIN,DISP=(OLD,DELETE)
//         DD  DSN=&LIBPRFX..SCEELIB(CELQS003),DISP=SHR
//         DD  DSN=&LIBPRFX..SCEELIB(IBMPQV11),DISP=SHR
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&&GOSET(&GOPGM),DISP=(MOD,PASS),UNIT=SYSALLDA,
//             SPACE=(1024,(50,20,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD  DUMMY
//SYSIN    DD  DUMMY
//*******************************************************************
//* RUN STEP
//*******************************************************************
//GO       EXEC PGM=*.BIND.SYSLMOD,COND=((8,LT,PLI),(8,LE,BIND))
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

図 *16.* カタログ式プロシージャー *IBMQCBG (64* ビット*)*

カタログ式プロシージャーの呼び出し

カタログ式プロシージャーを呼び出すには、EXEC ステートメントの PROC パラメーターにプロシージャー名を指定します。

例えば、カタログ式プロシージャー IBMZC を使用するには、入力ストリームにおいて他のジョブ制御ステートメント中の適切な位置に次のステートメントを組み込みます。

```
//stepname EXEC PROC=IBMZC
```

キーワード PROC のコーディングは必要ありません。 EXEC ステートメントの第 1 オペランドが PGM= または PROC= で始まっていない場合、ジョブ・スケジューラーはそのオペランドをカタログ式プロシージャーの名前と解釈します。次のステートメントは上記のステートメントと同等です。

```
//stepname EXEC IBMZC
```

JOB ステートメントにパラメーター MSGLEVEL=1 を入れると、オペレーティング・システムはオリジナルの EXEC ステートメントをそのリストに組み込み、カタログ式プロシージャーのステートメントを追加します。このリストでは、カタログ式プロシージャーのステートメントは最初の 2 文字の XX あるいは X/ によって識別されます。X/ は、カタログ式プロシージャーの現在の呼び出しで変更されたステートメントを意味します。

カタログ式プロシージャーを呼び出したジョブ・ステップが終わるまでの間、カタログ式プロシージャーのステートメントを変更しておかなければならないことがありますが、変更するには、DD ステートメントを付け加えるか、または EXEC ステートメントか DD ステートメント内の 1 つ以上のパラメーターを指定変更します。 例えば、コンパイラーを呼び出すカタログ式プロシージャーでは、ソース・ステートメントが入っているデータ・セットを定義するために、SYSIN という名前の DD ステートメントを追加する必要があります。 また、1 つのジョブで複数の標準リンク・エディットのプロシージャー・ステップを使用する場合、複数のロード・モジュールを実行したければ、呼び出すカタログ式プロシージャーのうち、最初のものを除くすべてのものを変更しなければなりません。

複数のカタログ式プロシージャー呼び出しを指定

同じジョブ内で、複数のカタログ式プロシージャーを呼び出すことができます。また、同じジョブ内で同じカタログ式プロシージャーを複数回呼び出すこともできます。

これらのカタログ式プロシージャーの 2 つ以上に同一のリンク・エディット・プロシージャー・ステップが関与しない限り、特別な問題が起こることはないと思われます。複数のカタログ式プロシージャーが同一のリンク・エディット・プロシージャー・ステップに関与する場合には、すべてのロード・モジュールを実行できるようにするために、以下に示す予防策をとる必要があります。

リンケージ・エディターがロード・モジュールを作成する場合、リンケージ・エディターはロード・モジュールを SYSLMOD という名前の DD ステートメントで定義された標準データ・セットに入れます。 バインダーがプログラム・オブジェクトを作成する場合、バインダーは、作成したプログラム・オブジェクトを、

SYSLMOD という名前の DD ステートメントで定義された PDSE に入れます。リンケージ・エディターの NAME ステートメントがない場合、リンケージ・エディターまたはバインダーは DSNAME パラメーターで指定されたメンバー名をモジュール名として使用します。 標準のカタログ式プロシージャーでは、SYSLMOD という名前の DD ステートメントは常に、メンバー名 GO の付いた一時ライブラリー &&GOSET を指定します。

2 つの PL/I プログラムをコンパイル、バインド、および実行するために、同じジョブの中でカタログ式プロシージャー IBMZCBG を 2 回使用し、バインダーが作成する 2 つのプログラム・オブジェクトのそれぞれに名前を付けなかった場合は、最初のプログラム・オブジェクトが 2 回実行され、2 番目のプログラム・オブジェクトはまったく実行されません。

これを防ぐには、次に挙げる方法のうちどれか 1 つを使用します。

- GO ステップの終わりにあるライブラリー &&GOSET を削除します。 GO ステップの終わりでの、カタログ式プロシージャーの最初の呼び出しで、次の構文の DD ステートメントを追加します。

```
//GO.SYSLMOD DD DSN=&&GOSET,
//   DISP=(OLD,DELETE)
```

- カタログ式プロシージャーの 2 番目以降の呼び出しで SYSLMOD という名前の DD ステートメントを変更し、ロード・モジュールの名前が違うものになるようにします (例: `//BIND.SYSLMOD DD DSN=&&GOSET(GO1)` など)。
- NAME リンケージ・エディター・オプションを使ってプログラム・オブジェクトごとに異なった名前を付け、各ジョブ・ステップの EXEC ステートメントを変更して、該当するジョブ・ステップ用の名前の付いたプログラム・オブジェクトの実行を指定します。

プログラム・オブジェクトにメンバー名を割り当てるには、SYSLMOD DD ステートメントに DSNAME パラメーターを指定して、リンケージ・エディターの NAME オプションを使用することができます。このプロシージャーを使用するとき、プログラムを実行する EXEC ステートメントが、実行されるモジュール名を SYSLMOD DD ステートメントで参照する場合は、メンバー名は NAME オプション上の名前と同じでなければなりません。

別の選択肢として、次の例にあるように、EXEC プロシージャー・ステートメントで GOPGM を使用して各プログラムに別々の名前を付けることもできます。

```
//  EXEC IBMZCBG,GOPGM=GO2
```

PL/I カタログ式プロシージャーの変更

カタログ式プロシージャーを呼び出す EXEC ステートメントにパラメーターを組み込んで、あるいは EXEC ステートメントの後に DD ステートメントを追加して、カタログ式プロシージャーを一時的に変更することができます。

一時変更は、プロシージャーが呼び出されるジョブ・ステップの期間だけ適用されます。 一時変更によって、プロシージャー・ライブラリー内のカタログ式プロシージャーのマスター・コピーが影響を受けることはありません。

一時変更は、カタログ式プロシージャーの EXEC ステートメントまたは DD ステートメントに適用できます。 EXEC ステートメントのパラメーターを変更するには、それに対応するパラメーターを、カタログ式プロシージャーを呼び出す EXEC ステートメントに入れなければなりません。 DD ステートメントの 1 つ以上のパラメーターを変更するには、カタログ式プロシージャーを呼び出す EXEC ステートメントの後に、対応する DD ステートメントを組みこまなければなりません。 カタログ式プロシージャーに新しい EXEC ステートメントを追加することはできませんが、追加の DD ステートメントはいつでも組み込むことができます。

EXEC ステートメント

カタログ式プロシージャーを呼び出す EXEC ステートメントにパラメーターを組み込めば、カタログ式プロシージャーを一時的に変更できます。

カタログ式プロシージャーを呼び出す EXEC ステートメントのパラメーターに非修飾名がある場合、そのパラメーターはカタログ式プロシージャー内のすべての EXEC ステートメントに適用されます。 カタログ式プロシージャーに対する影響は、次のようにパラメーターによって異なります。

- PARM は最初のプロシージャー・ステップに適用され、他のすべての PARM パラメーターを無効にします。
- COND と ACCT はすべてのプロシージャー・ステップに適用されます。
- TIME と REGION はすべてのプロシージャー・ステップに適用され、既存の値を指定変更します。

例えば、次のステートメントが実行されると、その下に示されていることが行われます。

```
//stepname EXEC IBMZCBG,PARM='OFFSET',REGION=32M
```

- カタログ式プロシージャー IBMZCBG を呼び出します。
- プロシージャー・ステップ PLI の EXEC ステートメントの中の OBJECT および OPTIONS を OFFSET で置き換えます。
- プロシージャー・ステップ BIND の EXEC ステートメントの中の PARM パラメーターを無効にします。
- 3 つのプロシージャー・ステップすべてに、領域サイズ 32M を指定します。

カタログ式プロシージャーの 1 つの EXEC ステートメントだけのパラメーターの値を変更したり、1 つの EXEC ステートメントに新しいパラメーターを追加したりするには、パラメーターの名前をプロシージャー・ステップの名前で修飾して、その EXEC ステートメントを識別する必要があります。 例えば、前の例でプロシージャー・ステップ PLI の領域サイズだけを変更するには、次のようにコーディングします。

```
//stepname EXEC PROC=IBMZCBG,PARM='OFFSET',REGION.PLI=90M
```

呼び出し側の EXEC ステートメントに指定されている新規パラメーターはプロシージャー EXEC ステートメントの対応パラメーターを指定変更します。

値を指定せずにキーワードと等号をコーディングすると、パラメーターに指定されたオプションをすべて無効にすることができます。 例えば、カタログ式プロシージ

ャー IBMZCBG を呼び出すときにリンケージ・エディター・リストを一括して抑止するには、次のようにコーディングします。

```
//stepname EXEC IBMZCBG,PARM.BIND=
```

DD ステートメント

DD ステートメントを EXEC ステートメントの後に追加すれば、カタログ式プロシージャーを一時的に変更できます。

カタログ式プロシージャーに DD ステートメントを追加したり、既存 DD ステートメントの 1 つ以上のパラメーターを変更するには、入力ストリーム内の適切な場所に procstepname.ddname の形の DD ステートメントを入れなければなりません。 ddname が procstepname として識別されているプロシージャー・ステップに既に存在する DD ステートメントの名前である場合に、新しい DD ステートメント内のパラメーターは既存の DD ステートメントの対応パラメーターをすべて指定変更します。そうでない場合は、新しい DD ステートメントがプロシージャー・ステップに追加されます。 例えば、次のステートメントは、カタログ式プロシージャー IBMZC のプロシージャー・ステップ PLI に DD ステートメントを追加します。

```
//PLI.SYSIN DD *
```

次のステートメントは、既存の DD ステートメント SYSPRINT を変更します (その結果、コンパイラー・リストはクラス C のシステム出力装置に伝送されます)。

```
//PLI.SYSPRINT DD SYSOUT=C
```

指定変更を行う DD ステートメントは、プロシージャー呼び出しの後で、カタログ式プロシージャーに出現するとおりの順序で出現しなければなりません。 該当ステップに指定変更を行う DD ステートメントを指定した後に、DD ステートメントを追加することができます。

DD ステートメントのパラメーターを指定変更するには、そのパラメーターを改訂した形でコーディングするか、同様の機能を実行する置換パラメーターをコーディングします (例えば、SPLIT を SPACE に置き換えます)。 パラメーターを無効にするには、値を指定せずにキーワードと等号をコーディングします。 DCB サブパラメーターは、変更したいものだけをコーディングして指定変更できます。つまり、指定変更を行う DD ステートメントの DCB パラメーターは、カタログ式プロシージャー内の対応ステートメントの DCB パラメーター全体を指定変更するとは限りません。

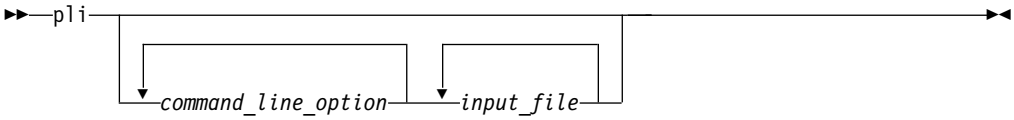
第 4 章 プログラムのコンパイル

この章では、z/OS UNIX システム・サービス (z/OS UNIX) の下でコンパイラーを呼び出す方法と、z/OS の下でコンパイルするのに使用するジョブ制御ステートメントについて説明します。

プログラムをコンパイルするためには、言語環境プログラム SCEERUN データ・セットがコンパイラーからアクセス可能になっている必要があります。

z/OS UNIX の下でのコンパイラーの呼び出し

z/OS UNIX 環境でプログラムをコンパイルするには、**pli** コマンドを使用します。



command_line_option

次のようにして **command_line_option** を指定できます。

- **-q***option*
- オプション・フラグ (通常は前に - を付けた単一文字)

コマンド行でコンパイル時オプションを指定する場合のフォーマットは、%PROCESS ステートメントを使用してソース・ファイルにコンパイル時オプションを設定する場合のフォーマットと異なります。 184 ページの『z/OS UNIX の下でのコンパイル時オプションの指定』を参照してください。

input_file

プログラム・ファイル用の z/OS UNIX ファイル指定。

ファイル指定から拡張子が省略された場合、コンパイラーは拡張子 **.pli** を想定します。パス全体が省略された場合、コンパイラーは現行ディレクトリーを想定します。

入力ファイル

pli コマンドは PL/I ソース・ファイルをコンパイルし、結果のオブジェクト・ファイルを、コマンド行で指定されたすべてのオブジェクト・ファイルおよびライブラリーに指定順にリンクし、単一の実行可能ファイルを生成します。

pli コマンドが受け入れるファイルのタイプは次のとおりです。

ソース・ファイル – .pli

.pli ファイルはすべてコンパイル用のソース・ファイルです。 **pli** コマンドは、リストされた順にソース・ファイルをコンパイラーに送ります。 コンパイラーは、指定されたソース・ファイルを検出できない場合はエラー・メッセージを生成します。その場合、**pli** コマンドは、次のファイルがあればそのファイルを処理します。

© Copyright IBM Corp. 1999, 2017

すべての HFS ソース・ファイルは、行区切りで、かつ EBCDIC でエンコードされている必要があります。

PDS(E) にあるソース・ファイルをコンパイルすることもできます。例えば、PDS ファイル USER.SOURCE.PLI のメンバー SAMPLE をコンパイルする場合、次のコマンドを使用できます。

```
pli -c "//'USER.SOURCE.PLI(SAMPLE)'"
```

データ・セット名 'USER.SOURCE.PLI(SAMPLE)' の前に "// があり、後ろに " があります。

オブジェクト・ファイル – .o

.o ファイルはすべてオブジェクト・ファイルです。 **-c** オプションを指定しない限り、**pli** コマンドはリンク・エディット時にすべてのオブジェクト・ファイルおよびライブラリー・ファイルをリンケージ・エディターに送ります。すべてのソース・ファイルをコンパイルした後、コンパイラーはリンケージ・エディターを呼び出して、結果のオブジェクト・ファイルを、入力ファイル・リストに指定されているすべてのオブジェクト・ファイルにリンク・エディットし、単一の実行可能出力ファイルを作成します。

ライブラリー・ファイル – .a

pli コマンドは、リンク・エディット時にすべてのライブラリー・ファイル (.a ファイル) をリンケージ・エディターに送信します。

z/OS UNIX の下でのコンパイル時オプションの指定

Enterprise PL/I には、コンパイラーのデフォルト設定を変更するためのコンパイル時オプションが備わっています。コマンド行でオプションを指定することができます。指定したオプションは、ソース・プログラムの中の %PROCESS ステートメントにより指定変更されない限り、ファイルのすべてのコンパイル単位に有効です。

これらのオプションの詳細については、 3 ページの『コンパイル時オプションの説明』を参照してください。

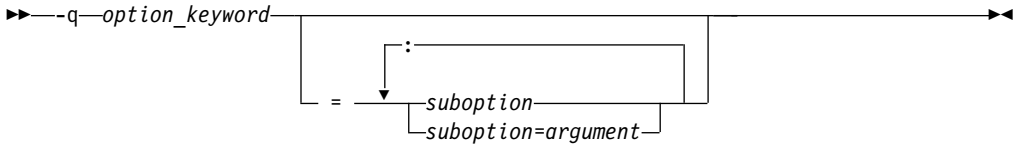
コマンド行で指定したオプションは、オプションのデフォルト設定を指定変更します。 指定したオプションは、ソース・ファイルで設定されているオプションにより指定変更されます。

コマンド行でコンパイル時オプションを指定するには、次の 3 つの方法があります。

- **-q***option_keyword* (コンパイラー特有)
- 単一フラグおよび複数文字フラグ
- -q+/u/myopts.txt

-qoption_keyword

-q*option* のフォーマットを使用して、コマンド行でオプションを指定できます。



同じコマンド行で複数の **-q***option* を指定できます。ただし、オプションをブランクで区切る必要があります。 オプション・キーワードは大文字または小文字のどちらでも指定できます。ただし、**-q** は小文字で指定しなければなりません。

一部のコンパイル時オプションではサブオプションを指定できます。 これらのサブオプションは、コマンド行では等号の後に **-q***option_keyword* を付けて示されます。複数のサブオプションは、ブランクを入れずにコロン (:) で区切らなければなりません。

複数のサブオプションを持つオプションの例として RULES があります。コマンド行で RULES(LAXDCL) を指定するには、次のコマンドを入力します。

```
-qrules=ibm:laxdcl
```

LIMITS オプションの場合は、そのサブオプションがそれぞれ引数をとるため、もう少し複雑になります。コマンド行で LIMITS(EXTNAME(31),FIXEDDEC(15)) を次の例にあるように指定できます。

```
-qlimits=extname=31:fixeddec=15
```

関連情報:

79 ページの『RULES』
RULES オプションを指定すると、ある種の言語機能を使用可能または 使用禁止にすることができ、代替の選択肢があればセマンティクスを選択できます。 これは一般プログラミング・エラーの診断に役立ちます。

51 ページの『LIMITS』
LIMITS オプションでは、各種のインプリメンテーションの制限を指定します。

単一フラグおよび複数文字フラグ

z/OS UNIX ファミリーのコンパイラーでは、共通の標準フラグをいくつも使用します。 各言語には、独自の追加フラグのセットがあります。

フラグ・オプションによっては、フラグの一部を形成する引数があります。次の例にある /home/test3/include は、組み込みファイルの検索場所となる組み込みディレクトリーです。

```
pli samp.pli -I/home/test3/include
```

それぞれのフラグ・オプションは、別々の引数として指定してください。

表 13. *z/OS UNIX* の下で *Enterprise PL/I* によりサポートされるコンパイル時オプション・フラグ

オプション	説明
-c	コンパイルのみ。
-e	フェッチ可能なロード・モジュールに名前とエントリーを作成します。
-I<*dir*>*	INCLUDE ファイルを検索するディレクトリーにパス <*dir*> を追加します。 -I の後にはパス名が必要です。1 つの -I オプションには 1 つのパスしか指定できません。複数のパスを追加するには、複数の -I オプションを使用します。 -I とパス名の間にスペースは入れないでください。
-0, -O2	生成されるコードを最適化します。 このオプションは -qOPT=2 と同等です。
-q<*option*>*	コンパイラーに渡します。 <*option*> はコンパイル時オプションです。各オプションはコンマで区切り、各サブオプションは等号またはコロンで区切ってください。**-q** と <*option*> の間にスペースは入れないでください。
-v	コンパイルおよびリンクのステップを表示し、それらのステップを実行します。
-#	コンパイルおよびリンクのステップを表示しますが、それらのステップを実行しません。

注: * 指示のある個所に引数を指定しなければなりません。そうしないと、予測できない結果となります。

JCL を使用した z/OS の下でのコンパイラーの呼び出し

コンパイラーを呼び出すジョブ・ステップに必要となる JCL ステートメントをすべて指定するわけではなくカタログ式プロシージャーを使用することになりますが、コンパイラーを可能な限り活用し、必要に応じてカタログ式プロシージャーのステートメントを指定変更できるように、JCL ステートメントに精通しておく必要があります。

いわゆる「バッチ・コンパイル」と呼ばれているもの (複数のオブジェクト・デックが 1 回で生成されるコンパイル) はサポートされていません。

また、BPXBATCH によるコンパイラーの呼び出しもサポートされていません。

以下のセクションでは、コンパイルに必要な JCL について説明します。 IBM 提供のカタログ式プロシージャー (167 ページの『IBM 提供のカタログ式プロシージャー』) には、これらのステートメントが入っています。 カタログ式プロシージャーを使用しない場合だけ、これらのステートメントを自分でコーディングする必要があります。

EXEC ステートメント

基本 EXEC ステートメントは `//stepname EXEC PGM` です。

このステートメントの REGION パラメーターには、512K が必要です。

最適化をオンにしてプログラムをコンパイルする場合は、必要な REGION サイズ (および時間) がはるかに大きくなることがあります。

EXEC ステートメントの PARM パラメーターを使用すれば、コンパイラーに備わったオプショナル機能のうちの 1 つ以上のものを指定することができます。 これらの機能は 191 ページの『EXEC ステートメントでのオプションの指定』で説明されています。 各種オプションの説明については、 3 ページの『第 1 章 コンパイラー・オプションと機能の使用』を参照してください。

標準データ・セット用の DD ステートメント

コンパイラーは標準データ・セットをいくつか要求します。データ・セットの数は、指定されたオプション機能によって異なります。そのようなデータ・セットは、DD ステートメントで定義する必要があります。

これらのデータ・セットは、標準 DD 名を持つ DD ステートメントに定義する必要があります。標準 DD 名は、他のデータ・セット特性と共に、表 14 に示されています。 DD ステートメント SYSIN、SYSUT1、および SYSPRINT は常に必要です。

標準データ・セットはどれも直接アクセス装置に保管できますが、DD ステートメントに SPACE パラメーターを組み込む必要があります。このパラメーターは、必要な補助記憶域の量を指定するためにデータ・セットを定義します。 IBM 提供のカタログ式プロシージャーに割り振られる補助記憶域の量は、大部分のアプリケーションにとって十分と思われます。

表 *14.* コンパイラーの標準データ・セット

標準 **DD** 名	データ・セットの内容	使用可能な装置クラス[1]	レコード・フォーマット **(RECFM)**	レコード・サイズ **(LRECL)**
SYSDEBUG	TEST(SEPARATE) 出力	SYSDA	F,FB	>=80 and <=1024
SYSDEFSD	XINFO(DEF) 出力	SYSDA	F,FB	128
SYSIN	コンパイラーへの入力	SYSSQ	F,FB,U	<101(100)
			VB,V	<105(104)
SYSLIB	INCLUDE ファイル用ソース・ステートメント	SYSDA	F,FB,U	<101
			V,VB	<105
SYSLIN	オブジェクト・モジュール	SYSSQ	FB	80
SYSPRINT	リスト (メッセージを含む)	SYSSQ	VBA	137 (MARGINS <= 100 の場合) 255 (MARGINS > 100 の場合)
SYSPUNCH	プリプロセッサー出力、コンパイラー出力	SYSSQ SYSCP	FB	80 または MARGINS() 値
SYSUT1	一時作業ファイル	SYSDA	F	4051
SYSUT2	一時作業ファイル	SYSDA	FB	3200

表 *14.* コンパイラーの標準データ・セット (続き)

標準 **DD** 名	データ・セットの内容	使用可能な装置クラス[1]	レコード・フォーマット **(RECFM)**	レコード・サイズ **(LRECL)**
SYSUT3	一時作業ファイル	SYSDA	FB	3200
SYSXMLSD	XINFO(XML) 出力	SYSDA	VB	16383
SYSADATA	XINFO(MSG) 出力	SYSDA	U	1024

1. 装置クラスの説明
   SYSSQ
   順次装置
   SYSDA
   直接アクセス装置

   SYSUT1 以外はブロック・サイズを指定できます。 SYSUT1 のブロック・サイズと論理レコード長はコンパイラーが選択します。

注:

1. 指定変更できるコンパイル時 SYSPRINT の値は BLKSIZE だけです。
2. SYSUT2 と SYSUT3 は、GONUMBER オプションまたは TEST オプションが有効になっている場合に限り、LP(64) でのみ必要となります。

入力 (SYSIN)

コンパイラーへの入力は、SYSIN という名前の DD ステートメントによって定義されたデータ・セットでなければなりません。

このデータ・セットは CONSECUTIVE 編成でなければなりません。入力は 1 つ以上の外部 PL/I プロシージャーでなければなりません。 単一のジョブまたはジョブ・ステップで複数の外部プロシージャーをコンパイルしたい場合は、各プロシージャーの前に %PROCESS ステートメントを置きます (最初のプロシージャーの前には不要な場合があります)。

PL/I ソース・プログラムの入力メディアとして、80 バイト・レコードが一般に使用されます。 入力データ・セットは直接アクセス装置または他の順次メディアに記録できます。入力データ・セットには、固定長レコード (ブロック化または非ブロック化のもの)、可変長レコード (コード化または非コード化のもの)、または不定長レコードのいずれでも入れることができます。 最大レコード・サイズは 100 バイトです。

入力ファイルの最大行数は 999999 です。

コンパイラーへの入力のためにデータ・セットを連結する場合は、連結されたデータ・セットは同じような特性 (例えば、ブロック・サイズやレコード・フォーマット) を持っていなければなりません。

出力 (SYSLIN、SYSPUNCH)

コンパイラーからの 1 つ以上のオブジェクト・モジュール形式の出力は、コンパイル時オプション OBJECT を指定した場合には、データ・セット SYSLIN に保管することができます。このデータ・セットは DD ステートメントで定義されます。

オブジェクト・モジュールは、常に、ブロック化または非ブロック化の 80 バイトの固定長レコードの形式になります。 BLKSIZE が SYSLIN に対して指定されていて 80 以外であれば、LRECL を 80 として指定しなければなりません。

SYSLIN DD は、一時データ・セットまたは永続データ・セットのいずれかを指定しなければなりません。 SYSLIN DD は、データ・セットのタイプに関係なく、連結したデータ・セットを指定できません。

SYSLIN DD は、PDS や PDSE ではなく、順次データ・セットを指定しなければなりません。

MDECK コンパイル時オプションを指定した場合、SYSPUNCH という名前の DD ステートメントで定義されるデータ・セットは、プリプロセッサーからの出力の保管にも使用されます。

一時作業ファイル (SYSUT1)

コンパイラーは、一時作業ファイルとして使用するためのデータ・セットを必要とします。このデータ・セットは、SYSUT1 という名前の DD ステートメントで定義され、予備ファイル と呼ばれます。 これは直接アクセス装置上にある必要があり、マルチボリューム・データ・セットとして割り振ってはなりません。

予備ファイルは、主記憶域の論理拡張部分として使用され、テキストと辞書情報を入れるためにコンパイラーとプリプロセッサーによって使用されます。 SYSUT1 の LRECL および BLKSIZE は、予備ファイル・ページに使用できるストレージの量に基づいて、コンパイラーが選択します。

本書で記述される DD ステートメントおよび SYSUT1 用のカタログ式プロシージャーの中の DD ステートメントは、1024 バイト・ブロック単位のスペース割り振りを要求します。 これは直接アクセス・ストレージ・スペースの適切な 2 次割り振りが取得されるようにするために行われます。

一時作業ファイル (SYSUT2、SYSUT3)

コンパイラーは、LP(64) オプションでプログラムをコンパイルするときに GONUMBER オプションまたは TEST オプションが有効になっていれば、一時データ・セットとして SYSUT2 および SYSUT3 を要求します。

プログラムが大きい場合に、SYSUT2 データ・セットまたは SYSUT3 データ・セットに使用可能なスペースが十分にないと、コンパイラーが異常終了する可能性があります。

リスト (SYSPRINT)

コンパイラーは、処理したすべてのソース・ステートメント、オブジェクト・モジュールに関連した情報、および、必要に応じてメッセージの入ったリストを生成します。

リストに入れられる情報の大半はオプショナル情報であり、適切なコンパイル時オプションを入れれば、ユーザーが必要とする情報部分を指定することができます。表示される可能性がある情報、および関連コンパイル時オプションについては、117 ページの『コンパイラー・リストの使用』を参照してください。

コンパイラーにそのリストを保管させるデータ・セットを、SYSPRINT という名前の DD ステートメントで指定する必要があります。 このデータ・セットは CONSECUTIVE 編成でなければなりません。リストは通常印刷されますが、任意の順次装置または直接アクセス装置に保管することもできます。 ご使用のシステムで出力クラス A がプリンターを指すきまりになっている場合は、印刷出力には次のステートメントで十分です。

```
//SYSPRINT DD SYSOUT=A
```

ソース・ステートメント・ライブラリー (SYSLIB)

%INCLUDE ステートメントを使用してライブラリーから PL/I プログラムにソース・ステートメントを取り込む場合は、SYSLIB という名前の DD ステートメントでライブラリーを定義するか、または独自の DD 名を選択して各 %INCLUDE ステートメントで DD 名を指定することができます。

DD ステートメントは、PDS または PDSE を指定しますが、実メンバーではありません。例えば、データ・セット INCLUDE.PLI を使用してライブラリー SYSLIB からファイル HEADER を組み込むには、以下のいずれかの %INCLUDE ステートメントを使用します。

- `%INCLUDE HEADER;`
- `%INCLUDE SYSLIB( HEADER );`

DD ステートメントは次のように指定します。

```
SYSLIB  DD DISP=SHR,DSN=INCLUDE.PLI
```

ただし、次のステートメントは無効です。

```
SYSLIB  DD DISP=SHR,DSN=INCLUDE.PLI(HEADER)
```

%INCLUDE ファイルはすべて、SYSIN ソース・ファイルのように、同じレコード・フォーマット (固定、可変、または未定義)、同じ論理レコード長を持ち、左右のマージンが一致するフォーマットのものでなければなりません。

ライブラリーの BLOCKSIZE は 32760 バイト以下でなければなりません。

どのインクルード・ファイルであれ、1 ファイル当たりの最大行数は 999999 です。

オプションの指定

それぞれのコンパイルごとに、%PROCESS ステートメントで、あるいは EXEC ステートメントの PARM パラメーターでオプションを指定して、デフォルトを指定変更した場合を除き、コンパイル時オプションとして IBM 提供のデフォルトまたはご使用のシステムのデフォルトが適用されます。

PARM パラメーターに指定されたオプションはデフォルト値を指定変更します。また、%PROCESS ステートメントで指定されたオプションは、PARM パラメーターで指定された値と、デフォルト値の両方を指定変更します。

注: 矛盾する属性が他のオプションの指定により明示的または暗黙的に指定された場合は、最新の暗黙的または明示的なオプションが受け入れられます。 このように指定変更されるオプションについては、診断メッセージは出されません。

EXEC ステートメントでのオプションの指定

EXEC ステートメントでオプションを指定するには、PARM= をコーディングし、その後にオプションのリストを指定します。オプションは任意の順序でリストできます。オプションはコンマで区切り、リストは単一引用符で囲む必要があります。

次の例を参照してください。

```
//STEP1 EXEC PGM=IBMZPLI,PARM='OBJECT,LIST'
```

MARGINI('c') のように引用符を伴うオプションの場合は、引用符を二重にする必要があります。オプション・リストの長さは、分離文字コンマを含めて 100 文字を超えてはなりません。ただし、使用可能であれば、オプションの省略構文を使用してスペースを節約できます。ステートメントが次行へ続くときには、オプション・リストを括弧 (引用符ではなく) で囲み、各行内のオプション・リストを引用符で囲み、そして、最終行を除く各行の最後のコンマが引用符の外側にくるようにしなければなりません。これらの点をすべて以下に例示します。

```
//STEP1 EXEC PGM=IBMZPLI,PARM=('AG,A',
//       'C,F(I)',
// 'M,MI(''X''),NEST,STG,X')
```

カタログ式プロシージャーを使用している場合に、オプションを明示的に指定するのであれば、以下のようにして、カタログ式プロシージャーを呼び出す EXEC ステートメントに PARM パラメーターを組み込み、コンパイラーを呼び出すプロシージャー・ステップの名前でキーワード PARM を修飾する必要があります。

```
//STEP1 EXEC nnnnnnn,PARM.PLI='A,LIST'
```

オプション・ファイルを使用した EXEC ステートメントでのオプションの指定

EXEC ステートメントでオプションを指定するもう 1 つの方法は、オプション・ファイルの中ですべてのオプションを宣言し、次のようにコーディングするというものです。

```
//STEP1 EXEC PGM=IBMZPLI,PARM='+DD:OPTIONS'
```

この方法を使うと、よく使用するオプションの整合性のあるセットを作成できます。 これは他のプログラマーに共通のオプション・セットを使ってもらいたい場合に特に効果的です。 また、100 文字の制限もなくなります。

MARGINS オプションは、オプション・ファイルには適用されません。1 桁目のデータは、オプションの一部として読み取られます。 また、ファイルが F フォーマットの場合、72 桁目より後のデータは無視されます。

parm ストリングには「通常」のオプションを使用でき、複数のオプション・ファイルを指定することもできます。 例えば、オプション LIST を指定し、さらに GROUP DD におけるファイルと PROJECT DD におけるファイルの両方からオプションを指定するには、次のように指定します。

```
PARM='LIST +DD:GROUP +DD:PROJECT'
```

PROJECT ファイルにおけるオプションは GROUP ファイルにおけるオプションより優先されます。

またこの例では、どちらかのオプション・ファイルに NOLIST オプションを指定することによって、LIST オプションをオフにすることもできます。 LIST オプションがオンになるようにするには、次のように指定します。

```
PARM='+DD:GROUP +DD:PROJECT LIST'
```

z/OS UNIX 環境では、オプション・ファイルも使用できます。例えば z/OS UNIX で、ファイル /u/pli/group.opt に含まれるオプションを使用して sample.pli をコンパイルするには、次のコマンドを使用します。

```
pli -q+/u/pli/group.opt  sample.pli
```

コンパイラーの旧リリースでは、オプション・ファイルの指定の前に付けるトリガー文字として、文字 '@' を使用していました。この文字は EBCDIC コード・ポイントのインバリアント・セットに含まれないため、文字 '+' (インバリアント) の使用をお勧めします。ただし、「@」文字は 16 進値 '7C'x を使用して指定される限りは引き続き使用できます。

第 5 章 31 ビット・プログラムに対するリンク・エディットおよび実行

LP(32) でのコンパイルが終わった 31 ビット・プログラムは、未解決の相互参照、および言語環境プログラム・ランタイム・ライブラリーに対する参照を含む 1 つ以上のオブジェクト・モジュールで構成されています。これらの参照は、リンク・エディット時 (静的) または実行時 (動的) に解決されます。

したがって、PL/I プログラムのコンパイルが終わった後で次にとるべきステップは、そのプログラムをテスト・データを用いてリンクして実行し、予想どおりの結果が出るかどうか確認することです。

言語環境プログラムは、ユーザーがプログラムを実行するのに必要なランタイム環境とサービスを提供します。PL/I およびその他すべての 言語環境プログラムに準拠した言語プログラムのリンクと実行については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。既存の PL/I プログラムから 言語環境プログラムへの移行については、「*Enterprise PL/I for z/OS* コンパイラーおよびランタイム 移行ガイド」を参照してください。

31 ビット・プログラムに関するリンク・エディットの考慮事項

オプション RENT またはオプション LIMITS(EXTNAME(*n*)) (*n* > 8) を使用してコンパイルを行う場合は、リンカー出力に PDSE を使用する必要があります。

31 ビット・プログラムでバインダーを使用

バインダー出力は PDSE 内に配置する必要があります。

DLL をリンクする際には、必要な定義サイド・デックをバインド・ステップの中で指定する必要があります。

ENTRY カードの使用

エントリー・ポイントとして Enterprise PL/I ルーチンを持つフェッチ予定のモジュールをビルドする場合は、その PL/I エントリー・ポイントの名前を ENTRY カードで指定する必要があります。

モジュールが Enterprise PL/I からフェッチされる場合、強くはお勧めしませんが、ENTRY カードに CEESTART を指定することができます。ただし、モジュールが COBOL またはアセンブラーからフェッチされる場合は、ENTRY カードには、モジュール内の PL/I エントリー・ポイントの名前を必ず指定するようにし、CEESTART を指定してはなりません。

© Copyright IBM Corp. 1999, 2017

31 ビット・プログラムに関する実行時の考慮事項

プログラム初期化ルーチンに渡されるパラメーターとしてランタイム・オプションを指定できます。また、PLIXOPT 変数の中でランタイム・オプションを指定できます。PLIXOPT 変数を使用してランタイム・オプションを指定し、既存のプログラムを変更してから再コンパイルするのも、パフォーマンスの観点から有利な場合があります。

PLIXOPT の使用法については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

端末の入出力を簡素化するため、端末に割り当てられるストリーム・ファイルには各種規則が取り入れられています。 次の 3 つの領域が影響を受けます。

1. PRINT ファイルのフォーマット設定
2. 自動プロンプト機能
3. 入力のスペーシングと句読法の規則

注**:** 端末でのレコード入出力にはプロンプトその他の機能は提供されていません。したがって、端末から、または端末への伝送にはストリーム入出力を使用することを強くお勧めします。

PRINT ファイルのフォーマット設定に関する規則

端末に PRINT ファイルが割り当てられる場合、そのファイルは印刷されるとおりに読み込まれると想定されています。 したがって、印刷時間を短縮するために、スペーシングは最小限に行われます。

PAGE、SKIP、および ENDPAGE キーワードには、次の規則が適用されます。

- PAGE オプションまたはフォーマット項目は、3 行スキップする原因となります。
- SKIP (2) より大きい SKIP オプションやフォーマット項目は、3 行スキップする原因となります。 SKIP (2) 以下は、通常の方法で処置されます。
- ENDPAGE 条件が発生することはありません。

PRINT ファイル上のフォーマットを 31 ビット・プログラム用に変更

端末で PRINT ファイルからの出力に通常のスペーシングを適合させたい場合は、PL/I 用に独自のタブ・テーブルを指定する必要があります。

以下の手順に従います。

1. メインプログラムにおいて、またはメインプログラムとリンクされたプログラムにおいて、外部構造 PLITABS を宣言します。
2. ご使用のページに収めることができる行数にエレメント PAGELENGTH を初期設定します。この値は PAGESIZE とは異なります。PAGESIZE は、ENDPAGE になる前にページに印刷したい行数を定義します (196 ページの図 18 を参照)。

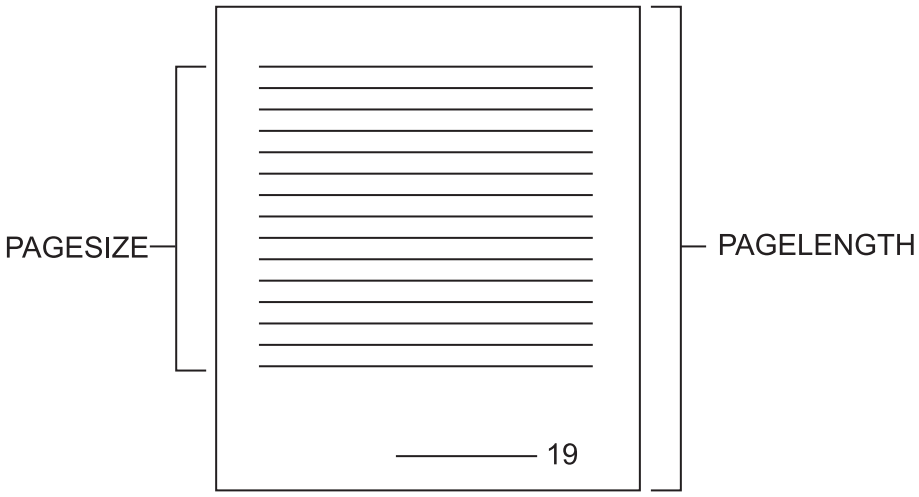
64 行の PAGELENGTH が必要な場合は、図 17 に示すように PLITABS を宣言します。 タブ・テーブルの指定変更の詳細については、 295 ページの『タブ制御テーブルの指定変更』を参照してください。

ご使用のコードに PLITABS の宣言が含まれる場合は、値と、PLITABS 構造体の先頭フィールドがすべて有効でなければなりません。このフィールドには、構造体が設定されるタブの数を指定しているフィールドへのオフセットが入っている必要があります。そうでない場合、Enterprise PL/I ライブラリー・コードは正しく機能しません。

RENT オプションを使用してコンパイルを行う場合、PLITABS を NONASGN 属性で宣言する必要があります。 NORENT によるコンパイル時に NONASGN 属性を指定することもできるため、PLITABS は常に NONASGN 属性で宣言するようお勧めします。

```
DCL 1 PLITABS STATIC EXTERNAL NONASGN,
  ( 2   OFFSET INIT (14),
    2    PAGESIZE INIT (60),
    2    LINESIZE INIT (120),
    2    PAGELENGTH INIT (64),
    2    FILL1 INIT (0),
    2    FILL2 INIT (0),
    2    FILL3 INIT (0),
    2    NUMBER_OF_TABS INIT (5),
    2    TAB1 INIT (25),
    2    TAB2 INIT (49),
    2    TAB3 INIT (73),
    2    TAB4 INIT (97),
    2    TAB5 INIT (121)) FIXED BIN (15,0);
```

図 *17. PLITABS* の宣言： これは標準ページ・サイズ、行サイズ、およびタブ位置を決める宣言です。



PAGELENGTH: 1 ページに印刷可能な行数

PAGESIZE: ENDPAGE 条件が生じるまでに 1 ページに印刷される行数

図 *18. PAGELENGTH* および *PAGESIZE*： PAGELENGTH は用紙のサイズを定義します。 PAGESIZE はメイン印刷域の行数を定義します。

自動プロンプト

プログラムは、端末に関連したファイルからの入力が必要になると、プロンプトを出します。 このプロンプトは、次の行にコロンを印刷してから、コロンの次の行の 1 桁目にスキップするという形をとります。 これにより、行全体を入力に使用できます。

次の例を参照してください。

```
:
(space for entry of your data)
```

この種のプロンプトを 1 次プロンプトと呼びます。

自動プロンプトの指定変更

1 次プロンプトを指定変更するには、データ要求の最後の項目をコロンにします。 2 次プロンプトは指定変更できません。

例えば、次の 2 つの PL/I ステートメントが実行されると、 197 ページの図 19 に示されている出力が端末に表示されます。

```
PUT SKIP EDIT ('ENTER TIME OF PERIHELION') (A);
GET EDIT (PERITIME) (A(10));
```

```
ENTER TIME OF PERIHELION
:         (automatic prompt)
(space for entry of data)
```

図 *19.* 自動プロンプトを使用した出力

ただし、次のように、最初のステートメントで出力の終わりにコロンがある場合、自動プロンプトは指定変更されます。端末に表示されるシーケンスを 図 20 に示します。

```
PUT EDIT ('ENTER TIME OF PERIHELION:') (A);
```

```
ENTER TIME OF PERIHELION: (space for entry of data)
```

図 *20.* 自動プロンプトを使用しない出力

注**:** 指定変更は 1 つのプロンプトにのみ有効です。 自動プロンプトを指定変更しない限り、次の項目に関するプロンプトが自動的に出されます。

長い入力行の句読法

端末で 2 行以上のスペース行を必要とするデータを 1 つのデータ項目として伝送するには、行継続文字として SBCS ハイフンを使用する必要があります。最後の行を除く各行の末尾に SBCS ハイフンを入力します。

例えば、`this data must be transmitted as one unit.` という文を伝送するには、次のようにデータを入力する必要があります。

```
:'this data must be transmitted -
+:as one unit.'
```

`unit.'` の後で ENTER を押すまで伝送は行われません。ハイフンは除去されます。 送信された項目は論理行 と呼ばれます。

注**:** 最後のデータ文字がハイフンまたは PL/I 負符号である行を送るには、行の終わりに 2 つのハイフンを入力し、次の行をヌル行にします。 次の例を参照してください。

```
xyz--
(press ENTER only on this line)
```

GET LIST ステートメントと GET DATA ステートメントの句読法

GET LIST ステートメントと GET DATA ステートメントの場合、プログラマーがコンマを省略すると、端末から送信される各論理行の終わりに、コンマが追加されます。 したがって、項目を別々の論理行に入力する場合は、項目を区切るためのブランクまたはコンマを入力する必要はありません。

PL/I ステートメント `GET LIST(A,B,C);` が指定された場合、端末で入力できるデータは次のとおりです。

```
:1
+:2
+:3
```

この規則は、文字ストリング・データの入力にも適用されます。したがって、1 つの文字ストリングは 1 つの論理行として送信する必要があります。 そうしないと、ブレークポイントにコンマが置かれます。 例えば、以下のデータを入力するとします。

```
:'COMMAS SHOULD NOT BREAK
+:UP A CLAUSE.'
```

結果のストリングは COMMAS SHOULD NOT BREAK, UP A CLAUSE. となります。

行継続文字としてハイフンが使用された場合、コンマは追加されません。

GET EDIT での自動埋め込み

GET EDIT ステートメントに関しては、行の終わりにブランクを入力する必要はありません。データは指定された長さまで埋め込まれます。

例えば、PL/I ステートメント GET EDIT (NAME) (A(15)); が指定された場合に SMITH という 5 文字を入力すると、プログラムが受け取る文字の数が 15 文字になるようにデータに 10 個のブランクが埋め込まれます。

```
'SMITH          '
```

注: 単一のデータ項目は 1 つの論理行として送信する必要があります。 そうしないと、送信される最初の行に必要なブランクが埋め込まれ、完了したデータ項目と見なされます。

端末入力での SKIP の使用

入力での SKIP の使用は、すべてファイルが端末に割り振られるときに SKIP(1) として解釈されます。 SKIP(1) は、現在使用可能な論理行にある未使用データをすべて無視するという命令として扱われます。

ENDFILE

端末でファイルの終わりを入力できます。それには、2 つの文字 /* からなる論理行を入力します。

以後、クローズせずにファイルを使用しようとすると、ENDFILE 条件になります。

31 ビット・プログラムに関する SYSPRINT の考慮事項

PL/I 標準 SYSPRINT ファイルは、アプリケーション内で複数のエンクレーブによって共用されます。 同じあるいは異なるエンクレーブから、例えば STREAM PUT などの入出力要求を出すことができます。 これらの要求は標準 PL/I SYSPRINT ファイルを、全アプリケーション共通のファイルとして使用して処理されます。 SYSPRINT ファイルは、エンクレーブの終了時ではなく、アプリケーションが終了するときにのみ暗黙的に閉じられます。

標準 PL/I SYSPRINT ファイルには、STREAM PUT などのユーザー開始出力のみが入っています。 ランタイム・ライブラリー・メッセージおよび他の類似診断出力は、言語環境プログラム MSGFILE へ向けられます。SYSPRINT ファイル出力を

言語環境プログラム MSGFILE にリダイレクトする処理の詳細については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

アプリケーション内で複数のエンクレーブによって共用されるためには、PL/I SYSPRINT ファイルは SYSPRINT のファイル名で EXTERNAL FILE 定数として宣言されなければならず、また属性 STREAM および OUTPUT ならびに暗黙の PRINT(OPEN 処理されるとき) も持たなくてはなりません。 これはコンパイラーによってデフォルトとされる標準 SYSPRINT ファイルです。

アプリケーション内にはただ 1 つの標準 PL/I SYSPRINT FILE が存在し、このファイルはそのアプリケーション内の全エンクレーブによって共用されます。 例えば、SYSPRINT ファイルはアプリケーション内の多重ネスト・エンクレーブによる共用が可能であり、また、言語環境プログラム事前初期化機能によってアプリケーション内で作成され終了される一連のエンクレーブによる共用が可能です。 アプリケーション内でエンクレーブによって共用されるためには、PL/I SYSPRINT ファイルはそのエンクレーブ内で宣言されなくてはなりません。 標準 SYSPRINT ファイルは、エンクレーブ間でファイル引数としてそれを渡すことによって共用することはできません。標準 SYSPRINT ファイルの宣言済み属性は、アプリケーション内では EXTERNAL として宣言された定数によるのと同様、同じでなくてはなりません。 PL/I はこの規則を強制しません。 TITLE オプションと MSGFILE(SYSPRINT) オプションは、どちらも SYSPRINT を別のデータ・セットへ経路指定しようとします。 したがって、この 2 つのオプションを一緒に使用すると、矛盾が起こり、TITLE オプションは無視されます。

共通 SYSPRINT ファイルをアプリケーション内で持つことは、互いに緊密に結ばれたエンクレーブを利用するアプリケーションにとっては利点となります。しかし、アプリケーション内のすべてのエンクレーブは、同じ共用データ・セットに書き込みを行うので、各エンクレーブ間での何らかの調整が必要となります。

SYSPRINT ファイルは、アプリケーションのエンクレーブ内で最初に参照が行われた時にオープンされます (暗黙的にあるいは明示的に)。 SYSPRINT ファイルが CLOSE 処理されると、ファイル・リソースは解放され (あたかもファイルがオープンされていなかったかのように)、また全エンクレーブは閉じられた状況を反映するように更新されます。

SYSPRINT が複数のエンクレーブ・アプリケーション内で利用される場合は、LINENO 組み込み関数はエンクレーブ内の最初の PUT あるいは OPEN が出されるまで現在の行番号を返すだけです。 これは旧プログラムとの完全な互換性を維持するために必要です。

COUNT 組み込み関数はエンクレーブ・レベルにおいて維持されます。これは常に、エンクレーブ内の最初の PUT が出されるまでゼロの値を返します。 ネスト化された子エンクレーブが親エンクレーブから呼び出される場合、COUNT 組み込み関数の値は、親エンクレーブが子エンクレーブから制御を取り戻したときは未定義です。

TITLE オプションを使用して、標準 SYSPRINT ファイルを異なるオペレーティング・システム・データ・セットと関連付けることができますが、特定のオープン関

連付けは、別のものをオープンする前にクローズする必要があることに留意してください。 この関連付けはオープン状態が続く間は各エンクレーブ間で保持されます。

標準 PL/I SYSPRINT ファイルと関連する PL/I 条件処理は、その現行のセマンティクスと有効範囲を保持します。 例えば、子エンクレーブ内で生じた ENDPAGE 条件は、その子エンクレーブ内で、設定された ON ユニットを呼び出すだけです。これは親エンクレーブ内の ON ユニットの呼び出しは行いません。

標準 PL/I SYSPRINT ファイルのタブは、エンクレーブがユーザー PLITABS テーブルを含む場合、PUT が異なるエンクレーブから実行されたときは変わる可能性があります。

PL/I SYSPRINT ファイルが RECORD ファイルあるいは STREAM INPUT ファイルとして利用される場合、PL/I は個々のエンクレーブあるいはタスク・レベルでそれをサポートしますが、エンクレーブ間の共用可能ファイルとしてはサポートしません。 同じアプリケーションの異なるエンクレーブ内の異なるファイル属性 (例えば RECORD および STREAM) において PL/I SYSPRINT ファイルがオープンされると、結果は予測不能のものとなります。

SYSPRINT は、Enterprise PL/I コンパイラーによってコンパイルされたコードと、以前の PL/I コンパイラーによってコンパイルされたコードとの間で共用することもできます。ただし、そのためには以下の条件がすべて当てはまらなければなりません。

- SYSPRINT は STREAM OUTPUT として宣言されなければならない。
- アプリケーションを TSO 環境で実行することはできない。
- ランタイム・オプション MSGFILE(SYSPRINT) が有効になっている場合、アプリケーション内に、事前初期設定されたプログラムおよびストアード・プロシージャーが存在していてはいけません。

MSGFILE(SYSPRINT) の使用

SYSPRINT STREAM PRINT のファイル宣言の ENVIRONMENT オプションに指定されたファイル属性はすべて無視されます。

SYSPRINT の OPEN ステートメントに指定された属性はすべて無視されます。

OPEN ステートメントを使用して `PL/I SYSPRINT STREAM PRINT` ファイルを開くと、PL/I 制御ブロック内でそのファイルに `opened` のマークが付けられますが、実際には言語環境プログラムによって開かれます。

CLOSE ステートメントを使用して `PL/I SYSPRINT STREAM PRINT` ファイルを閉じると、PL/I 制御ブロック内でそのファイルに `closed` のマークが付けられますが、言語環境プログラムはそのファイルを開いたままにします。

言語環境プログラムメッセージと PL/I ユーザー指定出力の間で同期化は行われないため、出力の順序は予測不能です。

MSGFILE(SYSPRINT) を使用すると、LINESIZE オプションで指定される行サイズは最大 225 文字に制限されます。

31 ビット・アプリケーションで自分のルーチンにおいて FETCH を使用

Enterprise PL/I では、PL/I、C、COBOL、またはアセンブラーによってコンパイルされたルーチンをフェッチできます。

31 ビット・アプリケーションで Enterprise PL/I ルーチンをフェッチ

旧 PL/I コンパイラーによって課せられた、フェッチされたモジュールの制限は、ほぼすべて除去されました。 そのため、現在、フェッチされたモジュールは以下の操作を実行できます。

- 他のモジュールをフェッチする。
- 任意の PL/I ファイルに入出力操作を実行する。 ファイルは、フェッチされたモジュール、メイン・モジュール、または他のフェッチされたモジュールによりオープンできます。
- 独自の CONTROLLED 変数の ALLOCATE と FREE を実行する。

しかし、フェッチの対象である Enterprise PL/I モジュールに対するいくつかの制限があります。

1. リンク・エディット・ステップで ENTRY カードが指定されていない場合は、フェッチされたルーチンの PROCEDURE ステートメントで OPTIONS(FETCHABLE) を指定するようにしてください。
2. ENTRY カードには、PL/I エントリー・ポイントの名前を指定する必要があります。
   - モジュールが Enterprise PL/I からフェッチされる場合、強くはお勧めしませんが、ENTRY カードに CEESTART を指定することができます。
   - ただし、モジュールが COBOL またはアセンブラーからフェッチされる場合は、ENTRY カードには、モジュール内の PL/I エントリー・ポイントの名前を必ず指定するようにし、CEESTART を指定してはなりません。
3. フェッチされる側のコードのいずれかをコンパイルするために RENT コンパイラー・オプションが使用された場合、そのモジュールは DLL としてリンクする必要があります。
4. フェッチする側のコードをコンパイルするために NORENT コンパイラー・オプションが使用された場合、フェッチされる側のモジュールは少なくとも以下のいずれかの条件を満たさなければなりません。
   - MAIN モジュールである。
   - NORENT コードのみから成る。
   - NORENT オプションが有効になっている C または Enterprise PL/I コンパイラーでコンパイルされたエントリー・ポイント・コードを持つ。この場合は、RENT オプションでコンパイルされたコードもモジュールに含まれている可能性がありますが、そのコードの呼び出しはサポートされていません。
5. フェッチする側のコードのコンパイルに RENT コンパイラー・オプションが使用された場合、FETCH される側の ENTRY が、フェッチする側のモジュールに OPTIONS(COBOL) または OPTIONS(ASM) として宣言されていてはなりま

せん。この状態で記述子を渡すのを避けたい場合には、ENTRY 宣言に OPTIONS(NODESCRIPTOR) 属性を指定する必要があります。

6. Enterprise PL/I ルーチンは自らをフェッチすることはできません。

   再入不可で再使用不可のモジュールが何度もロードされる場合、処理の順序は後入れ先出し法の順序になります。

   例えば、プログラム A がモジュール LOADMODA をロードしてからプログラム B を呼び出すとします。プログラム B も LOADMODA をロードし、LOADMODA に対して DELETE を発行するとします。この場合、削除される LOADMODA のコピーは、プログラム B に関連付けられているものです。この時点で、プログラム A に関連付けられている LOADMODA のコピーは依然として存在します。

   つまり、LOADMODA に対して要求された DELETE によって削除されるのは、最後にロードされたコピーです。要求を発行したプログラムは関係ありません。

NORENT WRITABLE コードは逐次使用可能です。そのため、FETCHABLE 定数を示すために使用されるポインターは、すべての NORENT WRITABLE ルーチンのプロローグ・コードにおいてゼロにリセットされます。これによって、コードが逐次再使用可能なコードであると同時に、正しい PL/I セマンティクスを提供することにもなりますが、NORENT WRITABLE コードにおいて TITLE を指定する FETCH の使用に制限が課されます。この制限によって、FETCH A TITLE('B') を実行したルーチンは終了して再入する場合に、FETCH A TITLE('B') を再実行してから CALL A ステートメントを実行しなければならなくなります (さもないと、CALL の実行前に暗黙の (TITLE のない) A の FETCH が実行されます)。

これらの制限を説明するために、コンパイラー・ユーザー出口を考えてみます。EXIT コンパイル時オプションを指定すると、コンパイラーは IBMUEXIT という名前の Enterprise PL/I モジュールをフェッチして呼び出します。

まず、RENT オプションを指定してコンパイラー・ユーザー出口をコンパイルする必要があるので注意してください。コンパイラーは、このユーザー出口が DLL であることを前提としているからです。

上の項目 1 により、このルーチンに関するコンパイラーの PROCEDURE ステートメントは次のようになります。

```
ibmuexit:
  proc ( addr_Userexit_Interface_Block,
         addr_Request_Area )
  options( fetchable  );

  dcl addr_Userexit_Interface_Block  pointer byvalue;

  dcl addr_Request_Area              pointer byvalue;
```

上の項目 3 により、ユーザー出口を DLL にリンクするときに、リンカー・オプション DYNAM=DLL を指定する必要があります。 DLL は PDSE または一時データ・セットのいずれかにリンクされなければなりません (一時データ・セットの場合は、DSNTYPE=LIBRARY を SYSLMOD DD ステートメントで指定する必要があります)。

ユーザー出口のコンパイル、リンク、および呼び出しを行うための JCL ステートメントはすべて、 204 ページの図 21 の JCL に示されています。 下記のサンプル・コードでは、フェッチされたユーザー出口は、構造体を指す 2 つの BYVALUE ポインターを受け取らず、代わりに 2 つの構造体を参照 (BYADDR) によって受け取ります。これが上のコード抜粋との大きな相違です。この変更を有効にするために、コードではその PROCEDURE ステートメントのそれぞれに OPTIONS(NODESCRIPTOR) を指定しています。

```
//*
//*********************************************************************
//* compile the user exit
//*********************************************************************
//PLIEXIT EXEC PGM=IBMZPLI,
//STEPLIB   DD DSN=IBMZ.V5R2M0.SIBMZCMP,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSLIN    DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(CYL,(3,1))
//SYSUT1    DD DSN=&&SYSUT1,UNIT=SYSDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//SYSIN     DD *
*Process or('|') not('!');
*Process limits(extname(31));
*Process RENT;

 /*********************************************************************/
 /*                                                                   */
 /*  NAME - IBMUEXIT.PLI                                              */
 /*                                                                   */
 /*  DESCRIPTION                                                      */
 /*    User-exit sample program.                                      */
 /*                                                                   */
 /*    Licensed Materials - Property of IBM                           */
 /*    Copyright IBM Corp. 1999, 2017 All Rights Reserved             */
 /*    All Rights Reserved.                                           */
 /*    US Government Users Restricted Rights-- Use, duplication or    */
 /*    disclosure restricted by GSA ADP Schedule Contract with        */
 /*    IBM Corp.                                                      */
 /*                                                                   */
 /*  DISCLAIMER OF WARRANTIES                                         */
 /*    The following "enclosed" code is sample code created by IBM    */
 /*    Corporation. This sample code is not part of any standard      */
 /*    IBM product and is provided to you solely for the purpose of   */
 /*    assisting you in the development of your applications.  The    */
 /*    code is provided "AS IS", without warranty of any kind.        */
 /*    IBM shall not be liable for any damages arising out of your    */
 /*    use of the sample code, even if IBM has been advised of the    */
 /*    possibility of such damages.                                   */
 /*                                                                   */
 /*********************************************************************/

 /*********************************************************************/
 /*                                                                   */
 /*  During initialization, IBMUEXIT is called.  It reads             */
 /*  information about the messages being screened from a text        */
 /*  file and stores the information in a hash table.  IBMUEXIT       */
 /*  also sets up the entry points for the message filter service     */
 /*  and termination service.                                         */
 /*                                                                   */
 /*  For each message generated by the compiler, the compiler         */
 /*  calls the message filter registered by IBMUEXIT.  The filter     */
 /*  looks the message up in the hash table previously created.       */
 /*                                                                   */
 /*  The termination service is called at the end of the compile      */
 /*  but does nothing.  It could be enhanced to generate reports     */
 /*  or do other cleanup work.                                        */
 /*                                                                   */
 /*********************************************************************/
```

図 *21.* ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL*

```
pack: package exports(*);

  Dcl
    1 Uex_UIB           native based,
      2 Uex_UIB_Length         fixed bin(31),

      2 Uex_UIB_Exit_token     pointer,        /* for user exit's use*/

      2 Uex_UIB_User_char_str  pointer,        /* to exit option str */
      2 Uex_UIB_User_char_len  fixed bin(31),

      2 Uex_UIB_Filename_str   pointer,        /* to source filename */
      2 Uex_UIB_Filename_len   fixed bin(31),

      2 Uex_UIB_return_code fixed bin(31),     /* set by exit procs  */
      2 Uex_UIB_reason_code fixed bin(31),     /* set by exit procs  */

      2 Uex_UIB_Exit_Routs,                    /* exit entries setat
                                                  initialization     */
        3 ( Uex_UIB_Termination,
            Uex_UIB_Message_Filter,            /* call for each msg  */
            *, *, *, * )
          limited entry (
            *,                                 /* to Uex_UIB         */
            *                                  /* to a request area  */
           );

  /*********************************************************************/
  /*                                                                   */
  /*   Request Area for Initialization exit                            */
  /*                                                                   */
  /*********************************************************************/

  Dcl 1 Uex_ISA native based,
        2 Uex_ISA_Length fixed bin(31);

  /*********************************************************************/
  /*                                                                   */
  /*   Request Area for Message_Filter exit                            */
  /*                                                                   */
  /*********************************************************************/

  Dcl 1 Uex_MFA native based,
        2 Uex_MFA_Length   fixed bin(31),
        2 Uex_MFA_Facility_Id  char(3),
        2 *                    char(1),
        2 Uex_MFA_Message_no   fixed bin(31),
        2 Uex_MFA_Severity     fixed bin(15),
        2 Uex_MFA_New_Severity fixed bin(15);  /* set by exit proc   */

  /*********************************************************************/
  /*                                                                   */
  /*   Request Area for Terminate exit                                 */
  /*                                                                   */
  /*********************************************************************/

  Dcl 1 Uex_TSA native based,
        2 Uex_TSA_Length fixed bin(31);
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL (*続き*)*

```
/*******************************************************************/
/*                                                                 */
/*   Severity Codes                                                */
/*                                                                 */
/*******************************************************************/

dcl uex_Severity_Normal             fixed bin(15) value(0);
dcl uex_Severity_Warning            fixed bin(15) value(4);
dcl uex_Severity_Error              fixed bin(15) value(8);
dcl uex_Severity_Severe             fixed bin(15) value(12);
dcl uex_Severity_Unrecoverable      fixed bin(15) value(16);

/*******************************************************************/
/*                                                                 */
/*   Return Codes                                                  */
/*                                                                 */
/*******************************************************************/

dcl uex_Return_Normal               fixed bin(15) value(0);
dcl uex_Return_Warning              fixed bin(15) value(4);
dcl uex_Return_Error                fixed bin(15) value(8);
dcl uex_Return_Severe               fixed bin(15) value(12);
dcl uex_Return_Unrecoverable        fixed bin(15) value(16);

/*******************************************************************/
/*                                                                 */
/*   Reason Codes                                                  */
/*                                                                 */
/*******************************************************************/

dcl uex_Reason_Output               fixed bin(15) value(0);
dcl uex_Reason_Suppress             fixed bin(15) value(1);

dcl hashsize fixed bin(15) value(97);
dcl hashtable(0:hashsize-1) ptr init((hashsize) sysnull());

dcl 1 message_item native based,
      2 message_Info,
        3 facid  char(3),
        3 msgno  fixed bin(31),
        3 newsev fixed bin(15),
        3 reason fixed bin(31),
      2 link pointer;
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL (続き)*

```
ibmuexit: proc ( ue, ia  )
  options( fetchable nodescriptor );

  dcl 1 ue like uex_Uib byaddr;
  dcl 1 ia like uex_Isa byaddr;

  dcl sysuexit     file stream input env(recsize(80));
  dcl p            pointer;
  dcl bucket       fixed bin(31);
| dcl based_Chars  char(31) based;
| dcl title_Str    char(31) var;

  ue.uex_Uib_Message_Filter = message_Filter;
  ue.uex_Uib_Termination = exitterm;


  on undefinedfile(sysuexit)
  begin;
    put edit ('** User exit unable to open exit file ')
             (A) skip;
    put skip;
    signal error;
  end;

  if ue.uex_Uib_User_Char_Len = 0 then
    do;
      open file(sysuexit);
    end;
  else
    do;
      title_Str
       = substr( ue.uex_Uib_User_Char_Str->based_Chars,
                 1, ue.uex_Uib_User_Char_Len );
      open file(sysuexit) title(title_Str);
    end;

  on error, endfile(sysuexit)
    goto done;

  allocate message_item set(p);

  /*****************************************************************/
  /*                                                               */
  /*  Skip header lines and read first data line                   */
  /*                                                               */
  /*****************************************************************/

  get file(sysuexit) list(p->message_info) skip(3);
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL (*続き*)*

```
    do loop;

      /*************************************************************/
      /*                                                           */
      /*  Put message information in hash table                    */
      /*                                                           */
      /*************************************************************/

      bucket = mod(p->msgno, hashsize);
      p->link = hashtable(bucket);
      hashtable(bucket) = p;

      /*************************************************************/
      /*                                                           */
      /*  Read next data line                                      */
      /*                                                           */
      /*************************************************************/

      allocate message_item set(p);
      get file(sysuexit) skip;
      get file(sysuexit) list(p->message_info);

    end;


    /*****************************************************************/
    /*                                                               */
    /*  Clean up                                                     */
    /*                                                               */
    /*****************************************************************/

    done:

    free p->message_Item;
    close file(sysuexit);

  end;

  message_Filter:
    proc ( ue, mf  )
    options( nodescriptor );

    dcl 1 ue like uex_Uib byaddr;
    dcl 1 mf like uex_Mfa byaddr;

    dcl p pointer;
    dcl bucket fixed bin(15);

    on error snap system;

    ue.uex_Uib_Reason_Code = uex_Reason_Output;
    ue.uex_Uib_Return_Code = 0;

    mf.uex_Mfa_New_Severity = mf.uex_Mfa_Severity;


    /*****************************************************************/
    /*                                                               */
    /*  Calculate bucket for error message                           */
    /*                                                               */
    /*****************************************************************/

    bucket = mod(mf.uex_Mfa_Message_No, hashsize);
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル ***JCL (続き)***

```
    /*************************************************************/
    /*                                                           */
    /*  Search bucket for error message                          */
    /*                                                           */
    /*************************************************************/

    do p = hashtable(bucket) repeat (p->link) while(p!=sysnull())
      until (p->msgno = mf.uex_Mfa_Message_No &
             p->facid = mf.Uex_Mfa_Facility_Id);
    end;


    if p = sysnull() then;
    else
      do;

        /*********************************************************/
        /*                                                       */
        /*  Filter error based on information in has table       */
        /*                                                       */
        /*********************************************************/

        ue.uex_Uib_Reason_Code = p->reason;
        if p->newsev < 0 then;
        else
          mf.uex_Mfa_New_Severity = p->newsev;
      end;
  end;


  exitterm:
    proc ( ue, ta  )
    options( nodescriptor );

    dcl 1 ue like uex_Uib byaddr;
    dcl 1 ta like uex_Tsa byaddr;

    ue.uex_Uib_return_Code = 0;
    ue.uex_Uib_reason_Code = 0;

  end;

end pack;

//*********************************************************************
//* link the user exit
//*********************************************************************
//LKEDEXIT EXEC PGM=IEWL,PARM='XREF,LIST,LET,DYNAM=DLL',
//    COND=(9,LT,PLIEXIT),REGION=5000K
//SYSLIB   DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=&&EXITLIB(IBMUEXIT),DISP=(NEW,PASS),UNIT=SYSDA,
//             SPACE=(TRK,(7,1,1)),DSNTYPE=LIBRARY
//SYSUT1   DD DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(CYL,(3,1)),
//            DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=X
//SYSDEFSD DD DUMMY
//SYSLIN    DD DSN=&&LOADSET,DISP=SHR
//         DD DDNAME=SYSIN
//LKED.SYSIN     DD *
  ENTRY IBMUEXIT
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL (*続き*)*

```
//*********************************************************************
//* compile main
//*********************************************************************
//PLI EXEC PGM=IBMZPLI,PARM='F(I),EXIT',
//            REGION=256K
//STEPLIB   DD  DSN=&&EXITLIB,DISP=SHR
//          DD DSN=IBMZ.V5R2M0.SIBMZCMP,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSLIN    DD DSN=&&LOADSET2,DISP=(MOD,PASS),UNIT=SYSSQ,
//             SPACE=(CYL,(3,1))
//SYSUT1    DD DSN=&&SYSUT1,UNIT=SYSDA,
//             SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
//SYSIN     DD *
*process;
 MainFet: Proc Options(Main);
   /* the exit will suppress the message for the next dcl */
   dcl one_byte_integer  fixed bin(7);
 End ;
//*
//SYSUEXIT DD  DISP=SHR,DSN=hlq.some.user.exit.input.file *
  Fac Id   Msg No   Severity   Suppress   Comment
+--------+--------+----------+----------+-------------------------------
  'IBM'     1042       -1          1      String spans multiple lines
  'IBM'     1044       -1          1      FIXED BIN 7 mapped to 1 byte
```

ユーザー出口のコンパイル、リンク、および呼び出しのためのサンプル *JCL (続き)*

31 ビット・アプリケーションで PL/I MAIN ルーチンをフェッチ

Enterprise PL/I アプリケーションでは、PL/I MAIN プログラムをフェッチすることもできます。PL/I MAIN プログラムの FETCH が行われると、子エンクレーブが作成されます。

以下の規則に従う必要があります。

- パラメーター・ストリング内のフェッチされた MAIN プログラムには、ランタイム・オプションを渡すことはできません。
- SYSTEM(MVS) コンパイラー・オプションが指定されている場合は、任意のパラメーター・リストを渡すことができますが、このパラメーターが単一の CHAR VARYING ストリングではないと、コンパイラーが MAIN ルーチンに警告メッセージでフラグを立てます。
- フェッチするプログラム内のフェッチされた MAIN ルーチンに対する ENTRY 宣言で、 OPTIONS(ASM) または OPTIONS(NODESCRIPTOR) を指定してはなりません。
- ランタイム・オプションは渡さないでください。ランタイム・オプションを解析しようとすると、無効なランタイム・オプションに関する LE 通知メッセージが生成されることがあるためです。フェッチされた MAIN ルーチンで NOEXECOPS が指定されている場合、渡された CHAR VARYING ストリングはランタイム・オプションに対して解析されません。
- フェッチされる MAIN プログラムにパラメーターが渡されない場合は、フェッチする側のプログラムが、フェッチされる MAIN ルーチンの ENTRY 宣言で OPTIONS(LINKAGE(SYSTEM)) を指定するか、または DEFAULT(LINKAGE (SYSTEM)) でコンパイルされなければなりません。

例

PL/I フェッチ MAIN プログラムのサンプルを以下に示します。

```
FMAIN: proc(parm) options(main,noexecops );
  DCL parm char(*) var;
  DCL SYSPRINT print;
  DCL PLIXOPT CHAR(11) VAR INIT('RPTOPTS(ON)')
      STATIC EXTERNAL;
  Put skip list("FMAIN parm: "|| parm);
  Put skip list("FMAIN finished ");
End FMAIN;
```

別の PL/I MAIN プログラムをフェッチする PL/I MAIN プログラムのサンプルを以下に示します。

```
MainFet: Proc Options(main);
    Dcl Parm char(1000) var;
    Dcl SYSPRINT print;
    Dcl Fmain entry(char(*) var) ;
    Put skip list("MainFet: start ");
    Parm = 'local-parm';
    Put skip list("MainFet parm: "|| Parm);
    Fetch Fmain;
    Call Fmain(Parm);
    Release Fmain;
    Put skip list("MainFet:testcase finished ");
End;
```

31 ビット・アプリケーションで z/OS C ルーチンをフェッチ

NORENT オプションが指定されている場合を除き、z/OS C ルーチンをフェッチするルーチン内の ENTRY 宣言では、OPTIONS(COBOL) または OPTIONS(ASM) は指定できません。これらは DLL としてリンクされていない COBOL ルーチンまたは ASM ルーチンだけに指定してください。

z/OS C DLL のコンパイルとリンクの方法についての指示は、z/OS C 資料にあります。

31 ビット・アプリケーションでアセンブラー・ルーチンをフェッチ

NORENT オプションが指定されている場合を除き、アセンブラー・ルーチンをフェッチするルーチン内の ENTRY 宣言は、OPTIONS(ASM) を指定する必要があります。

データ専用アセンブラー・モジュールをフェッチするときは、FETCH A SET(P) 構成を使用して、ポインター P を正しく設定する必要があります。

TSO/E のもとでの MAIN の呼び出し

SYSTEM(MVS) オプションで MAIN プログラムをコンパイルする場合は、TSO CALL コマンドを使用するか、または TSO コマンド・プロセッサーとしてプログラムを呼び出すことができます。ランタイム・オプションとパラメーターの両方を、MVS バッチ配下と同じ方法で渡すことができます。

例えば、TSOARG1 という MAIN プログラムが userid.TEST.load というデータ・セットのメンバーとしてリンク・エディットされた場合は、次のものによってこのプログラムを呼び出すことができます。

```
  CALL TEST(TSOARG1) 'RPTSTG(ON),TRAP(ON)/THIS IS MY ARGUMENT'
または
  CALL TEST(TSOARG1) '/THIS IS MY ARGMENT'
または
  TSOARG1 TRAP(ON)/THIS IS MY ARGUMENT
または
  TSOARG1 /THIS IS MY ARGUMENT
```

注: CALL ステートメントを使用しないでプログラムを実行するためには、TSOARG1 を含むデータ・セット (userid.TEST.load) が標準 TSO プログラム検索リストになければなりません。これは、次の TSO コマンドを発行することによって達成できます。

```
TSOLIB ACTIVATE DSN('userid.TEST.load')
```

ただし、MAIN プログラムを SYSTEM(TSO) オプションでコンパイルすると、コマンド・プロセッサー・パラメーター・リスト (CPPL) を指すポインターがプログラムに渡されます。 この場合は、NOEXECOPS が有効です。 プログラムを TSO コマンドとして呼び出せますが、TSO CALL ステートメントによって呼び出すことはできません。 次の例を参照してください。

```
TSOARG2 This is my argument
```

図 22 のプログラムは、SYSTEM(TSO) インターフェースを使用して、CPPL からプログラム実引数を指定して表示します。

```
*process system(tso);
 tsoarg2: proc (cppl_ptr) options(main);
   dcl cppl_ptr pointer;
   dcl 1 cppl based(cppl_ptr),
     2 cpplcbuf   pointer,
     2 cpplupt    pointer,
     2 cpplpscb   pointer,
     2 cpplect    pointer;
   dcl 1 cpplbuf based(cpplcbuf),
     2 len    fixed bin(15),
     2 offset fixed bin(15),
     2 argstr char(1000);
   dcl my_argument char(1000) varying;
   dcl my_argument_len fixed bin(31);
   dcl length builtin;

   my_argument_len = len - offset - 4;
   if my_argument_len = 0 then
     my_argument = '';
   else
     my_argument =  substr(argstr,offset + 1, my_argument_len);
   display('Program args: ' || my_argument);
 end tsoarg2;
```

図 *22. SYSTEM(STD)* オプションを使用したときに *TSO* のもとで *CPPL* からプログラム実引数を表示するサンプル・プログラム

MAIN プログラムをコマンドとして呼び出すか、CALL を介して呼び出すかに関係なく、PLIXOPT ストリングによって常にランタイム・オプションを指定できます。

z/OS UNIX を指定した場合の MAIN の呼び出し

z/OS UNIX では、SYSTEM(MVS) オプションまたは SYSTEM(OS) オプションを使用して MAIN プログラムをコンパイルできます。ただし、プログラムに渡されるパラメーターの数とフォーマットは、使用するオプションによって異なります。

SYSTEM(MVS) オプションを使用してコンパイルされた MAIN プログラムには、通常どおり、そのプログラムが呼び出されたときに指定されたパラメーターを含む 1 つの CHARACTER VARYING ストリングが渡されます。

しかし、SYSTEM(OS) オプションを使用してコンパイルされた MAIN プログラムには、z/OS UNIX の資料に指定されている 7 つのパラメーターが渡されます。これら 7 つのパラメーターには、以下が含まれます。

- 引数カウント (第 1 引数として実行可能な名前を含む)
- 複数の引数のアドレスを含む、単一の配列のアドレス
- ヌル終了文字ストリングである複数の引数のアドレスを含む、単一の配列のアドレス
- 環境変数セットのカウント
- 複数の環境変数の長さのアドレスを含む、単一の配列のアドレス
- ヌル終了文字ストリングである複数の環境変数のアドレスを含む、単一の配列のアドレス

214 ページの図 23 内のプログラムは、SYSTEM(OS) インターフェースを使用して、個々の引数と環境変数を参照して表示します。

```
*process display(std) system(os);

 sayargs:
  proc(argc, pArgLen, pArgStr, envc, pEnvLen, pEnvStr, pParmSelf)
  options( main, noexecops );

    dcl argc                fixed bin(31) nonasgn byaddr;
    dcl pArgLen             pointer nonasgn byvalue;
    dcl pArgStr             pointer nonasgn byvalue;
    dcl envc                fixed bin(31) nonasgn byaddr;
    dcl pEnvLen             pointer nonasgn byvalue;
    dcl pEnvStr             pointer nonasgn byvalue;
    dcl pParmSelf           pointer nonasgn byvalue;

    dcl q(4095)             pointer based;
    dcl bxb                 fixed bin(31) based;
    dcl bcz                 char(31) varz based;

    display( 'argc = ' || argc );
    do jx = 1 to argc;
      display( 'pargStr(jx) =' || pArgStr->q(jx)->bcz );
    end;
    display( 'envc = ' || envc );
    do jx = 1 to envc;
      display( 'pEnvStr(jx) =' || pEnvStr->q(jx)->bcz );
    end;

  end;
```

図 *23. z/OS UNIX* 引数と環境変数を表示するサンプル・プログラム

第 6 章 64 ビット・プログラムに対するリンク・エディットおよび実行

LP(64) でのコンパイルが終わった 64 ビット・プログラムは、未解決の相互参照、および言語環境プログラム・ランタイム・ライブラリーに対する参照を含む 1 つ以上のオブジェクト・モジュールで構成されています。これらの参照は、リンク・エディット時 (静的) または実行時 (動的) に解決されます。

したがって、PL/I プログラムのコンパイルが終わった後で次にとるべきステップは、そのプログラムをテスト・データを用いてリンクして実行し、予想どおりの結果が出るかどうか確認することです。

言語環境プログラムは、ユーザーがプログラムを実行するのに必要なランタイム環境とサービスを提供します。PL/I およびその他すべての 言語環境プログラムに準拠した言語プログラムのリンクと実行については、「*z/OS Language Environment 64* ビット仮想アドレッシング・モード向け プログラミング・ガイド」を参照してください。既存の PL/I プログラムから 言語環境プログラムへの移行については、「*Enterprise PL/I for z/OS* コンパイラーおよびランタイム 移行ガイド」を参照してください。

64 ビット・プログラムに関するリンク・エディットの考慮事項

リンカー出力に PDSE を使用する必要があります。

64 ビット・プログラムでバインダーを使用

バインダー出力は PDSE 内に配置する必要があり、バインド・ステップで DYNAM(DLL) をパラメーターとして指定する必要があります。

PL/I サイド・デック IBMPQV11 および C サイド・デック CELQS003 をバインダーへの入力として組み込む必要があります。これらのサイド・デックは SCEELIB データ・セットにあります。これらのサイド・デックを SYSLIN 連結に組み込む必要があります。 SCEEBND2 データ・セットを SYSLIB DD ステートメントで指定する必要があります。参考までに 175 ページの『コンパイルおよびバインド - 64 ビット (IBMQCB)』を参照してください。

バインド・ステップで DYNAM(DLL) をパラメーターとして指定する場合は、次のオプションを強くお勧めします。

```
CASE(MIXED), AMODE(64), RMODE(ANY)
```

64 ビット・プログラムで ENTRY カードを使用

フェッチされるモジュールをビルドするときに、エントリー・ポイントとして Enterprise PL/I ルーチンが使用される場合は、ENTRY カードで CELQSTRT が指定されなければなりません。

© Copyright IBM Corp. 1999, 2017

64 ビット・プログラムに関する実行時の考慮事項

プログラム初期化ルーチンに渡されるパラメーターとしてランタイム・オプションを指定できます。また、PLIXOPT 変数の中でランタイム・オプションを指定できます。PLIXOPT 変数を使用してランタイム・オプションを指定し、既存のプログラムを変更してから再コンパイルするのも、パフォーマンスの観点から有利な場合があります。

PLIXOPT の使用法については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

端末の入出力を簡素化するため、端末に割り当てられるストリーム・ファイルには各種規則が取り入れられています。 次の 3 つの領域が影響を受けます。

1. PRINT ファイルのフォーマット設定
2. 自動プロンプト機能
3. 入力のスペーシングと句読法の規則

注**:** 端末でのレコード入出力にはプロンプトその他の機能は提供されていません。したがって、端末から、または端末への伝送にはストリーム入出力を使用することを強くお勧めします。

64 ビット・プログラムに関する SYSPRINT の考慮事項

64 ビット・プログラムの場合、SYSPRINT は C stdout ファイルと同等です。また、共用 SYSPRINT はサポートされていません。

SYSPRINT の LINESIZE を 132 (C で許可される最大値) より大きくすることはできません。

64 ビット・アプリケーションで自分のルーチンにおいて FETCH を使用

Enterprise PL/I では、PL/I、C、またはアセンブラーによってコンパイルされたルーチンをフェッチできます。

64 ビット・アプリケーションで Enterprise PL/I ルーチンをフェッチ

フェッチされたモジュールは FETCH 命令、I/O 命令、ALLOCATE 命令、および FREE 命令を実行できます。ただし、依然として制限がいくつか適用されます。

フェッチされたモジュールは次の操作を実行できます。

- 他のモジュールをフェッチする。
- 任意の PL/I ファイルに入出力操作を実行する。このファイルは、フェッチされたモジュール、メイン・モジュール、または他のフェッチされたモジュールによりオープンできます。
- 独自の CONTROLLED 変数の ALLOCATE と FREE を実行する。

しかし、フェッチの対象である Enterprise PL/I モジュールに対するいくつかの制限があります。

1. リンク・エディット・ステップで ENTRY カードが指定されていない場合は、フェッチされたルーチンの PROCEDURE ステートメントで OPTIONS(FETCHABLE) を指定する必要があります。
2. ENTRY カードでは CELQSTRT が指定されなければなりません。
3. LP(64) では RENT は実際は常にオンであるため、フェッチされる側の ENTRY が、フェッチする側のモジュールで OPTIONS(COBOL) または OPTIONS(ASM) として宣言されていてはなりません。この状態で記述子を渡すのを避けたい場合には、ENTRY 宣言に OPTIONS(NODESCRIPTOR) 属性を指定する必要があります。
4. Enterprise PL/I ルーチンは自らをフェッチすることはできません。

64 ビット・アプリケーションで PL/I MAIN ルーチンをフェッチ

64 ビット PL/I MAIN ルーチンは 64 ビット PL/I MAIN ルーチンをフェッチできません。

64 ビット・アプリケーションでアセンブラー・ルーチンをフェッチ

アセンブラー・ルーチンをフェッチするルーチンにおける ENTRY 宣言では OPTIONS(ASM) が指定されなければなりません。そのアセンブラー・ルーチンは AMODE=64 でリンクされていなければなりません。 LINKAGE(SYSTEM) を指定する必要があります。

データ専用アセンブラー・モジュールをフェッチするときは、FETCH A SET(P) 構成を使用して、ポインター P を正しく設定する必要があります。

TSO/E のもとでの MAIN の呼び出し

SYSTEM(MVS) オプションで MAIN プログラムをコンパイルする場合は、TSO CALL コマンドを使用するか、または TSO コマンド・プロセッサーとしてプログラムを呼び出すことができます。ランタイム・オプションとパラメーターの両方を、MVS バッチ配下と同じ方法で渡すことができます。

例えば、TSOARG1 という MAIN プログラムが userid.TEST.load というデータ・セットのメンバーとしてリンク・エディットされた場合は、次のものによってこのプログラムを呼び出すことができます。

```
  CALL TEST(TSOARG1) 'RPTSTG(ON),TRAP(ON)/THIS IS MY ARGUMENT'
または
  CALL TEST(TSOARG1) '/THIS IS MY ARGMENT'
または
  TSOARG1 TRAP(ON)/THIS IS MY ARGUMENT
または
  TSOARG1 /THIS IS MY ARGUMENT
```

注: CALL ステートメントを使用しないでプログラムを実行するためには、TSOARG1 を含むデータ・セット (userid.TEST.load) が標準 TSO プログラム検索リストになければなりません。これは、次の TSO コマンドを発行することによって達成できます。

```
TSOLIB ACTIVATE DSN('userid.TEST.load')
```

ただし、MAIN プログラムを SYSTEM(TSO) オプションでコンパイルすると、コマンド・プロセッサー・パラメーター・リスト (CPPL) を指すポインターがプログラムに渡されます。 この場合は、NOEXECOPS が有効です。 プログラムを TSO コマンドとして呼び出せますが、TSO CALL ステートメントによって呼び出すことはできません。 パラメーター・リスト内のポインターは 31 ビット・ポインターであることに注意してください。 次の例を参照してください。

```
TSOARG2 This is my argument
```

図 24 のプログラムは、SYSTEM(TSO) インターフェースを使用して、CPPL からプログラム実引数を指定して表示します。

```
*process system(tso);
 tsoarg2: proc (cppl_ptr) options(main);
   dcl cppl_ptr pointer;
   dcl 1 cppl based(cppl_ptr),
     2 cpplcbuf   pointer(32),
     2 cpplupt    pointer(32),
     2 cpplpscb   pointer(32),
     2 cpplect    pointer(32);
   dcl 1 cpplbuf based(cpplcbuf),
     2 len    fixed bin(15),
     2 offset fixed bin(15),
     2 argstr char(1000);
   dcl my_argument char(1000) varying;
   dcl my_argument_len fixed bin(31);
   dcl length builtin;

   my_argument_len = len - offset - 4;
   if my_argument_len = 0 then
     my_argument = '';
   else
     my_argument =  substr(argstr,offset + 1, my_argument_len);
   display('Program args: ' || my_argument);
 end tsoarg2;
```

図 *24. SYSTEM(STD)* オプションを使用したときに *TSO* のもとで *CPPL* からプログラム実引数を表示するサンプル・プログラム

MAIN プログラムをコマンドとして呼び出すか、CALL を介して呼び出すかに関係なく、PLIXOPT ストリングによって常にランタイム・オプションを指定できます。

z/OS UNIX を指定した場合の MAIN の呼び出し

z/OS UNIX では、SYSTEM(MVS) オプションまたは SYSTEM(OS) オプションを使用して MAIN プログラムをコンパイルできます。ただし、プログラムに渡されるパラメーターの数とフォーマットは、使用するオプションによって異なります。

SYSTEM(MVS) オプションを使用してコンパイルされた MAIN プログラムには、通常どおり、そのプログラムが呼び出されたときに指定されたパラメーターを含む 1 つの CHARACTER VARYING ストリングが渡されます。

しかし、SYSTEM(OS) オプションを使用してコンパイルされた MAIN プログラムには、z/OS UNIX の資料に指定されている 7 つのパラメーターが渡されます。これら 7 つのパラメーターには、以下が含まれます。

- 引数カウント (第 1 引数として実行可能な名前を含む)
- 複数の引数のアドレスを含む、単一の配列のアドレス
- ヌル終了文字ストリングである複数の引数のアドレスを含む、単一の配列のアドレス
- 環境変数セットのカウント
- 複数の環境変数の長さのアドレスを含む、単一の配列のアドレス
- ヌル終了文字ストリングである複数の環境変数のアドレスを含む、単一の配列のアドレス

図 25 内のプログラムは、SYSTEM(OS) インターフェースを使用して、個々の引数と環境変数を参照して表示します。

```
*process display(std) system(os);

 sayargs:
  proc(argc, pArgLen, pArgStr, envc, pEnvLen, pEnvStr, pParmSelf)
  options( main, noexecops );

   dcl argc                fixed bin(31) nonasgn byaddr;
   dcl pArgLen             pointer nonasgn byvalue;
   dcl pArgStr             pointer nonasgn byvalue;
   dcl envc                fixed bin(31) nonasgn byaddr;
   dcl pEnvLen             pointer nonasgn byvalue;
   dcl pEnvStr             pointer nonasgn byvalue;
   dcl pParmSelf           pointer nonasgn byvalue;

   dcl q(4095)             pointer based;
   dcl bxb                 fixed bin(31) based;
   dcl bcz                 char(31) varz based;

   display( 'argc = ' || argc );
   do jx = 1 to argc;
     display( 'pargStr(jx) =' || pArgStr->q(jx)->bcz );
   end;
   display( 'envc = ' || envc );
   do jx = 1 to envc;
     display( 'pEnvStr(jx) =' || pEnvStr->q(jx)->bcz );
   end;

 end;
```

図 *25. z/OS UNIX* 引数と環境変数を表示するサンプル・プログラム

第 7 章 64 ビット・アプリケーションを開発する場合の考慮事項

Enterprise PL/I を使用すれば、31 ビット/64 ビット・アプリケーションを開発できます。ご使用のアプリケーションで 64 ビット環境がサポートされるようにするには、必要に応じてコードを調整しなければならない場合があります。このセクションでは、開発時およびコンパイル時に考慮すべき事項について説明します。

コンパイラー・オプションを使用して 64 ビット・アプリケーションをビルド

64 ビット・アプリケーション用にコードをコンパイルするには、LP(64) コンパイラー・オプションを使用する必要があります。 LP(64) では、一部のコンパイラー・オプションまたはコンパイラー・サブオプションがサポートされていないこと、および一部のオプションまたはサブオプションがコンパイル時に無視されることを理解しておく必要があります。

LP(64) でコードをコンパイルするときは、以下のコンパイラー・オプションに注意してください。

10 ページの『**BACKREG**』
このオプションは無視されます。

14 ページの『**CEESTART**』
このオプションは無視されます。

14 ページの『**CHECK**』
LP(64) では STORAGE サブオプションはサポートされていません。つまり、以下の組み込み関数は使用できません。

- ALLOCSIZE
- CHECKSTG
- UNALLOCATED

16 ページの『**CMPAT**』
このオプションは無視されます。実際上、LP(64) では CMPAT(V3) が常にオンになります。

18 ページの『**COMMON**』
このオプションは無視されます。

24 ページの『**DEFAULT**』
LP(64) では、以下のサブオプションは無視されます。

- LINKAGE

76 ページの『**RENT**』
このオプションは無視されます。実際上、LP(64) では RENT が常にオンになります。

© Copyright IBM Corp. 1999, 2017

97 ページの『**STDSYS**』
このオプションは無視されます。実際上、LP(64) では STDSYS が常にオンになります。

107 ページの『**WRITABLE**』
このオプションは無視されます。

関連情報:

54 ページの『LP』
LP オプションは、コンパイラーが 31 ビット・コードを生成するのか 64 ビット・コードを生成するのかを指定します。 また、このオプションは、
POINTER、HANDLE、および関連変数のデフォルト・サイズも決定します。

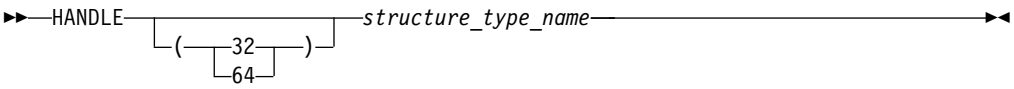
LP(64) で属性 HANDLE および POINTER を使用

LP(64) における HANDLE および POINTER のサイズと位置合わせのデフォルトは、LP(32) におけるデフォルトとは異なります。

HANDLE 属性

LP(32) ではデフォルトは HANDLE(32) です。 LP(64) ではデフォルトは HANDLE(64) です。

構文



HANDLE(32) は 4 バイトのサイズであり、デフォルトでフルワード位置合わせされます。

HANDLE(64) は 8 バイトのサイズであり、デフォルトでダブルワード位置合わせされます。

HANDLE(64) は LP(64) でのみ有効です。

HANDLE(32) を HANDLE(64) に変更することは常に有効です。その逆は、HANDLE(64) の先頭 4 バイトがゼロの場合にのみ有効です。

ハンドルのサイズと位置合わせが変更されたために、ハンドルを含む構造体で埋め込みバイトが使用されることがあります。

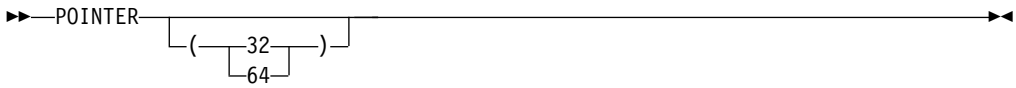
関連情報:

54 ページの『LP』
LP オプションは、コンパイラーが 31 ビット・コードを生成するのか 64 ビット・コードを生成するのかを指定します。 また、このオプションは、
POINTER、HANDLE、および関連変数のデフォルト・サイズも決定します。

POINTER 属性

LP(32) ではデフォルトは POINTER(32) です。 LP(64) ではデフォルトは POINTER(64) です。

構文



POINTER(32) は 4 バイトのサイズであり、デフォルトでフルワード位置合わせされます。

POINTER(64) は 8 バイトのサイズであり、デフォルトでダブルワード位置合わせされます。

POINTER(64) は LP(64) でのみ有効です。

POINTER(32) を POINTER(64) に割り当てることは常に有効です。その逆は、POINTER(64) の先頭 4 バイトがゼロの場合にのみ有効です。

ポインターのサイズと位置合わせが変更されたために、ポインターを含む構造体で埋め込みバイトが使用されることがあります。

関連情報:

54 ページの『LP』
LP オプションは、コンパイラーが 31 ビット・コードを生成するのか 64 ビット・コードを生成するのかを指定します。 また、このオプションは、POINTER、HANDLE、および関連変数のデフォルト・サイズも決定します。

LP(64) で ENTRY 変数を使用

LP(64) では、ENTRY 変数はすべて、LIMITED 属性を持つかどうかに関係なく 8 バイトのサイズであり、デフォルトでダブルワード位置合わせされます。したがって、現在は、ENTRY 変数を含む構造体には埋め込みバイトが使用されることがあります。

LP(64) で組み込み関数を使用

64 ビット・アプリケーションを開発するときは、一部の組み込み関数では引数の型と戻りの型が異なることを理解しておく必要があります。

LP(64) で FIXED BIN(63) 値を返す組み込み関数

以下の組み込み関数は、LP(64) では FIXED BIN(63) 値を返しますが、LP(32) では FIXED BIN(31) 値を返します。

AUTOMATIC
AVAILABLEAREA
BASE64DECODE16
BASE64DECODE8
BASE64ENCODE16
BASE64ENCODE8
CURRENTSTORAGE
FILEID
FILEREAD
FILESEEK
FILETELL
HEXDECODE
HEXDECODE8
JSONGETARRAYEND
JSONGETARRAYSTART
JSONGETCOLON
JSONGETCOMMA
JSONGETMEMBER
JSONGETOBJECTEND
JSONGETOBJECTSTART
JSONGETVALUE
JSONPUTARRAYEND
JSONPUTARRAYSTART
JSONPUTCOLON
JSONPUTCOMMA
JSONPUTMEMBER
JSONPUTOBJECTEND
JSONPUTOBJECTSTART
JSONPUTVALUE
JSONVALID
MEMCONVERT
MEMCU12
MEMCU14
MEMCU21
MEMCU24
MEMCU12
MEMCU41
MEMCU42
MEMINDEX
MEMSEARCH
MEMSEARCHR
MEMVERIFY
MEMVERIFYR
LOCATION
LOCSTG
OFFSETDIFF
POINTERDIFF
STORAGE
WSCOLLAPSE
WSCOLLAPSE16
WSREPLACE
WSREPLACE16
XMLCHAR
XMLSCRUB
XMLSCRUB16

LP(64) で整数引数が FIXED BIN(63) に変換される組み込み関数

以下の組み込み関数には、ストレージの断片のサイズを表す 1 つ以上の引数があります。その引数は、LP(64) では必要に応じて FIXED BIN(63) に変換されます。

ALLOCATE
BASE64DECODE8
BASE64DECODE16
BASE64ENCODE8
BASE64ENCODE16
CHECKSUM
COMPARE
FILEWRITE
HEXDECODE
HEXDECODE8
JSONGETARRAYEND
JSONGETARRAYSTART
JSONGETCOLON
JSONGETCOMMA
JSONGETMEMBER
JSONGETOBJECTEND
JSONGETOBJECTSTART
JSONGETVALUE
JSONPUTARRAYEND
JSONPUTARRAYSTART
JSONPUTCOLON
JSONPUTCOMMA
JSONPUTMEMBER
JSONPUTOBJECTEND
JSONPUTOBJECTSTART
JSONPUTVALUE
JSONVALID
MEMCONVERT
MEMCU12
MEMCU14
MEMCU21
MEMCU24
MEMCU41
MEMCU42
MEMINDEX
MEMSEARCH
MEMSEARCHR
MEMVERIFY
MEMVERIFYR
PLIASCII
PLIEBCDIC
PLIFILL
PLIMOVE
PLIOVER
PLISAXA
PLISAXB
PLISAXC
PLISAXD
PLITRAN11
PLITRAN12
PLITRAN21
PLITRAN22
WSCOLLAPSE
WSCOLLAPSE16
WSREPLACE
WSREPLACE16
XMLCHAR
XMLSCRUB
XMLSCRUB16

LP(64) でサポートされない組み込み関数

LP(64) では、CHECK コンパイラー・オプションの STORAGE サブオプションがサポートされていないため、以下の組み込み関数は使用できません。

- ALLOCSIZE
- CHECKSTG
- UNALLOCATED

注: 64 ビット環境では、PLISRT 組み込み関数は LE V2R2 以降でのみサポートされます。

SQL プログラムの考慮事項

同じソースを使用して 32 ビット・アプリケーションおよび 64 ビットのアプリケーションを開発するには、`EXEC SQL INCLUDE` ステートメントを使用して `sqlda` または `sqlda2` を宣言すること、および `sqlvar` のサイズとして `stg(sqlvar(1))` を使用するように `sqldabc` フィールドを設定することをお勧めします。ご使用の

SQL プログラムがこの推奨事項に従っていない場合は、64 ビット・アプリケーションを開発するためにコードを変更しなければならない可能性があります。

必要に応じて、既存の 32 ビット SQL プログラムを、その振る舞いを変えずに変更できます。こうした変更により、プログラムの移植がいっそう簡単になる可能性があります。

sqlda の宣言

EXEC SQL INCLUDE sqlda ステートメントで sqlda が宣言されていない場合は、sqllen フィールドの後に次のフィールドを追加して sqlda の宣言を変更する必要があります。

```
char( length(hex(sysnull()))/2 - 4 ),
```

例えば、プログラムで sqlda 構造体が以下のように宣言されているとします。

```
dcl
      1 fsqlda based(fsqldaptr),
        2 sqldaid      char(8),
        2 sqldabc      fixed bin(31),
        2 sqln         fixed bin(15),
        2 sqld         fixed bin(15),
        2 sqlvar(fsqlsize refer(sqln)),
          3 sqltype    fixed bin(15),
          3 sqllen     fixed bin(15),
          3 sqldata    pointer,
          3 sqlind     pointer,
          3 sqlname    char(30) var;
```

この宣言を以下のように変更する必要があります。

```
dcl
      1 fsqlda based(fsqldaptr),
        2 sqldaid      char(8),
        2 sqldabc      fixed bin(31),
        2 sqln         fixed bin(15),
        2 sqld         fixed bin(15),
        2 sqlvar(fsqlsize refer(sqln)),
          3 sqltype    fixed bin(15),
          3 sqllen     fixed bin(15),
          3 *          char( length(hex(sysnull()))/2 - 4 ),
          3 sqldata    pointer,
          3 sqlind     pointer,
          3 sqlname    char(30) var;
```

sqlda2 の宣言

EXEC SQL INCLUDE sqlda2 ステートメントで sqlda2 が宣言されていない場合は、sqlrsvdl フィールドの後に次のフィールドを追加して sqlda2 の宣言を変更する必要があります。

```
char( length(hex(sysnull())) - 8 ),
```

例えば、プログラムで sqlda2 構造体が以下のように宣言されているとします。

```
dcl
      1 fsqlda2 based(fsqldaptr),
        2 sqldaid2     char(8),
        2 sqldabc2     fixed bin(31),
        2 sqln2        fixed bin(15),
        2 sqld2        fixed bin(15),
        2 sqlvar2(fsqlsize refer(sqln2)),
```

```
          3 sqlbiglen,
            4 sqllongl fixed bin(31),
            4 sqlrsvdl fixed bin(31),
          3 sqldatal   pointer,
          3 sqltname   char(30) var;
```

この宣言を以下のように変更する必要があります。

```
dcl
      1 fsqlda2 based(fsqldaptr),
        2 sqldaid2     char(8),
        2 sqldabc2     fixed bin(31),
        2 sqln2        fixed bin(15),
        2 sqld2        fixed bin(15),
        2 sqlvar2(fsqlsize refer(sqln2)),
          3 sqlbiglen,
            4 sqllongl fixed bin(31),
            4 sqlrsvdl fixed bin(31),
          3 *          char( length(hex(sysnull())) - 8 ),
          3 sqldatal   pointer,
          3 sqltname   char(30) var;
```

sqldabc の設定

sqldabc フィールドが sqlvar のサイズとして 44 に設定されている場合は、代わりに stg(sqlvar(1)) を使用するようにそのフィールドを設定する必要があります。

例えば、ステートメント sqlda.sqldabc = 16 + (44 * sqlda.sqld); は sqlda.sqldabc = 16 + (stg(sqlda.sqlvar(1)) * sqlda.sqld); に変更する必要があります。

次のようにコーディングします。

```
SQLDABC = LEN_SQLDA + SQLN * LEN_SQLVAR;
```

LEN_SQLVAR の宣言は以下のように変更する必要があります。

```
元の宣言:          DCL LEN_SQLVAR FIXED BIN(15) VALUE( 44 );
更新された宣言:    DCL LEN_SQLVAR FIXED BIN(15) VALUE( STG(SQLDA.SQLVAR(1)) );
```

31 ビット・ルーチンとのやり取り

言語環境プログラムは、同じアプリケーションでの 64 ビット・プログラムと 31 ビット・プログラムとの混合をサポートしていません。例えば、LP(32) でコンパイルされた PL/I プログラムを、LP(64) でコンパイルされた PL/I プログラムから呼び出すことはできません。

非言語環境プログラム・ベース の 31 ビット・ルーチンとのやり取りを簡単に行えるように、31 ビット・ルーチンのロード、呼び出し、および解放を行うための汎用インターフェースが提供されています。ロードされたモジュールのエントリー・ポイント・アドレスを取得するための関数も提供されています。

IBMPC32I を呼び出して 31 ビット・ルーチンをロードします。これは、モジュールの名前を保持する 8 ビット文字ストリングを指すポインターを入力として使用し、呼び出し関数および公開関数に使用されるファイル・ハンドルを返します。

```
dcl <load32>   ext( "_IBMPC32I" )
              entry( char(8) byaddr inonly )
              returns( pointer byvalue )
              options( nodescriptor linkage(optlink) );
```

IBMPC32C を呼び出して 31 ビット・ルーチンを呼び出します。これは、2 つのパラメーターを入力として使用します。最初のパラメーターは、IBMPC32I 呼び出しから返されるファイル・ハンドルです。 2 番目のパラメーターは、31 ビット・ルーチンに渡されるユーザー・パラメーター・リストです。ユーザー・パラメーター・リストは 2 GB 境界より下のストレージに存在していなければならないことに注意してください。 IBMPC32C は、呼び出した 31 ビット・ルーチンからのレジスター 15 の値を返します。

```
dcl <call32>  ext( "_IBMPC32C" )
             entry( pointer byvalue, pointer byvalue )
             returns( fixed bin(31) byvalue )
             options( nodescriptor linkage(optlink) );
```

IBMPC32T を呼び出して 31 ビット・ルーチンを公開します。これは、ファイル・ハンドルをパラメーターとして使用します。

```
dcl <rel32>    ext( "_IBMPC32T" )
              entry( pointer byvalue )
              options( nodescriptor linkage(optlink) );
```

IBMPC32E を呼び出して、ロードしたモジュールのエントリー・ポイント・アドレスを取得します。これは、ファイル・ハンドルをパラメーターとして使用し、ロードされたモジュールのエントリー・ポイント・アドレスを返します。このアドレスに直接分岐することは無効であることに注意してください。

```
dcl <epa32>   ext( "_IBMPC32E" )
             entry( pointer byvalue )
             returns( pointer byvalue )
             options( nodescriptor linkage(optlink) );
```

次のコード・フラグメント例では、上記のインターフェースの使用方法を示したものです。 DSNALI は 31 ビット DB2 機能です。

```
....

dcl load32      ext( "_IBMPC32I" )
                     entry( char(8) byaddr inonly )
                     returns( pointer byvalue )
                     options( nodescriptor );

dcl call32     ext( "_IBMPC32C" )
                   entry( pointer byvalue, pointer byvalue )
                   returns( fixed bin(31) byvalue )
                   options( nodescriptor );

dcl  file_pointer   pointer;
dcl  plist_pointer pointer;
dcl 1 dsnali_plist based(plist_pointer),
3 args_list(10)        pointer(32),
3 content union,
5 connect,
7 functioncode  char(12),
7 subsystemid  char(4),
7 tecb              fixed bin(31),
7 secb             fixed bin(31),
7 ribpointer       ptr(32),
7 returncode     fixed bin(31),
7 reasoncode    fixed bin(31),
```

```
5 open,
7 functioncode  char(12),
7 subsystemid   char(4),
7 planname       char(8),
7 returncode     fixed bin(31),
7 reasoncode   fixed bin(31);

  dcl lastargflag bit(1) based;

  ....

 plist_pointer = alloc31( stg(dsnali_plist) );
 file_pointer = load32( "DSNALI" );

 /* set up argument list for CONNECT function for DSNALI call */
 args_list( 1 ) = addr( connect.functioncode );
 args_list( 2 ) = addr( connect.subsystemid );
 args_list( 3 ) = addr( connect.tecb );
 args_list( 4 ) = addr( connect.secb );
 args_list( 5 ) = addr( connect.ribpointer );
 args_list( 6 ) = addr( connect.returncode ) ;
 args_list( 7 ) = addr( connect.reasoncode ) ;

 /* mark last argument */
 addr( args_list( 7 ) ).lastargflag = '1'b;

 /* set up values for CONNECT function parameters */
 connect.functioncode = 'CONNECT';
 connect.subsystemid = 'DB2S';
 connect.tecb = 0;
 connect.secb = 0;
 connect.ribpointer = sysnull;
 connect.returncode = 0;
 connect.reasoncode = 0;

 /* invoke DSNALI with CONNECT function */
 rc = call32( file_pointer, plist_pointer );

 ...

 /* set up argument list for OPEN function for DSNALI call */
 args_list( 1 ) = addr( open.functioncode );
 args_list( 2 ) = addr( open.subsystemid );
 args_list( 3 ) = addr( open.planname );
 args_list( 4 ) = addr( open.returncode );
 args_list( 5 ) = addr( open.reasoncode );

 /* mark last argument */
 addr( args_list( 5 ) ).lastargflag = '1'b;

 /* set up values for OPEN function parameters */
 open.functioncode = 'OPEN';
 open.subsystemid = 'DB2S';
 open.planname = 'TESTCASE';
 open.returncode = 0;
 open.reasoncode = 0;

 /* invoke DSNALI with OPEN function */
 rc = call32( file_pointer, plist_pointer ) ;

  ....
```

第 2 部 入出力機能の使用

© Copyright IBM Corp. 1999, 2017

第 8 章 データ・セットとファイルの使用

本章では、ファイルを割り振り、ご使用のプログラム内で既知のファイルにデータ・セットを関連付ける方法について説明します。また、主要な 5 つのタイプのデータ・セットを概説し、そのデータ・セットをどのように編成するのか、およびそのデータ・セットにどのようにアクセスするのかについて説明します。この章は、ファイルやデータ・セットの特性をいくつか指定する方法を学習するときに役立ちます。

PL/I プログラムは、レコード と呼ばれる情報単位の処理と送信を行います。 レコードの集まりをデータ・セット と呼びます。 データ・セットは、PL/I プログラムの外部にある情報の物理的な集まりです。つまり、データ・セットは、PL/I またはその他の言語で書かれたプログラムや、オペレーティング・システムのユーティリティー・プログラムによって、作成、アクセス、または変更することができます。

PL/I プログラムは、ファイル と呼ばれるデータ・セットのシンボルによる表現または論理表現を使って、データ・セット内の情報を認識したり処理したりします。

注: INDEXED は VSAM を暗黙指定し、バッチ環境下でのみサポートされます。

注: PL/I V5.1 では、領域データ・セットは 64 ビット・プログラムに対してサポートされていません。

ファイルの割り振り

順次、VSAM、REGIONAL(1)、および HFS などのどのファイル・タイプに対しても、以下の方法を使用して外部名を定義することができます。

- MVS または TSO 環境の場合:
  - JCL (ジョブ制御言語) の *DD* 名
  - 環境変数名
  - OPEN ステートメントの TITLE オプション
- z/OS UNIX システム・サービス 環境の場合:
  - 環境変数名
  - OPEN ステートメントの TITLE オプション

バッチ環境の HFS ファイルには、次の規則が適用されます。

- FILEDATA が指定されていない場合、デフォルトの設定値は TEXT です。
- レコード・フォーマットまたは ENVIRONMENT オプションが指定されていない場合、デフォルトは LF タイプの V です。
- FILEDATA が BINARY の場合、固定長ファイルのみが有効です。可変長ファイルで FILEDATA=BINARY が指定された場合、エラー・メッセージ MSGIBM0210S が発行されます。
- 可変長ファイルの場合、ファイルは TYPE=LF であると想定されます。つまり、レコードは LF (x'15') 文字で区切られます。

© Copyright IBM Corp. 1999, 2017

- 区切り文字を含むデータ・ファイルの場合、FILEDATA=TEXT を指定して、PL/I ライブラリーが区切り文字を適切に処理できるようにしてください。

動的割り振り

PL/I プログラムは、環境変数または TITLE オプションによって指定された属性を使用して、ファイルを動的に割り振ります。

PL/I 動的割り振りを使用するには、DSN() 形式 (MVS データ・セットの場合)、または PATH() 形式 (HFS ファイルの場合) を使用してファイル名を指定する必要があります。大/小文字が区別される PATH オプションの **pathname** サブオプションを除き、すべてのオプションと属性は大文字でなければなりません。DSN() オプションに一時データ・セット名を使用しないでください。ファイル名を指定する以下の例を参照してください。

```
OPEN FILE(FILEIN) TITLE('DSN(USER.FILE.EXT),SHR');
OPEN FILE(FILEIN) TITLE('PATH(/usr/FILE.EXT)');

EXPORT DD_FILE="DSN(USER.FILE.EXT),SHR"
EXPORT DD_FILE="PATH(/usr/FILE.EXT)"
```

いつ動的割り振りを行うかを決定する際には、次の優先規則が適用されます。

1. ファイル用に以下の DD ステートメントの 1 つがある場合は、それが使用されます。この規則は z/OS UNIX システム・サービス 環境では無効です。
   - JCL DD
   - TSO ALLOCATE
   - ユーザー開始の動的割り振り
2. ファイル用の DD ステートメントがなく、TITLE オプションが OPEN ステートメントに指定されている場合は、ファイルの外部名に関連付けることによって TITLE オプションが使用されます。
3. ファイル用の DD ステートメントがなく、TITLE オプションが指定されていないが、ファイル用の環境変数はある場合は、ファイルの外部名に関連付けることによって環境変数が使用されます。

MVS データ・セットの場合、Enterprise PL/I ランタイムは、OPEN ステートメントごとに環境変数または TITLE オプションの内容を検査します。

- 同じ外部名を持つファイルが、先行する OPEN ステートメントによって動的に割り振られ、その OPEN ステートメント以後に環境変数または TITLE オプションの内容が変更された場合、ランタイムは前の割り振りの解除を動的に行い、現在、環境変数に設定されているか、TITLE オプションに指定されているオプションを使用してファイルを再割り振りします。
- 環境変数または TITLE オプションの内容が変更されなかった場合、ランタイムは割り振り解除または再割り振りを行わずに、現在の割り振りを使用します。

HFS ファイルの場合、DD ステートメントは割り振り解除され、後続の各 OPEN ステートメントで再割り振りされます。

注:

1. PL/I 動的割り振りに DSN() または PATH() 形式の指定を使用する際、DD ステートメントと、OPEN ステートメントの TITLE オプションの両方がファイルに指定されている場合は、DD ステートメントが使用され、TITLE オプションは無視されます。
2. z/OS UNIX システム・サービス 環境では、ユーザー開始の動的割り振りはサポートされません。 この動的割り振りが試行されると、TITLE オプションまたは環境変数が使用されていても、外部名が使用中であるため、UNDEFINEDFILE 条件が発生します。

z/OS でのデータ・セットとファイルの関連付け

PL/I プログラム内で使用されるファイルには、PL/I ファイル名が付きます。 プログラムの外部で存在する物理データ・セットの名前は、オペレーティング・システムが認識できる名前、すなわち、データ・セット名 または *dsname* をとります。名前のないデータ・セットもあります。その場合、データ・セットはそのデータ・セットが設定されている装置によってシステムに認識されます。

オペレーティング・システムには、使用しているプログラムが参照する物理データ・セットを識別する手段が必要なので、PL/I ファイル名を dsname に関連付ける、プログラムにとっては外部のデータ定義 つまり *DD* ステートメントを記述する必要があります。

例えば、ご使用のプログラムに次のファイル宣言がある場合は、PL/I ファイルの名前に一致するデータ定義名 (*dd* 名) を指定して DD ステートメントを作成しなければなりません。

```
DCL STOCK FILE STREAM INPUT;
```

つまり、DD ステートメントにより物理データ・セット名 (dsname) が指定され、その特性が指定されます。

```
//GO.STOCK  DD DSN=PARTS.INSTOCK, . . .
```

DD ステートメントの作成方法について詳しくは、ご使用のシステムのジョブ制御言語 (JCL) 資料を参照してください。

データ・セットを PL/I ファイルに関連付ける方法は、いくつかあります。 データ・セットを PL/I ファイルに関連付けるには、データ・セットを定義する DD ステートメントの dd 名を、次のいずれかと同じになるように指定します。

- 宣言された PL/I ファイル名
- 関連する OPEN ステートメントの TITLE オプションに指定されている式の文字ストリング値

また、対応する dd 名が次の条件を満たすように、PL/I ファイル名を選択する必要があります。

- ファイルが暗黙的にオープンされる場合、またはファイルを明示的にオープンする OPEN ステートメントに TITLE オプションが組み込まれていない場合、dd 名にはデフォルトでファイル名が使用されます。ファイル名が 8 文字を超える場合、デフォルトの dd 名はそのファイル名の最初の 8 文字から構成されます。

- JCL の文字セットには、区切り文字 (_) は含まれません。 したがって、区切り文字は dd 名の中に出てくることはできません。 有効な dd 名を式として使用する TITLE オプションによりファイルをオープンする場合を除き、ファイル名の最初の 8 文字に区切り文字は使用しないでください。 英字の拡張文字 $、@、および # も dd 名に使用することができますが、dd 名の最初の文字には A から Z までの英字でなければなりません。

外部名は 7 文字までに限定されているため、7 文字を超える外部ファイル名は、そのファイル名の最初の 4 文字と最後の 3 文字を連結したものに短縮されます。しかしこのような短縮名は、関連 DD ステートメント内で dd 名として使用される名前ではありません。

次の 3 つのステートメントを見てみましょう。

1. `OPEN FILE(MASTER);`
2. `OPEN FILE(OLDMASTER);`
3. `READ FILE(DETAIL) ...;`

ステートメント番号 1 を実行すると、ファイル名 MASTER は現行ジョブ・ステップの DD ステートメントの dd 名と同じであると見なされます。ステートメント番号 2 を実行すると、ファイル名 OLDMASTE は現行ジョブ・ステップの DD ステートメントの dd 名と同じであると見なされます。 (ファイル名の最初の 8 文字が dd 名になります。 例えば、OLDMASTER が外部名であるとすると、プログラム内ではコンパイラーによって短縮された OLDMTER という名前が使われます。) ステートメント番号 3 を使ってファイル DETAIL を暗黙的にオープンする場合、ファイル名 DETAIL は現行ジョブ・ステップの DD ステートメントの dd 名と同じであると見なされます。

上記の場合はいずれも、対応する DD ステートメントがジョブ・ストリーム内に存在しなければなりません。それが存在しないと、UNDEFINEDFILE 条件が発生します。 上記 3 つの DD ステートメントは、次のように始まります。

1. `//MASTER   DD ...`
2. `//OLDMASTE DD ...`
3. `//DETAIL   DD ...`

ファイルを明示的あるいは暗黙的にオープンするステートメント内のファイル参照がファイル定数でない場合は、その DD ステートメント名はファイル参照の値と同じでなければなりません。次の例は、DD ステートメントをどのようにファイル変数の値と関連付けるかを示します。

```
DCL PRICES FILE VARIABLE,
    RPRICE FILE;
      PRICES = RPRICE;
      OPEN FILE(PRICES);
```

この DD ステートメントでは、データ・セットをファイル定数 RPRICE と関連付ける必要があります。 RPRICE はファイル変数 PRICES の値です。

```
//RPRICE DD DSNAME=...
```

また、ファイル変数を使えば、1 つのステートメントで、いくつものファイルを何回も処理することができます。 次の例を参照してください。

```
DECLARE F FILE VARIABLE,
        A FILE,
        B FILE,
        C FILE;
          .
          .
          .
     DO F=A,B,C;
        READ FILE (F) ...;
          .
          .
          .
     END;
```

上記の場合、READ ステートメントにより 3 つのファイル A、B、C が読み取られ、各ファイルをそれぞれ異なるデータ・セットに関連付けることができます。 ファイル A、B、C は、それぞれの場合にこの READ ステートメントが実行されたあとも、オープンの状態に保たれます。

次の OPEN ステートメントは、TITLE オプションの使用を示したものです。

```
OPEN FILE(DETAIL) TITLE('DETAIL1');
```

このステートメントを正常に実行するには、現行ジョブ・ステップ内に dd 名が DETAIL1 の DD ステートメントがなければなりません。 このステートメントは、次のように始まります。

```
//DETAIL1 DD DSNAME=DETAILA,...
```

このように、dd 名 DETAIL1 を使ってデータ・セット DETAILA をファイル DETAIL に関連付けます。

複数のファイルと 1 つのデータ・セットの関連付け

同じタイトル名に対して 2 番目のファイル関連付けがオープンされる前に最初のファイル関連付けがクローズされるのであれば、TITLE オプションを使用して、複数の PL/I ファイルを同じ外部データ・セットに関連付けることができます。

次の例では、INVNTRY は 2 つのファイルに関連付けられるデータ・セットを定義する DD ステートメントの名前です。

```
OPEN FILE (FILE1) TITLE('INVNTRY');
.....
CLOSE FILE (FILE1);
.....
OPEN FILE (FILE2) TITLE('INVNTRY');
```

2 番目のファイルのオープンが行われる前に最初のファイルがクローズされていないと、subcode1 値が 59 の UNDEFINEDFILE 条件が発生し、既にオープンしているファイルに対してオープンが試みられたことを示します。

複数のデータ・セットと 1 つのファイルの関連付け

TITLE オプションを使用すれば、複数のデータ・セットを 1 つのファイルに関連付けることができます。

ファイル名は、時点が異なれば、まったく別のデータ・セットを表すことができます。 次の OPEN ステートメントを見てください。

```
OPEN FILE(DETAIL) TITLE('DETAIL1');
```

このステートメントを正常に実行するには、現行ジョブ・ステップにおいて dd 名として DETAIL1 を使用して DD ステートメントを指定する必要があります。

```
//DETAIL1 DD DSNAME=DETAILA,...
```

ファイル DETAIL1 が DD ステートメント DETAIL1 の DSNAME パラメーターで指定されているデータ・セットに関連付けられます。このファイルをクローズしてオープンし直せば、TITLE オプションで別の dd 名を指定して、そのファイルを別のデータ・セットに関連付けることができます。

TITLE オプションを使用すれば、オープン時に、特定のファイル名に関連付けるデータ・セットを複数のデータ・セットから 1 つ動的に選択できます。次の例を考えてみてください。

```
DO IDENT='A','B','C';
   OPEN FILE(MASTER)
        TITLE('MASTER1'||IDENT);
    .
    .
    .
   CLOSE FILE(MASTER);
END;
```

この例では、DO グループの最初の反復処理時に MASTER がオープンされるときに、関連 dd 名は MASTER1A になります。 処理が終了すると、そのファイルはクローズされ、ファイル名と dd 名との関連付けも解除されます。 DO グループの 2 回目の反復処理時に、MASTER がもう一度オープンされます。 このとき、MASTER は dd 名 MASTER1B と関連付けられます。 同様に、DO グループの最後の反復処理時では、MASTER が dd 名 MASTER1C に関連付けられます。

複数のデータ・セットの連結

入力に関してのみ、複数の順次データ・セットや領域データ・セットを連結できます (すなわち、連続する 1 つのデータ・セットとして処理されるように複数のデータ・セットをリンクできます。それには、そのようなデータ・セットを記述する最初の DD ステートメントを除くすべての DD ステートメントから dd 名を削除します。

例えば、次の DD ステートメントを定義すると、そのステートメントが出てくるジョブ・ステップの実行の間中、データ・セット LIST1、LIST2、LIST3 は 1 つのデータ・セットとして処理されます。

```
//GO.LIST DD DSNAME=LIST1,DISP=OLD
//        DD DSNAME=LIST2,DISP=OLD
//        DD DSNAME=LIST3,DISP=OLD
```

PL/I プログラムからの読み込みの場合は、連結データ・セットは同じボリューム上にある必要はありません。

連結データ・セットを逆方向に処理することはできません。

z/OS での HFS ファイルへのアクセス

バッチ・プログラムから HFS ファイルにアクセスするには、DD ステートメント内、または OPEN ステートメントの TITLE オプション内に HFS ファイル名を指定します。

例えば、DD HFS を使用して HFS ファイル /u/USER/sample.txt にアクセスするには、次のように DD ステートメントをコーディングします。

```
//HFS DD PATH='/u/USER/sample.txt',PATHOPTS=ORDONLY,DSNTYPE=HFS
```

OPEN ステートメントの TITLE オプションを使用して同じファイルにアクセスするには、次のようにコーディングします。

```
OPEN FILE(HFS) TITLE('//u/USER/sample.txt');
```

TITLE オプションにある 2 つのスラッシュに注意してください。最初のスラッシュはファイル名 (DD 名でなく) が後に続くことを示し、2 番目は完全修飾 HFS ファイル名の先頭を示します。バッチ環境で HFS ファイルが参照されるときは、ファイルを指定するために使用できる現行ディレクトリーが存在しないため、完全修飾名を使用する必要があります。

次のいずれかの方法で HFS ファイル名を指定することにより、PL/I 動的割り振りを使用してバッチ・プログラムから HFS ファイルにアクセスできます。

- DD ステートメント
- OPEN ステートメントの TITLE オプション
- PUTENV 組み込み関数
- ENVAR オプションを使用した PLIXOPT ストリング

次の例は、上記の方法を使用して HFS ファイルにアクセスする方法を示しています。

```
//HFS DD PATH='/u/USER/sample.txt',PATHOPTS=ORDONLY...

OPEN FILE(HFS) TITLE('PATH(/u/USER/sample.txt)');

xx = putenv('DD_HFS=/u/USER/sample.txt');

Dcl plixopt char(100) var ext static
    init('ENVAR("DD_HFS=PATH(/u/USER/sample.txt)")');
```

注: PL/I 動的割り振りを使用するには、DSN() 形式 (MVS データ・セットの場合)、または PATH() 形式 (HFS ファイルの場合) を使用してファイル名を指定してください。

バッチ環境では、PL/I は HFS ファイルの取り扱い方法を以下の順序で決定します。

1. ファイル宣言で ENV(F) が指定されている場合は、ファイルが固定長レコードで構成されていると想定します。
2. ファイル宣言で ENV(V) が指定されている場合は、ファイルが lf 区切りレコードで構成されていると想定します。
3. ファイルの DD ステートメントで FILEDATA=BINARY が指定されている場合は、ファイルが固定長レコードで構成されていると想定します。

4. 上記以外の場合は、ファイルが lf 区切りレコードで構成されていると想定します。

UNIX から固定長 z/OS ファイルにアクセスするには、ファイルのレコード・サイズをファイルの ENVIRONMENT 属性で、または OPEN ステートメントの TITLE オプションで指定する必要があります。

- ファイル宣言に ENV(F RECSIZE(...)) が含まれていない場合は、データ・セット名とそれらの属性を TITLE オプションで指定する必要があります。 例えば、固定長 80 バイト・レコードのファイルの場合は、TITLE オプションを次のように指定することができます。

  ```
  '/dataset.name,type(fixed),recsize(80)'
  ```

- ENVIRONMENT 属性が F または RECSIZE の一方のみを指定している場合は、データ・セット名と省略された属性を TITLE オプションで指定する必要があります。
- ENVIRONMENT 属性が F と RECSIZE の両方を指定している場合は、データ・セット名のみを TITLE オプションで指定する必要があります。

z/OS UNIX でのデータ・セットとファイルの関連付け

PL/I プログラム内で使用されるファイルには、PL/I ファイル名が付きます。 また、データ・セットには、オペレーティング・システムにより識別される名前が付きます。

PL/I では、ご使用のプログラムで PL/I ファイルが指すデータ・セットを認識する手段が必要であるため、使用するデータ・セットに ID を指定するか、PL/I によるデフォルト ID の使用を許可する必要があります。

環境変数、または OPEN ステートメントの TITLE オプションを使用することにより、明示的にデータ・セットを識別することができます。

PL/I 動的割り振りを使用するには、DSN() 形式 (MVS データ・セットの場合)、または PATH() 形式 (HFS ファイルの場合) を使用してファイル名を指定する必要があります。

PL/I 動的割り振りを使用して z/OS UNIX プログラムから HFS ファイルにアクセスするには、次のいずれかの方法で HFS ファイル名を指定できます。

- OPEN ステートメントの TITLE オプション
- PUTENV 組み込み関数
- ENVAR オプションを使用した PLIXOPT ストリング
- EXPORT ステートメント

環境変数の使用

export コマンドを使用して、PL/I ファイルに関連付けるデータ・セットを識別する環境変数を設定し、オプションでデータ・セットの特性を指定します。 環境変数から得られる情報を、データ定義 (または、DD) 情報と呼びます。

環境変数名の形は DD_*DDNAME* です。この場合、*DDNAME* は、PL/I ファイル定数 (または、代替 *DDNAME* (後述)) の名前です。ファイル名が HFS ファイルを参照

する場合は、ファイル名を適切に修飾する必要があります。これを行わないと、PL/I ライブラリーは、そのファイル名が MVS データ・セットであると想定します。

例

- `declare MyFile stream output;`
  `export DD_MYFILE=/datapath/mydata.dat`

  `/datapath/mydata.dat` は HFS ファイルです。 ファイル名は完全修飾されています。
- `export DD_MYFILE=./mydata.dat`

  `./mydata.dat` は、現行ディレクトリーにある HFS ファイルを指します。
- `export DD_MYFILE=mydata.dat`

  `mydata.dat` は MVS データ・セットです。

次の例は、PL/I 動的割り振りを使用した HFS ファイルへのアクセスを示しています。

```
export DD_HFS="PATH(/u/USER/sample.txt)"
export DD_FILE="DSN(USER.FILE.EXT),SHR"
```

IBM のメインフレーム環境に精通していれば、この環境変数は以下のステートメントと同様に考えることができます。
z/OS での DD ステートメント
TSO での ALLOCATE ステートメント

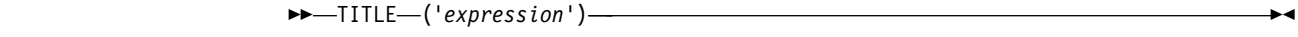
DD_DDNAME 環境変数を併用した構文およびオプションについて詳しくは、 243 ページの『DD_DDNAME 環境変数を使用した特性の指定』を参照してください。

z/OS UNIX 環境では、バッチ環境よりも多くの可変長 HFS ファイル・タイプがサポートされるため、PL/I は HFS ファイルを以下のように扱います。

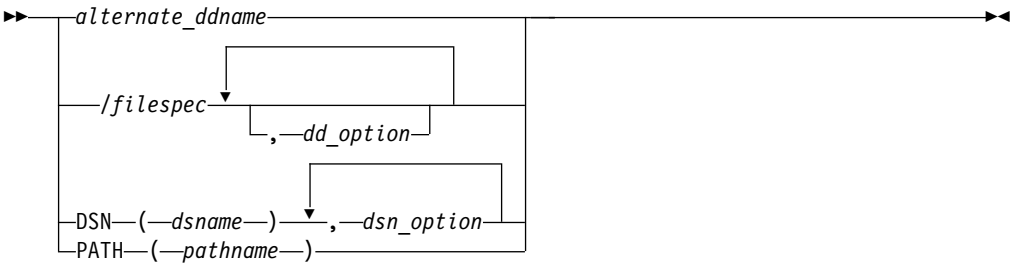
1. ファイル宣言で ENV(F) が指定されている場合は、ファイルが固定長レコードで構成されていると想定します。
2. ファイルの EXPORT ステートメントで TYPE が指定されている場合は、ファイルがそのタイプのレコードで構成されていると想定します。
3. 上記以外の場合は、ファイルが lf 区切りレコードで構成されていると想定します。

OPEN ステートメントの TITLE オプションの使用

OPEN ステートメントの TITLE オプションを使用すると、PL/I ファイルに関連付けるデータ・セットを識別することができます。また、オプショナルでデータ・セットの特性も設定することができます。



expression は、次の構文をとる文字ストリングを与えられなければなりません。



alternate_ddname
代替 DD_DDNAME 環境変数の名前

代替 DD_DDNAME 環境変数は、ファイル定数の後には指定しません。 例えば、プログラムに INVENTRY というファイルがある場合に、2 つの DD_DDNAME 環境変数 (最初が INVENTRY、2 番目が PARTS) を設定すると、次のステートメントを使用することで INVENTRY ファイルを 2 番目の環境変数に関連付けることができます。

```
open file(Inventry) title('PARTS');
```

filespec
使用しているシステムでの任意の有効なファイル指定

filespec の最大長は、1023 文字です。

dd_option
DD_DDNAME 環境変数で許可されている 1 つ以上のオプション。

DD_DDNAME 環境変数のオプションについて詳しくは、 243 ページの『DD_DDNAME 環境変数を使用した特性の指定』を参照してください。

dsname
完全修飾 MVS データ・セット名。

dsn_option
1 つ以上の DSN オプション。

DSN オプションについて詳しくは、 301 ページの『PL/I 動的割り振りを使用して QSAM ファイルを定義』、 319 ページの『PL/I 動的割り振りを使用した REGIONAL(1) データ・セットの定義』、および 331 ページの『PL/I 動的割り振りを使用した VSAM ファイルの定義』を参照してください。

pathname
完全修飾 HFS パス名。

次に、z/OS DSN を指定して上記の方法で OPEN ステートメントを使用している例を示します。

```
open file(Payroll) title('/June.Dat,append(n),recsize(52)');
```

TITLE オプションにある必須の先行スラッシュに注意してください。 この先行スラッシュは、ファイル名 (DD 名ではなく) が後に続くことを示しています。 この場合、June.Dat は MVS データ・セットです。

June.Dat が HFS ファイルの場合、OPEN ステートメントの例は次のようになります。

```
open file(Payroll) title('//u/USER/June.Dat,append(n),recsize(52)');
```

TITLE オプションにある 2 つのスラッシュに注意してください。最初のスラッシュはファイル名 (DD 名でなく) が後に続くことを示し、2 番目は完全修飾 HFS ファイル名の先頭を示します。

完全修飾名の代わりに相対 HFS ファイル名を指定することもできます。次の例を参照してください。

```
open file(Payroll) title('./June.Dat,append(n),recsize(52)');
```

データ・セット名 June.Dat の前には、現行 z/OS UNIX ディレクトリーのパス名が付けられます。

この形式の場合、PL/I プログラムはすべての DD 情報を TITLE 式またはファイル宣言の ENVIRONMENT 属性から入手します (DD_DDNAME 環境変数は参照されません)。

次の例は、PL/I 動的割り振りサポートで OPEN ステートメントを使用する方法を示しています。

- z/OS データ・セットの場合:

  ```
  OPEN FILE(QSAM01) title('DSN(USER.FILE.EXT),SHR');
  ```

- HFS ファイルの場合:

  ```
  OPEN FILE(QSAM01) title('PATH(/u/USER/sample.txt)');
  ```

データ・セットに関連付けられていないファイルの使用の試み

データ・セットに関連付けされていないファイルを (OPEN ステートメントの TITLE オプションを使って、あるいは、DD_DDNAME 環境変数を設定して) 使用しようとすると、UNDEFINEDFILE 条件が発生します。

SYSIN ファイルと SYSPRINT ファイルだけは例外です。つまり、この 2 つのファイルはデフォルトでそれぞれ stdin および stdout になります。

PL/I によるデータ・セットの検索方法

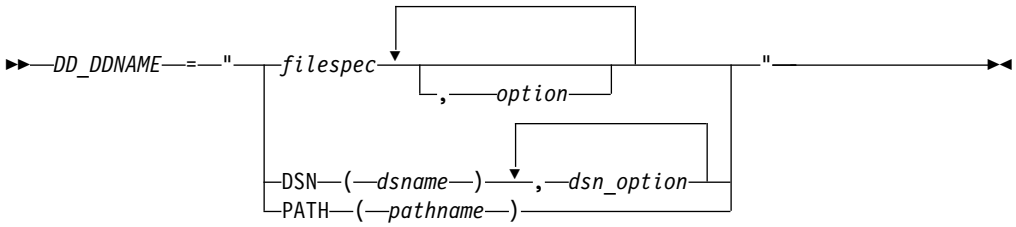
PL/I により、新規データ・セットを作成するためのパス、または既存データ・セットへアクセスするためのパスが、次のいずれかの方法で設定されます。

- 現行ディレクトリー。
- *export DD_DDNAME* 環境変数により定義されているパス。

DD_DDNAME 環境変数を使用した特性の指定

export コマンドを使用すると、PL/I ファイルに関連付けられるデータ・セットを識別する環境変数を設定することができます。また、オプショナルでデータ・セットの特性も指定することができます。 環境変数から得られるこのような情報を、データ定義 (または、dd) 情報と呼びます。

DD_DDNAME 構文



この構文ではブランクを使用することができます。 また、このステートメントの構文は、コマンド入力時にはチェックされません。データ・セットのオープン時に、このステートメントの構文が検証されます。 構文に誤りがあれば、ONCODE 96 により UNDEFINEDFILE 条件が発生します。

DD_DDNAME
環境変数の名前

DDNAME は大文字でなければなりません。また、DDNAME には、OPEN ステートメントの TITLE オプションで指定したファイル定数の名前または代替 DDNAME のいずれかを使用できます。詳しくは、 241 ページの『OPEN ステートメントの TITLE オプションの使用』を参照してください。

31 文字を超える代替 DDNAME を使用する場合、環境変数名の形成に使用される文字は最初の 31 文字のみです。

filespec
PL/I ファイルに関連付けるファイルまたは装置名の指定。

option
DD 情報として指定できるオプション。

dsname
完全修飾 MVS データ・セット名。

pathname
完全修飾 HFS パス名。

以下のトピックでは、DD 情報として指定できるオプションについて説明します。これらのオプションは、DSN() または PATH() の形式には適用されないことに注意してください。

APPEND

APPEND オプションは、既存データ・セットを拡張するか再作成するかを指定します。

Y 新規レコードを、順次データ・セットの終わりに追加する、あるいは相対データ・セットまたは索引付きデータ・セットに挿入することを指定します。

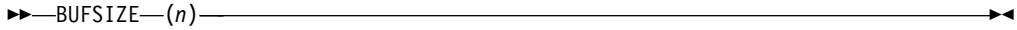
N ファイルが存在する場合、そのファイルを再作成することを指定します。

APPEND オプションを適用できるのは、OUTPUT ファイルだけです。 以下の条件下では APPEND は無視されます。

- 指定ファイルが存在しない場合
- 指定ファイルに OUTPUT 属性がない場合
- 指定ファイルの編成が REGIONAL(1) の場合

BUFSIZE

BUFSIZE オプションは、バッファーのバイト数を指定します。



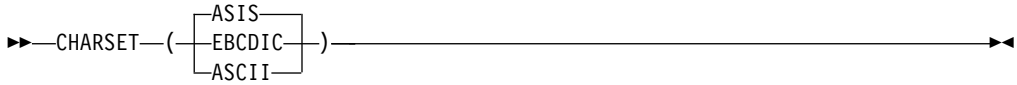
RECORD 出力はデフォルト設定でバッファーに入り、BUFSIZE のデフォルト値 64k です。 STREAM 出力はバッファーに入れられますが、デフォルトでは入れられません。 STREAM 出力のデフォルト BUFSIZE 値はゼロです。

BUFSIZE の値がゼロの場合、バッファーのバイト数は、RECSIZE オプションまたは LRECL オプションで指定された値と同じです。

BUFSIZE オプションが有効なのは、連続バイナリー・ファイルだけです。ファイルが端末入力に使用されている場合は、効率を上げるために、BUFSIZE に値をゼロを割り当てるべきです。

レコード入出力用の CHARSET

このバージョンの CHARSET オプションは、レコード入出力が使用される連続ファイルにのみ適用されます。ユーザーは、このオプションを使用することで、ASCII データ・ファイルを入力ファイルとして使用したり、出力ファイルの文字セットを指定したりできます。



入力ファイルの形式、または出力ファイルにとらせたい形式に基づいて、CHARSET のサブオプションを選択してください。

ストリーム入出力用の CHARSET

CHARSET オプションのこのバージョンは、ストリーム入力ファイルおよびストリーム出力ファイルにだけ適用されます。 ユーザーは、このオプションを使用することにより、ASCII データ・ファイルを入力ファイルとして使用したり、出力ファイルの文字セットを指定したりできます。

ストリーム入出力を使用しているときに ASIS を指定しようとしても、エラーは出されず、文字セットは EBCDIC として扱われます。

入力ファイルの形式、または出力ファイルにとらせたい形式に基づいて、CHARSET のサブオプションを選択してください。

DELAY

DELAY オプションは、システムでファイルやレコードのロックを取得できない場合に、失敗した操作をコンパイラーが再試行するのを遅延させる時間 (ミリ秒) を指定します。

このオプションを適用できるのは、VSAM ファイルだけです。

DELIMIT

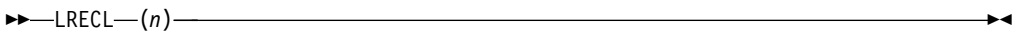
DELIMIT オプションは、入力ファイルにフィールド区切り文字が含まれているかを指定します。

フィールド区切り文字は、レコードのフィールドを分離するブランクかまたはユーザー定義文字です。 このオプションを適用できるのは、ソート入力ファイルだけです。

ソート・ユーティリティー・プログラムは、フィールド区切り文字の有無により、テキスト・ファイルとバイナリー・ファイルを区別します。 フィールド区切り文字が含まれている入力ファイルは、テキスト・ファイルとして処理され、区切り文字がない入力ファイルはバイナリー・ファイルと見なされます。 この情報は、ライブラリーが正しいパラメーターをソート・ユーティリティー・プログラムに渡すために必要です。

LRECL

LRECL オプションは RECSIZE オプションと同じです。



LRECL が指定されておらず LINESIZE 値による暗黙指定もされていない場合 (ただし TYPE(FIXED) ファイルを除く)、デフォルトは 1024 です。

LRMSKIP

LRMSKIP オプションを使用すれば、ファイルがオープンされた後で最初に実行される SKIP フォーマット項目の先頭ページの *n* 行目で出力を開始できます。 *n* は、PUT ステートメントまたは GET ステートメントの SKIP オプションで指定された値を指します。

n がゼロまたは 1 の場合、出力は先頭ページの 1 行目から開始されます。

PROMPT

PROMPT オプションは、コロンを端末からのストリーム入力のプロンプトとして表示するかどうかを指定します。

PUTPAGE

PUTPAGE オプションは、用紙送り文字の後ろに復帰文字を入れるかどうかを指定します。このオプションが適用されるのは、プリンター向けファイルだけです。

プリンター向けファイルは、PRINT 属性を指定して宣言されたストリーム出力ファイル、あるいは、CTLASA 環境オプションを指定して宣言されたレコード出力ファイルです。

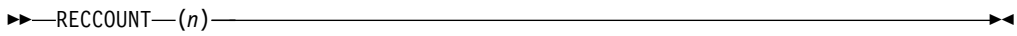
NOCR
用紙送り文字 ('0C'x) の後ろに復帰文字 ('0D'x) を入れないことを指します。

CR 用紙送り文字の後ろに復帰文字を追加することを指します。 このオプションは、出力が IBM 以外のプリンターに送られる場合に指定する必要があります。

RECCOUNT

RECCOUNT オプションは、PL/I ファイルのオープン・プロセス中に作成される、相対データ・セットまたは領域データ・セットにロードできる最大のレコード数を指定します。

注: PL/I V5.1 では、RECCOUNT オプションは 64 ビット・プログラムに対してサポートされていません。



PL/I がデータ・セットを作成も再作成もしない場合、RECCOUNT オプションは無視されます。

RECCOUNT オプションのデフォルトは 50 です。

注: z/OS において REGIONAL(1) データ・セットの機能性とパフォーマンスを向上させるには、/filespec パラメーターと RECCOUNT パラメーターの両方が指定された TITLE オプションを省略することをお勧めします。この場合、ファイルにロードされるレコードの数は、データ・セットの最初のエクステントに割り振られたスペースによって決まります。 詳しくは、 317 ページの『第 13 章 領域データ・セットの定義と使用』を参照してください。

RECSIZE

RECSIZE オプションは、データ・セット内のレコードの長さを指定します。領域データ・セットおよび固定長データ・セットの場合は、RECSIZE はデータ・セットの

各レコードの長さを指定します。その他のデータ・セット・タイプの場合は、RECSIZE はレコードがとれる最大の長さを指定します。

SAMELINE

SAMELINE オプションは、入力を求めるプロンプトのステートメントと同じ行で、システム・プロンプトを行わせるかどうかを指定します。

以下の例は、PROMPT オプションと SAMELINE オプションのいくつかの組み合わせの結果を示しています。

例 1

`PUT SKIP LIST('ENTER:');` というステートメントが与えられると、その出力結果は次のようになります。

prompt(y) sameline(y)	ENTER: (cursor)
prompt(y) sameline(n)	ENTER: (cursor)
prompt(n) sameline(y)	ENTER: (cursor)
prompt(n) sameline(n)	ENTER: (cursor)

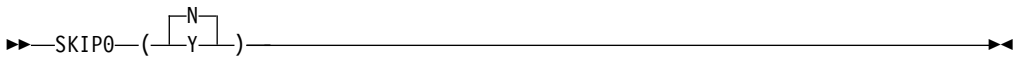
例 2

`PUT SKIP LIST('ENTER');` というステートメントが与えられると、その出力結果は次のようになります。

prompt(y) sameline(y)	ENTER: (cursor)
prompt(y) sameline(n)	ENTER : (cursor)
prompt(n) sameline(y)	ENTER (cursor)
prompt(n) sameline(n)	ENTER (cursor)

SKIP0

SKIP0 オプションは、ソース・プログラムに SKIP(0) ステートメントがコーディングされた場合、ライン・カーソルをどこに移動するかを指定します。 SKIP0 オプションは、PM アプリケーションとしてリンクされていない端末ファイルに適用されます。



SKIP0(N)
カーソルを次の行の先頭に移動することを指定します。

SKIP0(Y)
カーソルを現在行の先頭に移動することを指定します。

次の例は、現在の出力行の先頭にカーソルが移動するように、出力を端末スキップ・ゼロ行で行う方法を示しています。

```
export DD_SYSPRINT='stdout:,SKIP0(Y)'
```

TYPE

TYPE オプションは、ネイティブ・ファイル内のレコードのフォーマットを指定します。

CRLF
レコードを文字の組み合わせ CR - LF で区切ることを指定します。 ('CR' と 'LF' は、それぞれ復帰と改行の ASCII 値である '0D'x と '0A'x を表します。) 出力ファイルの場合、PL/I は各レコードの終わりにこれらの文字を挿入します。入力ファイルの場合、PL/I はこれらの文字を破棄します。 入力、出力のいずれの場合も、これらの文字は RECSIZE の考慮事項には入りません。

データ・セットのレコード長として決められている値よりも長いレコードをデータ・セットに入れることはできません。

LF レコードが LF 文字組み合わせで区切られることを指定します。 ('LF' は、ASCII コードの用紙送り、つまり '0A'x を表します。) 出力ファイルの場合、PL/I は各レコードの終わりにこれらの文字を挿入します。入力ファイルの場合、PL/I はこれらの文字を破棄します。 入力、出力のいずれの場合も、これらの文字は RECSIZE の考慮事項には入りません。

データ・セットのレコード長として決められている値よりも長いレコードをデータ・セットに入れることはできません。

TEXT
前述の LF と同じです。

FIXED
データ・セット内の各レコードの長さが同じであることを指定します。データ・セット内のレコード長として指定されている値は、レコードの境界を認識する場合に使用されます。

TYPE(FIXED) ファイル内のすべての文字は、制御文字も含め (ある場合)、データと見なされます。 指定したレコード長が、存在している文字を反映していること、あるいは、指定したレコード長がレコード内の全文字を扱えることを確認してください。

CRLFEOF
出力ファイルを除けば、このサブオプションは CRLF オプションと同じ情報を指定します。 ファイルの 1 つが出力についてクローズされるときに、ファイルの終わりマーカーが最後のレコードに追加されます。

U レコードが不定形式であることを表します。 これらの不定形式ファイルは、OPEN および CLOSE を除いて、どのレコード入出力またはストリーム入出力のステートメントでも使用できません。 TYPE(U) ファイルからの読み取りは、FILEREAD 組み込み関数を使用することのみにより行うことができます。 また、TYPE(U) ファイルへの書き込みは、FILEWRITE 組み込み関数を使用することのみにより行うことができます。

ASA(N) オプションを指定したプリンター向けファイルの場合このオプションが無視されるということを除き、TYPE オプションを適用できるのは CONSECUTIVE ファイルだけです。

使用しているプログラムが TYPE(FIXED) が有効である既存のデータ・セットにアクセスしようとしており、かつそのデータ・セット長がユーザーが指定した複数の論理レコード長の倍数でない場合は、PL/I は UNDEFINEDFILE 条件を発生させます。

TYPE(FIXED) 属性を持つ非印刷ファイルが使用されている場合は、SKIP が行の終わりまで末尾ブランクで置き換えられます。 TYPE(LF) が使用されている場合、SKIP は末尾ブランクなしに、LF で置き換えられます。

データ・セット特性の設定

データ・セットは、オペレーティング・システムのデータ管理ルーチンが理解できる特定のフォーマットで保管されているレコードから構成されています。ユーザーのプログラムでファイルの宣言またはオープンを行うときに、ユーザーはそのファイルに入っているレコードの特性を PL/I およびオペレーティング・システムに記述します。

また、JCL または OPEN ステートメントの TITLE オプションの式を使って、データ・セット内またはデータ・セットに関連付けられている PL/I ファイル内のデータ特性をオペレーティング・システムに記述することもできます。

必ずしも、プログラムの内と外の両方で、自分のデータを記述する必要はありません。多くの場合、1 回の記述が、データ・セットとその関連 PL/I ファイルの両方に役立ちます。 実際、データの特性は一箇所のみに記述する方が有利です。

使用するプログラム・データおよびデータ・セットを効率よく記述するには、オペレーティング・システムがどのようにデータを移動および保管するのかを理解する必要があります。

ブロックおよびレコード

データ・セット内のデータ項目は、ブロック間ギャップ (IBG) で区切られているブロックに配置されます。 (これをレコード間ギャップと呼んでいるマニュアルもあります。)

ブロック は、データ・セットに送られてくる、あるいはデータ・セットから送り出されるデータの単位です。 各ブロックには、1 つのレコード、レコードの一部分、または複数のレコードが入っています。 ブロック・サイズは、DD ステートメントの BLKSIZE パラメーター内、あるいは ENVIRONMENT 属性の BLKSIZE オプション内で指定することができます。

レコード は、プログラムに送られてくる、またはプログラムから送り出されるデータの単位です。 レコード長は、DD ステートメントの LRECL パラメーター内、OPEN ステートメントの TITLE オプション内、または ENVIRONMENT 属性の RECSIZE オプション内で指定することができます。

PL/I プログラムを作成する場合は、読み取りあるいは書き込みを行うレコードだけを考慮すれば済みます。しかし、自分のプログラムが作成あるいはアクセスするデータ・セットを記述する場合は、ブロックおよびレコード間の関係を知っている必要があります。

ブロック化によって、磁気ストレージ・ボリューム内のストレージ・スペースを節約することができます。ブロック化が、ブロック間ギャップ数を減らし、データ・セットを処理するのに必要な入出力操作回数を減らすことにより効率を上げるためです。また、レコードは、データ管理ルーチンにより、ブロック化およびブロック化解除されます。

情報交換コード

データが記録される通常のコードは拡張 2 進化 10 進コード (EBCDIC) です。

ASCII コードの各文字は 7 ビット・パターンで表されるので、このようなパターンが 128 通りあります。 また、ASCII セットには、有効な ASCII コードがない EBCDIC 文字を表すために使用される置換文字 (SUB 制御文字) があります。 ASCII 置換文字は、00111111 というビット・パターンを持つ EBCDIC SUB 文字に変換されます。

レコード・フォーマット

データ・セット内のレコードは、次のいずれかのフォーマットを持っています。

固定長
可変長
不定長

レコードは、必要に応じて、ブロック化することができます。 オペレーティング・システムは、固定長レコードと可変長レコードを非ブロック化しますが、不定長レコードを非ブロック化するには、ユーザーのプログラム内にコードを提供する必要があります。

レコード・フォーマットは、DD ステートメントの RECFM パラメーター内、OPEN ステートメントの TITLE オプション内、または ENVIRONMENT 属性のオプションとして指定します。

固定長レコード

固定長レコードには、次に挙げるフォーマットを指定することがでます。
F 固定長、非ブロック化
FB 固定長、ブロック化
FS 固定長、非ブロック化、標準
FBS 固定長、ブロック化、標準

固定長レコードを持っているデータ・セットの場合は、図 26 に示すように、すべてのレコードの長さが等しくなります。レコードがブロック化されていると、通常、各ブロックには同じ数の固定長レコードが入っています (ただし、ブロックは切り捨てられる場合もあります)。 レコードがブロック化されていない場合は、各レコードがブロックを構成します。

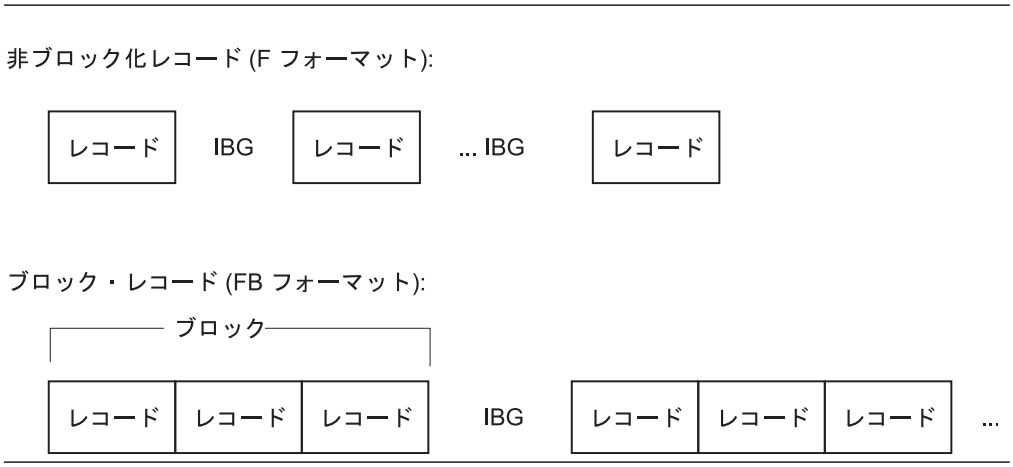


図 *26.* 固定長レコード

オペレーティング・システムでは、一定のレコード長に基づいてブロック化、非ブロック化が行われるため、可変長レコードより固定長レコードの方が速く処理されます。

可変長レコード

指定できる可変長レコードのフォーマットは、次のとおりです。
V 可変長、非ブロック化
VB 可変長、ブロック化
VS 可変長、非ブロック化、スパン
VBS 可変長、ブロック化、スパン

V フォーマットでは、可変長レコードと可変長ブロックの両方を使用することができます。 各レコードの 4 バイトの接頭部と、各ブロックの最初の 4 バイトには、オペレーティング・システムが使用する場合の制御情報が入ります (レコードまたはブロックのバイト単位の長さを含む)。 このような制御フィールドのため、可変長レコードは逆方向に読み込むことができません。

V フォーマットは、非ブロック化可変長レコードを表します。 各レコードは、レコードが 1 つだけを持つブロックとして扱われます。 ブロックの最初の 4 バイトにはブロック制御情報が入り、次の 4 バイトにはレコード制御情報が入ります。

VB フォーマットは、ブロック化可変長レコードを表します。 各ブロックには、そのブロックに収容可能なレコード数と同じ数のレコードが入ります。 ブロックの最初の 4 バイトにはブロック制御情報が入り、各レコードの 4 バイト接頭部にはレコード制御情報が入ります。

スパン・レコード: スパン・レコードは可変長レコードで、レコードの長さがブロック・サイズを超えることもできます。超えた場合は、レコード・フォーマットを VS または VBS のいずれかで指定して、レコードを複数のセグメントに分割し、2 つ以上の連続ブロックに置きます。 セグメンテーションおよび組み立ては、オペレーティング・システムによって処理されます。 スパン・レコードを使用すると、レコードの長さを問わず、ブロック・サイズが選択でき、補助記憶域を最大限に活用し、伝送の効果を最大限に高めます。

VS フォーマットは、V フォーマットに似ています。 それぞれのブロックは、1 つのレコードまたはレコードのセグメントのみを含みます。 ブロックの最初の 4 バイトにはブロック制御情報が入り、次の 4 バイトにはレコードまたはセグメント制御情報 (レコードが単体のものか、または最初、中間、最後のセグメントであるかということを示す情報を含む) が入ります。

VBS フォーマットでは、各ブロックが、単体のレコードまたはセグメントをできるだけ多く保持することができるという点において、VS フォーマットと異なっています。したがって、各ブロックのサイズがほとんど同じになります (ただし、各セグメントは最低 1 バイトのデータを含まなければならないため、最大で 4 バイトの変化幅があります)。

不定長レコード

U フォーマットでは、F フォーマットにも V フォーマットにも当てはまらないレコードを処理することができます。 オペレーティング・システムおよびコンパイラーは各ブロックをレコードとして扱います。ユーザー・プログラムで必要に応じてブロック化あるいは非ブロック化を行わなくてはなりません。

データ・セットの編成

オペレーティング・システムのデータ管理ルーチンは、いくつかのタイプのデータ・セットを処理できます。これらのデータ・セットは、その内部でのデータ保管方法、およびデータにアクセスするために許可される方法において異なります。

非 VSAM データ・セットの主な 3 つタイプと、それに対応するキーワード (PL/I 編成についての説明) を以下の表に示します[1]。

データ・セットのタイプ	**PL/I** 編成
順次	CONSECUTIVE または ORGANIZATION (連続)
索引	INDEXED または ORGANIZATION (索引)
直接	REGIONAL または ORGANIZATION (相対)

データ・セットの 4 つ目のタイプ 区分 には、対応する PL/I 編成はありません。

また、PL/I では、3 つのタイプの VSAM データ編成 (*ESDS*、*KSDS*、および *RRDS*) もサポートされています。 VSAM データ・セットの詳細については、331 ページの『第 14 章 VSAM データ・セットの定義と使用』を参照してください。

順次 (つまり CONSECUTIVE) データ・セット内では、各レコードは物理的順序で設定されます。あるレコードが与えられた場合、そのレコードの次のレコード位置は、そのレコードが物理的にデータ・セットのどこにあるかによって決まります。直接アクセス装置に対して順次編成を選択することができます。

索引順次 (または INDEXED) データ・セットは、直接アクセス・ボリューム上になければなりません。 オペレーティング・システムが維持する索引や索引のセットにより、特定の基本レコードの位置が与えられます。これによって、順次処理だけでなく、レコードの直接検索、置換、追加、および削除を行うことができます。

直接 (または REGIONAL) データ・セットは、直接アクセス・ボリューム上になければなりません。 データ・セットは複数の領域に分割され、各領域には 1 つ以上のレコードが入っています。領域番号を指定するキーを使えば、任意のレコードに直接アクセスすることができます。順次処理を行うことも可能です。

区分 データ・セットでは、順次編成されるデータから成り、それぞれがメンバーと呼ばれる独立したグループは、直接アクセス・データ・セット内に存在します。このタイプのデータ・セットには、各メンバーの位置をリストするディレクトリーが 1 つ含まれています。 区分データ・セットはしばしば ライブラリー と呼ばれます。このコンパイラーには、区分データ・セットを作成したり、区分データ・セットにアクセスするための特殊機能はありません。 各メンバーは、CONSECUTIVE データ・セットとして PL/I プログラムが処理することができます。 区分データ・セットをライブラリーとして使用する方法について詳しくは、273 ページの『第 9 章 ライブラリーの使用』を参照してください。

ラベル

オペレーティング・システムは内部ラベルを使って、直接アクセス・ボリュームを識別したり、データ・セット属性 (例えば、レコード長やブロック・サイズ) を保管します。 これらの属性情報は、最初は DD ステートメントまたはユーザー・プログラムから入手する必要があります。

---

1. 順次 および直接 という用語を PL/I ファイル属性 SEQUENTIAL および DIRECT と混同しないでください。この属性は、ファイルの処理方法を指すための属性であり、対応するデータ・セットの編成方法を指す属性ではありません。

IBM 標準ラベルには、初期ボリューム・ラベルとヘッダー・ラベルという 2 つの部分があります。初期ボリューム・ラベルは、特定のボリュームとその所有者を識別します。一方、ヘッダー・ラベルはボリューム上の各データ・セットの前後にあります。 ヘッダー・ラベルには、システム情報、装置依存情報 (記録手法など)、およびデータ・セット特性が含まれています。

直接アクセス・ボリュームには、IBM 標準ラベルが付きます。ボリューム・ラベルで各ボリュームが識別されます。 このラベルには、ボリューム通し番号とボリューム目録 (VTOC) のアドレスが入っています。 同様に、この目録には、ボリュームに保管されているデータ・セットごとにラベル (データ・セット制御ブロック (DSCB) と呼ばれる) が含まれています。

データ定義 (DD) ステートメント

データ定義 (DD) ステートメントは、オペレーティング・システムに対してデータ・セットを定義するためのジョブ制御ステートメントであり、入出力リソースの割り振りをオペレーティング・システムに要求するものです。 データ・セットを動的に割り振らない場合は、各ジョブ・ステップに、そのジョブ・ステップで処理するデータ・セットの DD ステートメントをすべて組み込んでおく必要があります。

ジョブ制御ステートメントの構文については、「*z/OS MVS JCL* ユーザーズ・ガイド」を参照してください。 DD ステートメントのオペランド・フィールドには、データ・セットの位置 (例えば、ボリューム通し番号や、ボリュームをマウントする装置の識別名) を記述したキーワード・パラメーター、およびデータそのものの属性 (例えば、レコード・フォーマット) を記述したキーワード・パラメーターを入れることができます。

DD ステートメントを使用すると、使用するデータ・セットおよび入出力装置から独立した PL/I ソース・プログラムを作成することができます。 また、ユーザー・プログラムを再コンパイルしなくても、データ・セットのパラメーターを変更したり、さまざまなデータ・セットを処理したりすることができます。

ENVIRONMENT 属性の LEAVE および REREAD オプションを使用すると、磁気テープ・ボリュームの終わりに達したとき、または磁気テープのデータ・セットが閉じたときにとるアクションを制御する DISP パラメーターを使用できます。 LEAVE オプションおよび REREAD オプションについては、 310 ページの『LEAVE|REREAD』を参照してください。

PL/I バージョン 1 の標準機能であった書き込み妥当性検査は、本バージョンでは実行されません。 書き込み妥当性検査を実行するには、JCL DD ステートメントの DCB パラメーターの OPTCD サブパラメーターを使って要求できます。資料「*OS/VS2 Job Control Language*」を参照してください。

条件付きサブパラメーターの用法

PL/I プログラムにより処理されるデータ・セットの DISP パラメーターの条件付きサブパラメーターを使用する場合は、ステップ異常終了機能を使う必要があります。

ステップ異常終了機能は、次のようなにして入手します。

1. アプリケーションの条件付きサブパラメーターを適用する必要がある障害が発生したためにプログラムの実行を終了する場合は常に、ERROR 条件を発生させるか、または ERROR 信号を出す。
2. PL/I のユーザー出口を ABEND 要求に変更する必要がある。

データ・セット特性

DD ステートメントの DCB (データ制御ブロック) パラメーターを使用すると、データ・セット内のデータの特性を記述したり、実行時のデータの処理方法を記述することができます。DD ステートメントの他のパラメーターは、主としてデータ・セットの識別、位置、処置を扱うのに対し、DCB パラメーターはレコード自体の処理に必要な情報を指定します。

DCB パラメーターのサブパラメーターについては、「*OS/VS2 Job Control Language*」を参照してください。

DCB パラメーターには、以下のデータ特性を記述するサブパラメーターが含まれます。

- データ・セットの編成とそのアクセス方法 (サブパラメーター CYLOFL、DSORG、LIMCT、NTM、および OPTCD)
- プリンターの行送りなどの装置依存情報 (サブパラメーター CODE、FUNC、MODE、OPTCD=J、 PRTSP、STACK、および STACK)
- レコード・フォーマット (サブパラメーター BLKSIZE、KEYLEN、LRECL、および RECFM)
- 各レコードの最初のバイト (RECFM サブパラメーター) に挿入される ASA 制御文字 (ある場合)

BLKSIZE、LRECL、KEYLEN、および RECFM (またはそれらと同等のもの) は、DCB パラメーターではなく、ユーザーの PL/I プログラムのファイル宣言の ENVIRONMENT 属性を使って指定することができます。

DCB パラメーターを使って、PL/I プログラム内でデータ・セット用に (宣言されたファイル属性と、その属性で暗黙指定されたその他の属性を使って) 既に設定されている情報を指定変更することはできません。 既に指定されている情報を変更しようとする DCB サブパラメーターは無視されます。

新規データ・セットに関しては、DD ステートメントとの競合が発生する場合は、プログラムで定義されたファイルの属性が使用されます。

PDS ファイルをクローズするときに、RC=4 を伴うメッセージ IEC225I が出される場合があります。このメッセージは安全であり、無視できます。

次の DCB パラメーターの例は、長さ 40 バイトの固定長レコードが 400 バイトの長さのブロックにグループ化されることを指定しています。

```
DCB=(RECFM=FB,BLKSIZE=400,LRECL=40)
```

OPEN ステートメントの TITLE オプションの使用

OPEN ステートメントの TITLE オプションを使用すると、PL/I ファイルに関連付けられるデータ・セットを識別することができます。また、オプショナルで追加のデータ・セット特性も与えることがきます。

関連情報:

241 ページの『OPEN ステートメントの TITLE オプションの使用』
OPEN ステートメントの TITLE オプションを使用すると、PL/I ファイルに 関連付けるデータ・セットを識別することができます。また、オプショナルで データ・セットの特性も設定することができます。

PL/I ファイルとデータ・セットの関連付け

PL/I OPEN ステートメントを実行することにより、ファイルとデータ・セットを関連付けることができます。 PL/I CLOSE ステートメントを実行することにより、関連付けられていたデータ・セットからファイルが切り離されます。

ファイルのオープン

PL/I OPEN ステートメントを実行することにより、ファイルとデータ・セットを関連付けることができます。これを行うには、ファイルを記述している情報とデータ・セットを記述している情報をマージする必要があります。ファイル属性とデータ・セット特性の間に矛盾が検出されると、UNDEFINEDFILE 条件が発生します。

PL/I ライブラリーのサブルーチンは、データ・セットの骨組みデータ制御ブロックが作成されます。このサブルーチンは、DECLARE ステートメントおよび OPEN ステートメントから得られるファイル属性と、宣言された属性で暗黙指定される属性を使用して、可能な限りデータ制御ブロックを完成させます。 (258 ページの図 27 を参照してください。) その後で、このサブルーチンは OPEN マクロ命令を発行します。このマクロ命令は、データ管理ルーチンを呼び出して、正しいボリュームがマウントされていることを検査し、データ制御ブロックを完成させます。

データ管理ルーチンは、さらに必要な情報がないか調べるためにデータ制御ブロックを検査し、次に、まず DD ステートメント内からその情報を検索し、最後に、データ・セットが存在するかどうか、かつデータ・セット・ラベル内に標準ラベルが付いているかどうかを調べます。 新規データ・セットの場合は、データ管理ルーチンは、ラベル (必要な場合) を作成し、そのラベルにデータ制御ブロックからの情報を入れます。

INPUT データ・セットの場合、属性が矛盾しない限り、PL/I プログラムは DCB 属性を指定変更できます。 OUTPUT データ・セットの場合、PL/I プログラムは DCB 属性を指定変更できません。ただし、データ・セットのオープン時に欠落 DCB 属性があれば、その属性は PL/I プログラムから取得されます (プログラムでその属性が指定されている場合)。

DCB フィールドがこれらのソースの情報で埋められると、制御は PL/I ライブラリー・サブルーチンに戻ります。依然として値が入力されていないフィールドには、PL/I OPEN サブルーチンがデフォルトの情報を提供します。例えば、LRECL が指定されていない場合は、BLKSIZE に与えられた値が提供されます。

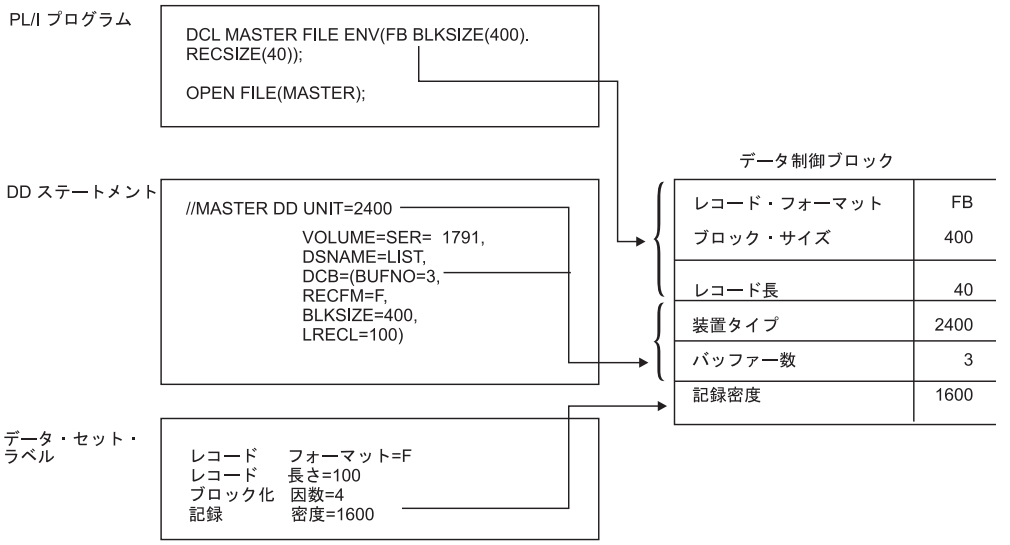


図 27. オペレーティング・システムによる *DCB* への情報の組み込みの方法

システム決定ブロック・サイズの使用**:**

データ機能プロダクト (DFP) のシステム決定ブロック・サイズ機能を z/OS 上で使用する場合は、割り当てられている装置タイプに最適なブロック・サイズが DFP により算出されます。

新規 DASD データ・セットを作成する場合、システムは最適なブロック・サイズを導き出し、以下の条件がすべて真である場合に、それをデータ・セット・ラベルに保存します。

- ブロック・サイズが利用できないか、ソースから指定されていません。BLKSIZE=0 を指定できます。
- LRECL を指定するか、それがデータ・クラスにあります。データ・セットは、SMS 管理される必要はありません。
- RECFM を指定するか、それがデータ・クラスにあります。固定または可変である必要があります。
- DSORG を PS または PO として指定します。 DSORG を省略した場合、DSORG はデータ・クラスにおける PS または PO です。

システム決定ブロック・サイズがアクティブであれば、DFP がブロック・サイズを決定し、それをデータ・セット・ラベルに置いてから、PL/I でファイルを開きます。 したがって、PL/I プログラムが ENVIRONMENT オプションによってブロック・サイズを指定する場合は、システム決定ブロック・サイズの値と競合するようなことがあってはなりません。

システム決定ブロック・サイズについて詳しくは、「*z/OS DFSMS* データ・セットの使用法」を参照してください。

ファイルのクローズ

PL/I CLOSE ステートメントを実行することにより、関連付けられていたデータ・セットからファイルが切り離されます。

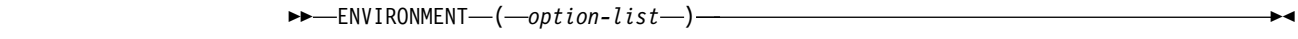
PL/I ライブラリー・サブルーチンは、最初に CLOSE マクロ命令を発行します。データ管理ルーチンから制御が戻ると、そのサブルーチンは、ファイルのオープン時に作成されたデータ制御ブロックを解放します。データ管理ルーチンは、新規データ・セットのラベルの作成を完了し、既存データ・セットのラベルを更新します。

ENVIRONMENT 属性での特性の指定

ENVIRONMENT 属性でさまざまなオプションを使用してデータ特性を指定できます。ファイルのタイプごとに、さまざまな属性および環境オプションがあります。

ENVIRONMENT 属性

PL/I ファイル宣言ファイルの ENVIRONMENT 属性を使用すれば、ファイルに関連付けられているデータ・セットの物理編成に関する情報を指定したり他の関連情報を記述したりできます。この情報のフォーマットは、括弧で囲んだオプション・リストでなければなりません。



省略形: ENV

ブランクまたはコンマで区切ったオプションを、任意の順序で指定することができます。

次の例は、完全なファイル宣言のコンテキストにおける ENVIRONMENT 属性の構文を示したものです (指定されているオプションは VSAM 用です)。

```
DCL FILENAME FILE RECORD SEQUENTIAL
    INPUT ENV(VSAM GENKEY);
```

260 ページの表 15 は、ENVIRONMENT オプションとファイル属性を要約したものです。 使用に関する特定の制限については、表に含まれる注記およびコメントを参照してください。

表 *15. PL/I* ファイル宣言の属性

データ・セット・タイプ	ストリーム	レコード									
		順次						直接			
		連続									
ファイル・タイプ	**Consecutive**	**Buffered**	**Unbuffered**	**Regional**	**Teleprocessing**	**Indexed**	**VSAM**	**Regional**	**Indexed**	**VSAM**	暗黙指定属性
ファイル属性[1]											
ファイル	I	I	I	I	I	I	I	I	I	I	
入力[1]	D	D	D	D	D	D	D	D	D	D	ファイル
出力	O	O	O	O	O	O	O	O	O	O	ファイル
環境	I	I	I	S	S	S	S	S	S	S	ファイル
ストリーム	D	-	-	-	-	-	-	-	-	-	ファイル
印刷[1]	O	-	-	-	-	-	-	-	-	-	ファイル・ストリーム出力
レコード	-	I	I	I	I	I	I	I	I	I	ファイル
更新	-	O	O	O	-	O	O	O	O	O	ファイル・レコード
順次	-	D	D	D	-	D	D	-	-	D	ファイル・レコード
バッファー付き	-	D	-	-	I	D	D	-	-	S	ファイル・レコード
キー順[2]	-	-	-	O	I	O	O	I	I	O	ファイル・レコード
直接	-	-	-	-	-	-	S	S	S	S	キー付きファイル・レコード

ENVIRONMENT オプション											コメント
F|FB|FS|FBS|V|VB|VS|VBS||U	I	S	S	-	-	-	N	-	-	N	VS および VBS はストリームでは無効
F|FB|U	S	S	-	-	-	-	N	-	-	N	ASCII データ・セットのみ
F|V|U	-	-	-	S	-	-	N	S	-	N	REGIONAL(1) の場合は F のみ
F|FB|V|VB	-	-	-	-	-	S	N	-	S	N	
RECSIZE(n)	I	I	I	I	S	I	C	I	I	C	連続ファイル、索引付きファイル、および領域ファイルに対しては、RECSIZE と BLKSIZE の一方または両方を指定する必要がある
BLKSIZE(n)	I	I	I	I	-	I	N	I	I	N	

凡例:

C	**VSAM** では検査
D	デフォルト
I	指定または暗黙指定が必要
N	**VSAM** では無視
O	オプション
S	指定が必要
-	無効

SCALARVARYING	-	O	O	O	-	O	O	O	O	O	ASCII データ・セットの場合は無効
CONSECUTIVE	D	D	D	-	-	-	O	-	-	O	VSAM ESDS の場合は指定可能
LEAVE|REREAD	O	O	O	-	-	-	-	-	-	-	
CTLASA|CTL360	-	O	O	-	-	-	-	-	-	-	ASCII データ・セットの場合は無効
GRAPHIC	O	-	-	-	-	-	-	-	-	-	
INDEXED	-	-	-	-	-	S	O	-	S	O	VSAM ESDS の場合は指定可能
KEYLOC(n)	-	-	-	-	-	O	-	-	O	-	
ORGANIZATION	D	-	-	-	-	-	-	-	-	-	
GENKEY	-	-	-	-	-	O	O	-	O	O	INPUT または UPDATE ファイルのみ (KEYED が必要)
REGIONAL(1)	-	-	-	S	-	-	-	S	-	-	
VSAM	-	-	-	-	-	-	S	-	-	S	
BKWD	-	-	-	-	-	-	O	-	-	O	
REUSE	-	-	-	-	-	-	O	-	-	O	OUTPUT ファイルのみ

注:

1. INPUT 属性が指定されているファイルは PRINT 属性をとることはできません。
2. INDEXED 出力および REGIONAL 出力にはキーが必要です。

複数のデータ・セット編成に適用される ENVIRONMENT オプションについては、後続のトピックで説明します。また、各オプションについては、オプションが適用される各データ・セット編成とともに後続のセクションで説明します。

関連情報:

331 ページの『第 14 章 VSAM データ・セットの定義と使用』
この章では、レコード単位データ伝送用 の VSAM (仮想記憶アクセス方式) 編成、VSAM ENVIRONMENT オプション、 他の PL/I データ・セット編成との 互換性、および PL/I がサポートする 3 つのタイプの VSAM データ・セット (入力順、キー順、および相対レコード) をロードしそれに アクセスするのに使用するステートメントについて述べます。

データ・セット編成オプション**:** 以下のオプションは、データ・セット編成を指定します。

各オプションについては、適用されるデータ・セット編成の項目で解説されています。

その他の **ENVIRONMENT** オプション**:**

ブロック・サイズやレコード長など、整数引数が必要な ENVIRONMENT オプションは、定数や変数と併用することができます。変数には、添え字や修飾を付けたりすることはできません。変数には、属性 FIXED BINARY(31,0) および STATIC が必要です。

ENVIRONMENT オプションおよび同等の DCB パラメーターを以下にリストします。

ENVIRONMENT オプション	**DCB** サブパラメーター
レコード・フォーマット	RECFM[1]
RECSIZE	LRECL
BLKSIZE	BLKSIZE
CTLASA|CTL360	RECFM
KEYLENGTH	KEYLEN

注: [1]VS は、DCB 中ではなく、 ENVIRONMENT オプションとして指定されなければなりません。

レコード単位データ伝送のレコード・フォーマット

サポートされるレコード・フォーマットは、データ・セット編成により異なります。

レコードのフォーマットは、次のいずれかです。

固定長	F	非ブロック化
	FB	ブロック化
	FS	非ブロック化、標準
	FBS	ブロック化、標準
可変長	V	非ブロック化
	VB	ブロック化
	VS	スパン
	VBS	ブロック化、スパン
不定長	U	(ブロック化できない)

U フォーマット・レコードが可変長ストリングに読み込まれる場合は、PL/I はストリングの長さを検索されたデータのブロック長に設定します。

上記のレコード・フォーマット・オプションは、VSAM データ・セットには適用されません。 VSAM データ・セットに関連付けられているファイルにレコード・フォーマット・オプションを指定すると、そのオプションは無視されます。

VS フォーマットのレコードは、連続編成のデータ・セットにのみ指定できます。

ストリーム指向データ伝送のレコード・フォーマット

ストリーム指向データ伝送のレコード・フォーマット・オプションについては、281 ページの『ストリーム指向データ伝送の用法』を参照してください。

RECSIZE オプション

RECSIZE オプションは、レコード長を指定します。

VSAM データ・セットに関連付けられているファイルに関しては、*record-length* は次の値の合計です。

1. データに必要な長さ

   可変長レコードおよび不定長レコードの場合は、この長さが最大長になります。

2. 必要な制御バイト

   可変長レコードには 4 バイトが必要です (レコード長の接頭部用)。固定長レコードおよび不定長レコードでは必要ありません。

VSAM データ・セットの場合、データ・セットが定義されるときに、レコードの最大長と平均長がアクセス方式サービス・プログラム・ユーティリティーに指定されます。 検査の目的で RECSIZE オプションをファイル宣言に組み込む場合は、レコードの最大長を指定する必要があります。指定した RECSIZE がデータ・セットに定義されている値と矛盾する場合は、UNDEFINEDFILE 条件が発生します。

record-length は、整数として、または属性 FIXED BINARY(31,0) STATIC を持つ変数として指定できます。

レコード長の値には、次のような規則があります。

最大長**:**

固定長、および不定長 (ASCII データ・セットを除く): 32760

UPDATE ファイルを持つ V フォーマット、ならびに VS および VBS フォーマット: 32756

入力ファイルおよび出力ファイルを持つ VS および VBS フォーマット: 16777215

ASCII データ・セット: 9999

VSAM データ・セット: 32761

注**:** 32,756 バイトより長い VS フォーマット・レコードおよび VBS フォーマット・レコードに関しては、ENVIRONMENT の RECSIZE オプションで長さを指定し、DD ステートメントの DCB サブパラメーターに対して LRECL=X を指定する必要があります。 RECSIZE が INPUT または OUTPUT に対して許可されている最大値を超える場合は、レコード条件が発生するか、またはレコードが切り捨てられます。 UPDATE ファイルは LRECL=X ではサポートされていません。

ゼロ値:

有効な値は、最初に、ファイルに関連付けられているデータ・セットの DD ステートメントで検索され、次にデータ・セット・ラベルで検索されます。

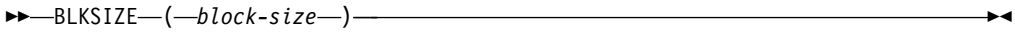
上記のいずれかで値を入手できない場合は、デフォルト・アクションが行われます (265 ページの『レコード・フォーマット、BLKSIZE、および RECSIZE のデフォルト』参照)。

負の値:

UNDEFINEDFILE 条件が発生します。

BLKSIZE オプション

BLKSIZE オプションは、データ・セット上の最大ブロック・サイズを指定します。



block-size (ブロック・サイズ) は、次の値の合計です。

1. 次のいずれかの全長
   - 1 つのレコード
   - 1 つのレコードと、1 つまたは 2 つのレコード・セグメント
   - 複数のレコード
   - 複数のレコードと、1 つまたは 2 つのレコード・セグメント
   - 2 つのレコード・セグメント
   - 1 つのレコード・セグメント

   可変長レコードの場合、各レコードの長さまたは各レコード・セグメントの長さには、レコードやレコード・セグメント用の 4 制御バイトが含まれています。

   前述のリストは、レコードおよびレコード・セグメントのオプション (固定長または可変長、ブロック化または非ブロック化、スパンまたは非スパン) の考えられるすべての組み合わせをまとめたものです。スパン・レコードのブロック・サイズを指定する場合、各レコードおよび各レコード・セグメントにはレコード長のための 4 制御バイトが必要であり、これらの数量は各ブロックに必要な 4 制御バイトに加算されるものであることに注意してください。
2. 必要なその他の制御バイト
   - 可変長ブロック・レコードには、4 バイトが必要です (ブロック・サイズの場合)。
   - 固定長レコードおよび不定長レコードには、追加の制御バイトは必要ありません。
3. 必要なブロック接頭部バイト (ASCII データ・セットのみ)

block-size (ブロック・サイズ) は、整数または属性 FIXED BINARY(31,0) STATIC を持つ変数として指定することができます。

レコード長の値には、次のような規則があります。

最大長:

32760

ゼロ値:

z/OS 上で BLKSIZE を 0 (ゼロ) に設定すると、データ機能プロダクトが

ブロック・サイズを設定します。詳しくは、『レコード・フォーマット、BLKSIZE、および RECSIZE のデフォルト』を参照してください。

負の値:
UNDEFINEDFILE 条件が発生します。

次のように、ブロック・サイズとレコード長の関係は、レコードのフォーマットによって異なります。

FB フォーマットまたは **FBS** フォーマット
このブロック・サイズは、レコード長の倍数でなければなりません。

VB フォーマット:
このブロック・サイズは、次の値の合計値以上でなければなりません。
1. 任意のレコードの最大長
2. 4 制御バイト

VS フォーマットまたは **VBS** フォーマット:
ブロック・サイズは、レコード長より小、等しい、またはより大きくすることができます。

注:

- 非ブロック化 (F フォーマットまたは V フォーマット) レコードで BLKSIZE オプションを使用するには、以下のいずれか方法を使用します。
  - BLKSIZE オプションを指定し、RECSIZE オプションを指定しない。 レコード長をブロック・サイズ (制御バイトまたは接頭語バイトを差し引く) と同じ値に設定し、レコード・フォーマットを未変更のままにする。
  - BLKSIZE と RECSIZE の両方を指定し、2 つの値の関係と使用するレコード・フォーマットのブロック化が矛盾しないことを確認する。 レコード・フォーマットを FB、VB に設定する (どちらでも該当する方)。
- FB フォーマットまたは FBS フォーマット・レコードでブロック・サイズがレコード長と等しい場合は、レコード・フォーマットは F に設定されます。
- BLKSIZE オプションは VSAM データ・セットには適用されません。そのため、指定しても無視されます。

レコード・フォーマット、BLKSIZE、および RECSIZE のデフォルト

非 VSAM データ・セットに対してレコード・フォーマット、ブロック・サイズ、レコード長を指定しないと、デフォルト・アクションが実行されます。

レコード・フォーマット:
関連付けられている DD ステートメントまたはデータ・セット・ラベル内で検索が行われます。 値が検出されない場合は、UNDEFINEDFILE 条件が発生します。ただし、ダミー・データ・セットまたはフォアグラウンド端末に関連付けられているファイルの場合を除きます。その場合は、レコード・フォーマットが U に設定されます。

ブロック・サイズあるいはレコード長:
ブロック・サイズまたはレコード長のいずれかが指定されている場合は、関連付けられている DD ステートメントまたはデータ・セット・ラベル内で、もう一方の検索が行われます。 その検索により値が検出されても、値

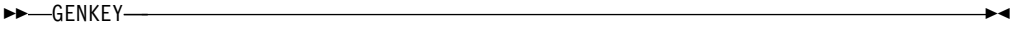
が指定オプションの値と矛盾している場合は、UNDEFINEDFILE 条件が発生します。 検索が成功しなかった場合は、指定オプションから値が導き出されます (制御バイトまたは接頭部バイトを追加あるいは除去して)。

ブロック・サイズもレコード長も指定されていない場合は、UNDEFINEDFILE 条件が発生します。ただし、ダミー・データ・セットと関連付けられているファイルの場合は除きます。その場合、BLKSIZE は 121 (F フォーマット・レコードや U フォーマット・レコードの場合)、または 129 (V フォーマット・レコードの場合) に設定されます。 フォアグラウンド端末に関連付けられているファイルの場合は、RECSIZE は 120 に設定されます。

z/OS 上でデータ機能プロダクト (DFP) のシステム決定ブロック・サイズ機能を使用している場合は、割り当てられている装置タイプに最適なブロック・サイズが DFP により算出されます。DD 割り当てまたは ENVIRONMENT ステートメント内で BLKSIZE(0) を指定する場合は、レコード長、レコード・フォーマット、および装置タイプを使って BLKSIZE が DFP により算出されます。

GENKEY オプション — キーの分類

GENKEY (総称キー) オプションは、INDEXED キー順データ・セットおよび VSAM キー順データ・セットにのみ適用されます。 このオプションを使用すれば、データ・セットに記録されているキーを分類したり、SEQUENTIAL KEYED INPUT ファイルや SEQUENTIAL KEYED UPDATE ファイルでキー・クラスに従ってレコードにアクセスしたりできます。



総称キーはキーのクラスを識別する文字ストリングです。このストリングで始まるすべてのキーはそのクラスのメンバーです。例えば、記録済みキー "ABCD"、"ABCE"、および "ABDF" はすべて、総称キー "A" および "AB" によって識別されるクラスのメンバーです。最初の 2 つは、クラス "ABC" のメンバーでもあります。これら 3 つの記録済みキーは、それぞれクラス "ABCD"、"ABCE"、"ABDF" の固有メンバーであると見なすことができます。

GENKEY オプションを使用すると、特定クラスのキーを持つ最初のレコードから、VSAM データ・セットを順次に読み取ることも更新することもできます。また、INDEXED データ・セットの場合は、このオプションを使用すると、特定のクラスのキーを持つ最初の非ダミー・レコードを順次に読み取ることも更新することもできます。 READ ステートメントの KEY オプションに総称キーを入れることにより、クラスを識別することができます。 KEY オプションを指定せずに、READ ステートメントで後続のレコードを読みとることができます。 キー・クラスの終わりに到達したときに、その旨の指摘は行われません。

KEY オプションを指定した READ ステートメントを使用することによって、特定クラスのキーを持つ最初のレコードを検索することができますが、KEYTO オプションは KEY オプションと同じステートメントで使用することができないため、レコードに組み込みキーがない限り、実際のキーを得ることはできません。

次の例では、3 バイトを超えているキーの長さが想定されます。

```
DCL IND FILE RECORD SEQUENTIAL KEYED
   UPDATE ENV (GENKEY);
          .
          .
          .
       READ FILE(IND) INTO(INFIELD)
                      KEY ('ABC');
          .
          .
          .
NEXT:  READ FILE (IND) INTO (INFIELD);
          .
          .
          .
       GO TO NEXT;
```

最初の READ ステートメントによって、「ABC」で始まるキーを持ったデータ・セット内の最初の非ダミー・レコードが、INFIELD に読み込まれることになります。第二の READ ステートメントが実行されるごとに、次に高位のキーを持つ非ダミー・レコードが取り出されます。 2 番目の READ ステートメントが繰り返し実行されると、より高位のキー・クラスからレコードが読み取られます。これは、キー・クラスの終わりに到達したときに標識が指定されていないためです。特定クラスのキーを超えて読み取りを続けたくない場合は、各自の責任で各キーを検査してください。 最初の READ ステートメントをもう一度実行すると、ファイルがキー・クラス「ABC」の最初のレコードの位置に再配置されます。

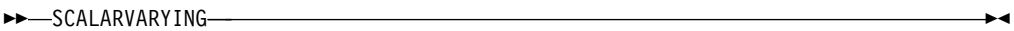
指定クラス内のキーを持つレコードがデータ・セットにない場合、または指定クラスのキーを持つレコードがすべてダミー・レコードの場合は、KEY 条件が発生します。 そのあと、データ・セットは上位のキーを持つ次のレコードに、あるいはファイルの終わりに位置付けられます。

GENKEY オプションの有無によって、KEYLEN サブパラメーターで指定されているキー長より短いソース・キーを提供する READ ステートメントの実行が影響を受けます。 KEYLEN サブパラメーターは、索引付きデータ・セットを定義する DD ステートメント内にあります。 GENKEY オプションを指定すると、それによってソース・キーが総称キーと解釈され、キーがソース・キーで始まるデータ・セット内の最初の非ダミー・レコードに、そのデータ・セットが位置付けられることになります。 GENKEY オプションを指定しないと、指定したキー長になるよう、READ ステートメントの短いソース・キーの右側にブランクが埋め込まれ、データ・セットは、この埋め込まれたキーを持ったレコード (このようなレコードが存在する場合) に位置付けられます。 WRITE ステートメントの場合は、短いソース・キーには常にブランクで埋め込まれます。

GENKEY オプションを使用しても、キー長が指定キー長以上であるソース・キーを提供してもその結果に影響はありません。 必要に応じて右側が切り捨てられるソース・キーは、特定のレコード (キーがそのクラスの唯一のメンバーだと考えられるレコード) を識別します。

SCALARVARYING オプション - 可変長ストリング

SCALARVARYING オプションは可変長ストリングの入出力について使用します。このオプションではどのフォーマットのレコードで使用することもできます。



ストレージを可変長ストリング用に割り当てるとき、コンパイラーは、ストリングの現行長を指定するための 2 バイト接頭部を組み込みます。 エレメント可変長ストリングの場合は、そのファイルに SCALARVARYING が指定されている場合にだけ、この接頭部が出力時に組み込まれるか、入力時に認識されます。

位置指定モード・ステートメント (LOCATE および READ SET) を使って、エレメント可変長ストリングを持つデータ・セットを作成し読み取る場合は、SCALARVARYING を指定して長さ接頭部の存在を認識させる必要があります。これは、バッファーの位置を指定するポインターは、常に長さ接頭部の開始位置を指すと想定されるからです。

SCALARVARYING を指定して、エレメント可変長ストリングが送信される場合は、長さ接頭部を組み込むためにレコード長に 2 バイトを与える必要があります。

SCALARVARYING を使用して作成されるデータ・セットは、SCALARVARYING を指定しているファイルだけがアクセスするようにします。

同じファイルに SCALARVARYING と CTLASA/CTL360 を指定することはできません。それを行うと、最初のデータ・バイトがあいまいになるからです。

KEYLENGTH オプション

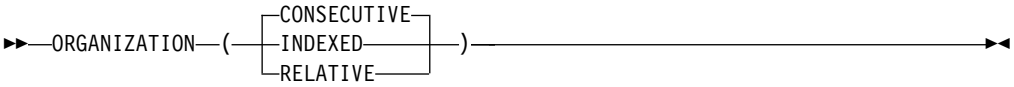
KEYLENGTH オプションは、KEYED ファイルに対して記録されるキーの長さを指定します。 INDEXED ファイルにも KEYLENGTH オプションを指定することができます。

n これは、KEYED ファイルに対して記録されるキーの長さを指定します。

検査目的で VSAM ファイル宣言に KEYLENGTH オプションを組み込む場合に、そのオプションで指定したキー長が、データ・セットに対して定義されている値と矛盾すると、UNDEFINEDFILE 条件が発生します。

ORGANIZATION オプション

ORGANIZATION は、PL/I ファイルに関連付けられているデータ・セットの編成を指定します。



CONSECUTIVE
該当のファイルが連続データ・セットに関連付けられていることを表します。連続ファイルは、ネイティブ・データ・セットでも、データ・セット VSAM、ESDS、RRDS、または KSDS でもかまいません。

RELATIVE
該当のファイルが相対データ・セットに関連付けられていることを表します。

RELATIVE は、そのデータ・セットに記録済みキーがないレコードが含まれていることを指定します。 相対ファイルは、VSAM 直接データ・セットです。 相対キーの範囲は、1 から nnnn までです。

PL/I レコード入出力で使用されるデータ・セットのタイプ

RECORD 属性を持ったデータ・セットは、データがプログラム変数に入っているとおりに補助記憶域との間で送受信されるレコード単位データ伝送によって処理されます。データ変換は行われません。 データ・セット内のレコードは、プログラム内の各変数に対応しています。

表 16 に、PL/I レコード入出力で使用できるさまざまなデータ・セット・タイプで使用可能な機能を示します。

表 *16. PL/I* レコード入出力で使用できるデータ・セット・タイプの比較

		VSAM KSDS	**VSAM ESDS**	**VSAM RRDS**	**INDEXED**	**CONSECUTIVE**	**REGIONAL (1)**
SEQUENCE		キー順	入力順	番号付け	キー順	入力順	領域ごと
装置		DASD	DASD	DASD	DASD	DASD、カードなど	DASD
ACCESS							
1	キー	123	123	123	12	2	12
2	順次						
3	逆方向						
前述のような代替索引アクセス		123	123	いいえ	いいえ	いいえ	いいえ
拡張する方法		新規キーあり	末尾	空のスロットを使用	新規キーあり	末尾	空のスロットを使用
DELETION							
		はい、1	いいえ	はい、1	はい、2	いいえ	はい、1
1	スペースは再使用可能						
2	スペースは再使用できない						

以下の各セクションは、各種データ・セットでのレコード入出力データ・セットの用法を説明しています。

- 281 ページの『第 10 章 連続データ・セットの定義と使用』
- 317 ページの『第 13 章 領域データ・セットの定義と使用』
- 331 ページの『第 14 章 VSAM データ・セットの定義と使用』

z/OS UNIX での環境変数の設定

いくつかの環境変数は、z/OS UNIX で使用するために設定したりエクスポートしたりできます。すべてのユーザーがアクセスできるように環境変数をシステム全体

用に設定するには、サブセクションで推奨されている行をファイル /etc/profile に追加します。 ある特定ユーザーだけに環境変数を設定するには、該当するユーザーのホーム・ディレクトリーにあるファイル .profile にその環境変数を追加します。

変数は、次にユーザーがログオンしたときに設定されます。

次の例に、環境変数の設定方法を示します。

```
LANG=ja_JP
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/prime/%N
LIBPATH=/home/joe/usr/lib:/home/joe/mylib:/usr/lib
export LANG NLSPATH LIBPATH
```

上の例にある最後のステートメントを使用する代わりに、その前の各行に export を追加することもできます (export LANG=ja_JP...)。

ECHO コマンドを使用すると、任意の環境変数の現行設定値を調べることができます。 BYPASS の値を定義するには、次の 2 つの例のどちらかを使用します。

```
echo $LANG

echo $LIBPATH
```

z/OS UNIX での PL/I 標準ファイル (SYSPRINT および SYSIN)

デフォルトにより、SYSIN は stdin から読み取られ、SYSPRINT は stdout に送られます。 関連付けを変更したい場合は、OPEN ステートメントの TITLE オプションを使うか、データ・セットまたは別の装置に名前を指定する DD_DDNAME 環境変数を設定する必要があります。

関連情報:

269 ページの『z/OS UNIX での環境変数の設定』

いくつかの環境変数は、z/OS UNIX で使用するために設定したりエクスポートしたりできます。すべてのユーザーがアクセスできるように環境変数をシステム全体用に設定するには、 サブセクションで推奨されている行をファイル /etc/profile に追加します。 ある特定ユーザーだけに環境変数を設定するには、 該当するユーザーのホーム・ディレクトリーにあるファイル .profile に その環境変数を追加します。

z/OS UNIX での標準入力装置、標準出力装置、および標準エラー装置のリダイレクト

標準入力装置、標準出力装置、および標準エラー装置をファイルにリダイレクトできます。

例えば、次のようなプログラムでリダイレクトを使用できます。

```
Hello2: proc options(main);
  put list('Hello!');
end;
```

プログラムをコンパイルしてリンクした後で、コマンド行で次のコマンドを入力して、そのプログラムを呼び出すことができます。

```
hello2 > hello2.out
```

stdout と stderr を 1 つのファイルにまとめる場合は、次のコマンドを入力します。

```
hello2 > hello2.out 2>&1
```

表示ステートメントの場合と同様に、(より大) 記号を使用すると、その後ろに指定されているファイル、この場合は hello2.out に出力がリダイレクトされます。 これは、'Hello' という語がファイル hello2.out に書き込まれることを意味します。PRINT 属性はデフォルトで SYSPRINT に適用されるため、出力にはプリンター制御文字も組み込まれるということに注意してください。

READ ステートメントは stdin からのデータにもアクセスできます。

第 9 章 ライブラリーの使用

z/OS オペレーティング・システムでは、区分データ・セット、区分データ・セット/拡張、およびライブラリー の 3 つの用語は同義であり、他のデータ・セット (通常、ソース・モジュール、オブジェクト・モジュール、またはロード・モジュールの形のプログラム) の保管に使用できるタイプのデータ・セットを指します。

ライブラリーは、直接アクセス・ストレージに保管する必要があり、全体が 1 つのボリューム内に入っていなければなりません。 ライブラリーには、連続して編成された、メンバーと呼ばれる独立したデータ・セットが入っています。各メンバーには、ライブラリーの一部であるディレクトリーに保管される 8 文字を超えない長さの固有名が付いています。 1 つのデータ・セット・ラベルだけが維持されるので、1 つのライブラリーに属するすべてのメンバーは同じデータ特性を持っている必要があります。

ディレクトリー内に新規項目を入れる十分なスペースが残っていないか、またはメンバーそのもののスペースが足りなくなるまでは、メンバーを個々に作成することができます。 メンバーの名前を指定すれば、メンバーを個別にアクセスすることができます。

DD ステートメントまたはそれらに対応する会話型での同等機能を使用することにより、メンバーの作成およびアクセスを行うことができます。

メンバーは、IBM ユーティリティー・プログラム IEHPROGM を使用して削除できます。このプログラムは、ディレクトリーからメンバー名を削除し、そのメンバーに今後アクセスできないようにします。ただし、IBM ユーティリティー・プログラム IEBCOPY などを使用してライブラリーを再作成したり未使用スペースを圧縮したりしない限り、そのメンバー自体が占有していたスペースは再利用できません。 DD ステートメントの DISP パラメーターを使ってメンバーを削除しようとすると、データ・セット全体が削除されてしまいます。

ライブラリーのタイプ

ライブラリーのタイプには、システム・プログラム・ライブラリー、システム・プロシージャー・ライブラリー、専用プログラム・ライブラリーなどがあります。

PL/I プログラムでは、次のタイプのライブラリーを使用することができます。

- システム・プログラム・ライブラリー SYS1.LINKLIB またはこれと同等のもの

  このライブラリーには、コンパイラーやリンケージ・エディターなどのすべてのシステム処理プログラムを収容することができます。
- 専用プログラム・ライブラリー

  これらのライブラリーには、通常、ユーザーが作成したプログラムが収容されます。多くの場合、一時専用ライブラリーを作成すれば、リンケージ・エディターからのロード・モジュール出力を、同一ジョブ内の後半のジョブ・ステップで実行するまでの間保管しておけるので便利です。 一時ライブラリーは、ジョブの

© Copyright IBM Corp. 1999, 2017

終了時に削除されます。 専用ライブラリーは、自動ライブラリー呼び出し用にもリンケージ・エディターやローダーによって使用されます。

- システム・プロシージャー・ライブラリー SYS1.PROCLIB またはこれと同等のもの

  このライブラリーには、ユーザーのシステム用としてカタログされているジョブ制御プロシージャーが入っています。

ライブラリーの使用

PL/I プログラムは、ライブラリーを直接使用することができます。

ユーザーが新たにメンバーをライブラリーに追加しようとすると、オペレーティング・システムは、その関連ファイルがクローズされるときに、データ・セット名の一部として指定されているメンバー名を使って新規メンバーのディレクトリー項目を作成します。

ユーザーがライブラリーのメンバーにアクセスする場合、オペレーティング・システムは、ユーザーがデータ・セット名の一部として指定したメンバー名からそのディレクトリー項目を見つけ出すことができます。

同一ライブラリーの複数のメンバーを 1 つの PL/I プログラムで処理することはできますが、このようなファイルは同時に 1 つしか出力としてオープンすることはできません。 DD ステートメントに入っているメンバー名を指定することにより、異なるメンバーにアクセスすることができます。

ライブラリーの作成

ライブラリーを作成するには、ライブラリーの作成に必要となる情報が含まれている DD ステートメントをジョブ・ステップに組み込みます。

ライブラリーの作成時に必要となる情報については、表 17 を参照してください。ライブラリーの作成に必要な情報は、SPACE パラメーターを除けば、連続して編成されたデータ・セット (306 ページの『レコード入出力を使用したファイルの定義』参照) の場合とほとんど同じです。

表 *17.* ライブラリー作成時に必要な情報

必須情報	**DD** ステートメントのパラメーター
使用する装置のタイプ	UNIT=
ライブラリーを入れるボリュームの通し番号	VOLUME=SER
ライブラリー名	DSNAME=
ライブラリーのスペース必須容量	SPACE=
ライブラリーの後処理	DISP=

SPACE パラメーター

DD ステートメントで SPACE パラメーターを使用すれば、作成するライブラリーに必要となるスペースの量を指定できます。

ライブラリーを定義するための DD ステートメント内の SPACE パラメーターは、常に次の形でなければなりません。

```
SPACE=(units,(quantity,increment,directory))
```

3 番目の項目 (increment) はオプションであり、省略する場合はコンマを残すことで、その項目が省略されていることを示すことができます。最後の項目 (割り振られるディレクトリー・ブロックの数を指定する) は必須です。

ライブラリーに必要な補助記憶域の大きさは、保管するメンバーの数とサイズと、メンバーがどのくらい頻繁に追加または置換されるかによって異なります。 (削除されたメンバーのスペースは解放されません。) 必要なディレクトリー・ブロック数は、メンバー数および別名数によって異なります。 SPACE パラメーターに増分容量を指定すると、データ・セット作成時や新規メンバー追加時に必要が生じた場合、オペレーティング・システムがデータ・セットにさらに必要な領域を確保することができます。ただし、ディレクトリー・ブロック数は作成時に固定されるため、増やすことはできません。

例

次の例にある DD ステートメントは、ボリューム通し番号 3412 を持つ DASD の 5 つのシリンダーを新規ライブラリー名 ALIB に対して割り振ること、およびこの名前をシステム・カタログに登録することをジョブ・スケジューラーに要求します。 SPACE パラメーターの最後の項目は、データ・セットに割り振られているスペースの一部を 10 個のディレクトリー・ブロック用に予約しています。

```
//   PDS DD UNIT=SYSDA,VOL=SER=3412,
//   DSNAME=ALIB,
//   SPACE=(CYL,(5,,10)),
//   DISP=(,CATLG)
```

ライブラリー・メンバーの作成と更新

ライブラリー・メンバーを作成および更新するときは、このトピックにあるガイドラインに従う必要があります。

各ライブラリー・メンバーは、同じ特性を持っていなければなりません。同じ特性メンバーを持っていない場合は、あとのメンバー検索が難しくなります。 同じ特性が必要な理由は、ボリューム目録 (VTOC) には、ライブラリー用のデータ・セット制御ブロック (DSCB) が 1 つ入っているだけで、各メンバー別用のものはないからです。 PL/I プログラムを使ってメンバーを作成する場合は、オペレーティング・システムによりディレクトリー項目が作成されます。つまり、ユーザーはユーザー・データ・フィールドに情報を入力することはできません。

ライブラリーとメンバーを同時に作成するときには、DD ステートメント内に 274 ページの『ライブラリーの作成』の下にリストアップされているすべてのパラメーター (ただし、データ・セットを一時的なものにするのであれば、DISP パラメーターは省くことができる) を入れなければなりません。DSNAME パラメーターは、メンバー名を括弧で囲んで指定する必要があります。例えば、DSNAME=ALIB(MEM1) によりデータ・セット ALIB 内のメンバー MEM1 の名前が指定されます。 メンバーがリンケージ・エディターによってライブラリー内に入れられると、DSNAME パラメーター内にメンバー名を入れる代わりに、リンケ

ージ・エディターの NAME ステートメントまたは NAME コンパイル時オプションを使用することができます。 また、メンバーの特性 (レコード・フォーマットなど) を DCB パラメーターまたは PL/I プログラムに記述する必要があります。これらの特性は、そのデータ・セットに追加される他のメンバーにも適用されます。

既存ライブラリーに追加するメンバーを作成するときには、SPACE パラメーターは必要ありません。 元のスペース割り振りは、個々のメンバーに対してではなく、ライブラリー全体に対して適用されるからです。 さらに、そのメンバーの特性を記述する必要もありません。その理由は、ライブラリー用の DSCB 内に既に特性が記録されているからです。

1 つのジョブ・ステップで複数のメンバーをライブラリーに追加するには、各メンバーごとに DD ステートメントを組み込み、そしてそのライブラリーを参照するファイルをクローズしてから次のファイルをオープンするようにしなければなりません。

例: コンパイルされたオブジェクト・モジュール用の新規ライブラリーの作成

以下の例では、カタログ式プロシージャー IBMZC を使用して簡単な PL/I プログラムをコンパイルし、オブジェクト・モジュールを EXLIB という新規ライブラリーに配置します。新規ライブラリーを定義してオブジェクト・モジュールを指定する DD ステートメントは、カタログ式プロシージャーにおける DD ステートメント SYSLIN を指定変更します。

PL/I プログラムは、関数プロシージャーであり、TIME 組み込み関数で作成される文字ストリングの形に 2 つの値が指定されている場合、ミリ秒単位でその値の差を戻します。

```
//OPT10#1 JOB
//TR        EXEC  IBMZC
//PLI.SYSLIN  DD  UNIT=SYSDA,DSNAME=HPU8.EXLIB(ELAPSE),
//      SPACE=(TRK,(1,,1)),DISP=(NEW,CATLG)
//PLI.SYSIN   DD  *
   ELAPSE:  PROC(TIME1,TIME2);
     DCL (TIME1,TIME2) CHAR(9),
         H1 PIC '99' DEF TIME1,
         M1 PIC '99' DEF TIME1 POS(3),
         MS1 PIC '99999' DEF TIME1 POS(5),
         H2 PIC '99' DEF TIME2,
         M2 PIC '99' DEF TIME2 POS(3),
         MS2 PIC '99999' DEF TIME2 POS(5),
         ETIME FIXED DEC(7);
     IF H2<H1 THEN H2=H2+24;
     ETIME=((H2*60+M2)*60000+MS2)-((H1*60+M1)*60000+MS1);
     RETURN(ETIME);
   END ELAPSE;
/*
```

図 *28.* コンパイルされたオブジェクト・モジュール用の新規ライブラリーの作成

例: ロード・モジュールの既存ライブラリーへの配置

以下の例では、カタログ式プロシージャー IBMZCL を使用して PL/I プログラムをコンパイルおよびリンク・エディットし、ロード・モジュールを既存ライブラリー HPU8.CCLM に配置します。

```
//OPT10#2  JOB
//TRLE      EXEC  IBMZCL
//PLI.SYSIN   DD  *
   MNAME:  PROC  OPTIONS(MAIN);
     .
     .
     .
     program
     .
     .
     .

   END MNAME;
/*
//LKED.SYSLMOD  DD  DSNAME=HPU8.CCLM(DIRLIST),DISP=OLD
```

図 *29.* ロード・モジュールの既存ライブラリーへの配置

例: ライブラリー・メンバーの更新

PL/I プログラムを使って、ライブラリーのメンバー内の 1 つ以上のレコードを追加または削除するには、そのライブラリー内の別の部分でそのメンバーを全部作成し直さなければなりません。 メンバーがそれまで占有していたスペースは再度使用することができないため、これはあまり経済的な提案とは言えません。ユーザーの PL/I プログラム内でファイルを 2 つ使う必要がありますが、2 つとも同じ DD ステートメントと関連付けることができます。

278 ページの図 31 に示すプログラムは、 278 ページの図 30 のプログラムで作成されたメンバーを更新します。 このプログラムは、ブランクだけのレコードを除き、元のメンバーのレコードをすべてコピーします。

```
//OPT10#3  JOB
//TREX  EXEC IBMZCBG
//PLI.SYSIN      DD *
  NMEM:  PROC OPTIONS(MAIN);
     DCL IN FILE RECORD SEQUENTIAL INPUT,
         OUT FILE RECORD SEQUENTIAL OUTPUT,
         P POINTER,
         IOFIELD CHAR(80) BASED(P),
         EOF BIT(1) INIT('0'B);
     OPEN FILE(IN),FILE (OUT);
     ON ENDFILE(IN) EOF='1'B;
     READ FILE(IN) SET(P);
     DO WHILE (¬EOF);
     PUT FILE(SYSPRINT) SKIP EDIT (IOFIELD) (A);
     WRITE FILE(OUT) FROM(IOFIELD);
     READ FILE(IN) SET(P);
     END;
     CLOSE FILE(IN),FILE(OUT);
  END NMEM;
/*
//GO.OUT   DD  UNIT=SYSDA,DSNAME=HPU8.ALIB(NMEM),
//     DISP=(NEW,CATLG),SPACE=(TRK,(1,1,1)),
//     DCB=(RECFM=FB,BLKSIZE=3600,LRECL=80)
//GO.IN DD *
  MEM:  PROC OPTIONS(MAIN);
        /* this is an incomplete dummy library member */
```

図 *30. PL/I* プログラム内でのライブラリー・メンバーの作成

```
//OPT10#4  JOB
//TREX  EXEC IBMZCBG
//PLI.SYSIN      DD *
  UPDTM: PROC OPTIONS(MAIN);
     DCL (OLD,NEW) FILE RECORD SEQUENTIAL,
         EOF BIT(1) INIT('0'B),
         DATA CHAR(80);
     ON ENDFILE(OLD)  EOF = '1'B;
     OPEN FILE(OLD) INPUT,FILE(NEW) OUTPUT TITLE('OLD');
     READ FILE(OLD) INTO(DATA);
     DO WHILE (¬EOF);
     PUT FILE(SYSPRINT) SKIP EDIT (DATA) (A);
     IF DATA=' ' THEN ;
     ELSE WRITE FILE(NEW) FROM(DATA);
     READ FILE(OLD) INTO(DATA);
     END;
   CLOSE FILE(OLD),FILE(NEW);
  END UPDTM;
/*
//GO.OLD DD  DSNAME=HPU8.ALIB(NMEM),DISP=(OLD,KEEP)
```

図 *31.* ライブラリー・メンバーの更新

ライブラリー・ディレクトリーからの情報の取り出し

ライブラリー・ディレクトリーは、データ・セットの先頭に置かれる一連のレコード (項目) です。 1 つ以上のディレクトリー項目が各メンバーに存在します。 各項目には、メンバー名、ライブラリー内のメンバーの相対アドレス、および可変量のユーザー・データが入っています。

ユーザー・データは、メンバーを作成したプログラムによって挿入される情報です。 リンケージ・エディターで作成されたメンバー (ロード・モジュール) を参照する項目は、システムのマニュアルに記載されている標準形式のユーザー・データを組み込みます。

PL/I プログラムを使ってメンバーを作成するときには、オペレーティング・システムがユーザーに代わってディレクトリー項目を作成するので、ユーザーはユーザー・データを書き込むことはできません。 ただし、アセンブラー言語マクロ命令を使用すると、メンバーを作成したり、独自のユーザー・データを作成することができます。 このためにマクロ命令を使用する方法は、データ管理資料に説明されています。

第 10 章 連続データ・セットの定義と使用

この章では、連続データ・セット編成について、さらにストリーム指向データ伝送およびレコード単位データ伝送用の連続データ・セットを定義するための ENVIRONMENT オプションについて述べています。 その次に、伝送タイプごとに、連続データ・セットの作成、アクセスおよび更新方法についても説明します。

連続編成のデータ・セット内では、各レコードは連続する物理的な位置のみに基づいて編成されます。データ・セットが作成されるときは、レコードは提示される順番で連続的に書き込まれます。 なお、レコードは、レコードが書き込まれた順序でのみ検索できます。 連続データ・セットの場合の有効ファイル属性と ENVIRONMENT オプションに関しては、 260 ページの表 15 を参照してください。

ストリーム指向データ伝送の用法

ここでは、STREAM 属性の PL/I ファイルで使用するデータ・セットの定義方法について説明します。 使用できる ENVIRONMENT オプション、データ・セットを作成しデータ・セットにアクセスする方法について説明します。 また、データ・セットの作成、アクセス時に使用する DD ステートメントの必須パラメーターを表にまとめ、文章による記述を説明するために PL/I プログラムの例もいくつか掲載しています。

STREAM 属性を持つデータ・セットは、ストリーム指向データ伝送で処理されます。そのため、PL/I プログラムは、ブロックやレコードの境界を無視して、各データ・セットを、文字の形またはグラフィックの形のデータ値の 1 つの連続するストリームとして扱うことができます。

ストリーム指向データ伝送用のデータ・セットを作成し、それにアクセスするには、「*PL/I 言語解説書*」で説明しているリスト指示、データ指示、および編集指示の入出力ステートメントを使用します。

出力の場合、PL/I は必要に応じてデータ項目をプログラム変数から文字の形に変換し、文字またはグラフィックスのストリームを、データ・セットに伝送するためにレコードに構築します。

入力の場合、PL/I はデータ・セットからレコードを取り出し、ユーザー・プログラムが要求した複数のデータ項目に分割し、さらにそれをプログラム変数に割り当てるのに適した形に変換します。

ストリーム指向データ伝送は、グラフィック・データの読み取りと書き込みに使用できます。 適切なプログラミング・サポートがあれば、グラフィックスを表示し、印刷し、入力することができる端末、プリンター、およびデータ入力装置があります。 ユーザーのデータが使用する装置または印刷ユーティリティー・プログラムで受け入れられているフォーマットになっているかどうか確認する必要があります。

© Copyright IBM Corp. 1999, 2017

ストリーム入出力を用いたファイルの定義

ストリーム指向データ伝送用のファイルはファイル宣言で定義できます。

```
DCL filename FILE STREAM
             INPUT | {OUTPUT [PRINT]}
             ENVIRONMENT(options);
```

デフォルト・ファイル属性については、 260 ページの表 15 を参照してください。 FILE 属性については、「*PL/I* 言語解説書」を参照してください。 PRINT 属性について詳しくは、 292 ページの『ストリーム入出力による PRINT ファイルの使用』を参照してください。ENVIRONMENT 属性のオプションについては、『ENVIRONMENT オプションの指定』を参照してください。

PL/I 動的割り振りを使用したストリーム・ファイルの定義

ストリーム・ファイルを定義するには、DD ステートメント、環境変数、または OPEN ステートメントの TITLE オプションを使用できます。

環境変数または TITLE オプションを使用する場合、名前が大文字でなければなりません。 次のいずれかの方法で MVS データ・セットを指定してください。

- DSN(*data-set-name*)
- DSN(*data-set-name*(*member-name*))

  data-set-name は完全修飾名でなければなりません。また、data-set-name は一時データ・セットにすることはできません。例えば、先頭が & であってはなりません。

次のように HFS ファイルを指定してください。

```
PATH(absolute-path-name)
```

DSN キーワードの後に、以下の属性を任意の順序で指定できます。

```
NEW、OLD、SHR、または MOD
TRACKS または CYL
SPACE(n,m)
VOL(volser)
UNIT(type)
KEEP、DELETE、CATALOG、または UNCATALOG
STORCLAS(storageclass)
MGMTCLAS(managementclass)
DATACLAS(dataclass)
```

注**:** 環境変数、または OPEN ステートメントの TITLE オプションを使用して PDS または PDSE を作成することはできませんが、既存の PDS または PDSE に新規メンバーを作成することは可能です。

ENVIRONMENT オプションの指定

このトピックでは、ストリーム指向データ伝送およびレコード単位データ伝送に対して連続データ・セットを定義する ENVIRONMENT オプションについて説明します。

260 ページの表 15 に、ENVIRONMENT オプションの要約があります。 以下のオプションをストリーム指向データ伝送に適用できます。

```
CONSECUTIVE or ORGANIZATION(CONSECUTIVE)
F|FB|FS|FBS|V|VB|VS|VBS|U
RECSIZE(record-length)
BLKSIZE(block-size)
GRAPHIC
LEAVE|REREAD
```

上記オプションをストリーム指向データ伝送に対して指定する方法については、以下のトピックを参照してください。

オプション	参照先のトピック
CONSECUTIVE	『CONSECUTIVE』
F|FB|FS|FBS|V|VB|VS|VBS|U	『レコード・フォーマット・オプション』
RECSIZE	284 ページの『RECSIZE』
BLKSIZE	264 ページの『BLKSIZE オプション』
GRAPHIC	285 ページの『GRAPHIC』
LEAVE|REREAD	310 ページの『LEAVE|REREAD』

CONSECUTIVE

STREAM ファイルは CONSECUTIVE データ・セット編成を持っていなくてはなりませんが、CONSECUTIVE はデフォルトのデータ・セット編成なので、これを ENVIRONMENT オプション内に指定する必要はありません。

STREAM ファイルの CONSECUTIVE オプションは、 253 ページの『データ・セットの編成』に記載されているものとまったく同じです。

レコード・フォーマット・オプション

ストリーム指向データ伝送では、レコード境界は無視されますが、データ・セットを作成する場合、レコード・フォーマットは重要な意味を持ちます。これは、レコード・フォーマットがデータ・セットが占有するストレージの容量と、データを処理するプログラムの効率に影響するためだけでなく、データ・セットが後で、レコード単位データ伝送でも処理できるようにするためです。

いったんレコード・フォーマットを指定したならば、ストリーム指向データ伝送を使う限り、レコードおよびブロックを意識する必要はありません。 データ・セットは、行に配置された一連の文字またはグラフィックスと見なすことができます。また、SKIP オプションまたはフォーマット項目 (PRINT ファイルでは、PAGE オプション、LINE オプションおよびフォーマット項目) を使って、新しい行を選択することができます。

レコードのフォーマットは、以下のいずれかにすることができます。詳しくは、251 ページの『レコード・フォーマット』を参照してください。

固定長	F	非ブロック化
	FB	ブロック化
	FS	非ブロック化、標準
	FBS	ブロック化、標準
可変長	V	非ブロック化
	VB	ブロック化
	VS	
	VBS	
不定長	U	(ブロック化できない)

なお、レコードのブロック化も非ブロック化も自動的に行われます。

RECSIZE

ストリーム指向データ伝送の場合の RECSIZE は 259 ページの『ENVIRONMENT 属性での特性の指定』 に記載されているものと同じです。 また、OPEN ステートメントの LINESIZE オプションで指定された値は、RECSIZE オプションで指定された値を指定変更します。 LINESIZE については、「*PL/I* 言語解説書」で説明されています。

グラフィックスのリスト指示伝送およびデータ指示伝送についてのレコード・サイズに関する追加の考慮事項は、「*PL/I* 言語解説書」に記載されています。

レコード・フォーマット、BLKSIZE および RECSIZE のデフォルト値

ENVIRONMENT 属性や関連 DD ステートメント/データ・セット・ラベルでレコード・フォーマット、BLKSIZE、RECSIZE オプションのいずれも指定しないと、PL/I がデフォルト値を決定します。

- 入力ファイル:

  デフォルトは、 265 ページの『レコード・フォーマット、BLKSIZE、および RECSIZE のデフォルト』で説明されているレコード単位データ伝送と同様に適用されます。

- 出力ファイル

  レコード・フォーマット
  VB フォーマットに設定されます。

レコード長
指定した、またはデフォルトの LINESIZE の値が使用されます。
– PRINT ファイル:
  - F、FB、FBS、または U: 行サイズ + 1
  - V または VB: 行サイズ + 5
– 非 PRINT ファイル:
  - F、FB、FBS、または U: linesize
  - V または VB: linesize + 4

ブロック・サイズ
SYSOUT に関連付けられたファイル:
– F、FB、または FBS: レコード長
– V または VB: レコード長 + 4

新規/一時データ・セット:
– DFP によって決定される最適ブロック・サイズ

GRAPHIC

編集指示入出力の GRAPHIC オプションを指定します。

入力データや出力データにグラフィックスが含まれているのに、GRAPHIC オプションが指定されていなければ、リスト指示入出力およびデータ指示入出力の ERROR 条件が発生します。

編集指示入出力で GRAPHIC オプションを指定すると、出力時に DBCS 変数と定数の左右に区切り文字が追加され、グラフィック入力時にも左右に区切り文字が付け加えられます。 GRAPHIC オプションを指定しない場合は、出力データの左右に区切り文字は追加されず、また、入力のグラフィックスにも左右の区切り文字は必要ありません。 また、GRAPHIC オプションを指定した場合は、入力データの左右に区切り文字がないと、ERROR 条件が発生します。

グラフィック・データ・タイプについて、また編集指示入出力の G フォーマット項目については、「*PL/I* 言語解説書」を参照してください。

ストリーム入出力によるデータ・セットの作成

データ・セットを作成するには、ユーザーの PL/I プログラム内で、またはデータ・セットを定義する DD ステートメント内で、特定の情報をオペレーティング・システムに与える必要があります。

z/OS UNIX の場合は、以下のいずれかの方法を使用して追加情報を指定します。
- OPEN ステートメントの TITLE オプション
- DD_DDNAME 環境変数
- ENVIRONMENT 属性

以下のトピックでは、データ・セットを作成するために指定しなければならない必須情報と、指定可能な一部のオプション情報について説明します。

必須情報

ご使用のアプリケーションで STREAM ファイルを作成する場合、PL/I は、そのファイルの行サイズ値を導き出します。

PL/I は、次のいずれかのソースから優先順位順 (降順) に行サイズ値を導き出します。

- OPEN ステートメントの LINESIZE オプション
- ENVIRONMENT 属性の RECSIZE オプション
- OPEN ステートメントの TITLE オプションの RECSIZE オプション
- DD_DDNAME 環境変数の RECSIZE オプション
- PL/I 提供のデフォルト値

LINESIZE の値が与えられていて、RECSIZE の値が与えられていない場合は、PL/I は、次のようにレコード長を導き出します。

- V フォーマット PRINT ファイルの場合、値は LINESIZE + 5。
- V フォーマット非 PRINT ファイルの場合、値は LINESIZE + 4。
- F フォーマット PRINT ファイルの場合、値は LINESIZE + 1。
- 上記以外の場合はすべて、値は LINESIZE。

LINESIZE の値が与えられておらず、RECSIZE の値が与えられている場合は、PL/I は、次のように RECSIZE から行サイズの値を導き出します。

- V フォーマット PRINT ファイルの場合、値は RECSIZE - 5。
- V フォーマット非 PRINT ファイルの場合、値は RECSIZE - 4。
- F フォーマット PRINT ファイルの場合、値は RECSIZE - 1。
- 上記以外の場合はすべて、値は RECSIZE。

LINESIZE も RECSIZE も与えられていない場合は、PL/I は、ファイル属性および関連付けられたデータ・セットのタイプに基づいて、デフォルトの行サイズの値を決定します。PL/I が、適切なデフォルト行サイズを提供できない場合は、UNDEFINEDFILE 条件が発生します。

以下の条件下では、OUTPUT ファイルに対してデフォルト行サイズ値が付与されます。

- ファイルは PRINT 属性を持っている。 この場合、値はタブ制御テーブルから得られます。
- 関連データ・セットが端末 (stdout または stderr) である。この場合、値は 120 です。

なお、LINESIZE オプションが (OPEN ステートメントで) 指定されていて、(ENVIRONMENT 属性、TITLE オプション、または DD ステートメントで) RECSIZE も指定されていて、(レコード・フォーマットおよび適切な制御バイト・オーバーヘッドを考慮に入れて) レコード・サイズ値が小さ過ぎて LINESIZE を保持できない場合は、以下のことが行われます。

- 言語環境プログラム for z/OS 1.9 以前のリリースを使用している場合、DD SYSOUT= ファイルに関しては、指定された LINESIZE に一致する新規レコー

ド・サイズを決定するために LINESIZE オプションが使用され、DD DSN= ファイルおよび他のすべてのファイルに関しては、UNDEFINEDFILE 条件が発生します。

- 言語環境プログラム for z/OS 1.9 より後のリリースを使用している場合は、すべてのファイルに関して UNDEFINEDFILE 条件が発生します。

例: ストリーム指向データ伝送によるデータ・セットの作成

以下の例では、編集指示ストリーム指向データ伝送を使用して直接アクセス・ストレージ装置上にデータ・セットを作成する方法が示されています。

ファイル SYSIN によって入力ストリームから読み取られるデータには、名前の付いていない 7 文字サブフィールドが入っているフィールド VREC が含まれており、フィールド NUM は情報が入っているこれらのサブフィールドの数を定義します。出力ファイル WORK からは、フィールド FREC 全体と、情報が含まれている VREC のサブフィールドのみがデータ・セットに伝送されます。

```
//EX7#2  JOB
//STEP1  EXEC IBMZCBG
//PLI.SYSIN     DD *
 PEOPLE: PROC OPTIONS(MAIN);
         DCL WORK FILE STREAM OUTPUT,
             1 REC,
               2 FREC,
                 3 NAME CHAR(19),
                 3 NUM CHAR(1),
                 3 PAD CHAR(25),
               2 VREC CHAR(35),
             EOF BIT(1) INIT('0'B),
             IN CHAR(80) DEF REC;
         ON ENDFILE(SYSIN) EOF='1'B;
         OPEN FILE(WORK) LINESIZE(400);
         GET FILE(SYSIN) EDIT(IN)(A(80));
         DO WHILE (¬EOF);
         PUT FILE(WORK) EDIT(IN)(A(45+7*NUM));
         GET FILE(SYSIN) EDIT(IN)(A(80));
         END;
         CLOSE FILE(WORK);
         END PEOPLE;
/*
//GO.WORK DD DSN=HPU8.PEOPLE,DISP=(NEW,CATLG),UNIT=SYSDA,
//           SPACE=(TRK,(1,1))
//GO.SYSIN DD *
R.C.ANDERSON       0 202848 DOCTOR
B.F.BENNETT        2 771239 PLUMBER          VICTOR HAZEL
R.E.COLE           5 698635 COOK             ELLEN  VICTOR JOAN   ANN    OTTO
J.F.COOPER         5 418915 LAWYER           FRANK  CAROL  DONALD NORMAN BRENDA
A.J.CORNELL        3 237837 BARBER           ALBERT ERIC   JANET
E.F.FERRIS         4 158636 CARPENTER        GERALD ANNA   MARY   HAROLD
/*
```

図 32. ストリーム指向データ伝送によるデータ・セットの作成

例: グラフィック・データのストリーム・ファイルへの書き込み

以下の例では、リスト指示出力を使用してグラフィックスをストリーム・ファイルに書き込むプログラムが示されています。

この例では、グラフィック・データを印刷できる出力装置があることを想定しています。このプログラムは、従業員レコードを読み取り、特定の地域に在住する従業員を選択します。このプログラムはさらに、住所フィールドを編集して、各住所項目間にグラフィック・ブランクを挿入し、従業員番号、氏名、および住所を印刷します。

```
//EX7#3  JOB
//STEP1 EXEC IBMZCBG
//PLI.SYSIN      DD *
% PROCESS GRAPHIC;
  XAMPLE1:  PROC OPTIONS(MAIN);
            DCL  INFILE FILE INPUT RECORD,
                 OUTFILE FILE OUTPUT STREAM ENV(GRAPHIC);
 /* GRAPHIC OPTION MEANS DELIMITERS WILL BE INSERTED ON OUTPUT FILES. */
           DCL
                1 IN,
                   3 EMPNO CHAR(6),
                   3 SHIFT1 CHAR(1),
                   3 NAME,
                      5 LAST G(7),
                      5 FIRST G(7),
                   3 SHIFT2 CHAR(1),
                   3 ADDRESS,
                      5 ZIP CHAR(6),
                      5 SHIFT3 CHAR(1),
                      5 DISTRICT G(5),
                      5 CITY G(5),
                      5 OTHER G(8),
                      5 SHIFT4 CHAR(1);
           DCL EOF BIT(1) INIT('0'B);
           DCL ADDRWK G(20);
     ON ENDFILE (INFILE) EOF = '1'B;
     READ FILE(INFILE) INTO(IN);
     DO WHILE(¬EOF);
             DO;
                 IF SUBSTR(ZIP,1,3)¬='300'
                    THEN LEAVE;
                 L=0;
                 ADDRWK=DISTRICT;
                 DO I=1 TO 5;
                 IF SUBSTR(DISTRICT,I,1)= <   >
                    THEN LEAVE;               /* SUBSTR BIF PICKS 3P  */
                 END;                         /* THE ITH GRAPHIC CHAR */
                 L=L+I+1;                     /* IN DISTRICT          */
                 SUBSTR(ADDRWK,L,5)=CITY;
                 DO I=1 TO 5;
                 IF SUBSTR(CITY,I,1)= <   >
                    THEN LEAVE;
                 END;
                 L=L+I;
                 SUBSTR(ADDRWK,L,8)=OTHER;
                 PUT FILE(OUTFILE) SKIP           /* THIS DATA SET    */
                 EDIT(EMPNO,IN.LAST,FIRST,ADDRWK) /* REQUIRES UTILITY */
                     (A(8),G(7),G(7),X(4),G(20)); /* TO PRINT GRAPHIC */
                                                  /* DATA             */
                 END;                        /* END OF NON-ITERATIVE DO */
         READ FILE(INFILE) INTO (IN);
         END;                                 /* END OF DO WHILE(¬EOF) */
     END XAMPLE1;
 /*
 //GO.OUTFILE   DD  SYSOUT=A,DCB=(RECFM=VB,LRECL=121,BLKSIZE=129)
 //GO.INFILE DD *
 ABCDEF<
>300099<           3 3 3 3 3 3 3   >
 ABCD  <
>300011<           3 3 3 3       >
 /*
```

図 *33.* グラフィック・データのストリーム・ファイルへの書き込み

ストリーム入出力によるデータ・セットへのアクセス

ストリーム指向データ伝送を使用してアクセスされるデータ・セットは、ストリーム指向データ伝送で作成する必要はありませんが、そのようなデータ・セットには CONSECUTIVE 編成が必要であり、そのようなデータ・セットに含まれるデータはすべて文字またはグラフィックの形式になっていなければなりません。入力のための関連ファイルをオープンし、データ・セットに入っているレコードを読み取るか、あるいは出力のためにファイルをオープンし、終わりにレコードを追加してデータ・セットを拡張することができます。

データ・セットにアクセスするには、次のいずれかの方法でデータ・セットを識別する必要があります。
- ENVIRONMENT 属性
- DD_DDNAME 環境変数
- OPEN ステートメントの TITLE オプション

以下のトピックでは、DD ステートメントに組み込まなければならない必須情報について説明します。また、指定可能なオプション情報についてもいくつか説明します。ただし、ここで述べる内容は、入力ストリームのデータ・セットには当てはまりません。

必須情報

ユーザー・アプリケーションで既存の STREAM ファイルにアクセスするには、PL/I はそのファイルのレコード長を入手する必要があります。

データ・セットがレコード長を持っていない場合は、次のいずれかのソースから値を取得できます。
- OPEN ステートメントの LINESIZE オプション
- ENVIRONMENT 属性の RECSIZE オプション
- DD_DDNAME 環境変数の RECSIZE オプション
- OPEN ステートメントの TITLE オプションの RECSIZE オプション
- PL/I 提供のデフォルト値

既存の OUTPUT ファイルを使用する場合、および RECSIZE の値を提供する場合、PL/I は、 285 ページの『ストリーム入出力によるデータ・セットの作成』に説明されているようにレコード長を決定します。

以下の条件では、PL/I は INPUT ファイルに対してデフォルトのレコード長の値を使用します。
- ファイルが SYSIN で、値が 80 の場合
- ファイルが端末 (stdout: または stderr:) に関連付けられており、値が 120 の場合

レコード・フォーマット

ストリーム指向データ伝送を使ってデータ・セットにアクセスするときには、そのデータ・セットのレコード・フォーマットを知っていなくてもかまいません (ブロ

ック・サイズを指定しなければならない場合を除く)。各 GET ステートメントがそれぞれ個別の数の文字あるいはグラフィックスをデータ・ストリームからプログラムへ転送します。

ユーザーがレコード・フォーマットの情報を与える場合は、その情報はデータ・セットの実際の構造と矛盾しないものである必要があります。 例えば、F フォーマット・レコード、600 バイトのレコード・サイズ、および 3600 バイトのブロック・サイズを使って作成したデータ・セットの場合、3600 バイトの最大ブロック・サイズを持った U フォーマット・レコードの場合と同様にそのレコードにアクセスすることができます。ただし、ブロック・サイズ 3500 を指定すると、データは切り捨てられます。

例: ストリーム指向データ伝送によるデータ・セットへのアクセス

以下の例では、ストリーム指向データ伝送を使用してデータ・セットにアクセスするプログラムが示されています。

292 ページの図 34 にあるプログラムは、 287 ページの図 32 のプログラムで作成したデータ・セットを読み取って、ファイル SYSPRINT を使ってその中に入っているデータをリストします。

データのセットはそれぞれ GET ステートメントによって 2 つの変数 (FREC および VREC) に読み込まれます。 FREC には常に 45 文字が含まれ、VREC には常に 35 文字が含まれます。 GET ステートメントが実行されるたびに、VREC には、7*NUM という式で算出される数の文字が含まれます。 35 文字に満たない場合はブランクが埋め込まれます。 DD ステートメントの DISP パラメーターは単に DISP=OLD とすることもできます。DELETE を省略すると既存のデータ・セットは削除されません。

```
//EX7#5  JOB
//STEP1  EXEC IBMZCBG
//PLI.SYSIN      DD *
 PEOPLE: PROC OPTIONS(MAIN);
         DCL WORK FILE STREAM INPUT,
             1 REC,
               2 FREC,
                 3 NAME CHAR(19),
                 3 NUM CHAR(1),
                 3 SERNO CHAR(7),
                 3 PROF CHAR(18),
               2 VREC CHAR(35),
             IN CHAR(80) DEF REC,
             EOF BIT(1) INIT('0'B);
         ON ENDFILE(WORK) EOF='1'B;
         OPEN FILE(WORK);
         GET FILE(WORK) EDIT(IN,VREC)(A(45),A(7*NUM));
         DO WHILE (¬EOF);
         PUT FILE(SYSPRINT) SKIP EDIT(IN)(A);
         GET FILE(WORK) EDIT(IN,VREC)(A(45),A(7*NUM));
         END;
         CLOSE FILE(WORK);
         END PEOPLE;
/*
//GO.WORK DD DSN=HPU8.PEOPLE,DISP=(OLD,DELETE)
```

図 *34.* ストリーム指向データ伝送によるデータ・セットへのアクセス

ストリーム入出力による **PRINT** ファイルの使用

オペレーティング・システムにも PL/I 言語にも、出力データのフォーマット設定を簡単に行えるようにするための機能がいくつかあります。

オペレーティング・システムで、ユーザーが各レコードの最初のバイトを印刷制御文字として使用することができるようにしています。 制御文字は、印刷されずにプリンターに改行や改ページを行わせます。 (印刷制御文字については、 309 ページの図 37 および 309 ページの表 19 を参照してください。)

PL/I プログラムでは、PRINT ファイルの使用は、ストリーム指向データ伝送からの印刷出力のレイアウトを制御するのに便利な方法です。 コンパイラーは、PAGE、 SKIP、LINE オプション、およびフォーマット項目に対応して印刷制御文字を自動的に挿入します。

関連付けられたデータ・セットを直接印刷するつもりでない場合でも、PRINT 属性を任意の STREAM OUTPUT ファイルに適用することができます。 PRINT ファイルが直接アクセス・データ・セットに関連付けられていると、印刷制御文字はこのデータ・セットのレイアウトには効力がありませんが、レコード内のデータの一部として現れます。

FB または VB でオープンされた PRINT ファイルは、UNDEFINEDFILE 条件が生じる原因になります。 PRINT ファイルは、「A」オプション (すなわち FBA または VBA) でオープンする必要があります。

コンパイラーは、PRINT ファイルで伝送される各レコードの最初のバイトが米国標準規格 (ANS) の印刷制御文字用に予約し、自動的に適切な文字を挿入します。

PRINT ファイルは、次の 5 つの印刷制御文字だけを使用します。

文字	処置
	1 行空けて (ブランク文字) から印刷する
0	2 行空けてから印刷する
-	3 行空けてから印刷する
+	1 行目から印刷する
1	改ページする

コンパイラーは、現行レコードの残りにブランクを埋め込み、次のレコードに適切な制御文字を挿入することによって、PAGE、SKIP、LINE の各オプションやフォーマット項目を処理します。 SKIP または LINE に 3 行を超えるスペースが指定されている場合は、コンパイラーは、適切な制御文字を使って必要な数のブランク・レコードを挿入し、必要なスペーシングを行います。印刷制御オプションやフォーマット項目がない場合は、レコードがフルになると、コンパイラーは、次のレコードの最初のバイトにブランク文字 (1 行のスペース) を挿入します。

PRINT ファイルの伝送先が端末の場合は、出力フォーマットを指定しない限り、PAGE、SKIP、LINE の各オプションに 3 行を超えるスキップをさせることはありません。

印刷する行の長さの制御

ユーザーの PL/I プログラムあるいは DD ステートメントにレコード長を指定する (ENVIRONMENT 属性) か、あるいは OPEN ステートメントに行サイズを指定する (LINESIZE オプション) ことによって、PRINT ファイルで作成された印刷行の長さを制限することができます。

レコード長には印刷制御文字用の余分のバイト数を含める必要があります。すなわち、レコード長は印刷される行の長さより 1 バイト (V フォーマット・レコードの場合は、5 バイト) 長くなければなりません。ユーザーが LINESIZE オプションに指定した値は、印刷される行の文字数を参照します。コンパイラーは印刷制御文字を追加します。

レコードをブロック化しても、PRINT ファイルで生成される出力の外観に影響はありませんが、直接アクセス装置上でファイルがデータ・セットと関連付けられていれば、補助記憶域をより効率よく使えるようになります。 なお、LINESIZE オプションを指定する場合は、行サイズとブロック・サイズとの互換性を確認する必要があります。 F フォーマットのレコードの場合、ブロック・サイズは正確に (行サイズ +1) の倍数でなくてはならず、V フォーマットのレコードの場合、ブロック・サイズは行サイズよりも最低 9 バイト大きくなければなりません。

ファイルをいったんクローズし、新しい行サイズでもう一度オープンすれば、実行中に PRINT ファイルの行サイズを変更できますが、PRINT ファイルを使って直接アクセス装置上にデータ・セットを作成する場合は、注意が必要です。 ファイルを初めてオープンしたとき、データ・セットに設定したレコード・フォーマットは変更できません。 OPEN ステートメントに指定した行サイズと、既に設定されているレコード長が矛盾すると、UNDEFINEDFILE 条件が発生します。 このような矛

盾が生じないように、使用する予定の最大行サイズより最低 9 バイト大きいブロック・サイズを指定して V フォーマット・レコードを指定するか、最初の OPEN ステートメントで最大行サイズを必ず指定するようにしてください。 (プリンターに送る予定の出力は、直接プリンターへ送ろうとする場合でも、UNIT= を使ってプリンターを指定しない限り、一時的に直接アクセス装置に保管することができます。)

PRINT ファイルは 120 文字のデフォルトの行サイズを持っています。したがって、PRINT ファイルのレコード・フォーマットを指定する必要はありません。また、他の情報がない場合には、コンパイラーは V フォーマットのレコードであると見なします。 完全なデフォルト値は、次のとおりです。

```
BLKSIZE=129
LRECL=125
RECFM=VBA
```

例

295 ページの図 35 は、PRINT ファイルと、ストリーム指向データ伝送ステートメントのオプションを使って、テーブルをフォーマット設定して、それを後で印刷できるように直接アクセス装置に書き込む方法を例示しています。このテーブルは、6' 間隔の、0° から 359° 54' までの角度の正弦から成ります。

ENDPAGE ON ユニット内のステートメントによって、各ページの下にページ番号が挿入され、次ページの見出しがセットアップされます。

このプログラムで作成されたデータ・セットを定義する DD ステートメントには、レコード・フォーマットに関する情報は含まれていません。 ファイル TABLE をオープンするステートメントに指定されたファイル宣言および行サイズから、コンパイラーは次のように推測します。

レコード・フォーマット =
: V (PRINT ファイルのデフォルト値)

レコード・サイズ =
: 98 (行サイズ + 1 バイト (印刷制御文字用) + 4 バイト (レコード制御フィールド用))

ブロック・サイズ =
: 102 (レコード長 + 4 バイト (ブロック制御フィールド用))

316 ページの図 39 のプログラムは、レコード単位データ伝送を使用して、 295 ページの図 35 のプログラムで作成されたテーブルを出力します。

```
%PROCESS INT F(I) AG A(F) OP STG NEST X(F) SOURCE ;
%PROCESS LIST;

 SINE: PROC OPTIONS(MAIN);
   DCL TABLE       FILE STREAM OUTPUT PRINT;
   DCL DEG         FIXED DEC(5,1) INIT(0);  /* INIT(0) FOR ENDPAGE  */
   DCL MIN         FIXED DEC(3,1);
   DCL PGNO        FIXED DEC(2)   INIT(0);
   DCL ONCODE      BUILTIN;

   ON ERROR
     BEGIN;
       ON ERROR SYSTEM;
       DISPLAY ('ONCODE = '|| ONCODE);
     END;

   ON ENDPAGE(TABLE)
     BEGIN;
       DCL I;
       IF PGNO ¬= 0 THEN
         PUT FILE(TABLE) EDIT ('PAGE',PGNO)
             (LINE(55),COL(80),A,F(3));
       IF DEG ¬= 360 THEN
         DO;
           PUT FILE(TABLE) PAGE EDIT ('NATURAL SINES') (A);
           IF PGNO ¬= 0 THEN
             PUT FILE(TABLE) EDIT ((I DO I = 0 TO 54 BY 6))
                                  (SKIP(3),10 F(9));
           PGNO = PGNO + 1;
         END;
       ELSE
         PUT FILE(TABLE) PAGE;
     END;

   OPEN FILE(TABLE) PAGESIZE(52) LINESIZE(93);
   SIGNAL ENDPAGE(TABLE);

   PUT FILE(TABLE) EDIT
    ((DEG,(SIND(DEG+MIN) DO MIN = 0 TO .9 BY .1) DO DEG = 0 TO 359))
    (SKIP(2), 5 (COL(1), F(3), 10 F(9,4) ));
   PUT FILE(TABLE) SKIP(52);
 END SINE;
```

図 *35.* ストリーム・データ伝送による印刷ファイルの作成： 316 ページの図 39 の例では、結果ファイルが印刷されます。

タブ制御テーブルの指定変更

PRINT ファイルへのデータ指示出力およびリスト指示出力は、事前設定されているタブ位置に合わせて配置されます。 31 ビット・プログラムに関しては、タブ制御テーブルをカスタマイズできます。ただし、64 ビット・プログラムに関しては、ユーザー定義の PLITAB がサポートされていません。したがって、タブ制御テーブルを指定変更できません。

タブ・テーブルの宣言方法の例については、 195 ページの図 17 および 297 ページの図 36 の例を参照してください。テーブル内のフィールドの定義は以下のとおりです。

OFFSET OF TAB COUNT:
ハーフワード 2 進整数で表された**タブ・カウント** のオフセット。これは、使用するタブ数を示すためのフィールドです。

PAGESIZE:
デフォルトのページ・サイズを定義するハーフワード 2 進整数

ページ・サイズは、ストリーム出力時、および PLIDUMP データ・セットへのダンプ出力時に使われる値です。

LINESIZE:
デフォルトの行サイズを定義するハーフワード 2 進整数

PAGELENGTH:
端末での印刷で使われるデフォルトのページ長を定義するハーフワード 2 進整数

FILLERS:
3 ハーフワードの 2 進整数。将来の利用のために予約済み

TAB COUNT:
テーブル内のタブ位置の数 (最大 255) を定義するハーフワード 2 進整数

タブ・カウント = 0 の場合は、指定されたタブ位置はすべて無視されます。

Tab1–Tabn:
印刷行内のタブ位置を定義する n 個のハーフワード 2 進整数

最初のタブ位置には 1 という番号がつき、最大値は 255 です。 各タブの値は、テーブル内でその前にあるタブの値より大きくなければなりません。さもなければその値は無視されます。 印刷される出力の最初のデータ・フィールドは、次の有効なタブ位置から始まります。

リンケージ・エディターを使って PLITABS への外部参照を解決すれば、ユーザー・プログラムで PL/I のデフォルトのタブ設定を変更することができます。この外部参照を解決するには、PLITABS という名前のテーブルを前述のフォーマットで与えます。

RENT オプションを使用してコンパイルを行う場合、PLITABS を NONASGN 属性で宣言する必要があります。 NORENT によるコンパイル時に NONASGN 属性を指定することもできるため、PLITABS は常に NONASGN 属性で宣言するようお勧めします。

このタブ・テーブルを供給するには、ソース・プログラムに PLITABS という名前の PL/I 構造体を組み込みます。この構造体を、MAIN プロシージャー内、または MAIN プロシージャーとリンクされるプログラム内で、STATIC EXTERNAL として宣言する必要があります。 297 ページの図 36 は、PL/I 構造体の例を示したものです。 この例では、3 つの位置 (30、60、および 90) にタブが設定され、ページ・サイズおよび行サイズにデフォルト値が使用されています。 TAB1 は行に印刷される 2 番目の項目の位置を識別するものであることに注意してください。行の最初の項目は常に左マージンから始まります。構造体の第 1 項目は NO_OF_TABS フィールドへのオフセットです。 FILL フィールドは省略できません。

```
DCL 1 PLITABS STATIC EXT NONASGN,
    2 (OFFSET INIT(14),
      PAGESIZE INIT(60),
      LINESIZE INIT(120),
      PAGELENGTH INIT(0),
      FILL1 INIT(0),
      FILL2 INIT(0),
      FILL3 INIT(0),
      NO_OF_TABS INIT(3),
      TAB1 INIT(30),
      TAB2 INIT(60),
      TAB3 INIT(90)) FIXED BIN(15,0);
```

図 *36.* 事前設定済みのタブ設定を変更する場合の *PL/I* 構造体 *PLITABS*

31 ビット・プログラムに SYSIN ファイルおよび SYSPRINT ファイルを使用

ユーザー・プログラムに FILE オプションを指定せずに GET ステートメントをコーディングした場合は、コンパイラーは SYSIN というファイル名を挿入します。また、FILE オプションを指定せずに PUT ステートメントをコーディングする場合は、コンパイラーは SYSPRINT という名前を挿入します。

SYSPRINT を宣言しないと、コンパイラーは、通常のデフォルト属性の他に、属性 PRINT をファイルに与えます。 属性の完全セットは次のとおりです。

```
FILE STREAM OUTPUT PRINT EXTERNAL
```

SYSPRINT は PRINT ファイルの一種であるので、コンパイラーはデフォルトの行サイズ (120 文字) や V フォーマット・レコードも与えます。ユーザーは最小限の情報のみを該当する DD ステートメントに与えるだけで済みます。クラス A のシステム出力装置がプリンターであるという通常の規則を使用するユーザーのシステムの場合には、次のステートメントで十分です。

```
 //SYSPRINT DD SYSOUT=A
```

注**:** SYSIN および SYSPRINT は、初期化中のユーザー出口で設定されます。SYSIN および SYSPRINT の IBM 提供のデフォルト値は両方とも端末に送られます。

コンパイラーによって SYSPRINT に与えられた属性は、ファイルを明示的に宣言またはオープンすることによって指定変更することができます。 SYSPRINT と z/OS 言語環境プログラム メッセージ・ファイル・オプション間の対話の詳細については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

コンパイラーは入力ファイル SYSIN のために特別な属性は指定しません。その宣言を行わない場合は、デフォルトの属性のみを受け取ります。 SYSIN に関連するデータ・セットは通常は入力ストリーム内にあります。それが入力ストリーム内にない場合には、完全な DD 情報を与えなくてはなりません。

関連情報:

198 ページの『31 ビット・プログラムに関する SYSPRINT の考慮事項』

64 ビット・プログラムに SYSIN ファイルおよび SYSPRINT ファイルを使用

64 ビット・プログラムの場合、SYSPRINT は C stdout ファイルと同等であり、SYSIN は C stdin ファイルと同等です。

関連情報:

216 ページの『64 ビット・プログラムに関する SYSPRINT の考慮事項』
64 ビット・プログラムの場合、SYSPRINT は C stdout ファイルと同等です。 また、共用 SYSPRINT はサポートされていません。

端末からの入力の制御

次に挙げることを行えば、ユーザーの PL/I プログラムで、入力ファイルへのデータを端末から入力することができます。

1. CONSECUTIVE 環境オプションを指定して、明示的または暗黙的に入力ファイルを宣言する (ストリーム・ファイルはすべてこの条件を満たしています)。
2. 入力ファイルを端末に割り振る

ユーザーは通常、標準のデフォルト入力ファイル SYSIN を使用することができます。このファイルはストリーム・ファイルであり、端末に割り振ることができるからです。

ストリーム・ファイルへの入力をユーザーに促すプロンプトは、コロン (:) で示されます。 プログラムで GET ステートメントが実行されると、その度にコロンが表示されます。また、GET ステートメントが実行されると、システムは次の行に移動します。 ユーザーはそこに必要なデータを入力することができます。 GET ステートメントの実行を完了するのに十分なデータの入っていない行を入力すると、コロンの前に正符号の付いた別のプロンプト (+:) が表示されます。

継続させたい行の最後にハイフンを付け加えると、2 行以上のデータが入力されるまで、ユーザー・プログラムへのデータ伝送を遅らせることができます。

ユーザーのプログラムで、ユーザーに入力を求めるプロンプトを出す出力ステートメントを組み込んだ場合、ユーザー自身のプロンプトをコロンで終了して、初期システム・プロンプトが出ないようにすることができます。 例えば、GET ステートメントの前に次のような PUT ステートメントを置くことができます。

```
PUT SKIP LIST('ENTER NEXT ITEM:');
```

その次の GET ステートメントに対してシステム・プロンプトが表示されないようにするには、独自のプロンプトが次の条件を満たしていなければなりません。

1. 独自のプロンプトは、リスト指示か編集指示のどちらかでなければなりません。また、リスト指示の場合は、PRINT ファイルにあてたものでなければなりません。
2. プロンプトを伝送するファイルは、端末に割り振る必要があります。端末でファイルをコピーするだけの場合は、システム・プロンプトが表示されないようにすることはできません。

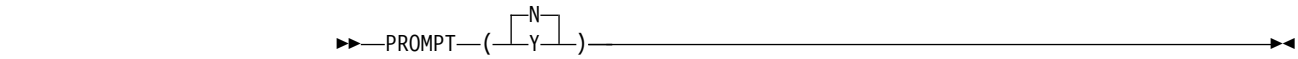
TSO のもとでは、端末からの入力を上記のデフォルトの方法より柔軟な方法で制御することを支援する TSO_INPUT_OPT という環境変数がサポートされています。

TSO_INPUT_OPT 環境変数 (大文字でなければならない) の構文は次のとおりです。

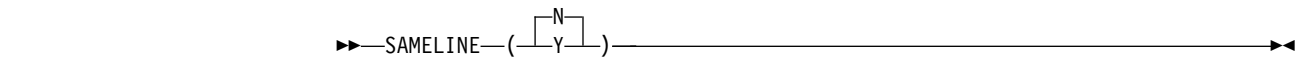
大文字にも小文字にもできるオプションを指定する場合は、ブランクを使用できません。複数のオプションを指定する場合は、それらのオプションをコンマで区切る必要があります。 また、このステートメントの構文は、コマンド入力時にはチェックされません。 データ・セットのオープン時に、このステートメントの構文が検証されます。 構文が間違っていると、ONCODE 96 で UNDEFINEDFILE 条件が発生します。

指定できるオプションは以下のとおりです。

PROMPT
PROMPT オプションは、コロンを端末からのストリーム入力のプロンプトとして表示するかどうかを指定します。



SAMELINE
SAMELINE オプションは、入力を求めるプロンプトのステートメントと同じ行で、システム・プロンプトを行わせるかどうかを指定します。



これらのオプションが端末に対する入出力にどのように影響するのかについて詳しくは、 248 ページの『SAMELINE』にある例を参照してください。

TSO のもとでこの環境変数を指定する 1 つの方法は PLIXOPT ストリングを使用する方法です。次の例を参照してください。

```
DCL PLIXOPT char(50) var ext static
    init('ENVAR("TSO_INPUT_OPT=PROMPT(Y),SAMELINE(Y)")');
```

データのフォーマット

端末で入力するデータのフォーマットは、バッチ・モードのストリーム入力データと完全に同じでなければなりません。ただし、例外がいくつかあります。

- 入力のために簡略化された句読法

  別々の入力項目を別々の行に入力する場合に、間にブランクやコンマを入力する必要はありません。コンパイラーは各行の末尾にコンマを挿入します。

  例えば、次のステートメントに応答するとします。

  ```
  GET LIST(I,J,K);
  ```

端末での対話は、次のようになります。

```
:
1
+:2
+:3
```

各項目の後ろに改行が付きます。 これは、次に示すものとまったく同じです。

```
:
1,2,3
```

ある項目を次の行まで続けたい場合は、1 行目の最後に継続文字を入力します。継続文字がなければ、GET LIST ステートメントまたは GET DATA ステートメントではコンマが挿入され、GET EDIT ステートメントでは埋め込みが行われます (以下参照)。

- GET EDIT での自動埋め込み

  GET EDIT ステートメントに関しては、入力行の末尾にブランクを入力する必要はありません。ユーザーが入力する項目には、正しい長さになるまで埋め込みが行われます。

  例えば、次の PL/I ステートメントを考えてみてください。

  ```
  GET EDIT(NAME)(A(15));
  ```

  SMITH という 5 文字を入力して即時に改行を行うことができます。プログラムが 15 文字からなるストリングを受け取れるように、この項目には 10 個のブランクが埋め込まれます。 ある項目を 2 番目の行または後続行に続けたければ、最終行を除くすべての行の終わりに連結文字を入れなければなりません。そうでなければ伝送される最初の行には、埋め込み処理が行われ、完結したデータ項目として扱われます。

- SKIP オプションまたはフォーマット項目

  GET ステートメントにおける SKIP は、まだ入力されていないデータを無視するようにプログラムに要求します。 SKIP(*n*) の *n* が 1 より大きい場合、その SKIP(n) はすべて SKIP(1) とみなされます。 SKIP(1) は、現在の行にある未使用データはすべて無視されることを意味します。

ストリーム・ファイルおよびレコード・ファイル

ストリーム・ファイルとレコード・ファイルを両方とも端末に割り振ることができます。しかし、この場合、レコード・ファイル用のプロンプトは表示されません。

端末に複数のファイルを割り振ったとき、そのうちの 1 つ以上がレコード・ファイルであると、ファイル出力は必ずしも同期化されるとは限りません。なお、端末へのデータ伝送および端末からのデータ伝送の順序が、対応する PL/I の入出力ステートメントの実行順序と等しくなる保証はありません。

また、端末からのレコード・ファイルの入力は、TCAM の制限により、大文字で受け取られます。問題を避けるため、可能であればストリーム・ファイルをお使いください。

PL/I 動的割り振りを使用して QSAM ファイルを定義

QSAM ファイルや HFS ファイルを定義するには、DD ステートメントや環境変数を使用できるほか、OPEN ステートメントの TITLE オプションも使用できます。

環境変数または TITLE オプションを使用する場合、名前が大文字でなければなりません。 次のいずれかの方法で MVS データ・セットを指定してください。

- DSN(*data-set-name*)
- DSN(*data-set-name* (*member-name*))

data-set-name は完全修飾名でなければなりません。また、data-set-name は一時データ・セットにすることはできません。例えば、先頭が & であってはなりません。

次のように HFS ファイルを指定してください。

PATH (*absolute-path-name*)

DSN キーワードの後に、以下の属性を任意の順序で指定できます。

```
NEW, OLD, SHR, or MOD
TRACKS or CYL
SPACE(n,m)
VOL(volser)
UNIT(type)
KEEP, DELETE, CATALOG, or UNCATALOG
STORCLAS(storageclass)
MGMTCLAS(managementclass)
DATACLAS(dataclass)
```

注: 環境変数、または OPEN ステートメントの TITLE オプションを使用して PDS または PDSE を作成することはできませんが、既存の PDS または PDSE に新規メンバーを作成することは可能です。

大文字と小文字

ストリーム・ファイルでは、文字ストリングは小文字または大文字で入力した通りにプログラムに送られます。 一方、レコード・ファイルでは、文字はすべて大文字になります。

ファイルの終わり

桁 1 および桁 2 に /* があり他の文字はない行の /* は、ファイルの終わりマーカーとして扱われます。すなわち、これらの文字は ENDFILE 条件を発生させます。

UNIX システム・サービス環境では、キー・シーケンス ESC-D もファイルの終わりマーカーとして使用できます。 ESC は特定の文字として事前定義されており、この事前定義された文字をシーケンス内で使用する必要があります。

GET ステートメントの COPY オプション

GET ステートメントは COPY オプションを指定することができますが、入力ファイルとともに COPY ファイルが端末に割り振られた場合は、データのコピーは印刷されません。

第 11 章 端末への出力の制御

端末では、以下の条件を満たす PL/I ファイルからデータを取得できます。

1. CONSECUTIVE 環境オプションを使用して明示的/暗黙的にファイルが宣言されている。ストリーム・ファイルはすべてこの条件を満たしている。
2. ファイルが端末に割り振られている。

標準の印刷ファイル SYSPRINT は、通常、これらの条件を 2 つとも満たしています。

PRINT ファイルのフォーマット

SYSPRINT または他の PRINT ファイルからのデータは、一般的に、端末でページの形にフォーマットされません。 PAGE オプション、LINE オプション、およびフォーマット項目については、常に、3 行がスキップされます。 通常、ENDPAGE 条件が発生することはありません。 SKIP(*n*) の *n* が 3 より大きい場合も、3 行のみがスキップされます。SKIP(0) はバックスペースによってインプリメントされるので、バックスペース機能の備わっていない端末で使用してはなりません。

PRINT ファイルは、ユーザー・プログラムにタブ制御テーブルを挿入することによって、ページの形にフォーマット設定することができます。 このテーブルは PLITABS という名前でなければなりません。また、そのテーブルの内容については、 295 ページの『タブ制御テーブルの指定変更』に説明があります。 ユーザーは、エレメント PAGELENGTH を必要なページ長、すなわち印刷可能な最大行数で表された、各ページを印刷する用紙の長さに初期化しなければなりません。 エレメント PAGESIZE は、各ページに印刷しようとする実際の行数に初期化しなければなりません。 PAGESIZE に指定した行数をページに印刷し終わると、ENDPAGE 条件が生じますが、これに対する標準システム動作は、PAGELENGTH から PAGESIZE を引いた値に等しい行数をスキップし、その後で次ページの印刷を開始することです。 標準以外のレイアウトの場合は、PLITABS 内の他のエレメントを、 195 ページの図 17 で示されている値に初期化しなければなりません。 PLITABS を使って、リスト指示およびリスト指示出力のタブ位置を変更することもできます。 ILC アプリケーションで改ページをフォーマット設定する必要があるとき、SYSPRINT の代わりに PLITABS を使用することができます。 改ページを制御するには、PAGESIZE を 32767 に設定して、PUT PAGE ステートメントを使用します。

端末の中にはタブ設定機能を備えたものもありますが、リスト指示およびリスト指示出力では、常に、ブランク文字を伝送することによってタブが設定されます。

ストリーム・ファイルおよびレコード・ファイル

ストリーム・ファイルとレコード・ファイルを両方とも端末に割り振ることができます。ただし、端末に複数のファイルを割り振り、それらのファイルの 1 つが SYSPRINT またはレコード・ファイルである場合、ファイルの出力は必ずしも同期化されるとは限りません。

© Copyright IBM Corp. 1999, 2017

プログラムと端末との間でやりとりされる時のデータの順序が、それに対応する PL/I 出力ステートメントが実行されるのと同じ順序になるという保証はありません。

PUT EDIT コマンドの出力

PUT EDIT コマンドをからの端末への出力のフォーマットは、行モード TPUT であり、フィールドの始まり およびフィールドの末尾 の文字は、画面上ではブランクとして表示されます。

第 12 章 レコード単位データ伝送の使用

PL/I は、RECORD 属性を持つさまざまなタイプのデータ・セットをサポートしています。このセクションでは、連続データ・セットの使用法について説明します。

表 18 は、レコード単位データ伝送を使って、連続データ・セットを作成したり、連続データ・セットにアクセスする場合に使用できるステートメントとオプションをリストしています。

表 *18.* 連続データ・セットの作成と連続データ・セットへのアクセスで使用できるステートメントとオプション

ファイル宣言[1]	有効ステートメント[2] および必須オプション	指定できるその他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference);	
	LOCATE 基底付き変数 FILE(file-reference);	SET(pointer-reference)
SEQUENTIAL OUTPUT	WRITE FILE(file-reference) FROM(reference);	
SEQUENTIAL INPUT BUFFERED	READ FILE(file-reference) INTO(reference);	
	READ FILE(file-reference) SET(pointer-reference);	
	READ FILE(file-reference) IGNORE(expression);	
SEQUENTIAL INPUT	READ FILE(file-reference) INTO(reference);	
	READ FILE(file-reference) IGNORE(expression);	
SEQUENTIAL UPDATE BUFFERED	READ FILE(file-reference) INTO(reference);	
	READ FILE(file-reference) SET(pointer-reference);	
	READ FILE(file-reference) IGNORE(expression);	
	REWRITE FILE(file-reference);	FROM(reference)

© Copyright IBM Corp. 1999, 2017

表 18. 連続データ・セットの作成と連続データ・セットへのアクセスで使用できるステートメントとオプション (続き)

ファイル宣言[1]	有効ステートメント[2] および必須オプション	指定できるその他のオプション
SEQUENTIAL UPDATE	READ FILE(file-reference) INTO(reference); READ FILE(file-reference) IGNORE(expression); REWRITE FILE(file-reference) FROM(reference);	

注:

1. 完全なファイル宣言には、属性 FILE、RECORD、および ENVIRONMENT が組み込まれています。
2. ステートメント `READ FILE (file-reference);` は有効なステートメントであり、`READ FILE(file-reference) IGNORE (1);` と同等です。

関連情報:

310 ページの『レコード入出力によるデータ・セットの作成』
連続データ・セットを作成するには、SEQUENTIAL OUTPUT の 関連ファイルをオープンする必要があります。 WRITE あるいは LOCATE ステートメントを使用してレコードを書くことができます。

レコード・フォーマットの指定

レコード・フォーマット情報を与える場合には、その情報はデータ・セットの実際の構造に矛盾しないものである必要があります。

例えば、レコード・フォーマット FB、レコード・サイズ 600 バイト、およびブロック・サイズ 3600 バイトでデータ・セットを作成した場合は、最大ブロック・サイズ 3600 バイトの U フォーマット・レコードであるかのようにレコードにアクセスできます。ブロック・サイズを 3500 バイトに指定すると、データが切り捨てられます。

レコード入出力を使用したファイルの定義

ファイル宣言を使用すれば、レコード単位データ伝送用のファイルを定義できます。

```
DCL filename FILE RECORD
             INPUT | OUTPUT | UPDATE
             SEQUENTIAL
             BUFFERED
             ENVIRONMENT(options);
```

デフォルト・ファイル属性については、 260 ページの表 15 を参照してください。
ファイル属性については、「*PL/I* 言語解説書」で説明されています。
ENVIRONMENT 属性のオプションについては、 307 ページの『ENVIRONMENT オプションの指定』を参照してください。

ENVIRONMENT オプションの指定

このセクションでは、連続データ・セットに適用できる ENVIRONMENT オプションについて説明します。

連続データ・セットに適用できる ENVIRONMENT オプションは以下のとおりです。

```
F|FB|FS|FBS|V|VB|U
RECSIZE(record-length)
BLKSIZE(block-size)
SCALARVARYING

CONSECUTIVE or ORGANIZATION(CONSECUTIVE)
CTLASA|CTL360
LEAVE|REREAD
```

これらのオプションについては、以下のトピックを参照してください。

ENVIRONMENT オプション	参照先のトピック
F|FB|FS|FBS|V|VB|U	• 259 ページの『ENVIRONMENT 属性』 • 262 ページの『レコード単位データ伝送のレコード・フォーマット』
RECSIZE(record-length)	• 259 ページの『ENVIRONMENT 属性』 • 263 ページの『RECSIZE オプション』
BLKSIZE(block-size)	• 259 ページの『ENVIRONMENT 属性』 • 264 ページの『BLKSIZE オプション』
SCALARVARYING	• 259 ページの『ENVIRONMENT 属性』 • 267 ページの『SCALARVARYING オプション - 可変長ストリング』
CONSECUTIVE	『CONSECUTIVE』
ORGANIZATION(CONSECUTIVE)	308 ページの『ORGANIZATION (CONSECUTIVE)』
CTLASA|CTL360	308 ページの『CTLASA|CTL360』
LEAVE|REREAD	310 ページの『LEAVE|REREAD』

どのオプションを指定する必要があるか、どれがオプショナルで、どれがデフォルト・オプションであるかを知るには、 260 ページの表 15 を参照してください。

CONSECUTIVE

CONSECUTIVE オプションは、連続データ・セット編成のファイルを定義します。

CONSECUTIVE はデフォルト値です。

関連情報:

253 ページの『データ・セットの編成』
オペレーティング・システムのデータ管理ルーチンは、いくつかのタイプのデータ・セットを処理できます。 これらのデータ・セットは、その内部でのデータ保管方法、およびデータにアクセスするために許可される方法において異なります。

ORGANIZATION(CONSECUTIVE)

ORGANIZATION(CONSECUTIVE) オプションは、ファイルが連続データ・セットに関連付けられるように指定します。

連続ファイルは、ネイティブ・データ・セットでも、VSAM データ・セットでも構いません。

関連情報:

268 ページの『ORGANIZATION オプション』
ORGANIZATION は、PL/I ファイルに関連付けられているデータ・セットの編成を指定します。

CTLASA|CTL360

印刷制御機構オプション CTLASA および CTL360 は、連続データ・セットに関連付けられた OUTPUT ファイルだけに適用されます。 この 2 つのオプションは、レコードの最初の文字を制御文字として解釈するように指定します。

CTLASA オプションは、米国標準規格垂直紙送り位置決め文字 (American National Standard Vertical Carriage Positioning Character) または米国標準規格ポケット選択文字 (American National Standard Pocket Select Character) (レベル 1) を指定するためのオプションです。 また、CTL360 オプションは、IBM のマシン・コード制御文字を指定するためのオプションです。

309 ページの図 37 にリストされている米国標準規格の制御文字は、関連付けられたレコードが印刷、穿孔される前に、指定の処置を実行するための文字です。

なお、マシン・コード制御文字は、装置タイプによって異なります。 プリンター用の IBM マシン・コード制御文字は、 309 ページの表 19 にリストしてあります。

コード	処置
	1 行空けて (ブランク・コード) から印刷する
0	2 行空けてから印刷する
-	3 行空けてから印刷する
+	1 行目から印刷する
1	チャネル 1 にスキップする
2	チャネル 2 にスキップする
3	チャネル 3 にスキップする
4	チャネル 4 にスキップする
5	チャネル 5 にスキップする
6	チャネル 6 にスキップする
7	チャネル 7 にスキップする
8	チャネル 8 にスキップする
9	チャネル 9 にスキップする
A	チャネル 10 にスキップする
B	チャネル 11 にスキップする
C	チャネル 12 にスキップする
V	スタッカー 1 を選択する
W	スタッカー 2 を選択する

図 *37.* 米国標準規格の印刷およびカード穿孔制御文字 *(CTLASA)*

表 *19. IBM* マシン・コード印刷制御文字 *(CTL360)*

印刷してから実行	処置	直ちに処置 **(**印刷は行わない**)**
コード・バイト		コード・バイト
00000001	印刷のみ (スペースなし)	—
00001001	スペース 1 行	00001011
00010001	スペース 2 行	00010011
00011001	スペース 3 行	00011011
10001001	チャネル 1 にスキップする	10001011
10010001	チャネル 2 にスキップする	10010011
10011001	チャネル 3 にスキップする	10011011
10100001	チャネル 4 にスキップする	10100011
10101001	チャネル 5 にスキップする	10101011
10110001	チャネル 6 にスキップする	10110011
10111001	チャネル 7 にスキップする	10111011
11000001	チャネル 8 にスキップする	11000011
11001001	チャネル 9 にスキップする	11001011
11010001	チャネル 10 にスキップする	11010011
11011001	チャネル 11 にスキップする	11011011
11100001	チャネル 12 にスキップする	11100011

LEAVE|REREAD

磁気テープ処理オプション LEAVE および REREAD は、磁気テープ・ボリュームの終わりに達したとき、または磁気テープ・ボリューム上のデータ・セットがクローズされたときに実行されるアクションを指定します。

LEAVE オプションは、テープが巻き戻されないようにします。 REREAD オプションは、テープを巻き戻して、データ・セットの再処理を可能にします。 これらのいずれかを指定しない場合、ボリュームの終わりまたはデータ・セットのクローズ時のアクションは、関連 DD ステートメントの DISP パラメーターによって制御されます。

同じプログラムでデータ・セットを最初は順方向に読み取るか書き込み、次に逆方向で読み取る場合は、ファイルが閉じられたとき (または、マルチボリューム・データ・セットの場合、ボリューム切り替えが生じるとき) にボリュームが巻き戻されないように LEAVE オプションを指定してください。

表 20 は、LEAVE オプションおよび REREAD オプションの影響を要約したものです。

表 *20. LEAVE* および *REREAD* オプションの影響

ENVIRONMENT オプション	**DISP** パラメーター	処置
REREAD	—	データ・セットを再処理する現行ボリュームの位置を決めます。
LEAVE	—	現行ボリュームをデータ・セットの論理終了に位置決めします。
REREAD も LEAVE も指定しない	PASS	データ・セットの終わりにボリュームの位置を決める。
	DELETE	現行ボリュームを巻き戻す。
	KEEP CATLG UNCATLG	現行ボリュームを巻き戻し・アンロードする。

レコード入出力によるデータ・セットの作成

連続データ・セットを作成するには、SEQUENTIAL OUTPUT の関連ファイルをオープンする必要があります。 WRITE あるいは LOCATE ステートメントを使用してレコードを書くことができます。

305 ページの表 18 は、連続データ・セットを作成するためのステートメントとオプションを示しています。

データ・セットを作成する際、DD ステートメント内で、オペレーティング・システムに対してそのデータ・セットを識別しなければなりません。 DD ステートメントに組み込まなければならない必須情報、および指定可能なオプション情報を 表 21 にまとめます。

表 *21.* レコード入出力による連続データ・セットの作成*: DD* ステートメントの必須パラメーター

ストレージ装置	必要な場合	指定すべき事項	パラメーター
全種	常時	出力装置	UNIT= または SYSOUT= または VOLUME=REF=
		ブロック・サイズ[1]	DCB=(BLKSIZE=...
直接アクセスのみ	常時	必要なストレージ・スペース	SPACE=
直接アクセス	データ・セットが別のジョブ・ステップによって使用されるが、ジョブ終了時に必要ではない場合	後処理	DISP=
	ジョブ終了後も保持されるデータ・セット	後処理	DISP=
		データ・セットの名前	DSNAME=
	特定装置上に配置されるデータ・セット	ボリューム通し番号	VOLUME=SER= または VOLUME=REF=

注:

1. または、ENVIRONMENT 属性を用いれば、PL/I プログラム内でブロック・サイズを指定することもできます。

必須情報

データ・セットを作成する際、DD ステートメント内で、オペレーティング・システムに対してそのデータ・セットを識別しなければなりません。 このトピックでは、ステートメントに組み込む必要がある必須情報について説明します。

連続データ・セットを作成する場合は、以下の必須情報を DD ステートメントに指定する必要があります。

- PL/I ファイルと関連付けるデータ・セットの名前

  なお、連続編成のデータ・セットは、どのタイプの装置上にも存在できます。
- レコード長

  レコード長は、ENVIRONMENT 属性、DD_DDNAME 環境変数、または OPEN ステートメントの TITLE オプションの RECSIZE オプションを使って指定することができます。

  端末装置 (stdout: または stderr:) に関連付けられたファイルでは、RECSIZE オプションが指定されていなければ、PL/I はデフォルトのレコード長の 120 を使用します。

レコード入出力によるデータ・セットのアクセスおよび更新

連続データ・セットを作成し終えれば、順次入力、順次出力、または、直接アクセス装置上のデータ・セットの場合は、更新を行うために、その連続データ・セットにアクセスするためのファイルをオープンすることができます。

314 ページの図 38 は、連続データ・セットにアクセスし、それを更新するプログラムの例です。 出力用のファイルをオープンして、その終わりにレコードを追加してデータ・セットを拡張するには、DD ステートメント内に DISP=MOD を指定しなければなりません。それを指定しないと、そのデータ・セットは上書きされます。 更新用のファイルをオープンしても、その既存順にしかレコードを更新することはできず、レコードを挿入したければ、新たにデータ・セットを作成しなければなりません。 連続データ・セットにアクセスしたり連続データ・セットを更新したりするためのステートメントとオプションを 305 ページの表 18 に示します。

SEQUENTIAL UPDATE ファイルで連続データ・セットにアクセスするには、READ ステートメントを使ってレコードを取り出してから、REWRITE ステートメントでそれを更新しなければなりません。しかし、検索される各レコードの再書き込みは必要ありません。 REWRITE ステートメントは、常に、最後に読み取られたレコードを更新します。

次の ステートメントを見てください。

```
READ FILE(F) INTO(A);
     .
     .
     .
READ FILE(F) INTO(B);
     .
     .
     .
REWRITE FILE(F) FROM(A);
```

REWRITE ステートメントによって、2 つ目の READ ステートメントで読み取ったレコードが更新されます。 最初のステートメントで読み取ったレコードは、2 つ目の READ ステートメントが実行されると再書き込みできません。

磁気テープ上の連続データ・セットの場合、末尾へのレコードの追加を除き、更新はできません。 レコードを置換または挿入するには、データ・セットを読み取り、更新したレコードを新規データ・セットに書き込む必要があります。

磁気テープ上の連続データ・セットの読み取りは、順方向にのみ行うことができます。 逆方向読み取りはサポートされていません。

データ・セットにアクセスするには、DD ステートメントでそのデータ・セットをオペレーティング・システムに識別する必要があります。 表 22 は、連続データ・セットにアクセスするのに必要な DD ステートメントのパラメーターを要約しています。

表 *22.* レコード入出力による連続データ・セットへのアクセス*: DD* ステートメントの必須パラメーター

パラメーター	指定すべき事項	必要な場合
DSNAME=	データ・セットの名前	常時

表 22. レコード入出力による連続データ・セットへのアクセス: *DD* ステートメントの必須パラメーター (続き)

パラメーター	指定すべき事項	必要な場合
DISP=	データ・セットの後処理	常時
UNIT= または VOLUME=REF=	入力装置	データ・セットがカタログされていない場合 (すべての装置)
VOLUME=SER=	ボリューム通し番号	データ・セットがカタログされていない場合 (直接アクセス)
DCB=(BLKSIZE=	ブロック・サイズ[1]	データ・セットに標準ラベルが付いていない場合

注:

1. または、ENVIRONMENT 属性を用いれば、PL/I プログラム内でブロック・サイズを指定することもできます。

以下のトピックでは、DD ステートメントに組み込まなければならない必須情報、および指定可能なオプション情報について説明します。ただし、ここで述べる内容は、入力ストリームのデータ・セットには当てはまりません。

必須情報

データ・セットがカタログされている場合は、DD ステートメントに次の情報だけを指定する必要があります。

- データ・セットの名前 (DSNAME パラメーター)

  オペレーティング・システムは、システム・カタログ内でそのデータ・セットを記述した情報を検索し、また、必要があれば、オペレーターに、そのデータ・セットが入っているボリュームをマウントするよう要求します。

- データ・セットが存在することの確認 (DISP パラメーター)

  データ・セットの終わりにレコードを追加して、データ・セットを拡張するために出力用のデータ・セットをオープンするには、DISP=MOD とコーディングします。それを行わないと、出力のためのデータ・セットのオープンが行われたときに、そのデータ・セットは上書きされます。

データ・セットがカタログされていない場合は、データ・セットを読み取る装置を追加で指定する必要があり、直接アクセス装置に関しては、データ・セットが含まれているボリュームの通し番号を指定する必要があります (UNIT パラメーターおよび VOLUME パラメーター)。

連続データ・セットの例

例: 連続データ・セットのマージ、ソート、作成と連続データ・セットへのアクセス

314 ページの図 38 は、連続データ・セットの作成、および連続データ・セットへのアクセスを示したものです。プログラムは、2 つのデータ・セットの内容を入力ストリーム内で組み合わせてから、それを新規データ・セット &&TEMP; に書き込みますが、元のデータ・セットにはおのおの、EBCDIC 照合順序に並べられた 15

バイトの固定長レコードが入っています。 INPUT1 と INPUT2 の 2 つの入力ファイルはデフォルト属性 BUFFERED を持ち、関連データ・セットからレコードをそれぞれのバッファーに読み取るのに位置指定モードが使われます。 バッファー内の基底付き変数へのアクセスは、ファイルがクローズされた後は行わないでください。

```
//EXAMPLE  JOB
//STEP1 EXEC IBMZCBG
//PLI.SYSIN   DD *
%PROCESS INT F(I) AG A(F) OP STG NEST X(F) SOURCE ;
%PROCESS LIST;

 MERGE: PROC OPTIONS(MAIN);
   DCL (INPUT1,                              /* FIRST INPUT FILE     */
        INPUT2,                              /* SECOND INPUT FILE    */
        OUT )      FILE RECORD SEQUENTIAL;   /* RESULTING MERGED FILE*/
   DCL SYSPRINT    FILE PRINT;               /* NORMAL PRINT FILE    */

   DCL INPUT1_EOF  BIT(1) INIT('0'B);        /* EOF FLAG FOR INPUT1  */
   DCL INPUT2_EOF  BIT(1) INIT('0'B);        /* EOF FLAG FOR INPUT2  */
   DCL OUT_EOF     BIT(1) INIT('0'B);        /* EOF FLAG FOR OUT     */
   DCL TRUE        BIT(1) INIT('1'B);        /* CONSTANT TRUE        */
   DCL FALSE       BIT(1) INIT('0'B);        /* CONSTANT FALSE       */

   DCL ITEM1       CHAR(15) BASED(A);        /* ITEM FROM INPUT1     */
   DCL ITEM2       CHAR(15) BASED(B);        /* ITEM FROM INPUT2     */
   DCL INPUT_LINE  CHAR(15);                 /* INPUT FOR READ INTO  */
   DCL A           POINTER;                  /* POINTER VAR          */
   DCL B           POINTER;                  /* POINTER VAR          */

   ON ENDFILE(INPUT1) INPUT1_EOF = TRUE;
   ON ENDFILE(INPUT2) INPUT2_EOF = TRUE;
   ON ENDFILE(OUT)    OUT_EOF    = TRUE;

   OPEN FILE(INPUT1) INPUT,
        FILE(INPUT2) INPUT,
        FILE(OUT)    OUTPUT;

   READ FILE(INPUT1) SET(A);                 /* PRIMING READ         */
   READ FILE(INPUT2) SET(B);

   DO WHILE ((INPUT1_EOF = FALSE) & (INPUT2_EOF = FALSE));
     IF ITEM1 > ITEM2 THEN
       DO;
         WRITE FILE(OUT) FROM(ITEM2);
         PUT FILE(SYSPRINT) SKIP EDIT('1>2', ITEM1, ITEM2)
             (A(5),A,A);
         READ FILE(INPUT2) SET(B);
       END;
     ELSE
       DO;
         WRITE FILE(OUT) FROM(ITEM1);
         PUT FILE(SYSPRINT) SKIP EDIT('1<2', ITEM1, ITEM2)
             (A(5),A,A);
         READ FILE(INPUT1) SET(A);
       END;
   END;
```

図 *38.* 連続データ・セットのマージ、ソート、作成と連続データ・セットへのアクセス

```
   DO WHILE (INPUT1_EOF = FALSE);           /* INPUT2 IS EXHAUSTED  */
     WRITE FILE(OUT) FROM(ITEM1);
     PUT FILE(SYSPRINT) SKIP EDIT('1', ITEM1) (A(2),A);
     READ FILE(INPUT1) SET(A);
   END;

   DO WHILE (INPUT2_EOF = FALSE);           /* INPUT1 IS EXHAUSTED  */
     WRITE FILE(OUT) FROM(ITEM2);
     PUT FILE(SYSPRINT) SKIP EDIT('2', ITEM2) (A(2),A);
     READ FILE(INPUT2) SET(B);
   END;

   CLOSE FILE(INPUT1), FILE(INPUT2), FILE(OUT);
   PUT FILE(SYSPRINT) PAGE;
   OPEN FILE(OUT) SEQUENTIAL INPUT;

   READ FILE(OUT) INTO(INPUT_LINE);         /* DISPLAY OUT FILE     */
   DO WHILE (OUT_EOF = FALSE);
     PUT FILE(SYSPRINT) SKIP EDIT(INPUT_LINE) (A);
     READ FILE(OUT) INTO(INPUT_LINE);
   END;
   CLOSE FILE(OUT);

 END MERGE;
/*
//GO.INPUT1 DD *
AAAAAA
CCCCCC
EEEEEE
GGGGGG
IIIIII
/*
//GO.INPUT2 DD *
BBBBBB
DDDDDD
FFFFFF
HHHHHH
JJJJJJ
KKKKKK
/*
//GO.OUT DD DSN=&&TEMP,DISP=(NEW,DELETE),UNIT=SYSDA,
//          DCB=(RECFM=FB,BLKSIZE=150,LRECL=15),SPACE=(TRK,(1,1))
```

連続データ・セットのマージ、ソート、作成と連続データ・セットへのアクセス *(続き)*

例: レコード単位データ伝送の印刷

316 ページの図 39 のプログラムは、レコード単位データ伝送を使用して、 295 ページの図 35 のプログラムで作成されたテーブルを出力します。

```
%PROCESS INT F(I) AG A(F) OP STG NEST X(F) SOURCE ;
%PROCESS LIST;

 PRT: PROC OPTIONS(MAIN);
   DCL TABLE       FILE RECORD INPUT SEQUENTIAL;
   DCL PRINTER     FILE RECORD OUTPUT SEQL
                        ENV(V BLKSIZE(102) CTLASA);
   DCL LINE        CHAR(94) VAR;

   DCL TABLE_EOF   BIT(1) INIT('0'B);       /* EOF FLAG FOR TABLE   */
   DCL TRUE        BIT(1) INIT('1'B);       /* CONSTANT TRUE        */
   DCL FALSE       BIT(1) INIT('0'B);       /* CONSTANT FALSE       */


   ON ENDFILE(TABLE) TABLE_EOF = TRUE;

   OPEN FILE(TABLE),
        FILE(PRINTER);

   READ FILE(TABLE) INTO(LINE);             /* PRIMING READ         */

   DO WHILE (TABLE_EOF = FALSE);
     WRITE FILE(PRINTER) FROM(LINE);
     READ FILE(TABLE) INTO(LINE);
   END;

   CLOSE FILE(TABLE),
         FILE(PRINTER);
 END PRT;
```

図 *39.* レコード単位データ伝送の印刷

第 13 章 領域データ・セットの定義と使用

この章では、領域データ・セットの編成、データ伝送ステートメント、および領域データ・セットを定義する ENVIRONMENT オプションについて述べます。また、領域編成のタイプごとに、領域データ・セットを作成したり領域データ・セットにアクセスしたりする方法についても説明します。

注: PL/I V5.1 では、領域データ・セットは 64 ビット・プログラムに対してサポートされていません。

領域編成のデータ・セットは 2 つの領域に分かれますが、それぞれの領域は領域番号で識別され、またそのおのおのに、領域編成のタイプに応じて、単数または複数のレコードを入れることができます。 これらの領域には、ゼロから始まる連続番号が付けられ、レコードはデータ伝送ステートメント内に領域番号と一緒にキーを指定することによってアクセスすることができます。

領域データ・セットは、直接アクセス装置に限られます。

データ・セットを領域編成にすれば、データ・セット内でのレコードの物理配置を制御することができ、また、特定アプリケーションへのアクセス時間を最適化することができます。 このような最適化は、連続編成または索引編成では使用することはできません。それは、これらの編成では、昇順キー値に応じて、連続レコードが厳密な物理順序または論理順序で書き込まれるからです。これらの方式のいずれも直接アクセス・ストレージ・デバイスの特性を十分には利用しません。

領域データ・セットは、連続データ・セットまたは索引付きデータ・セットと似た方法で、昇順の領域番号順にレコードを提示することによって作成することができます。別の方法として、直接アクセスを用いることができ、その場合、レコードはランダム順序で提示し、それらを事前にフォーマット設定された領域に直接挿入します。 領域データ・セットを作成した後は、INPUT または UPDATE だけでなく SEQUENTIAL または DIRECT 属性を持ったファイルを使用してそのデータ・セットにアクセスすることができます。データ・セットが SEQUENTIAL INPUT ファイルまたは SEQUENTIAL UPDATE ファイルと関連付けていれば、領域番号またはキーを指定する必要はありません。 ファイルに DIRECT 属性があれば、無作為にレコードを検索、追加、削除、および置換することができます。

領域データ・セット内のレコードは、有効なデータが入っている実際のレコードであるか、またはダミー・レコードのいずれかです。

領域編成での、他のタイプのデータ・セット編成よりも大きな利点は、ユーザーがレコードの相対配置を制御できることにあります。適切なプログラミングにより、装置の能力およびアプリケーションの要件に合わせレコード・アクセスを最適化することができます。

領域データ・セットの直接アクセスは、索引付きデータ・セットのアクセスより早く行うことができますが、領域データ・セットには、順次処理はレコードをランダ

© Copyright IBM Corp. 1999, 2017

ム順に提示することがあるという欠点があります。順次検索の順序は必ずしもレコードが提示される順序ではなく、また必ずしも相対キー値と関連する必要もありません。

表 23 は、領域データ・セットを作成し、また領域データ・セットへアクセスできるためのデータ伝送ステートメントとオプションをリストしています。

表 23. 領域データ・セットの作成と領域データ・セットへのアクセスで使用できるステートメントとオプション

ファイル宣言[1]	有効ステートメント[2] および必須オプション	含めることができるその他のオプション
SEQUENTIAL OUTPUT	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
	LOCATE 基底付き変数 FROM(file-reference) KEYFROM(expression);	SET(pointer-reference)
SEQUENTIAL INPUT	READ FILE(file-reference) INTO(reference);	KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEYTO(reference)
	READ FILE(file-reference) IGNORE(expression);	
SEQUENTIAL UPDATE[3]	READ FILE(file-reference) INTO(reference);	KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEYTO(reference)
	READ FILE(file-reference) IGNORE(expression);	
	REWRITE FILE(file-reference);	FROM(reference)
DIRECT OUTPUT	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
DIRECT INPUT	READ FILE(file-reference) INTO(reference) KEY(expression);	

表 23. 領域データ・セットの作成と領域データ・セットへのアクセスで使用できるステートメントとオプション (続き)

ファイル宣言[1]	有効ステートメント[2] および必須オプション	含めることができるその他のオプション
DIRECT UPDATE	READ FILE(file-reference) INTO(reference) KEY(expression); REWRITE FILE(file-reference) FROM(reference) KEY(expression); WRITE FILE(file-reference) FROM(reference) KEYFROM(expression); DELETE FILE(file-reference) KEY(expression);	

注:

1. 完全なファイル宣言には属性 FILE、 RECORD、および ENVIRONMENT が含まれています。オプション KEY、KEYFROM、あるいは KEYTO のいずれかを使用する場合は、属性 KEYED も含めなくてはなりません。
2. ステートメント `READ FILE(file-reference);` は、ステートメント `READ FILE(file-reference) IGNORE (1);` と同等です。
3. 新たにデータ・セットを作成する場合は、ファイルに UPDATE 属性が含まれていてはなりません。

PL/I 動的割り振りを使用した REGIONAL(1) データ・セットの定義

REGIONAL(1) データ・セットは、DD ステートメントや環境変数を使用したり、OPEN ステートメントの TITLE オプションを使用したりして定義できます。

環境変数または TITLE オプションを使用する場合、名前が大文字でなければなりません。 次のように MVS データ・セットを指定してください。

```
DSN(data-set-name)
```

data-set-name は完全修飾名でなければなりません。また、data-set-name は一時データ・セットにすることはできません。例えば、先頭が & であってはなりません。

DSN キーワードの後に次のいずれかの属性を指定する必要があります。

```
OLD
SHR
```

領域データ・セット用ファイルの定義

ファイル宣言を使用すれば、領域データ・セットを定義できます。

順次領域データ・セットの定義

順次領域データ・セットを定義するには、次の属性が指定されたファイル宣言を使用します。

```
DCL filename FILE RECORD
             INPUT | OUTPUT | UPDATE
             SEQUENTIAL
             BUFFERED
            [KEYED]
             ENVIRONMENT(options);
```

REGIONAL(1) データ・セットの BUFFERED と UNBUFFERED は同じ扱いになるため、ENVIRONMENT オプションではいずれのオプションを指定してもかまいません。例えば、UNBUFFERED が指定されていても、SEQUENTIAL UNBUFFERED ファイルの REWRITE には FROM オプションの必要がなく、OUTPUT SEQUENTIAL データ・セットに対する LOCATE ステートメントが許可されます。

直接領域データ・セットの定義

直接領域データ・セットを定義するには、次の属性が指定されたファイル宣言を使用します。

```
DCL filename FILE RECORD
             INPUT | OUTPUT | UPDATE
             DIRECT
             ENVIRONMENT(options);
```

デフォルト・ファイル属性については、 260 ページの表 15 を参照してください。 ファイル属性について詳しくは、「*PL/I* 言語解説書」を参照してください。ENVIRONMENT オプションのサブオプションについては、『ENVIRONMENT オプションの指定』を参照してください。

ENVIRONMENT オプションの指定

このセクションでは、領域データ・セットに適用できる ENVIRONMENT オプションについて説明します。

領域データ・セットに適用できる ENVIRONMENT オプションは以下のとおりです。

```
REGIONAL({1})
F
RECSIZE(record-length)
BLKSIZE(block-size)
SCALARVARYING
```

REGIONAL

REGIONAL オプションを使用すれば、領域編成のファイルを定義できます。

1 REGIONAL(1) を指定します。

REGIONAL(1)
データ・セットに、記録済みキーのない F フォーマットのレコードが入っていることを表します。データ・セット内の各領域にはただ 1 つのレコードが入っており、したがって、各領域番号はデータ・セット内の相対レコードに対応しています (すなわち、領域番号はデータ・セットの始めから 0 で始まります)。

REGIONAL(1) データ・セットには記録済みキーが 1 つもありませんが、REGIONAL(1) DIRECT INPUT ファイルまたは REGIONAL(1) DIRECT UPDATE ファイルを使えば、記録済みキーのないデータ・セットも処理できます。

RECSIZE(record-length)
BLKSIZE(block-size)
RECSIZE と BLKSIZE の両方を指定する場合は、それぞれに同じ値を指定する必要があります。

重複した領域番号がなく、また大半の領域がいっぱいになる (データ・セット内の無駄なスペースが削減される) アプリケーションの場合には、REGIONAL(1) 編成が最適です。

REGIONAL データ・セットでのキーの使用

キーには、記録済みキー とソース・キー の 2 種類があります。

記録済みキー は、レコードを識別するためにデータ・セット内の各キーの直前に付く文字ストリングです。その長さは 255 文字を超えることはできません。 ソース・キー は、ステートメントが参照するレコードを識別するためにデータ伝送ステートメントの KEY オプションまたは KEYFROM オプション内に現れる式の文字値です。 領域データ・セット内のレコードにアクセスする場合は、ソース・キーは領域番号を与え、同時に、記録済みキーも与えることができます。

索引付きデータ・セット用のキーと異なり、領域データ・セットの記録済みキーはレコード内に埋め込まれることはありません。

REGIONAL(1) データ・セットの使用

REGIONAL(1) データ・セットでは、記録済みキーがないため、領域番号は特定のレコードを識別する唯一のキーとしての役割を果たします。

ソース・キーの文字値は、16777215 を超えてはならない符号なし 10 進整数を表さなければなりません (許容される実際のレコード数は、レコード・サイズ、装置容量、および個々のアクセス方式での制限をどのように組み合わせるかによって、これより小さくなることがあります。固定フォーマット・レコードを持つ直接 regional(1) ファイルに関しては、相対トラック・アドレッシングでアドレスできるトラックの最大数は 65536 です。領域番号がこの数値を超える場合、領域番号はモジュロ 16777216 として扱われ、例えば、16777226 は 10 として扱われます。 0 から 9 までの文字とブランク文字だけが、ソース・キー内では有効です。先行ブランクはゼロとして解釈されます。 領域番号に埋め込まれたブランクを使用することはできません。したがって、埋め込まれたブランクが最初に見つかった時点で、その領域番号は終了します。 ソース・キー中に 8 文字を超えて存在する場合は、右端の 8 文字のみが領域番号として使用され、8 文字未満の場合は、左側にブランク (ゼロとして解釈される) が挿入されます。

ダミー・レコード

REGIONAL(1) データ・セットには、有効なデータが入っている実際のレコード、またはダミー・レコードのいずれかが入っています。 REGIONAL(1) データ・セット内のダミー・レコードは、先頭バイトにある定数 (8)'1'B で識別されます。

このようなダミー・レコードは、データ・セットの作成時かまたはレコードの削除時にデータ・セット内に挿入されますが、データ・セットが読み取られるときに無視されません。ユーザーの PL/I プログラムはそれらを認識するように作成されなくてはなりません。 ダミー・レコードは、有効データで置き換えることができます。 ダミー・レコードの最初のバイトに (8)'1'B を挿入した場合は、検索されないダミー・レコードを持っているデータ・セットにファイルをコピーすると、そのレコードは失われることがあります。

REGIONAL(1) データ・セットの作成

REGIONAL(1) データ・セットは、順次に作成することも直接アクセスによって作成することもできます。

318 ページの表 23 は、領域データ・セットを作成するためのステートメントとオプションを示しています。

SEQUENTIAL OUTPUT ファイルを使ってデータ・セットを作成するとき、ファイルをオープンすると、データ・セット上のすべてのトラックが消去され、各トラックの先頭に、そのトラック上で使用できるスペースの大きさを記録する容量レコードが書き込まれることになります。 レコードは領域番号の昇順で提示しなくてはならず、そのシーケンスから省略された領域はダミー・レコードによって埋められます。 このシーケンスにエラーがあると、あるいは、重複キーを提示すると、KEY 条件が発生します。 ファイルがクローズされるときに、現行エクステントの終わりのスペースにはダミー・レコードが埋め込まれます。

DIRECT OUTPUT ファイルを使ってデータ・セットを作成すると、そのデータ・セットに割り振られた 1 次エクステント全体が、ファイルのオープン時にダミー・レコードで埋められます。 レコードをランダム順で提示することができますが、レコードを重複して提示した場合、既存レコードは上書きされます。

順次作成の場合、データ・セットは最大 15 までのエクステントを持つことができ、また複数のボリューム上にまたがっていてもかまいません。 直接作成の場合、データ・セットは 1 つしかエクステントを持つことはできず、したがって 1 つのボリューム上にしか存在することはできません。

例

REGIONAL(1) データ・セットの作成方法を 323 ページの図 40 に示します。この例のデータ・セットは、電話番号とその電話番号を割り当てる加入者の氏名リストです。電話番号は領域データ・セット内の領域番号と対応しており、各領域番号が占める領域には加入者名のデータが入っています。

```
//EX9    JOB
//STEP1  EXEC IBMZCBG,PARM.PLI='NOP,MAR(1,72)',PARM.BIND='LIST'
//PLI.SYSIN  DD  *
 CRR1:   PROC OPTIONS(MAIN);
  /*  CREATING A REGIONAL(1) DATA SET  -  PHONE DIRECTORY     */

   DCL NOS FILE RECORD OUTPUT DIRECT KEYED ENV(REGIONAL(1));
   DCL  SYSIN FILE INPUT RECORD;
   DCL  SYSIN_REC BIT(1) INIT('1'B);
   DCL 1   CARD,
        2   NAME   CHAR(20),
        2   NUMBER CHAR( 2),
        2   CARD_1 CHAR(58);
   DCL IOFIELD CHAR(20);

      ON ENDFILE (SYSIN) SYSIN_REC = '0'B;
      OPEN FILE(NOS);
      READ FILE(SYSIN) INTO(CARD);

      DO WHILE(SYSIN_REC);
         IOFIELD = NAME;
         WRITE FILE(NOS) FROM(IOFIELD) KEYFROM(NUMBER);
         PUT FILE(SYSPRINT) SKIP EDIT (CARD) (A);
         READ FILE(SYSIN) INTO(CARD);
      END;

      CLOSE FILE(NOS);
  END CRR1;
/*
//GO.SYSLMOD DD DSN=&&GOSET,DISP=(OLD,DELETE)
//GO.NOS     DD DSN=MYID.NOS,UNIT=SYSDA,SPACE=(20,100),
//           DCB=(RECFM=F,BLKSIZE=20,DSORG=DA),DISP=(NEW,KEEP)
//GO.SYSIN DD *
ACTION,G.           12
BAKER,R.            13
BRAMLEY,O.H.        28
CHEESNAME,L.        11
CORY,G.             36
ELLIOTT,D.          85
FIGGINS,E.S.        43
HARVEY,C.D.W.       25
HASTINGS,G.M.       31
KENDALL,J.G.        24
LANCASTER,W.R.      64
MILES,R.            23
NEWMAN,M.W.         40
PITT,W.H.           55
ROLF,D.E.           14
SHEERS,C.D.         21
SURCLIFFE,M.        42
TAYLOR,G.C.         47
WILTON,L.W.         44
WINSTONE,E.M.       37
/*
```

図 *40. REGIONAL(1)* データ・セットの作成

REGIONAL(1) データ・セットへのアクセスと更新

いったん REGIONAL(1) データ・セットを作成すると、SEQUENTIAL INPUT および SEQUENTIAL UPDATE、または DIRECT INPUT および DIRECT

UPDATE のためにそのデータ・セットにアクセスするファイルをオープンすることができます。そのファイルは、既存のデータ・セットを上書きする場合にのみ OUTPUT に関してオープンできます。

318 ページの表 23 は、領域データ・セットにアクセスするためのステートメントとオプションを示しています。

順次アクセス

REGIONAL(1) データ・セットを処理する SEQUENTIAL ファイルをオープンするには、INPUT 属性または UPDATE 属性を使用します。

データ伝送ステートメントには KEY オプションを指定してはなりませんが、KEYTO オプションは使えるため、ファイルは KEYED 属性を持つことができます。KEYTO オプションで参照されるターゲット文字ストリングが 8 文字を超えると、返される値 (8 文字の領域番号) の左側にブランクが埋め込まれます。ターゲット・ストリングが 8 文字より短い場合は、返された値の左側が切り捨てられます。

順次アクセスは、領域番号の昇順で行われます。ダミー・レコードか実レコードかに関係なく、レコードはすべて検索されるため、PL/I プログラムにダミー・レコードを認識させておく必要があります。

REGIONAL(1) データ・セットで順次入力を使用すれば、領域番号の昇順ですべてのレコードを読み取ることができます。また、順次更新では、順番に各レコードを読み取り、もう一度書き込むことが可能です。

REGIONAL(1) データ・セットにアクセスする SEQUENTIAL UPDATE ファイルに対する READ ステートメントと REWRITE ステートメント間の関係を決める規則は、連続データ・セットの場合と同じです。

関連情報:

281 ページの『第 10 章 連続データ・セットの定義と使用』
この章では、連続データ・セット編成について、さらにストリーム指向データ 伝送およびレコード単位データ伝送用の連続データ・セットを定義するため の ENVIRONMENT オプションについて述べています。 その次に、伝送タイプごとに、連続データ・セットの作成、 アクセスおよび更新方法についても説明します。

直接アクセス

REGIONAL(1) データ・セットの処理に使用される DIRECT ファイルをオープンするには、INPUT 属性または UPDATE 属性を使用します。すべてのデータ伝送ステートメントはソース・キーを持っていなければならず、DIRECT 属性は KEYED 属性を暗黙指定します。

次の規則に従って REGIONAL(1) データ・セット内のレコードを検索、追加、削除、または置換するには、DIRECT UPDATE ファイルを使用します。

検索　ダミー・レコードも実際のレコードもすべて検索されます。 したがって、ユーザー・プログラムがダミー・レコードを認識できなくてはなりません。

加算　WRITE ステートメントは、ソース・キーで指定された領域の既存レコード (実際のレコードまたはダミー・レコード) を新規レコードで置き換えます。

削除　DELETE ステートメントでソース・キーを使って指定したレコードは、ダミー・レコードに変換されます。

置換　REWRITE ステートメントでソース・キーを使って指定したレコードは、ダミー・レコードであれ実際のレコードであれ変換されます。

例

ここでは、REGIONAL(1) データ・セットの更新方法について例を交えて説明します。

326 ページの図 41 に示されているプログラムは、データ・セットを更新し、その内容をリストします。別のレコードや更新済みレコードを書き込む前に、その領域内の既存レコードをテストし、それがダミー・レコードであるかどうかを確認します。これは、たとえダミーでなくても WRITE ステートメントは REGIONAL(1) データ・セット中の既存レコードを上書きできるためです。同様に、データ・セットの内容が連続して読み取られたり印刷されたりするときは、各レコードがテストされ、ダミー・レコードは印刷されません。

```
//EX10    JOB
//STEP2   EXEC  IBMZCBG,PARM.PLI='NOP,MAR(1,72)',PARM.BIND='LIST'
//PLI.SYSIN  DD  *
 ACR1: PROC OPTIONS(MAIN);
   /*  UPDATING A REGIONAL(1) DATA SET  -  PHONE DIRECTORY      */
    DCL NOS FILE RECORD  KEYED ENV(REGIONAL(1));
    DCL  SYSIN FILE INPUT RECORD;
    DCL  (SYSIN_REC,NOS_REC) BIT(1) INIT('1'B);
    DCL 1   CARD,
         2   NAME   CHAR(20),
         2   (NEWNO,OLDNO)  CHAR( 2),
         2   CARD_1 CHAR( 1),
         2   CODE   CHAR( 1),
         2   CARD_2 CHAR(54);
    DCL IOFIELD CHAR(20);
    DCL BYTE    CHAR(1) DEF IOFIELD;

   ON ENDFILE(SYSIN) SYSIN_REC = '0'B;
   OPEN FILE (NOS) DIRECT UPDATE;
   READ FILE(SYSIN) INTO(CARD);

   DO WHILE(SYSIN_REC);
      SELECT(CODE);
         WHEN('A','C') DO;
             IF CODE = 'C' THEN
                DELETE FILE(NOS) KEY(OLDNO);
             READ FILE(NOS) KEY(NEWNO) INTO(IOFIELD);
             IF UNSPEC(BYTE) = (8)'1'B
                THEN WRITE FILE(NOS) KEYFROM(NEWNO) FROM(NAME);
             ELSE PUT FILE(SYSPRINT) SKIP LIST ('DUPLICATE:',NAME);
         END;
         WHEN('D') DELETE FILE(NOS) KEY(OLDNO);
      OTHERWISE PUT FILE(SYSPRINT) SKIP LIST ('INVALID CODE:',NAME);
      END;
      READ FILE(SYSIN) INTO(CARD);
   END;

   CLOSE FILE(SYSIN),FILE(NOS);
   PUT FILE(SYSPRINT) PAGE;
   OPEN FILE(NOS) SEQUENTIAL INPUT;
   ON ENDFILE(NOS) NOS_REC = '0'B;
   READ FILE(NOS) INTO(IOFIELD) KEYTO(NEWNO);
   DO WHILE(NOS_REC);
      IF UNSPEC(BYTE) ¬= (8)'1'B
         THEN PUT FILE(SYSPRINT) SKIP EDIT (NEWNO,IOFIELD)(A(2),X(3),A);
      PUT FILE(SYSPRINT) SKIP EDIT (IOFIELD) (A);
      READ FILE(NOS) INTO(IOFIELD) KEYTO(NEWNO);
   END;
  CLOSE FILE(NOS);
  END ACR1;
/*
```

図 *41. REGIONAL(1)* データ・セットの更新

```
//GO.NOS   DD DSN=J44PLI.NOS,DISP=(OLD,DELETE),UNIT=SYSDA,VOL=SER=nnnnnn
//GO.SYSIN DD *
NEWMAN,M.W.         5640 C
GOODFELLOW,D.T.     89   A
MILES,R.              23 D
HARVEY,C.D.W.       29   A
BARTLETT,S.G.       13   A
CORY,G.               36 D
READ,K.M.           01   A
PITT,W.H.             55
ROLF,D.F.             14 D
ELLIOTT,D.          4285 C
HASTINGS,G.M.         31 D
BRAMLEY,O.H.        4928 C
/*
```

REGIONAL(1) データ・セットの更新 *(*続き*)*

領域データ・セットの作成時、および領域データ・セットへのアクセス時の必須情報

領域データ・セットを作成するには、PL/I プログラムまたは DD ステートメントにおいて、データ・セットを定義する特定の情報をオペレーティング・システムに渡す必要があります。

領域データ・セットを作成するには、次の情報を与える必要があります。

- データ・セットを書き込む装置 (DD ステートメントの UNIT パラメーターまたは VOLUME パラメーター)
- ブロック・サイズ

  ブロック・サイズは、PL/I プログラム (ENVIRONMENT 属性の BLKSIZE オプションで) または DD ステートメント (BLKSIZE サブパラメーター) 中で指定することができます。レコード長を指定しないと、非ブロック化レコードがデフォルトになり、レコード長はそのブロック・サイズから決められます。 レコード長を指定する場合は、ブロック・サイズと等しくなければなりません。

データ・セットを保持したい場合 (すなわち、ジョブ終了時にオペレーティング・システムによってデータ・セットが削除されないようにするには)、DD ステートメントを使って、そのデータ・セット名とデータ・セットをどのように処理するのかを指定する必要があります (DSNAME パラメーターおよび DISP パラメーター)。後のステップでデータ・セットを使用したいが、ジョブが終了すればそのデータ・セットは必要なくなるのであれば、DISP パラメーターだけで十分です。

データ・セットを特定の直接アクセス装置に保管したければ、DD ステートメント内でボリューム通し番号を指示しなければなりません (VOLUME パラメーターの SER サブパラメーターまたは REF サブパラメーター)。 とっておきたいデータ・セット用の通し番号を指定しないと、オペレーティング・システムが番号を割り振り、それをオペレーターに通知してから、ユーザーのプログラム・リスト上にその番号を印刷します。

328 ページの表 24 には、領域データ・セットの作成に DD ステートメントで必要となるすべての必須パラメーターが要約されています。 329 ページの表 25 に

は、必要な DCB サブパラメーターがリストされています。 DCB サブパラメーターについては、「*z/OS MVS JCL* ユーザーズ・ガイド」を参照してください。

領域データ・セットは、システム出力 (SYSOUT) 装置上に置くことはできません。

DCB パラメーターで DSORG パラメーターを指定する場合は、DSORG=DA とコーディングすることによって、データ・セット編成を直接アクセスと指定しなければなりません。 領域データ・セット用の DD ステートメント内では、DUMMY パラメーターまたは DSN=NULLFILE パラメーターを指定することはできません。 DSORG=DA を使用すると、メッセージ IEC225I が出されることがあります。このメッセージは安全であり、無視できます。

表 *24.* 領域データ・セットの作成*: DD* ステートメントの必須パラメーター

パラメーター	指定すべき事項	必要な場合
UNIT= または VOLUME=REF=	出力装置[1]	常時
SPACE=	必要なストレージ・スペース[2]	常時
DCB=	データ制御ブロック情報 329 ページの表 25を参照してください。	常時
DISP=	後処理	データ・セットが別のジョブ・ステップで使われるが、別のジョブでは必要ない場合。
DISP=	後処理	ジョブ終了後も保持されるデータ・セット
DSNAME=	データ・セットの名前	
VOLUME=SER= または VOLUME=REF=	ボリューム通し番号	特定のボリューム上に存在するデータ・セット

注:

1. 領域データ・セットは、直接アクセス装置に限られます。
2. 順次アクセスの場合、データ・セットには最大 15 個のエクステントを指定できます。そのエクステントは複数のボリューム上に配置できます。直接アクセスによる作成の場合、データ・セットはエクステントを 1 つだけ持つことができます。

領域データ・セットにアクセスするには、DD ステートメントを使ってそのデータ・セットをオペレーティング・システムに識別する必要があります。 次項に、DD ステートメントに入れなければならない最低限の情報を示します。この情報は 329 ページの表 26 に要約されています。

データ・セットがカタログされている場合は、DD ステートメントに次の情報だけを指定する必要があります。

- データ・セットの名前 (DSNAME パラメーター)

オペレーティング・システムは、システム・カタログ内で該当のデータ・セットを記述する情報を検出し、また、必要があれば、オペレーターに、そのデータ・セットが入っているボリュームをマウントするよう要求します。

- データ・セットが存在することの確認 (DISP パラメーター)

データ・セットがカタログされていない場合は、さらにデータ・セットを読み取る装置、およびデータ・セットが入っているボリュームの通し番号 (UNIT パラメーターおよび VOLUME パラメーター) を指定する必要があります。

順次更新用にマルチボリューム領域データ・セットをオープンすると、最初のボリューム終了時に ENDFILE 条件が発生します。

表 *25.* 領域データ・セットの *DCB* サブパラメーター

サブパラメーター	指定するもの	必要な場合
RECFM=F	レコード・フォーマット[1]	常時
BLKSIZE=	ブロック・サイズ[1]	
DSORG=DA	データ・セットの編成	

[1] あるいは、ENVIRONMENT 属性内でブロック・サイズを指定することもできます。

表 *26.* 領域データ・セットへのアクセス*: DD* ステートメントの必須パラメーター

パラメーター	指定すべき事項	必要な場合
DSNAME=	データ・セットの名前	常時
DISP=	データ・セットの後処理	
UNIT= または VOLUME=REF=	入力装置	データ・セットがカタログされていない場合
VOLUME=SER=	ボリューム通し番号	

第 14 章 VSAM データ・セットの定義と使用

この章では、レコード単位データ伝送用の VSAM (仮想記憶アクセス方式) 編成、VSAM ENVIRONMENT オプション、他の PL/I データ・セット編成との互換性、および PL/I がサポートする 3 つのタイプの VSAM データ・セット (入力順、キー順、および相対レコード) をロードしそれにアクセスするのに使用するステートメントについて述べます。

そしてこの章の終わりには、VSAM データ・セットを作成してそれにアクセスするのに必要な、PL/I ステートメント、アクセス方式サービス・コマンド、および JCL ステートメントの一連の例を示しています。

Enterprise PL/I は、ISAM データ・セットをサポートしていません。

VSAM の各種機能、VSAM データ・セットと索引の構造、それらをアクセス方式サービスが定義する方法、および必須 JCL ステートメントに関する詳細は、ご使用のシステムの VSAM 資料を参照してください。

PL/I 動的割り振りを使用した VSAM ファイルの定義

VSAM データ・セットは、DD ステートメントや環境変数を使用したり、OPEN ステートメントの TITLE オプションを使用したりして定義できます。

環境変数または TITLE オプションを使用する場合、名前が大文字でなければなりません。 次のように MVS データ・セットを指定してください。

```
DSN(data-set-name)
```

data-set-name は完全修飾名でなければなりません。また、data-set-name は一時データ・セットにすることはできません。例えば、先頭が & であってはなりません。

DSN キーワードの後に次のいずれかの属性を指定する必要があります。

```
OLD
SHR
```

VSAM データ・セットの使用

ご使用のプログラムで VSAM データ・セットを使用する必要がある場合は、DD ステートメントを指定して、データ・セットへのアクセス権をプログラムに付与する必要があります。また、ご使用のプログラムは、代替索引パスを使用してキー順データ・セットおよび入力順データ・セットにアクセスすることもできます。

VSAM データ・セットを使用してプログラムを実行

VSAM データ・セットにプログラムがアクセスできるようにするには、特定の情報を DD ステートメントで指定する必要があります。

VSAM データ・セットにアクセスするプログラムを実行する前に、以下の情報を知っておく必要があります。

© Copyright IBM Corp. 1999, 2017

- VSAM データ・セット名
- PL/I ファイルの名前
- データ・セットを他のユーザーと共用するかどうか

その後で、そのデータ・セットにアクセスするのに必要な DD ステートメントを作成することができます。

```
//filename DD DSNAME=dsname,DISP=OLD|SHR
```

例えば、ファイル名が PL1FILE であり、データ・セット名が VSAMDS である場合に、そのデータ・セットを排他的に制御したいのであれば、次のステートメントを入力します。

```
//PL1FILE DD DSNAME=VSAMDS,DISP=OLD
```

データ・セットを他のユーザーと共用するには、DISP=SHR を使用します。

Enterprise PL/I は、ISAM データ・セットをサポートしていません。

データ・セットに使用する VSAM バッファー数を制御して VSAM のパフォーマンスを最適化するには、VSAM 資料を参照してください。

代替索引パスとファイルをペア化

代替索引を使用するには、基本データ・セット/代替索引のペアとユーザーの PL/I ファイルを関連付ける DD ステートメントの DSNAME パラメーターに *path* の名前を指定するだけです。

代替索引を使用する前に、処理についての制約事項に気を付ける必要があります。制限事項は 338 ページの表 30 にまとめてあります。

PL1FILE という PL/I ファイルと PERSALPH という代替索引パスの場合、必要な DD ステートメントは次のようになります。

```
//PL1FILE DD DSNAME=PERSALPH,DISP=OLD
```

VSAM 編成

VSAM データ・セットには 3 つのタイプがあります。各タイプは、大体、PL/I データ・セット編成に対応しています。 VSAM データ・セットは 3 つのタイプともすべて順序付けされます。これらのデータ・セットでは、キーをレコードに関連付けることができます。 3 つのタイプ全部において、順次アクセスとキーによるアクセスの両方を行うことができます。

表 *27. VSAM* データ・セットのタイプ、および対応する *PL/I* データ・セット編成

VSAM データ・セット・タイプ	対応する **PL/I** データ・セット編成
キー順データ・セット (KSDS)	索引付きデータ・セット
入力順データ・セット (ESDS)	連続データ・セット
相対レコード・データ・セット (RRDS)	領域データ・セット

キー順データ・セットだけがその論理レコードの一部としてキーを持つことができますが、入力順データ・セット (相対バイト・アドレスを使用) および相対レコード・データ・セット (相対レコード番号を使用) でもキー順アクセスを行うことができます。

VSAM データ・セットはすべて直接アクセス・ストレージ・デバイスに保持され、そのデータ・セットを使用するには、仮想記憶オペレーティング・システムが必要です。

VSAM データ・セットの物理編成は、他のアクセス方式で使用するものとは異なります。 VSAM ではブロック化の概念は適用されず、また、相対レコード・データ・セットの場合を除き、レコードは固定長でなくてもかまいません。 VSAM 編成のデータ・セットでは、データ項目は制御インターバル に並べられ、さらにそれが制御域 内に並べられます。 処理に備えて、制御インターバル内のデータ項目は論理レコード内に並べられます。 制御インターバルは、1 つ以上の論理レコードを収容することができ、1 つの論理レコードが複数の制御インターバルにスパンしてもかまいません。 VSAM では、ブロック化因数とレコード長に対する配慮は大きく軽減されますが、レコードは、最大指定サイズを超えてはなりません。 VSAM では制御インターバルへアクセスすることもできますが、このタイプのアクセスは PL/I ではサポートされていません。

VSAM データ・セットは、2 つの索引タイプを持つことができます。それは、基本と代替です。基本索引 はデータ・セットを定義するときに設定される KSDS に対する索引で、常に存在し、また KSDS で可能な唯一の索引です。 KSDS または ESDS に対して、1 つ以上の 代替索引 を使用することができます。 ESDS に代替索引 を定義すると、一般に ESDS を KSDS として使用できます。 KSDS の代替索引は、基本索引とは異なる論理レコードのフィールドをキー・フィールドとして使用できます。 代替索引は重複キーが使用できる非固有、重複キーが使用できない固有 のいずれかにすることができます。 基本索引には、重複キーがあってはなりません。

代替索引を持つデータ・セット内の変更は、今後も使用するすべての索引に反映されなければなりません。 この活動は、索引アップグレード として知られ、データ・セット内の索引アップグレード・セット 内の任意の索引について、VSAM が行います。 (KSDS の場合は、基本索引は常に索引アップグレード・セットのメンバーです。) しかし、基本索引または固有代替索引に重複キーを作成する、データ・セット内の変更は避けてください。

VSAM データ・セットを初めて使用する前に、アクセス方式サービスの DEFINE コマンドを使ってそのデータ・セットをシステムに対して定義しなければなりません。これは、データ・セットのタイプ、構造、および必要スペースを完全に定義するのに使用できるコマンドです。 このコマンドはまた、データ・セットが KSDS であるかまたは 1 つ以上の代替索引を持つ場合に、そのデータ・セットの索引 (キーの長さと位置も一緒に) と索引アップグレード・セットも定義します。したがって、VSAM データ・セットはアクセス方式サービスによって「作成」されます。

初期データを新たに作成した VSAM データ・セットに書き込む操作を、本書ではロードする という表現で示しています。

3 つのタイプのデータ・セットは、次の各目的別に使用します。

- 入力順データ・セット は、主として作成された順序 (またはその逆の順序) でアクセスするデータの場合に使用します。
- キー順データ・セット は、通常どおりにレコード内のキーを介してレコードにアクセスする場合に使用します (例えば、レコードにアクセスするのに部品番号が使用される在庫制御ファイル)。
- 相対レコード・データ・セット は、各項目が特定の番号を持っており、その番号によって相対レコードに通常アクセスするようなデータに使用します (例えば、各番号に関連したレコードを持った電話システム)。

VSAM データ・セットのタイプに関係なく、レコードには、キーを使用して直接アクセスすることも、順次に (逆方向でも順方向でも) アクセスすることもできます。2 方向の組み合わせを使用することもできます。キーで開始点を選択し、その点から順方向あるいは逆方向に読み取りを行います。

キー順および入力順データ・セットに代替索引 を作成できます。 作成後は、多くのシーケンスを使用して、または多くのキーのうちのいずれかを使用して、ユーザー・データにアクセスできます。 例えば、従業員番号順に保持されているか、索引を付けられているデータ・セットを使用して、代替索引 内に名前でそれに索引を付けることができます。 それから、英字順、逆英字順、または名前を直接キーとして使用して、そのデータ・セットにアクセスできます。従業員番号の同じ種類の組み合わせによって、データ・セットにアクセスすることもできます。

表 28 に、同一データをどのようにこの 3 つの異なったタイプの VSAM データ・セット内に保持できるかを示し、それぞれの利点と欠点を示しています。

表 *28. VSAM* データ・セットのタイプと利点

データ・セット・タイプ	ロード方式	読み取り方式	更新方式	利点と欠点
キー順	順次に並んだ索引または固有でなければならない基本索引。	基本索引でレコードのキーを指定した KEYED。 任意の索引を逆方向または順方向に SEQUENTIAL。 キーで位置決めした後に、逆方向または順方向の順次読み取り。	任意の索引内で固有キーを指定して KEYED。 固有キーで位置決めした後に SEQUENTIAL。 レコードの削除が許可される。 レコードの追加が許可される。	利点 : 完全アクセスおよび更新。 欠点 : ロード前にレコードが基本索引順にソートされていなければならない。 使用 : アクセスがキーと関連する場合に使用。
入力順	順次 (順方向のみ)。 各レコードの RBA を入手して、キーとして使用可能。	SEQUENTIAL 逆方向または順方向。 RBA を使用して KEYED。 キーで位置決めをした後で、逆方向または順方向に順次。	新しいレコードは終わりにのみ。 既存レコード長は変更不可。 レコードの削除不可。	利点 : 簡単迅速に作成。 固有索引は不要。 欠点 : 限定された更新機能。 使用 : データが主として順次式でアクセスされる場合。

表 28. VSAM データ・セットのタイプと利点 (続き)

データ・セット・タイプ	ロード方式	読み取り方式	更新方式	利点と欠点
相対レコード	スロット 1 から順次。 スロット番号を指定して KEYED。 キーで位置決めした後で、順次書き込み。	番号をキーとして指定して KEYED。 空きレコードを省いて、順方向または逆方向へ順次。	指定したスロットから開始し、次のスロットへ順次。 番号をキーとして指定して KEYED。 レコードの削除が許可される。 空きスロットへのレコード追加が許可される。	利点 : 番号によるレコードへの高速アクセス。 欠点 : 構造が番号付けに拘束される。 固定長レコード。 使用 : レコードが番号によってアクセスされる場合に使用。

VSAM データ・セットのキー

VSAM データ・セットはすべて、それぞれのレコードに関連したキーを持つことができます。

キー順データ・セット、および代替索引 を介して入力順データ・セットにアクセスする場合は、そのキーは論理レコード内の定義されたフィールドになります。入力順データ・セットの場合、キーは、レコードの相対バイト・アドレス (RBA) になります。 相対レコード・データ・セットの場合、キーは、相対レコード番号 になります。

索引付き VSAM データ・セットのキー

キー順データ・セットのキー、および代替索引 を使用してアクセスされる入力順データ・セットのキーは、データ・セットに記録されている論理レコードの一部です。データ・セットの作成時に、キーの長さと位置を定義します。

KEY オプション、KEYFROM オプション、および KEYTO オプションでキーを参照する方法については、「*PL/I* 言語解説書」にある KEY(expression) オプション、KEYFROM(expression) オプション、および KEYTO(reference) オプションに関するトピックを参照してください。

相対バイト・アドレス (RBA)

相対バイト・アドレスを使えば、KEYED SEQUENTIAL ファイルに関連した ESDS 上でキーによるアクセスを使用することができます。

RBA (すなわちキー) は 4 文字の長さの文字ストリングです。その値は VSAM によって定義されます。 PL/I では RBA を構成または操作することはできません。しかし、データ・セット内のレコードの相対位置を判別するためにそれらの値を比較することはできます。RBA は通常は印刷できません。

レコードの RBA を求めるには、データ・セットをロードまたは拡張するときに WRITE ステートメント上で、またはデータ・セットを読み取るときに READ ステートメント上で KEYTO オプションを使用します。これで、READ ステートメント

または REWRITE ステートメントの KEY オプションで、上記のどちらかの方法で取得した RBA を後になってから使用することができます。

WRITE ステートメントの KEYFROM オプションでは、RBA を使用しないでください。

VSAM では、相対バイト・アドレスを KSDS に対するキーとして使用できますが、この使用は PL/I ではサポートされていません。

相対レコード番号

RRDS 内のレコードは相対レコード番号で識別されます。その相対レコード番号は 1 から始まり、後続のレコードごとに 1 ずつ増えていきます。この相対レコード番号は、データ・セットへのキー順アクセスで、キーとして使用することができます。

相対レコード番号として使用されるキーは、長さ 8 の文字ストリングです。 KEY オプションや KEYFROM オプションで使用するソース・キーの文字値は、符号なし整数を表していなければなりません。ソース・キーが 8 文字の長さでなければ、左側で切り捨てられるか、あるいは、ブランク (ゼロと解釈される) が埋められます。 KEYTO オプションが戻す値は、先行ゼロを抑止された長さ 8 の文字ストリングです。

データ・セット・タイプの選択

プログラムを計画するときは、使用するデータ・セットのタイプを最初に決定しなければなりません。 VSAM データ・セットには 3 つのタイプを使用でき、非 VSAM データ・セットには 5 つのタイプを使用できます。

VSAM データ・セットには、他のタイプのデータ・セットにあるすべての機能に加えて、VSAM だけで使用できる機能が備わっています。VSAM はパフォーマンスの点では、通常、他のデータ・セットと互角であり、上回ることも多くあります。ただし、VSAM の方が、機能が誤用された場合に、パフォーマンスが低下する可能性が高くなります。

8 つのタイプのデータ・セットをすべて比較するには、 269 ページの表 16 を参照してください。ただし、大規模インストール・システムでデータ・セット・タイプを選択するときに影響する要因の多くは、本書で説明できる範囲を超えています。

各種 VSAM データ・セットのうちのいずれかを選択するときには、個々のデータを必要とする最も一般的な順序に基づいて選択する必要があります。 次の手順は、データ・セットと索引を組み合わせて必要な機能を得るときに役立ちます。

1. データ・タイプを決めて、それにアクセスする方法を決めます。
   a. 主として順次の場合 - ESDS を優先
   b. 主としてキーによる場合 - KSDS を優先
   c. 主として番号による場合 - RRDS を優先
2. データ・セットのロード方法を決めます。 KSDS はキー・シーケンスでロードしなければならないことに注意してください。したがって、一部のアプリケーションでは、代替索引 パスを持つ ESDS をより実用的な代替として使用することもできます。

3. 代替索引 パスを使用してアクセスする必要があるかどうかを決めます。 これらは、KSDS と ESDS でのみサポートされます。 代替索引 パスが必要な場合は、代替索引 が固有キーを持つか、非固有キーを持つかを決めます。 非固有キーを使用する場合は、キー処理の制限が生じます。しかし、今後のレコードすべてに固有キーを使用するということは実用的ではありません。固有キー用に作成した索引に非固有キーを持つレコードを挿入しようとすると、エラーが発生します。
4. ユーザーが必要とするデータ・セットとパスが決まったら、ユーザーが想定している操作がサポートされているかどうか確認してください。 表 29 を参考にしてください。

ダミーの VSAM データ・セットにはアクセスしないでください。未定義ファイルがあることを示すエラー・メッセージを受け取るためです。

表 29 は、VSAM データ・セットのタイプに基づく、矛盾のないファイル属性の組み合わせを示しています。

表 *29. VSAM* データ・セットと使用できるファイル属性

	SEQUENTIAL	**KEYED SEQUENTIAL**	**DIRECT**
INPUT	ESDS	ESDS	KSDS
	KSDS	KSDS	RRDS
	RRDS	RRDS	Path(U)
	Path(N)	Path(N)	
	Path(U)	Path(U)	
OUTPUT	ESDS	ESDS	KSDS
	RRDS	KSDS	RRDS
		RRDS	Path(U)
UPDATE	ESDS	ESDS	KSDS
	KSDS	KSDS	RRDS
	RRDS	RRDS	Path(U)
	Path(N)	Path(N)	
	Path(U)	Path(U)	

表 29. VSAM データ・セットと使用できるファイル属性 (続き)

	SEQUENTIAL	KEYED SEQUENTIAL	DIRECT

キー:

ESDS 入力順データ・セット

KSDS キー順データ・セット

RRDS 相対レコード・データ・セット

Path(N)
非固有キーを持つ代替索引パス

Path(U)
固有キーを持つ代替索引パス

注:

- 図の左側にある属性と、図の上にある属性を組み合わせて、表示されているデータ・セットとパスを求めることができます。例えば、SEQUENTIAL OUTPUT で使用可能なのは ESDS と RRDS のみです。
- PL/I はダミーの VSAM データ・セットをサポートしていません。

表 30. 代替索引パスで実行できる処理

基本クラスター・タイプ	代替索引キー・タイプ	処理	制約事項
KSDS	固有キー	通常の KSDS として	アクセスのキーは変更できません。
	非固有キー	制限つきキー・アクセス	アクセスのキーは変更できません。
ESDS	固有キー	KSDS として	削除なし。 アクセスのキーは変更できません。
	非固有キー	制限つきキー・アクセス	削除なし。 アクセスのキーは変更できません。

関連情報:

344 ページの『入力順データ・セット』
このトピックでは、入力順データ・セット (ESDS) に関連付けられているファイルに対して使用できるステートメントとオプションについて説明します。

348 ページの『キー順および索引付き入力順データ・セット』
索引付きデータ・セットは、基本索引を持つキー順データ・セット (KSDS) になることも、代替索引 を持つ KSDS または入力順データ・セット (ESDS) になることもあります。 このトピックでは、索引付き VSAM データ・セットに関連付けられているファイルに対して使用できるステートメントとオプションについて説明します。

363 ページの『相対レコード・データ・セット』
このトピックでは、VSAM 相対レコード・データ・セット (RRDS) に関連付けられているファイルに対して使用できるステートメントとオプションについて説明しま

す。

VSAM データ・セットのファイルの定義

このトピックでは、順次 VSAM データ・セットまたは直接 VSAM データ・セットの定義に使用できるファイル宣言について説明します。

順次 VSAM データ・セットの定義

順次 VSAM データ・セットを定義するには、次の属性を指定したファイル宣言を使用します。

```
DCL filename FILE RECORD
             INPUT | OUTPUT | UPDATE
             SEQUENTIAL
             BUFFERED
            [KEYED]
             ENVIRONMENT(options);
```

直接 VSAM データ・セットの定義

直接 VSAM データ・セットを定義するには、次の属性を指定したファイル宣言を使用します。

```
DCL filename FILE RECORD
             INPUT | OUTPUT | UPDATE
             DIRECT
            [KEYED]
             ENVIRONMENT(options);
```

260 ページの表 15 はデフォルト属性を示しています。 ファイル属性については、「*PL/I* 言語解説書」で説明されています。 ENVIRONMENT 属性のオプションについては、『ENVIRONMENT オプションの指定』を参照してください。

ファイル属性 INPUT、OUTPUT、または UPDATE と、ファイル属性 DIRECT、SEQUENTIAL、または KEYED SEQUENTIAL との組み合わせのいくつかは、特定タイプの VSAM データ・セットに対してのみ使用できます。 337 ページの表 29 に互換性のある組み合わせを示しています。

ENVIRONMENT オプションの指定

このセクションでは、VSAM データ・セットに適用できる ENVIRONMENT オプションについて説明します。

データ・セット構造に影響を与える ENVIRONMENT 属性のオプションの多くは、VSAM データ・セットでは必要ありません。 このオプションを指定しても、無視されるか、または検査の目的で使用されるだけです。そこで検査されたものと、そのデータ・セット用に定義された値が矛盾していると、ファイルをオープンしようとした場合、UNDEFINEDFILE 条件が生じます。

VSAM データ・セットに使用できる ENVIRONMENT オプションは以下のとおりです。

```
BKWD
BUFND (n)
BUFNI (n)
BUFSP (n)
```

```
GENKEY
PASSWORD (password-specification)
REUSE
SCALARVARYING
SKIP
VSAM
```

GENKEY および SCALARVARYING オプションは、非 VSAM データ・セットで使用されたときと同じ効果を持ちます。 VSAM RLS 環境では、オプション BUFND、BUFNI、および BUFSP は無視されるので注意してください。

VSAM データ・セットで検査されるオプションは RECSIZE で、キー順データ・セットで検査されるのは KEYLENGTH と KEYLOC の各オプションです。 260 ページの表 15 は VSAM ではどのオプションが無視されるかを示しています。 260 ページの表 15 はまた必須オプションおよびデフォルト・オプションも示しています。

VSAM データ・セットの場合、データ・セットを定義するときに、アクセス方式サービス・ユーティリティーに対して、レコードの最大長と平均長を指定します。 検査の目的でファイル宣言に RECSIZE オプションを組み込む場合は、最大レコード・サイズを指定します。 指定した RECSIZE 値が、データ・セットに対して定義されている値と矛盾する場合は、UNDEFINEDFILE 条件が発生します。

BKWD

BKWD オプションは、VSAM データ・セットに関連付けられている SEQUENTIAL INPUT ファイルまたは SEQUENTIAL UPDATE ファイルに対して逆方向処理を指定します。

順次読み取り (すなわち、KEY オプションなしの読み取り) では、直前の順序にあるレコードが検索されます。 索引付きデータ・セットの場合は、直前のレコードは、一般的に、その次の低位キーを持つレコードのことです。 しかし、非固有な代替索引 を介してデータ・セットにアクセスしている場合は、同じキーを持つレコードは通常のシーケンスでリカバリーされます。例えば、次のレコードを考えてみます。 C1、C2、および C3® のキーは同じです。

```
A B C1 C2 C3 D E
```

レコードは次の順序でリカバリーされます。

```
E D C1 C2 C3 B A
```

BKWD オプションを指定したファイルをオープンすると、データ・セットは最終レコードに位置決めされます。 そのデータ・セットの先頭まで来ると、通常どおりに ENDFILE が生じます。

REUSE オプションや GENKEY オプションと一緒に、BKWD オプションを指定しないでください。 また、BKWD オプションを指定して宣言されたファイルには、WRITE ステートメントは使用できません。

BUFND

BUFND オプションは、VSAM データ・セットに必要となるデータ・バッファーの数を指定します。

n 整数、または属性 FIXED BINARY(31) STATIC を持つ変数を指定します。

ファイルが SEQUENTIAL 属性を持ち、連続レコードの長いグループを順次処理する場合は、複数のデータ・バッファーを使用するとパフォーマンス面で有効です。

BUFNI

BUFNI オプションは、VSAM キー順データ・セットに必要となる索引バッファーの数を指定します。

n 整数、または属性 FIXED BINARY(31) STATIC を持つ変数を指定します。

ファイルが KEYED 属性を持つ場合は、複数の索引バッファーを使用するとパフォーマンス面で有効です。 少なくとも、索引のレベル数と同数の索引バッファー数を指定してください。

BUFSP

BUFSP オプションは、VSAM データ・セットに必要となるバッファー・スペースの合計 (データ・コンポーネントと索引コンポーネントの両方) をバイト単位で指定します。

n 整数、または属性 FIXED BINARY(31) STATIC を持つ変数を指定します。

通常は BUFSP でなく、BUFNI と BUFND オプションを指定することをお勧めします。

GENKEY

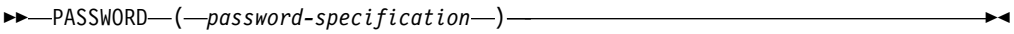
GENKEY (総称キー) オプションは、INDEXED キー順データ・セットおよび VSAM キー順データ・セットにのみ適用されます。 このオプションを使用すれば、データ・セットに記録されているキーを分類したり、SEQUENTIAL KEYED INPUT ファイルや SEQUENTIAL KEYED UPDATE ファイルでキー・クラスに従ってレコードにアクセスしたりできます。

詳しくは、 266 ページの『GENKEY オプション — キーの分類』を参照してください。

PASSWORD

システムに VSAM データ・セットを定義すると (アクセス方式サービスの DEFINE コマンドを使用)、READ パスワードと UPDATE パスワードをシステム

に関連付けることができます。その時点から、データ・セットへのアクセスに使用する PL/I ファイルの宣言には、該当するパスワードを組み込む必要があります。

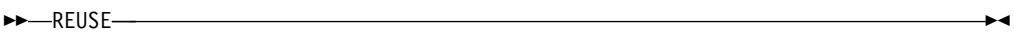


password-specification
プログラムで必要とされるアクセスのタイプに対するパスワードを指定する文字定数または文字変数を指定します。定数を指定する場合は、反復因数を含めることはできません。変数を指定する場合は、レベル 1、エレメント、静的、および添え字なしでなければなりません。

文字ストリングは埋め込みまたは切り捨てが行われて 8 文字にされた後、VSAM に渡されて検査されます。 誤ったパスワードだった場合、システム・オペレーターには、正しいパスワードを指定するための機会が何回か与えられます。その許容回数は、データ・セットを定義するときに指定します。 失敗した試行がこの回数に達すると、UNDEFINEDFILE 条件が発生します。

REUSE

REUSE オプションは、VSAM データ・セットに関連付けられている OUTPUT ファイルを作業ファイルとして使用することを指定します。



データ・セットは、ファイルをオープンするたびに、空のデータ・セットとして扱われます。 そのデータ・セット用のすべての 2 次割り振りが解放され、データ・セットは初めてオープンされたときと同様に取り扱われます。

REUSE オプションが指定されたファイルを、代替索引 を持つデータ・セット、あるいは BKWD オプションが指定されているデータ・セットに関連付けたり、INPUT や UPDATE を行うためにオープンしたりしないでください。

REUSE オプションが有効なのは、アクセス方式サービス DEFINE CLUSTER コマンド内で REUSE を指定した場合だけです。

SKIP

SKIP オプションは、可能であれば常に VSAM OPTCD の「SKP」を使用することを指定します。このオプションは、KEYED SEQUENTIAL INPUT または UPDATE ファイルを使用してアクセスするキー順データ・セットに適用できます。

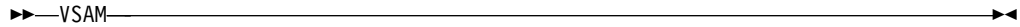
このオプションは、データ・セット全体に分散している個々のレコードにプログラムがアクセスし、そのアクセスが主に昇順のキー順で行われる場合に、ファイルに指定します。

プログラムが KEY オプションを使用せずに多数のレコードを順次読み取る場合、またはプログラムがデータ・セット内の特定のポイントに多数のレコードを挿入する場合は (大量順次挿入)、このオプションを省略してください。

SKIP オプションを指定 (または省略) することは、決して間違った使い方ではありません。このオプションがパフォーマンスに大きく影響するのは、前述の場合のみです。

VSAM

VSAM データ・セットに VSAM オプションを指定する必要があります。



パフォーマンス・オプション

DD ステートメントの AMP パラメーター内にバッファー・オプションを指定することもできます。これらは、アクセス方式サービスの資料に解説があります。

代替索引パスのファイルを定義

VSAM では、キー・シーケンスおよび入力順データ・セットで代替索引を定義できます。

代替索引を使用すれば、基本索引以外のいくつかの方法でキー順データ・セットにアクセスできます。また、入力順データ・セットに索引を付け、キーでアクセスすることも、キーの順番にアクセスすることもできます。その結果、1 つの形式で作成されたデータに、さまざまな方法でアクセスできるようになります。 例えば、ある従業員ファイルが、個人番号、名前、および部門番号によって索引付けられていたとします。

代替索引が作成されると、ユーザーは、実際は代替索引パス (代替索引とデータ・セット間の接続として機能する) とされた 3 番目のオブジェクトを通じてデータ・セットにアクセスします。

ここでは、2 つのタイプの代替索引が使用できます。固有キーと非固有キーです。固有キー代替索引の場合は、それぞれのレコードが別の代替キーを持っていなければなりません。 非固有キー代替索引の場合は、任意の数のレコードが同じ代替キーを持つことができます。 上記の例では、名前が使用されている代替索引は固有キー代替索引にすることができます (各人の名前が異なる場合)。部門番号を使用している代替索引は、各部門に複数の人が所属することができるので非固有キー代替索引になります。

ほとんどの点で、固有キー代替索引パスを通じてアクセスされたデータ・セットを、基本索引を通じてアクセスされた KSDS の場合と同様に扱うことができます。レコードはキーによって、または順次にアクセスでき、ユーザーはレコードを更新し、さらに新規のレコードを追加できます。 データ・セットが KSDS であれば、レコードを削除し、更新したレコードの長さを変更できます。 制約事項と許可されている処理については、 338 ページの表 30 で示しています。 レコードを追加または削除したら、データ・セットに関連したすべての索引が、新しい状況を反映するために変更されます (デフォルト)。

非固有キー代替索引パスを使用してデータ・セットにアクセスする場合、アクセスされるレコードはキーとシーケンスによって判別されます。 キーを使用して位置決めを行えば、順次アクセスを続けることができます。 キーを使用して、最初のレコ

ードにアクセスできます。 データ・セットを逆方向に読み取る場合は、キーの順番のみが逆になります。 同じキーを持つレコードの順番は、データ・セットをいずれの方向で読み取るにしても、同じままです。

VSAM データ・セットの定義

VSAM データ・セットを定義し、カタログするには、アクセス方式サービスの DEFINE CLUSTER コマンドを使用できます。

DEFINE コマンドを使用するには、以下の情報を知っておく必要があります。

- マスター・カタログがパスワード保護されていれば、マスター・カタログの名前とパスワード。
- マスター・カタログを使用しないのであれば、使用する VSAM 専用カタログの名前とパスワード。
- VSAM スペースをデータ・セットに使用できるかどうか。
- 作成する VSAM データ・セットのタイプ。
- データ・セットを入れる先のボリューム。
- データ・セット内の平均および最大レコード・サイズ。
- 索引付きデータ・セット用のキーの位置と長さ。
- データ・セットに割り振られるスペース。
- DEFINE コマンドのコーディング法。
- アクセス方式サービス・プログラムの使用方法。

上記の情報が得られたら、DEFINE コマンドをコーディングし、アクセス方式サービスを使用してデータ・セットを定義およびカタログできます。

入力順データ・セット

このトピックでは、入力順データ・セット (ESDS) に関連付けられているファイルに対して使用できるステートメントとオプションについて説明します。

表 *31. VSAM* 入力順データ・セットのロードと入力順データ・セットへのアクセスで使用できるステートメントとオプション

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference);	KEYTO(reference)
	LOCATE 基底付き変数 FILE(file-reference);	SET(pointer-reference)
SEQUENTIAL INPUT BUFFERED	READ FILE(file-reference) INTO(reference);	KEYTO(reference) または KEY(expression)[3]
	READ FILE(file-reference) SET(pointer-reference);	KEYTO(reference) または KEY(expression)[3]
	READ FILE(file-reference);	IGNORE(expression)

表 *31. VSAM* 入力順データ・セットのロードと入力順データ・セットへのアクセスで使用できるステートメントとオプション *(*続き*)*

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
SEQUENTIAL UPDATE BUFFERED	READ FILE(file-reference) INTO(reference);	KEYTO(reference) または KEY(expression)[3]
	READ FILE(file-reference) SET(pointer-reference);	KEYTO(reference) または KEY(expression)[3]
	READ FILE(file-reference)[2]	IGNORE(expression)
	WRITE FILE(file-reference) FROM(reference);	KEYTO(reference)
	REWRITE FILE(file-reference);	FROM(reference) および/または KEY(expression)[3]

注:

1. 完全なファイル宣言には属性 FILE、RECORD、および ENVIRONMENT が含まれます。オプションの KEY あるいは KEYTO のいずれかを使用する場合は、属性 KEYED も含めなくてはなりません。
2. ステートメント `READ FILE(file-reference);` は、ステートメント `READ FILE(file-reference) IGNORE (1);` と同等です。
3. KEY オプションにおいて使用される式は、あらかじめ KEYTO オプションで取得された相対バイト・アドレスでなければなりません。

ESDS のロード

ESDS がロードされるときには、SEQUENTIAL OUTPUT 用に関連ファイルをオープンする必要があります。 レコードは、提示された順序で保持されます。

KEYTO オプションを使用すれば、各レコードが書き込まれるときの相対バイト・アドレスを入手することができます。 後でこれらのキーを使用すれば、このデータ・セットにキーによるアクセスを行うことができます。

SEQUENTIAL ファイルを使用した ESDS へのアクセス

ESDS にアクセスするのに使用する SEQUENTIAL ファイルは、INPUT 属性または UPDATE 属性を使用してオープンすることができます。 KEY オプションまたは KEYTO オプションを使用する場合には、ファイルには、KEYED 属性も必要です。

順次アクセスの順序は、レコードをデータ・セットに初めにロードしたときと同じです。 読み取られるレコードの RBA を回復するには、READ ステートメントで KEYTO オプションを使用します。 KEY オプションを使用すると、回復されるレコードは、ユーザーが指定する RBA を持つレコードになります。 次の順次アクセスは、データ・セットの新しい場所から開始されます。

UPDATE ファイルの場合、WRITE ステートメントは、データ・セットの終わりに新たにレコードを付け加えます。 REWRITE ステートメントでは、再書き込みの行

われるレコードは、KEY オプションを使用する場合は、指定された RBA を持つものであり、そうでない場合には、直前の READ でアクセスされたレコードです。REWRITE ステートメントで置き換えようとするレコードの長さを変更してはなりません。

入力順データ・セットでは、DELETE ステートメントは使用できません。

ESDS の定義とロード

このトピックでは、入力順データ・セット (ESDS) を定義およびロードするサンプル PL/I プログラムを示します。このプログラムは、SEQUENTIAL OUTPUT ファイルを使用してデータ・セットに書き込みを行います。

347 ページの図 42 に示されているプログラムでは、データ・セットは DEFINE CLUSTER コマンドで定義され、PLIVSAM.AJC1.BASE という名前が付けられています。 NONINDEXED キーワードを使用すると、ESDS が定義されることになります。

PL/I プログラムは、SEQUENTIAL OUTPUT ファイルと WRITE FROM ステートメントを使ってデータ・セットを書き込みます。このファイル用の DD ステートメントには、DEFINE CLUSTER コマンドの NAME パラメーターで与えられたデータ・セットの DSNAME が入っています。

レコードの RBA は、後で使用するために KEYED ファイルにキーとして取得できます。これを行うには、適切な変数を宣言してキーを保持し、WRITE...KEYTO ステートメントを使用する必要があります。次の例を参照してください。

```
DCL CHARS CHAR(4);
WRITE FILE(FAMFILE) FROM (STRING)
 KEYTO(CHARS);
```

通常、キーは印刷できませんが、後で使用するときのためにとっておくことができることに注意してください。

カタログ式プロシージャー IBMZCBG を使用しています。 同じプログラム (347 ページの図 42 参照) を使ってデータ・セットにレコードを追加できるため、このプログラムはライブラリーに保持されています。 レコードを追加する手順について詳しくは、 347 ページの『ESDS の更新』にある例を参照してください。

```
//OPT9#7  JOB
//STEP1    EXEC   PGM=IDCAMS,REGION=512K
//SYSPRINT DD SYSOUT=A
//SYSIN    DD    *
     DEFINE CLUSTER -
      (NAME(PLIVSAM.AJC1.BASE) -
      VOLUMES(nnnnnn) -
      NONINDEXED -
      RECORDSIZE(80 80) -
      TRACKS(2 2))
/*
//STEP2  EXEC IBMZCLG
//PLI.SYSIN      DD *
   CREATE: PROC OPTIONS(MAIN);

      DCL
        FAMFILE FILE SEQUENTIAL OUTPUT ENV(VSAM),
        IN FILE RECORD INPUT,
        STRING CHAR(80),
        EOF BIT(1) INIT('0'B);

      ON ENDFILE(IN) EOF='1'B;

      READ FILE(IN) INTO (STRING);
      DO I=1 BY 1 WHILE (¬EOF);
        PUT FILE(SYSPRINT) SKIP EDIT (STRING) (A);
        WRITE FILE(FAMFILE) FROM (STRING);
        READ FILE(IN) INTO (STRING);
      END;

      PUT SKIP EDIT(I-1,' RECORDS PROCESSED')(A);
    END;
/*
//LKED.SYSLMOD  DD  DSN=HPU8.MYDS(PGMA),DISP=(NEW,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(1,1,1))
//GO.FAMFILE DD   DSNAME=PLIVSAM.AJC1.BASE,DISP=OLD
//GO.IN      DD     *
FRED                     69          M
ANDY                     70          M
SUZAN                    72          F
/*
```

図 *42. ESDS* の定義とロード

ESDS の更新

このトピックでは、ESDS の終わりに新しいレコードを追加する例を示します。

この例は、 346 ページの『ESDS の定義とロード』トピックの 図 42 に基づいています。このプログラムは、レコードをデータ・セットに追加する場合にも使用できます。このプログラムでは、データ・セット PLIVSAM.AJC1.BASE が DEFINE CLUSTER コマンドで定義されています。

348 ページの図 43 では、ESDS の終わりに新規レコードが追加されています。SEQUENTIAL OUTPUT ファイルが使用されていて、それに関連付けられているデータ・セット PLIVSAM.AJC1.BASE が DSNAME パラメーターで指定されています。

```
//OPT9#8   JOB
//STEP1    EXEC  PGM=PGMA
//STEPLIB   DD   DSN=HPU8.MYDS(PGMA),DISP=(OLD,KEEP)
//          DD   DSN=CEE.SCEERUN,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//FAMFILE   DD   DSN=PLIVSAM.AJC1.BASE,DISP=SHR
//IN        DD   *
JANE                    75          F
//
```

図 *43. ESDS* の更新

レコードの長さを変えないかぎり、ESDS 内の既存レコードを再書き込みすることができます。 これには、SEQUENTIAL または KEYED SEQUENTIAL 更新ファイルを使用します。 キーを使用する場合は、RBA または代替索引 パスのキーを使用することができます。

DELETE ステートメントは ESDS に対しては許可されていません。

キー順および索引付き入力順データ・セット

索引付きデータ・セットは、基本索引を持つキー順データ・セット (KSDS) になることも、代替索引 を持つ KSDS または入力順データ・セット (ESDS) になることもあります。このトピックでは、索引付き VSAM データ・セットに関連付けられているファイルに対して使用できるステートメントとオプションについて説明します。

特に断り書きがなければ、表 32 における説明はすべての索引付き VSAM データ・セットに当てはまります。

表 *32. VSAM* 索引付きデータ・セットのロードとそれへのアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
	LOCATE 基底付き変数 FILE(file-reference) KEYFROM(expression);	SET(pointer-reference)
SEQUENTIAL INPUT BUFFERED	READ FILE(file-reference) INTO(reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference);[2]	IGNORE(expression)

表 32. *VSAM* 索引付きデータ・セットのロードとそれへのアクセスに使用できるステートメントとオプション (続き)

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
SEQUENTIAL UPDATE BUFFERED	READ FILE(file-reference) INTO(reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference);[2]	IGNORE(expression)
	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
	REWRITE FILE(file-reference);	FROM(reference) および/または KEY(expression)
	DELETE FILE(file-reference)	KEY(expression)
DIRECT BUFFERED	READ FILE(file-reference) INTO(reference) KEY(expression);	
	READ FILE(file-reference) SET(pointer-reference) KEY(expression);	
DIRECT OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
DIRECT BUFFERED	READ FILE(file-reference) INTO(reference) KEY(expression);	
	READ FILE(file-reference) SET(pointer-reference) KEY(expression);	
	REWRITE FILE(file-reference) FROM(reference) KEY(expression);	
	DELETE FILE(file-reference) KEY(expression);	
	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	

表 32. *VSAM* 索引付きデータ・セットのロードとそれへのアクセスに使用できるステートメントとオプション *(*続き*)*

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション

注:

1. 完全なファイル宣言には、属性 FILE と RECORD が組み込まれています。KEY、KEYFROM または KEYTO オプションのどれか 1 つを使用するときは、宣言内に KEYED 属性も入れる必要があります。
2. ステートメント `READ FILE(file-reference);` は、ステートメント `READ FILE(file-reference) IGNORE (1);` と同等です。
3. SEQUENTIAL OUTPUT ファイルを、代替索引 を介してアクセスするデータ・セットに関連させないでください。
4. DIRECT ファイルを、非固有代替索引 を介してアクセスするデータ・セットに関連させないでください。
5. DELETE ステートメントは、代替索引 を介してアクセスする ESDS と関連させたファイルで使用することはできません。

KSDS または索引付き ESDS のロード

KSDS をロードするときには、KEYED SEQUENTIAL OUTPUT 用の関連ファイルをオープンする必要があります。 レコードは昇順のキー順で指定しなければならず、KEYFROM オプションを使用する必要があります。

データ・セットをロードするためには基本索引を使用しなければならないことに注意してください。代替索引 を介して VSAM データ・セットをロードすることはできません。

KSDS にレコードが既に含まれている場合に、SEQUENTIAL 属性と OUTPUT 属性を持つ関連ファイルをオープンするとき、レコードはデータ・セットの末尾にのみ追加できます。前項のルールが適用されます。特に、指定する最初のレコードはデータ・セット上に存在する最高位のキーより大きいキーを持っていなくてはなりません。

351 ページの図 44 は、KSDS を定義するのに使用する DEFINE コマンドを示しています。 データ・セットには PLIVSAM.AJC2.BASE という名前が与えられ、INDEXED オペランドが使用されているため、KSDS と定義されています。レコード内のキーの位置は、KEYS オペランド内で定義されます。

PL/I プログラム内では、WRITE...FROM...KEYFROM ステートメントで KEYED SEQUENTIAL OUTPUT ファイルが使用されます。データは、昇順のキー順で示されます。KSDS はこの方法でロードしなければなりません。

ファイルは、DEFINE コマンドで付けられた名前を DSNAME パラメーターとして使用する DD ステートメントによってデータ・セットに関連付けられます。

```
//OPT9#12  JOB
//   EXEC  PGM=IDCAMS,REGION=512K
//SYSPRINT  DD  SYSOUT=A
//SYSIN     DD  *
   DEFINE CLUSTER -
     (NAME(PLIVSAM.AJC2.BASE) -
     VOLUMES(nnnnnn) -
     INDEXED -
     TRACKS(3 1) -
     KEYS(20 0) -
     RECORDSIZE(23 80))
/*
//   EXEC IBMZCBG
//PLI.SYSIN      DD *
 TELNOS: PROC OPTIONS(MAIN);

        DCL DIREC FILE RECORD SEQUENTIAL OUTPUT KEYED ENV(VSAM),
            CARD CHAR(80),
            NAME CHAR(20) DEF CARD POS(1),
            NUMBER CHAR(3) DEF CARD POS(21),
            OUTREC CHAR(23) DEF CARD POS(1),
            EOF BIT(1) INIT('0'B);

        ON ENDFILE(SYSIN) EOF='1'B;

        OPEN FILE(DIREC) OUTPUT;

        GET FILE(SYSIN) EDIT(CARD)(A(80));
        DO WHILE (¬EOF);
        WRITE FILE(DIREC) FROM(OUTREC) KEYFROM(NAME);
        GET FILE(SYSIN) EDIT(CARD)(A(80));
        END;

       CLOSE FILE(DIREC);

        END TELNOS;
/*
//GO.DIREC  DD  DSNAME=PLIVSAM.AJC2.BASE,DISP=OLD
//GO.SYSIN  DD  *
ACTION,G.           162
BAKER,R.            152
BRAMLEY,O.H.        248
CHEESEMAN,D.        141
CORY,G.             336
ELLIOTT,D.          875
FIGGINS,S.          413
HARVEY,C.D.W.       205
HASTINGS,G.M.       391
KENDALL,J.G.        294
LANCASTER,W.R.      624
MILES,R.            233
NEWMAN,M.W.         450
PITT,W.H.           515
ROLF,D.E.           114
SHEERS,C.D.         241
SUTCLIFFE,M.        472
TAYLOR,G.C.         407
WILTON,L.W.         404
WINSTONE,E.M.       307
//
```

図 *44.* キー順データ・セット *(KSDS)* の定義とロード

SEQUENTIAL ファイルを使用した KSDS または索引付き ESDS へのアクセス

KSDS にアクセスするのに使用する SEQUENTIAL ファイルは、INPUT 属性または UPDATE 属性を使用してオープンすることができます。

KEY オプションを指定しない READ ステートメントの場合、レコードは昇順のキー順 (ただし、BKWD オプションを使用すると、降順のキー順) で回復されます。KEYTO オプションを使用することによって、このように回復されたレコードのキーを得ることができます。

KEY オプションを使用すると、READ ステートメントで回復されるレコードは、指定したキーを持つレコードになります。 このような READ ステートメントはデータ・セットを指定されたレコードに位置付け、その後の順次読み取りは後続のレコードを順番に回復します。

KEYFROM オプションのある WRITE ステートメントを、 KEYED SEQUENTIAL UPDATE ファイルで使用することができます。 前回行ったアクセスの位置に関係なく、データ・セット内の任意の位置に挿入を行うことができます。 固有索引によってデータ・セットにアクセスするときは、そのデータ・セットに既にあるレコードと同じキーを持つレコードを挿入しようとすると、KEY 条件が生じます。非固有索引の場合には、同一キーを持つレコードを後になって検索すると、レコードがデータ・セットに追加された順に、検索が行われます。

UPDATE ファイルでは、REWRITE ステートメントは、KEY オプションのあるなしに関係なく使用できます。 KEY オプションを使用した場合、再書き込みされるレコードは、指定されたキーを持つ最初のレコードであり、それ以外の場合は直前の READ ステートメントによってアクセスされたレコードです。 代替索引 を使用してレコードを再書き込みする場合は、レコードの基本キーを変更しないでください。

DIRECT ファイルを使用した KSDS または索引付き ESDS へのアクセス

索引付き VSAM データ・セットにアクセスするのに使用する DIRECT ファイルをオープンするには、INPUT 属性、OUTPUT 属性または UPDATE 属性を指定します。 DIRECT ファイルを、非固有索引によってデータ・セットにアクセスするのに使用しないでください。

DIRECT OUTPUT ファイルを使ってデータ・セットにレコードを追加するときに、そのデータ・セットに既にあるレコードと同じキーを持つレコードを挿入しようとすると、KEY 条件が生じます。

DIRECT INPUT ファイルまたは DIRECT UPDATE ファイルを使用すれば、KEYED SEQUENTIAL ファイルの場合と同様に、レコードの読み取り、書き込み、再書き込み、または削除が行えます。

基本索引を使用して KSDS を更新する方法の 1 つを 353 ページの図 45 に示します。

```
//OPT9#13  JOB
//STEP1  EXEC IBMZCBG
//PLI.SYSIN      DD *
 DIRUPDT: PROC OPTIONS(MAIN);

         DCL DIREC FILE RECORD KEYED ENV(VSAM),
             ONCODE BUILTIN,
             OUTREC CHAR(23),
             NUMBER CHAR(3) DEF OUTREC POS(21),
             NAME CHAR(20) DEF OUTREC,
             CODE CHAR(1),
             EOF BIT(1) INIT('0'B);

         ON ENDFILE(SYSIN) EOF='1'B;

         ON KEY(DIREC) BEGIN;
          IF ONCODE=51 THEN PUT FILE(SYSPRINT) SKIP EDIT
                            ('NOT FOUND: ',NAME)(A(15),A);
          IF ONCODE=52 THEN PUT FILE(SYSPRINT) SKIP EDIT
                           ('DUPLICATE: ',NAME)(A(15),A);
          END;

         OPEN FILE(DIREC) DIRECT UPDATE;

       GET FILE(SYSIN) EDIT (NAME,NUMBER,CODE)
         (COLUMN(1),A(20),A(3),A(1));
       DO WHILE (¬EOF);
       PUT FILE(SYSPRINT) SKIP EDIT (' ',NAME,'#',NUMBER,' ',CODE)
       (A(1),A(20),A(1),A(3),A(1),A(1));
       SELECT (CODE);
          WHEN('A') WRITE FILE(DIREC) FROM(OUTREC) KEYFROM(NAME);
          WHEN('C') REWRITE FILE(DIREC) FROM(OUTREC) KEY(NAME);
          WHEN('D') DELETE FILE(DIREC) KEY(NAME);
          OTHERWISE PUT FILE(SYSPRINT) SKIP EDIT
             ('INVALID CODE: ',NAME) (A(15),A);
       END;
       GET FILE(SYSIN) EDIT (NAME,NUMBER,CODE)
         (COLUMN(1),A(20),A(3),A(1));
       END;

         CLOSE FILE(DIREC);
         PUT FILE(SYSPRINT) PAGE;
         OPEN FILE(DIREC) SEQUENTIAL INPUT;

         EOF='0'B;
         ON ENDFILE(DIREC) EOF='1'B;

         READ FILE(DIREC) INTO(OUTREC);
         DO WHILE(¬EOF);
         PUT FILE(SYSPRINT) SKIP EDIT(OUTREC)(A);
         READ FILE(DIREC) INTO(OUTREC);
         END;
         CLOSE FILE(DIREC);
   END DIRUPDT;
```

図 *45. KSDS* の更新

```
/*
//GO.DIREC DD DSNAME=PLIVSAM.AJC2.BASE,DISP=OLD
//GO.SYSIN DD *
NEWMAN,M.W.         516C
GOODFELLOW,D.T.     889A
MILES,R.               D
HARVEY,C.D.W.       209A
BARTLETT,S.G.       183A
CORY,G.                D
READ,K.M.           001A
PITT,W.H.
ROLF,D.F.              D
ELLIOTT,D.          291C
HASTINGS,G.M.          D
BRAMLEY,O.H.        439C
/*
```

KSDS* の更新 *(*続き*)

DIRECT 更新ファイルが使用され、ファイル SYSIN 内のレコードに渡されたコードにしたがって、データが変更されます。

A 新しいレコードの追加
C 既存名の番号の変更
D レコードの削除

ラベル NEXT で、名前、番号、およびコードが読み取られ、そのコードの値に従って処理されます。 KEY ON ユニットを使用して、誤ったキーに対する処理がとられます。 更新が (ラベル PRINT で) 終了すると、ファイル DIREC はクローズされてから、属性 SEQUENTIAL INPUT でもう一度オープンされます。 次に、このファイルは順次読み取られ、印刷されます。

このファイルは、 351 ページの図 44 のアクセス方式サービスの DEFINE CLUSTER コマンドで定義された DSNAME の PLIVSAM.AJC2.BASE を使用する DD ステートメントによって、データ・セットと関連付けられます。

KSDS の更新

KSDS を更新する方法は、いくつかあります。 大量順次挿入の場合は、KEYED SEQUENTIAL UPDATE ファイルを使用します。

この方法では、データは、各 WRITE ステートメントの後ではなく本当に必要な場合にのみデータ・セットに書き込まれるため、またデータ・セット内のフリー・スペースの平衡が保たれるため、パフォーマンスが向上します。

以下のステートメントを使用すれば、効率的な大量順次挿入を行うことができます。

```
DCL DIREC KEYED SEQUENTIAL UPDATE
    ENV(VSAM);
WRITE FILE(DIREC) FROM(OUTREC)
 KEYFROM(NAME);
```

PL/I 入出力ルーチンは、キーが順々になっていることを検出すると、VSAM に正しい要求を出します。 キーが順々になっていない場合には、そのことが検出され、パフォーマンスの利点は失われますがエラーにはなりません。

353 ページの図 45 に示されている例では、KSDS の更新に DIRECT ファイルが使用されています。この方法は、この例に示されているようにデータに適しています。

KSDS または索引付き ESDS の代替索引

代替索引を使用すると、固有キーまたは非固有キーを使用して、さまざまな方法で KSDS または索引付き ESDS にアクセスできます。

ESDS の固有キー代替索引パスを作成

このトピックでは、ESDS の固有キー代替索引パスを作成する方法を説明する例を示します。

356 ページの図 46 では、 347 ページの図 42 に定義されロードされた ESDS の固有キー代替索引パスを作成する方法を説明しています。このパスを使用すると、レコードの最初の 15 バイトの子の名前によって、データ・セットへの索引付けが行われます。

以下のアクセス方式サービス・コマンドが使用されます。

DEFINE ALTERNATEINDEX
代替索引をデータ・セットとして VSAM に定義します。

BLDINDEX
関連レコードへのポインターを代替索引に入れます。

DEFINE PATH
PL/I ファイルと関連付けるエンティティーを、DD ステートメントで定義します。

DD ステートメントは、BLDINDEX の INFILE および OUTFILE オペランド、ならびにソート・ファイルで必要です。 各所で正しい名前が指定されていることを確認してください。

```
//OPT9#9    JOB
//STEP1     EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   DEFINE ALTERNATEINDEX -
     (NAME(PLIVSAM.AJC1.ALPHIND) -
     VOLUMES(nnnnnn) -
     TRACKS(4 1) -
     KEYS(15 0) -
     RECORDSIZE(20 40) -
     UNIQUEKEY -
    RELATE(PLIVSAM.AJC1.BASE))
/*
//STEP2     EXEC PGM=IDCAMS,REGION=512K
//DD1       DD   DSNAME=PLIVSAM.AJC1.BASE,DISP=SHR
//DD2       DD   DSNAME=PLIVSAM.AJC1.ALPHIND,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   BLDINDEX  INFILE(DD1)  OUTFILE(DD2)
   DEFINE  PATH -
     (NAME(PLIVSAM.AJC1.ALPHPATH) -
     PATHENTRY(PLIVSAM.AJC1.ALPHIND))
//
```

図 *46. ESDS* の固有キー代替索引パスを作成

ESDS の非固有キー代替索引パスを作成

このトピックでは、ESDS の非固有キー代替索引パスを作成する方法を説明する例を示します。

357 ページの図 47 では、ESDS の非固有キー代替索引パスを作成する方法を説明しています。代替索引を使用すれば、子の性によってデータを選択できます。これにより少女または少年のデータに別々にアクセスでき、キーを使用して、それぞれのグループの各メンバーにアクセスできます。

以下のアクセス方式サービス・コマンドが使用されます。

DEFINE ALTERNATEINDEX
代替索引をデータ・セットとして VSAM に定義します。

BLDINDEX
関連レコードへのポインターを代替索引に入れます。

DEFINE PATH
PL/I ファイルと関連付けるエンティティーを、DD ステートメントで定義します。

DD ステートメントは、BLDINDEX の INFILE および OUTFILE オペランド、ならびにソート・ファイルで必要です。 各所で正しい名前が指定されていることを確認してください。

この例で、NONUNIQUEKEY オペランドは、索引に非固有キーがあることを指定します。非固有キーを持つ索引を作成する場合は、十分な大きさの RECORDSIZE を指定してください。非固有代替索引では、それぞれの代替索引レコードが、関連

した索引キーを持つすべてのレコードへのポインターを含みます。 ポインターは、ESDS の場合は RBA の形式で、KSDS の場合は基本キーの形式です。 多くのレコードが同じキーを持つ場合は、大きなレコードが必要です。

```
//OPT9#10   JOB
//STEP1     EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   /* care must be taken with recordsize */
   DEFINE ALTERNATEINDEX -
     (NAME(PLIVSAM.AJC1.SEXIND) -
     VOLUMES(nnnnnn) -
     TRACKS(4 1) -
     KEYS(1 37) -
     RECORDSIZE(20 400) -
     NONUNIQUEKEY -
    RELATE(PLIVSAM.AJC1.BASE))
/*
//STEP2     EXEC PGM=IDCAMS,REGION=512K
//DD1       DD   DSNAME=PLIVSAM.AJC1.BASE,DISP=SHR
//DD2       DD   DSNAME=PLIVSAM.AJC1.SEXIND,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   BLDINDEX  INFILE(DD1)  OUTFILE(DD2)
   DEFINE  PATH -
     (NAME(PLIVSAM.AJC1.SEXPATH) -
     PATHENTRY(PLIVSAM.AJC1.SEXIND))
//
```

図 *47. ESDS* の非固有キー代替索引パスを作成

KSDS の固有キー代替索引パスを作成

このトピックでは、KSDS の固有キー代替索引パスを作成する方法を説明する例を示します。

358 ページの図 48 では、KSDS の固有キー代替索引パスを作成する方法を説明しています。 電話番号でデータ・セットに索引が付けられます。これにより、電話番号をキーとして使用して、電話番号の持ち主の名前を検索できるようになります。また、どの番号が使用されていないかを示すためにデータ・セットを番号順にリストすることができます。

この例で、UNIQUEKEY オペランドは、キーが固有であることを指定します。

以下のアクセス方式サービス・コマンドが使用されます。

DEFINE ALTERNATEINDEX
: 代替索引データを保持するデータ・セットを定義します。

BLDINDEX
: 関連レコードへのポインターを代替索引に入れます。

DEFINE PATH
: PL/I ファイルと関連付けるエンティティーを、DD ステートメントで定義します。

DD ステートメントは、BLDINDEX の INFILE および OUTFILE、ならびにソート・ファイルで必要です。 組み込まれる名前を混同することがないようにしてください。

```
//OPT9#14   JOB
//STEP1     EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   DEFINE ALTERNATEINDEX -
     (NAME(PLIVSAM.AJC2.NUMIND) -
     VOLUMES(nnnnnn) -
     TRACKS(4 4) -
     KEYS(3 20) -
     RECORDSIZE(24 48) -
     UNIQUEKEY -
    RELATE(PLIVSAM.AJC2.BASE))
/*
//STEP2     EXEC PGM=IDCAMS,REGION=512K
//DD1       DD   DSNAME=PLIVSAM.AJC2.BASE,DISP=SHR
//DD2       DD   DSNAME=PLIVSAM.AJC2.NUMIND,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
   BLDINDEX  INFILE(DD1)  OUTFILE(DD2)
   DEFINE  PATH -
     (NAME(PLIVSAM.AJC2.NUMPATH) -
     PATHENTRY(PLIVSAM.AJC2.NUMIND))
//
```

図 *48. KSDS* の固有キー代替索引パスを作成

固有キーを使用して代替索引を作成する場合は、同じ代替キーでさらにレコードが組み込まれることがないようにしてください。実際、固有キー代替索引は、電話帳として使用するには、完全に満足のいくものとはいえません。なぜなら、固有キー代替索引では、2 人の人間が同じ番号を持つことができないためです。同様に、基本キーでは 1 人の人間が 2 つの電話番号を持てません。 解決策として、ESDS に 2 つの非固有キー代替索引を持たせるか、または 1 人の人間が複数の番号を持てるようにデータ・フォーマットを再構築して、複数の番号に対応した非固有キー代替索引を 1 つ持たせるようにします。

非固有代替索引キーの検出

代替索引パスを使用して VSAM データ・セットにアクセスする場合は、SAMEKEY 組み込み関数によって非固有キーの有無を検出できます。

非固有キーを検出した場合、SAMEKEY は、検索されたレコードと同じ代替索引キーを持つレコードが他に存在するかどうかを示します。このため、非固有キーを持つ一連のレコードの最後で停止できます (最後のレコードより先を読み取る必要はありません)。 SAMEKEY (ファイル参照) は、入出力ステートメントが正しく完了し、アクセスしたレコードの次に、同じキーを持つ別のレコードが続く場合は '1'B を戻します。そうでない場合は '0'B を戻します。

ESDS での代替索引の使用

360 ページの図 49 は、ESDS での代替索引の使用と、逆方向読み取りについて説明します。このプログラムは以下の 4 つのファイルを使用します。

BASEFLE
基本データ・セットを順方向に読み取る。

BACKFLE
基本データ・セットを逆方向に読み取る。

ALPHFLE
名前によって子の索引付けを行う英字代替索引パス。

SEXFILE
子の性別に対応する代替索引パス。

すべてのファイルに DD ステートメントがあります。 DSNAME パラメーターに基本データ・セット名を指定して、BASEFLE と BACKFLE を基本データ・セットに接続し、 356 ページの図 46 と 357 ページの図 47 で示したパス名を指定して ALPHFLE と SEXFILE を接続します。

プログラムではデータにアクセスするために SEQUENTIAL ファイルを使用し、最初はそれを通常の順で書き込みし、それから逆順で書き込みします。 AGEQUERY ラベルでは、固有代替索引の代替索引キーに関連付けられたデータを読み取るために、DIRECT ファイルが使用されます。

最後に、SPRINT ラベルでは、非固有キー代替索引パスを使用して、ファミリー内の女性のリストを書き込むために KEYED SEQUENTIAL ファイルが使用されます。 同じキーを持つすべてのレコードを読み取るために、SAMEKEY 組み込み関数が使用されます。 女性名はもともと入力された順でアクセスされます。 これが行われるのはファイルが順方向または逆方向に読み取られる場合です。 非固有キー・パスの場合は、BKWD オプションはキーの読み取り順にのみ影響します。同じキーを持つ項目の順番はファイルが順方向に読み取られる場合と同じです。

例の終わりでは、アクセス方式サービスの DELETE コマンドが基本データ・セットを削除するために使用されています。これが行われると、関連する代替索引とパスも削除されます。

KSDS で代替索引を使用

362 ページの図 50 では、KSDS を更新するための、固有な代替索引キーを持つパスの使用法と代替索引の順にそれにアクセスし出力する方法を説明します。

代替索引パスは、パスの名前 (PLIVSAM.AJC2.NUMPATH、 358 ページの図 48 の DEFINE PATH コマンドで指定されたもの) を DSNAME として指定する DD ステートメントによって、PL/I ファイルに関連付けられています。

プログラムの最初のセクションでは、代替索引キーを使用して新規のレコードを挿入するために、DIRECT OUTPUT ファイルが使用されます。代替索引を用いて変更を加える場合は、既存のレコードへのアクセス用の基本キーや代替索引キーを変更しないでください。また、変更の際に、基本索引や固有キー代替索引に、重複するキーを追加しないでください。

プログラムの 2 番目のセクションでは (PRINTIT ラベルで)、データ・セットが SEQUENTIAL INPUT ファイルを使用して代替索引キーの順で読み取られます。その後、SYSPRINT に書き込まれます。

```
//OPT9#15  JOB
//STEP1  EXEC IBMZCLG
//PLI.SYSIN      DD *
  READIT: PROC OPTIONS(MAIN);
    DCL   BASEFLE FILE SEQUENTIAL INPUT ENV(VSAM),
                 /*File to read base data set forward */
          BACKFLE FILE SEQUENTIAL INPUT ENV(VSAM BKWD),
                 /*File to read base data set backward */
          ALPHFLE FILE DIRECT INPUT ENV(VSAM),
          /*File to access via unique alternate index path */
          SEXFILE FILE KEYED SEQUENTIAL INPUT ENV(VSAM),
           /*File to access via nonunique alternate index path */
          STRING CHAR(80),  /*String to be read into */
          1 STRUC DEF (STRING),
            2 NAME CHAR(25),
            2 DATE_OF_BIRTH CHAR(2),
            2 FILL CHAR(10),
            2 SEX CHAR(1);
            DCL NAMEHOLD CHAR(25),SAMEKEY BUILTIN;
            DCL EOF BIT(1) INIT('0'B);

       /*Print out the family eldest first*/

       ON ENDFILE(BASEFLE) EOF='1'B;
       PUT EDIT('FAMILY ELDEST FIRST')(A);
       READ FILE(BASEFLE) INTO (STRING);
       DO WHILE(¬EOF);
          PUT SKIP EDIT(STRING)(A);
          READ FILE(BASEFLE) INTO (STRING);
       END;
       CLOSE FILE(BASEFLE);
       PUT SKIP(2);
        /*Close before using data set from other file not
          necessary but good practice to prevent potential
          problems*/

       EOF='0'B;
       ON ENDFILE(BACKFLE) EOF='1'B;
       PUT SKIP(3) EDIT('FAMILY YOUNGEST FIRST')(A);
       READ FILE(BACKFLE) INTO(STRING);
       DO WHILE(¬EOF);
         PUT SKIP EDIT(STRING)(A);
         READ FILE(BACKFLE) INTO (STRING);
       END;

       CLOSE FILE(BACKFLE);
       PUT SKIP(2);

       /*Print date of birth of child specified in the file
         SYSIN*/
       ON KEY(ALPHFLE) BEGIN;
          PUT SKIP EDIT
             (NAMEHOLD,' NOT A MEMBER OF THE SMITH FAMILY') (A);
          GO TO SPRINT;
       END;
```

図 *49. ESDS* での代替索引パスと逆方向読み取り

```
AGEQUERY:
   EOF='0'B;
   ON ENDFILE(SYSIN) EOF='1'B;
   GET SKIP EDIT(NAMEHOLD)(A(25));
   DO WHILE(¬EOF);
         READ FILE(ALPHFLE) INTO (STRING) KEY(NAMEHOLD);
         PUT SKIP (2) EDIT(NAMEHOLD,' WAS BORN IN ',
             DATE_OF_BIRTH)(A,X(1),A,X(1),A);
         GET SKIP EDIT(NAMEHOLD)(A(25));
   END;
SPRINT:
   CLOSE FILE(ALPHFLE);
   PUT SKIP(1);

  /*Use the alternate index to print out all the females in the
    family*/
      ON ENDFILE(SEXFILE) GOTO FINITO;
      PUT SKIP(2) EDIT('ALL THE FEMALES')(A);
      READ FILE(SEXFILE) INTO (STRING) KEY('F');
      PUT SKIP EDIT(STRING)(A);
      DO WHILE(SAMEKEY(SEXFILE));
         READ FILE(SEXFILE) INTO (STRING);
         PUT SKIP EDIT(STRING)(A);
      END;

FINITO:
    END;
/*
//GO.BASEFLE   DD   DSN=PLIVSAM.AJC1.BASE,DISP=SHR
//GO.BACKFLE   DD   DSN=PLIVSAM.AJC1.BASE,DISP=SHR
//GO.ALPHFLE   DD   DSN=PLIVSAM.AJC1.ALPHPATH,DISP=SHR
//GO.SEXFILE   DD   DSN=PLIVSAM.AJC1.SEXPATH,DISP=SHR
//GO.SYSIN     DD    *
ANDY
/*
//STEP2    EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT    DD    SYSOUT=A
//SYSIN       DD     *
     DELETE -
          PLIVSAM.AJC1.BASE
//
```

ESDS での代替索引パスと逆方向読み取り *(続き)*

```
//OPT9#16   JOB
//STEP1     EXEC IBMZCLG,REGION.GO=256K
//PLI.SYSIN DD   *
   ALTER: PROC OPTIONS(MAIN);
     DCL NUMFLE1 FILE RECORD DIRECT OUTPUT ENV(VSAM),
         NUMFLE2 FILE RECORD SEQUENTIAL INPUT ENV(VSAM),
         IN FILE RECORD,
         STRING CHAR(80),
         NAME CHAR(20) DEF STRING,
         NUMBER CHAR(3) DEF STRING POS(21),
         DATA CHAR(23) DEF STRING,
         EOF BIT(1) INIT('0'B);

     ON KEY (NUMFLE1) BEGIN;
       PUT SKIP EDIT('DUPLICATE NUMBER')(A);
     END;

     ON ENDFILE(IN) EOF='1'B;

     READ FILE(IN) INTO (STRING);
     DO WHILE(¬EOF);
       PUT FILE(SYSPRINT) SKIP EDIT (STRING) (A);
       WRITE FILE(NUMFLE1) FROM (STRING) KEYFROM(NUMBER);
       READ FILE(IN) INTO (STRING);
     END;

     CLOSE FILE(NUMFLE1);

     EOF='0'B;
     ON ENDFILE(NUMFLE2) EOF='1'B;

     READ FILE(NUMFLE2) INTO (STRING);
     DO WHILE(¬EOF);
       PUT SKIP EDIT(DATA)(A);
       READ FILE(NUMFLE2) INTO (STRING);
     END;

     PUT SKIP(3) EDIT('****SO ENDS THE PHONE DIRECTORY****')(A);
  END;
/*
//GO.IN    DD   *
RIERA L            123
/*
//NUMFLE1    DD   DSN=PLIVSAM.AJC2.NUMPATH,DISP=OLD
//NUMFLE2    DD   DSN=PLIVSAM.AJC2.NUMPATH,DISP=OLD
//STEP2    EXEC PGM=IDCAMS,COND=EVEN
//SYSPRINT   DD  SYSOUT=A
//SYSIN   DD    *
     DELETE -
     PLIVSAM.AJC2.BASE
//
```

図 *50. KSDS* アクセス用の固有代替索引パスの使用

相対レコード・データ・セット

このトピックでは、VSAM 相対レコード・データ・セット (RRDS) に関連付けられているファイルに対して使用できるステートメントとオプションについて説明します。

表 *33. VSAM* 相対レコード・データ・セットのロードとそれへのアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference);	KEYFROM(expression) または KEYTO(reference)
	LOCATE 基底付き変数 FILE(file-reference);	SET(pointer-reference)
SEQUENTIAL INPUT BUFFERED	READ FILE(file-reference) INTO(reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference);[2]	IGNORE(expression)
SEQUENTIAL UPDATE BUFFERED	READ FILE(file-reference) INTO(reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference) SET(pointer-reference);	KEY(expression) または KEYTO(reference)
	READ FILE(file-reference);[2]	IGNORE(expression)
	WRITE FILE(file-reference) FROM(reference);	KEYFROM(expression) または KEYTO(reference)
	REWRITE FILE(file-reference);	FROM(reference) および/または KEY(expression)
	DELETE FILE(file-reference);	KEY(expression)
DIRECT OUTPUT BUFFERED	WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	
DIRECT INPUT BUFFERED	READ FILE(file-reference) INTO(reference) KEY(expression);	
	READ FILE(file-reference) SET(pointer-reference) KEY(expression);	

表 33. *VSAM* 相対レコード・データ・セットのロードとそれへのアクセスに使用できるステートメントとオプション *(続き)*

ファイル宣言[1]	有効ステートメントおよび	含めることができるその他のオプション
DIRECT UPDATE BUFFERED	READ FILE(file-reference) INTO(reference) KEY(expression); READ FILE(file-reference) SET(pointer-reference) KEY(expression); REWRITE FILE(file-reference) FROM(reference) KEY(expression); DELETE FILE(file-reference) KEY(expression); WRITE FILE(file-reference) FROM(reference) KEYFROM(expression);	

注:

1. 完全なファイル宣言には、属性 FILE と RECORD が組み込まれています。 KEY、KEYFROM、KEYTO オプションのどれか 1 つを使用する場合は、宣言に属性 KEYED も指定する必要があります。

   DIRECT UPDATE ファイルでの UNLOCK ステートメントは VSAM RRDS に関連したファイルに使用されると無視されます。

2. ステートメント `READ FILE(file-reference);` は、ステートメント `READ FILE(file-reference) IGNORE (1);` と同等です。

RRDS のロード

RRDS をロードするときには、OUTPUT 用の関連ファイルをオープンする必要があります。 DIRECT ファイルまたは SEQUENTIAL ファイルを使用します。

DIRECT OUTPUT ファイルの場合、各レコードは、WRITE ステートメントの KEYFROM オプションの相対レコード番号 (つまりキー) で指定された位置に配置されます (335 ページの『VSAM データ・セットのキー』を参照)。

SEQUENTIAL OUTPUT ファイルの場合、KEYFROM オプションの指定の有無に関係なく WRITE ステートメントを使用します。 KEYFROM オプションを指定すると、レコードは指定されたスロット内に置かれ、省略した場合は、レコードは現在位置に続くスロット内に置かれます。 レコードを昇順の相対レコード番号順に並べる必要はありません。 KEYFROM オプションを省略しても、KEYTO オプションを使用することにより、書き込まれたレコードの相対レコード番号を取得することができます。

KEYFROM オプションも KEYTO オプションも使わずに、RRDS を順次にロードする場合は、ファイルは KEYED 属性を持っている必要はありません。

既にレコードが存在する位置にレコードをロードしようとすることは誤りです。KEYFROM オプションを使用する場合は、KEY 条件が生じ、それを省略する場合は ERROR 条件が発生します。

366 ページの図 51 では、データ・セットは DEFINE CLUSTER コマンドで定義され、PLIVSAM.AJC3.BASE という名前が与えられます。それが RRDS であるという事実は、NUMBERED キーワードによって判断されます。 PL/I プログラムでは、データ・セットは DIRECT OUTPUT ファイルによってロードされ、WRITE...FROM...KEYFROM ステートメントが使用されます。

データが順に並んでいてキーが順序どおりになっていれば、SEQUENTIAL ファイルを使用して、先頭からデータ・セットに書き込むことができたはずです。 この後レコードは、次に使用できるスロットに入れられ、該当する番号が付けられます。各レコード用のキーの番号が、KEYTO オプションを使って戻されたはずです。

PL/I ファイルは、DEFINE CLUSTER コマンドで指定された名前を DSNAME として使用する DD ステートメントによってデータ・セットへ関連付けられます。

```
//OPT9#17  JOB
//STEP1 EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEFINE CLUSTER -
             (NAME(PLIVSAM.AJC3.BASE) -
             VOLUMES(nnnnnn) -
             NUMBERED -
             TRACKS(2 2) -
             RECORDSIZE(20 20))
/*
//STEP2  EXEC IBMZCBG
//PLI.SYSIN      DD *
 CRR1:   PROC OPTIONS(MAIN);
         DCL NOS FILE RECORD OUTPUT DIRECT KEYED ENV(VSAM),
             CARD CHAR(80),
             NAME CHAR(20) DEF CARD,
             NUMBER CHAR(2) DEF CARD POS(21),
             IOFIELD CHAR(20),
             EOF BIT(1) INIT('0'B);
         ON ENDFILE (SYSIN) EOF='1'B;
         OPEN FILE(NOS);
         GET FILE(SYSIN) EDIT(CARD)(A(80));
        DO WHILE (¬EOF);
         PUT FILE(SYSPRINT) SKIP EDIT (CARD) (A);
         IOFIELD=NAME;
         WRITE FILE(NOS) FROM(IOFIELD) KEYFROM(NUMBER);
         GET FILE(SYSIN) EDIT(CARD)(A(80));
         END;
         CLOSE FILE(NOS);
 END CRR1;
/*
//GO.NOS DD DSN=PLIVSAM.AJC3.BASE,DISP=OLD
//GO.SYSIN DD *
ACTION,G.           12
BAKER,R.            13
BRAMLEY,O.H.        28
CHEESNAME,L.        11
CORY,G.             36
ELLIOTT,D.          85
FIGGINS.E.S.        43
HARVEY,C.D.W.       25
HASTINGS,G.M.       31
KENDALL,J.G.        24
LANCASTER,W.R.      64
MILES,R.            23
NEWMAN,M.W.         40
PITT,W.H.           55
ROLF,D.E.           14
SHEERS,C.D.         21
SURCLIFFE,M.        42
TAYLOR,G.C.         47
WILTON,L.W.         44
WINSTONE,E.M.       37
//
```

図 *51.* 相対レコード・データ・セット *(RRDS)* の定義とロード

SEQUENTIAL ファイルを使用した RRDS へのアクセス

RRDS へのアクセスに使用される SEQUENTIAL ファイルは、INPUT 属性または UPDATE 属性を使用してオープンできます。KEY、KEYTO、KEYFROM オプションのいずれかを使用する場合は、属性 KEYED も指定する必要があります。

KEY オプションが指定されていない READ ステートメントの場合、レコードは、昇順の相対レコード番号順に回復されます。 データ・セット内に空きスロットがあれば、すべてスキップされます。

KEY オプションを指定すると、READ ステートメントで回復されるレコードは、指定した相対レコード番号を持つレコードになります。 このような READ ステートメントはデータ・セットを指定されたレコードに位置付け、その後の順次読み取りは後続のレコードを順番に回復します。

KEYFROM オプションが指定されていてもいなくても、WRITE ステートメントは、KEYED SEQUENTIAL UPDATE ファイルに使用することができます。 前回行ったアクセスの位置に関係なく、データ・セット内の任意の位置に挿入を行うことができます。 KEYFROM オプションが指定された WRITE ステートメントの場合、そのデータ・セットに既に存在するレコードと同じ相対レコード番号を持つレコードを挿入しようとすると、KEY 条件が発生します。 KEYFROM オプションを省略すると、現在位置から見て次のスロットにレコードが書き込まれます。 このスロットが空いていなければ、ERROR 条件が発生します。

KEYTO オプションを使えば、KEYFROM オプションを指定していない WRITE ステートメントにより追加されるレコードのキーを回復することができます。

REWRITE ステートメントは、KEY オプションのあるなしに関係なく、UPDATE ファイルで使用できます。 KEY オプションを使用した場合、再書き込みされるレコードは指定された相対レコード番号を持つレコードであり、そうでない場合は直前の READ ステートメントによってアクセスされたレコードです。

KEY オプションの有無に関係なく DELETE ステートメントを使用すれば、データ・セットからレコードを削除できます。

DIRECT ファイルを使った RRDS へのアクセス

RRDS にアクセスするのに使用する DIRECT ファイルは、OUTPUT 属性、INPUT 属性、UPDATE 属性を持つことができます。 KEYED SEQUENTIAL ファイルを使用する場合と同様に、レコードの読み取り、書き込み、再書き込み、または削除が行えます。

368 ページの図 52 は、RRDS の更新の例を示しています。 DIRECT UPDATE ファイルが使用され、キーによって新しいレコードが書き込まれます。 VSAM では空のレコードは使用できないため、レコードが空かどうかを検査する必要はありません。

ラベル PRINT で始まるプログラムの後半では、更新済みファイルが印刷されます。 ここでも、REGIONAL(1) にあるので、空のレコードの検査は必要ありません。

PL/I ファイルは、図 52 の DEFINE CLUSTER コマンド内で定義された DSNAME の PLIVSAM.AJC3.BASE を指定する DD ステートメントで、データ・セットと関連づけられます。

例の終わりに、DELETE コマンドを使ってデータ・セットの削除が示されています。

```
//* NOTE: WITH A WRITE STATEMENT AFTER THE DELETE FILE STATEMENT,
//*       A DUPLICATE MESSAGE IS EXPECTED FOR CODE 'C' ITEMS
//*       WHOSE NEWNO CORRESPONDS TO AN EXISTING NUMBER IN THE LIST,
//*       FOR EXAMPLE, ELLIOT.
//*       WITH A REWRITE STATEMENT AFTER THE DELETE FILE STATEMENT,
//*       A NOT FOUND MESSAGE IS EXPECTED FOR CODE 'C' ITEMS
//*       WHOSE NEWNO DOES NOT CORRESPOND TO AN EXISTING NUMBER IN
//*       THE LIST, FOR EXAMPLE, NEWMAN AND BRAMLEY.
//OPT9#18  JOB
//STEP1  EXEC IBMZCBG
//PLI.SYSIN      DD *
 ACR1:  PROC OPTIONS(MAIN);
                DCL NOS FILE RECORD KEYED ENV(VSAM),NAME CHAR(20),
                  (NEWNO,OLDNO) CHAR(2),CODE CHAR(1),IOFIELD CHAR(20),
                  BYTE CHAR(1) DEF IOFIELD, EOF BIT(1) INIT('0'B),
                  ONCODE BUILTIN;
        ON ENDFILE(SYSIN) EOF='1'B;
        OPEN FILE(NOS) DIRECT UPDATE;
        ON KEY(NOS) BEGIN;
           IF ONCODE=51 THEN PUT FILE(SYSPRINT) SKIP EDIT
                    ('NOT FOUND:',NAME)(A(15),A);
           IF ONCODE=52 THEN PUT FILE(SYSPRINT) SKIP EDIT
                    ('DUPLICATE:',NAME)(A(15),A);
        END;
        GET FILE(SYSIN) EDIT(NAME,NEWNO,OLDNO,CODE)
           (COLUMN(1),A(20),A(2),A(2),A(1));
        DO WHILE (¬EOF);
        PUT FILE(SYSPRINT) SKIP EDIT (' ',NAME,'#',NEWNO,OLDNO,' ',CODE)
           (A(1),A(20),A(1),2(A(2)),X(5),2(A(1)));
        SELECT(CODE);
           WHEN('A') WRITE FILE(NOS) KEYFROM(NEWNO) FROM(NAME);
           WHEN('C') DO;
            DELETE FILE(NOS) KEY(OLDNO);
            WRITE FILE(NOS) KEYFROM(NEWNO) FROM(NAME);
           END;
           WHEN('D') DELETE FILE(NOS) KEY(OLDNO);
           OTHERWISE PUT FILE(SYSPRINT) SKIP EDIT
             ('INVALID CODE: ',NAME)(A(15),A);
        END;
```

図 *52. RRDS* の更新

```
        GET FILE(SYSIN) EDIT(NAME,NEWNO,OLDNO,CODE)
           (COLUMN(1),A(20),A(2),A(2),A(1));
        END;
         CLOSE FILE(NOS);
            PRINT:
         PUT FILE(SYSPRINT) PAGE;
         OPEN FILE(NOS) SEQUENTIAL INPUT;
         EOF='0'B;
         ON ENDFILE(NOS) EOF='1'B;
          READ FILE(NOS) INTO(IOFIELD) KEYTO(NEWNO);
          DO WHILE (¬EOF);
          PUT FILE(SYSPRINT) SKIP EDIT(NEWNO,IOFIELD)(A(5),A);
          READ FILE(NOS) INTO(IOFIELD) KEYTO(NEWNO);
          END;
       CLOSE FILE(NOS);
 END ACR1;

/*
//GO.NOS     DD DSN=PLIVSAM.AJC3.BASE,DISP=OLD
//GO.SYSIN   DD  *
NEWMAN,M.W.         5640C
GOODFELLOW,D.T.     89  A
MILES,R.              23D
HARVEY,C.D.W.       29  A
BARTLETT,S.G.       13  A
CORY,G.               36D
READ,K.M.           01  A
PITT,W.H.             55
ROLF,D.F.             14D
ELLIOTT,D.          4285C
HASTINGS,G.M.         31D
BRAMLEY,O.H.        4928C
//STEP3    EXEC PGM=IDCAMS,REGION=512K,COND=EVEN
//SYSPRINT DD SYSOUT=A
//SYSIN   DD   *
      DELETE -
          PLIVSAM.AJC3.BASE
//
```

RRDS の更新 *(*続き*)*

非 VSAM データ・セットに対して定義されたファイルの使用

共用データ・セットの使用

PL/I では、データ・セットの領域間またはシステム間での共用が許されます。このタイプの共用に対するサポートは VSAM によって提供されます。

上記のサポートについての詳細は、以下の DFSMS の資料を参照してください。

- *DFSMS:* データ・セットの使用法
- *DFSMS:* カタログのためのアクセス方式サービス・プログラム

第 3 部 プログラムの改良

© Copyright IBM Corp. 1999, 2017

第 15 章 パフォーマンスの向上

ユーザーのプログラムの速度を向上することに関する多数の考慮事項は、使用するコンパイラーとそれを実行するプラットフォームには関係ありません。 しかし、この章では、考慮事項の中でも PL/I コンパイラーとそれによって生成されるコードに特有な考慮事項を識別して説明します。

最適なパフォーマンスのためのコンパイラー・オプションの選択

選択するコンパイラー・オプションに応じて、コンパイラーによって生成されるコードのパフォーマンスを大幅に向上できることがあります。ただし、ほとんどのパフォーマンスの考慮事項と同様に、オプションの選択にはトレードオフがあります。

幸いなことに、コンパイラー・オプションはコマンド行または構成ファイルで指定できるため、ソース・コードを編集しなくても、コンパイル時オプションに伴うトレードオフについて比較検討することができます。

詳細を省きたい場合は、生成されるコードのパフォーマンスを向上させる最も簡単な方法として、次の (非デフォルトの) コンパイラー・オプションを指定する方法があります。

OPT(2) または OPT(3)
DFT(REORDER)

次のトピックでは、パフォーマンスの向上と、特定のコンパイラー・オプションに伴うトレードオフについて、より詳しく説明します。

OPTIMIZE

OPTIMIZE オプションを指定すると、プログラムの速度を上げることができます。このオプションを指定しないと、コンパイラーは基本的な最適化のみを実行します。

OPTIMIZE(2) を選択すると、コンパイラーは、パフォーマンスを改善するコードを生成するように指示されます。 通常、結果として生成されるコードは、プログラムが NOOPTIMIZE を使ってコンパイルされるときより短くなります。 しかし、場合によっては、長い命令シーケンスが、短い命令シーケンスよりも実行時間が短いこともあります。 例えば、WHEN 文節の値にギャップが指定されている SELECT ステートメント用にブランチ・テーブルが作成された場合などがこれに該当します。 生成される命令の数が増えた分は、通常、ほかの場所での命令の実行数が減ることによって相殺されます。

OPTIMIZE(3) を選択すると、コンパイラーは、パフォーマンスを改善するコードを生成するように指示されます。 しかし、OPTIMIZE(3) オプションを指定すると、OPTIMIZE(2) を指定したときよりもコンパイルに長い (ときには非常に長い) 時間がかかります。

© Copyright IBM Corp. 1999, 2017

GONUMBER

GONUMBER オプションを指定すれば、デバッグに使用されるステートメント番号表を生成できます。

この追加情報は、デバッグ時に非常に役立つことがありますが、ステートメント番号表を含めると、実行可能ファイルのサイズが大きくなります。実行可能ファイルが大きくなると、ロードに時間がかかります。

ARCH

ARCH オプションで最高値を使用すれば、z/OS において使用できる最も幅広い命令セットから命令を選択するようにコンパイラーに指示することができます。これにより、コンパイラーは最適なコードを生成できます。

REDUCE

REDUCE オプションは、コンパイラーが、構造体に対するヌル・ストリングの割り当てを縮小して、単純なコピー操作にすることを認められることを指定します。その操作が、埋め込みバイトの上書きを意味するにすぎない場合もあります。

REDUCE オプションが指定されると、ヌル・ストリングを構造体に割り当てるために生成されるコードの行の数が少なくなり、その結果として通常はコンパイルが高速になり、コードの実行速度が大きく向上します。しかし、埋め込みバイトはゼロにリセットされることがあります。

例えば次の構造体では、`field11` と `field12` の間に 1 バイトの埋め込みがあります。

```
dcl
 1 sample ext,
   5  field10          bin fixed(31),
   5  field11          dec fixed(13),
   5  field12          bin fixed(31),
   5  field13          bin fixed(31),
   5  field14          bit(32),
   5  field15          bin fixed(31),
   5  field16          bit(32),
   5  field17          bin fixed(31);
```

ここで、割り当て `sample = '';` について考えてみます。

NOREDUCE オプションでは、8 つの割り当てが生成されますが、埋め込みバイトは変更されません。

ただし、REDUCE では、割り当ては 3 つの演算に削減されます。

RULES

RULES サブオプションのほとんどは、特定のコーディング方法 (変数を宣言しないことなど) にフラグを立てる基準となる重大度だけに影響し、パフォーマンスには影響しません。 ただし、次のサブオプションはパフォーマンスに影響します。

IBM/ANS

RULES(IBM) オプションを使うと、コンパイラーはスケールされた FIXED BINARY をサポートします。パフォーマンスについてより重大なことは、いくつかの操作により、コンパイラーは、スケールされた FIXED BINARY の結果を生成します。

RULES(ANS) を指定すると、スケールされた FIXED BINARY はサポートされず、スケールされた FIXED BINARY の結果は生成されません。つまり、RULES(ANS) を指定して生成されたコードは、常に RULES(IBM) を指定して生成されたコードと少なくとも同じ速さで実行され、場合によっては実行速度が速くなります。

例えば、次のような部分コードがあるとします。

```
dcl (i,j,k) fixed bin(15);
   .
   .
   .
i = j / k;
```

RULES(IBM) を指定した場合は、割り算の結果に属性 FIXED BIN(31,16) が含まれます。つまり、割り算の前にはシフト命令が必要で、割り当てを実行するためにさらにいくつかの命令が必要になります。

RULES(ANS) のもとでは、割り算の結果は属性 FIXED BIN(15,0) を持ちます。 つまり、割り算の前にシフトは必要なく、割り当てを実行するために追加の命令は必要ありません。

(NO)LAXCTL

RULES(LAXCTL) のもとでは、CONTROLLED 変数を固定エクステントによって宣言しても、異なるエクステントに割り振ることができます。ただし、このコーディング方法はパフォーマンスに重大な影響を及ぼします。 RULES(NOLAXCTL) オプションを使用して、そのような方法を許可しないようにすることをお勧めします。

例えば、RULES(LAXCTL) を指定すると、次のような構造体を宣言できます。

```
dcl
  1 a controlled,
    2 b char(17),
    2 c char(29);
```

しかし、その後でこの構造体を次のように割り振ることができます。

```
allocate
  1 a,
    2 b char(170),
    2 c char(290);
```

コンパイラーは構造体 A に対する参照や、その構造体のメンバーに対する参照を見つけると必ず、構造体におけるフィールドの長さ、次元、またはオフセットに関して何の情報もないと強制的に想定します。このため、パフォーマンスが著しく低下する可能性があります。

RULES(NOLAXCTL) では、可変エクステントを使用して CONTROLLED 変数を割り振る場合は、そのエクステントを、アスタリスクまたは非定数式を使用して宣言する必要があります。したがって、RULES(NOLAXCTL) を指定し、固定エクステントを使用して CONTROLLED 変数を宣言すれば、コンパイラーはその変数を参照するためのコードをはるかに適切に生成できます。

PREFIX

PREFIX オプションは、選択された PL/I 条件がデフォルトで使用可能になっているかどうかを判別します。PREFIX のデフォルトのサブオプションは、PL/I 言語定義に適合するように設定されますが、デフォルトを指定変更すると、ユーザーのプログラムのパフォーマンスに大きい影響を与えることがあります。

デフォルトのサブオプションを以下に示します。

CONVERSION
INVALIDOP
FIXEDOVERFLOW
OVERFLOW
INVALIDOP
NOSIZE
NOSTRINGRANGE
NOSTRINGSIZE
NOSUBSCRIPTRANGE
UNDERFLOW
ZERODIVIDE

SIZE、STRINGRANGE、STRINGSIZE、または SUBSCRIPTRANGE サブオプションを指定すると、コンパイラーによってエクストラ・コードが生成されます。このコードは、他の方法では見つけるのが難しいソース内のいろいろな問題領域の位置を正確に示すのに役立ちます。 ただし、この追加のコードにより、プログラムのパフォーマンスが大幅に低下することがあります。

CONVERSION

CONVERSION 条件を使用不可にすると、いくつかの文字から数値への変換がインラインで、ソースの妥当性検査が行われずに実行されます。したがって、NOCONVERSION を指定した場合も、プログラムのパフォーマンスに影響があります。

FIXEDOVERFLOW

プラットフォームによっては、ハードウェアによって FIXEDOVERFLOW 条件が生じるため、コンパイラーはその検出に追加のコードを生成しなくてもよい場合があります。

DEFAULT

DEFAULT オプションを使うと、属性のデフォルトを選択できます。 PREFIX オプションの場合と同様に、DEFAULT のサブオプションは PL/I 言語定義に従うように設定されます。 デフォルトを変更すると、パフォーマンスに影響が及ぶ場合があります。

IBM/ANS および ASSIGNABLE/NONASSIGNABLE など、サブオプションのいくつかはプログラムのパフォーマンスに影響しません。ただし、ほかのすべてのサブオプションは、多かれ少なかれパフォーマンスに影響を与えることがあり、不適切に適用された場合は、プログラムが無効になることもあります。 以下のトピックでは、より重要ないくつかのサブオプションについて詳しく説明します。

BYADDR または BYVALUE

DEFAULT(BYADDR) オプションが有効な場合は、入り口宣言の属性でほかのものが指定されていない限り、参照によって (PL/I の必要性に応じて) 引数が渡されます。 参照によって引数が渡されると、変数そのものが渡されるときに、引数のアドレスが、あるルーチン (呼び出しルーチン) から別のルーチン (呼び出されたルーチン) に渡されます。 呼び出されたルーチン側に制御があるときに引数が変更されると、呼び出しルーチンの実行が再開されたときに呼び出しルーチンにそれが反映されます。

多くの場合、プログラム・ロジックは、参照によって渡される変数に応じて異なります。しかし、参照によって変数を渡すと、次の 2 つのようなパフォーマンスの低下が起きる場合があります。

1. そのパラメーターに対するすべての参照に、追加の命令が必要になる。
2. 変数のアドレスが別のルーチンに渡されると、コンパイラーは、その変数が変更されるときを想定して、その変数の参照用のコードとして非常に保守的なコードを生成することを強制される。

したがって、プログラム・ロジック上で認められるときはいつでも、BYVALUE サブオプションを使い、値によってパラメーターを渡す必要があります。 BYADDR 属性を使って、1 つのパラメーターが参照によって渡されるように指示するときでも、DEFAULT(BYVALUE) オプションを使って、ほかのすべてのパラメーターが値によって渡されるように設定できます。

BYVALUE 引数はレジスターで正当に渡すことができる引数でなければなりません。 したがって、そのタイプは以下のいずれかでなければなりません。

- REAL FIXED BIN
- REAL FLOAT
- POINTER
- OFFSET
- HANDLE
- LIMITED ENTRY
- FILE
- ORDINAL
- CHAR(1)
- WCHAR(1)
- ALIGNED BIT(*n*) (*n* は 8 以下)

さらに、BYVALUE パラメーターをアセンブラー・コードと一緒に使用するときは、パラメーター・リストで 4 バイトが使用されるように、4 の倍数バイトを必要としない BYVALUE パラメーターが拡張されることに注意する必要があります。

プロシージャーが BYADDR によって渡される 1 つのパラメーターだけを受け取り、それを変更する場合は、値によってそのパラメーターを受け取る関数にプロシージャーを変換することを考慮してください。 その関数は、パラメーターの更新値が含まれた RETURN ステートメントで終了します。

BYADDR パラメーターが指定されたプロシージャー

```
a: proc( parm1, parm2, ..., parmN );

  dcl parm1 byaddr  ...;
  dcl parm2 byvalue ...;
   .
   .
   .
  dcl parmN byvalue ...;

  /* program logic */

end;
```

BYVALUE パラメーターが指定された、より高速の同等な関数

```
a: proc( parm1, parm2, ..., parmN )
   returns( ... /* attributes of parm1 */ );

  dcl parm1 byvalue  ...;
  dcl parm2 byvalue ...;
   .
   .
   .
  dcl parmN byvalue ...;

  /* program logic */

  return( parm1 );

end;
```

(NON)CONNECTED

DEFAULT(NONCONNECTED) オプションは、すべての集合体パラメーターが NONCONNECTED であるとコンパイラーが想定するように指示します。 NONCONNECTED 集合体パラメーターのエレメントに対する参照では、コンパイラーがパラメーターの記述子にアクセスするためのコードを生成する必要があります。これは、集合体が固定エクステントを使って宣言されている場合にも同様です。

集合体パラメーターに固定エクステントが指定されており、CONNECTED が指定されている場合、コンパイラーはこれらの命令を生成しません。 したがって、アプリケーションが NONCONNECTED パラメーターを渡さない場合は、DEFAULT(CONNECTED) オプションを使うとコードがより最適化されます。

(NO)DESCRIPTOR

DEFAULT(DESCRIPTOR) オプションは、デフォルトでは、ストリング、領域、または集合体パラメーター用に記述子が渡されることを示します。しかし、記述子は、パラメーターに非固定エクステントが指定されている場合、またはパラメーターが NONCONNECTED 属性を持つ配列である場合にだけ、使われます。

この場合は、記述子を渡すために必要となる命令とスペースは効果がなく、かなりのコストがかかります (多くの場合、構造化記述子のサイズは構造体そのもののサイズよりも大きくなります)。したがって、PROCEDURE ステートメントと ENTRY 宣言で必要なときにだけ DEFAULT(NODESCRIPTOR) を指定し、OPTIONS(DESCRIPTOR) を使うと、コードの実行がより最適化されます。

(NO)INLINE

NOINLINE サブオプションは、プロシージャーと開始ブロックがインライン化されないように指示します。インライン化は、最適化を指定するときにだけ発生します。

ユーザー・コードをインライン化すると、関数呼び出しとリンケージのオーバーヘッドが除去され、関数のコードが最適化プログラムに公開されて、結果としてコード・パフォーマンスが向上します。 インライン化は、関数のオーバーヘッドが無視できるようなものではないとき、例えば、関数がネストされたループ内で呼び出されるときなどに、最高の結果をもたらします。 また、インライン化は、インライン化された関数によってさらに最適化の機会が与えられるとき、例えば、定数引数が使われるときなどにも、有益です。

ネストされていない多くのプロシージャーが含まれているプログラムに関しては、以下の情報を考慮してください。

- プロシージャーの規模が小さく、ほんのいくつかの場所からしか呼び出されない場合は、INLINE を指定する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。
- プロシージャーの規模が大きく、複数の場所から呼び出される場合は、インライン化によって、プログラム全体にわたってコードの重複が起こります。 このようなプログラム・サイズの増大は、速度の増大を相殺します。 この場合は、NOINLINE をデフォルトのままにして、個別に選択したプロシージャーだけで OPTIONS(INLINE) を指定することをお勧めします。

インライン化を使う場合は、スタック・スペースを拡大する必要があります。 関数が呼び出されると、そのローカル・ストレージが呼び出し時に割り振られ、呼び出し関数に戻るときに解放されます。 その関数がインライン化されると、そのストレージは、それを呼び出す関数に入ったときに割り振られ、呼び出し関数が終了するまで解放されません。インライン化した関数のローカル・ストレージ用に、十分なスタック・スペースがあることを確認してください。

LINKAGE

この LINKAGE サブオプションは、OPTIONS 属性の LINKAGE サブオプション、または入力のオプションが指定されなかったときにコンパイラーが使用しなければならないデフォルト・リンケージを指定します。コンパイラーは、それぞれが固有のパフォーマンス特性を持つ各種のリンケージをサポートします。

外部エンティティー (例えば、オペレーティング・システムなど) によって提供される ENTRY を呼び出すときには、その ENTRY に対してあらかじめ定義されているリンケージを使用する必要があります。

ただし、独自のアプリケーションを作成するときには、リンケージ規則を選択できます。 OPTLINK リンケージは、ほかのリンケージ規則よりも大幅にパフォーマンスを向上させるので、これを選択することをお勧めします。

(RE)ORDER

DEFAULT(ORDER) オプションは、ORDER オプションがすべてのブロックに適用されることを示します。つまり、ON ユニットで参照されるそのブロック (または ON ユニットから動的に派生するブロック) 内の変数に最新の値が適用されることを示します。これにより、そのような変数でのほとんどすべての最適化が実際上禁止されます。

したがって、プログラム・ロジックによって認められる場合は、DEFAULT(REORDER) を使って、上位コードを生成してください。

NOOVERLAP

DEFAULT(NOOVERALP) オプションを指定すると、コンパイラーは割り当て時にソースとターゲットがオーバーラップしていないと想定するため、より小さく速いコードを生成することができます。

ただし、このオプションを使用する場合は、割り当て時にソースとターゲットがオーバーラップしないようにする必要があります。例えば、DEFAULT(NOOVERLAP) オプションでは、次の例にある割り当ては無効です。

```
dcl c char(20);
substr(c,2,5) = substr(c,1,5);
```

RETURNS(BYVALUE) または RETURNS(BYADDR)

DEFAULT(RETURNS(BYVALUE)) オプションが有効な場合は、BYADDR が指定されないすべての RETURNS 記述リストに BYVALUE 属性が適用されます。つまり、これらの関数は、最適なコードを生成するために、可能なときにはレジスターに値を戻します。

パフォーマンスを向上させるコンパイラー・オプションの要約

要約すると、次に挙げるオプション (ユーザーのアプリケーションに適用できる場合) はパフォーマンスを向上させることができます。

- OPTIMIZE(3)
- ARCH(12)
- REDUCE
- RULES(ANS NOLAXCTL)
- 次のサブオプションが指定された DEFAULT
  - BYVALUE
  - CONNECTED
  - NODESCRIPTOR
  - INLINE
  - LINKAGE(OPTLINK)
  - REORDER
  - NOOVERLAP
  - RETURNS(BYVALUE)

パフォーマンス向上のためのコーディング

コードを作成するときには、指定されたタスクを実行するために適する方法が複数あるのが普通です。 多くの重要な要素、例えば、読み易さや保守容易性などによって、コーディングするスタイルの選択は変わってきます。 次のトピックでは、コーディングを行うときにプログラムのパフォーマンスに影響を及ぼす可能性がある選択肢について説明します。

DATA ディレクティブ入出力

デバッグに GET DATA ステートメントと PUT DATA ステートメントを使うと、非常に有効であることがあります。 ただし、これらのステートメントを使うと、一般的にはパフォーマンスが低下という犠牲が伴います。 このパフォーマンスの低下は、変数リストを使わずに GET DATA または PUT DATA を使うと、非常に大きくなります。

多くのプログラマーは、次の例に示すように、ON ERROR コード内で PUT DATA ステートメントを使います。

```
on error
  begin;
    on error system;
   .
   .
   .
    put data;
   .
   .
   .
  end;
```

この場合、PUT DATA ステートメントで、選択された変数のリストを組み込むことにより、プログラムがより最適化されます。

上記の例の ON ERROR ブロックには、PUT DATA ステートメントの前に ON ERROR システム・ステートメントが含まれています。 これにより、PUT DATA ステートメントでエラーが起きても (このエラーは、リストされる変数に無効な FIXED DECIMAL 値が含まれている場合に起きる可能性がある)、ON ERROR ブロックのほかの場所でエラーが起きても、プログラムが無限ループに入ることが回避されます。

入力専用パラメーター

プロシージャーに、入力専用に使われる BYADDR パラメーターが含まれている場合は、そのパラメーターを NONASSIGNABLE として宣言するのが (ASSIGNABLE のデフォルト属性を取得させるのではなく) 最良の方法です。プロシージャーがあとからそのパラメーターの定数を使って呼び出された場合、コンパイラーは静的ストレージに定数を入れ、その静的領域のアドレスを渡します。

この方法は、レジスターに渡すことができないストリングやその他のパラメーターに特に役立ちます (レジスターに渡すことができる入力専用パラメーターは、BYVALUE として宣言するのが最良です)。

例えば次の宣言では、getenv に対する最初のパラメーターが、入力専用の CHAR VARYINGZ ストリングです。

```
dcl getenv     entry( char(*) varyingz nonasgn byaddr,
                      pointer byaddr )
               returns( native fixed bin(31) optional )
               options( nodescriptor );
```

ストリング IBM_OPTIONS が指定されてこの関数が呼び出されると、コンパイラーは、コンパイラー生成一時記憶域にそのストリングを割り当て、その領域のアドレスを渡すのではなく、ストリングのアドレスを渡すことができます。

GOTO ステートメント

別のブロック内のラベルまたはラベル変数を使う GOTO ステートメントは、コンパイラーが実行する最適化を厳しく制限します。

ラベル配列が初期化され、暗黙的または明示的に AUTOMATIC と宣言された場合、その配列のエレメントへの GOTO は最適化の妨げとなります。 しかし、配列が STATIC と宣言された場合は、コンパイラーがその配列に対して CONSTANT 属性を想定し、最適化は妨げられません。

ストリングの割り当て

あるストリングが別のストリングに割り当てられているとき、コンパイラーは、ソースとターゲットがオーバーラップする場合でもターゲットの値を正しい値とし、ソース・ストリングがターゲットよりも長い場合はソース・ストリングを切り捨てます。これには、いくつかの追加命令が必要になります。

コンパイラーは、これらの追加の命令を必要なときにだけ生成しようとしますが、コンパイラーが必要でないということに確信を持てない場合、ユーザーは、プログラマーとして、その追加の命令が必要ないということを知っていることが多いのです。 例えば、ソースとターゲットが基底付き文字ストリングであって、ユーザーはそれらがオーバーラップすることがあり得ないことを知っている場合、コンパイラーであれば生成するように強制されるところを、PLIMOVE 組み込み関数を使ってその追加のコードを除去することができます。

次の例では、2 番目の割り当てステートメント用に、より速いコードが生成されます。

```
dcl based_Str   char(64) based;
dcl target_Addr pointer;
dcl source_Addr pointer;

target_Addr->based_Str = source_Addr->based_Str;

call plimove( target_Addr, source_Addr, stg(based_Str) );
```

ユーザーがソースとターゲットがオーバーラップするのではないかと疑いがある場合、またはターゲットがソースを収容できる大きさであるかどうかが疑われる場合は、ユーザーは PLIMOVE 組み込み関数を使ってはなりません。

ループ制御変数

ループ制御変数を適切に定義すれば、プログラムのパフォーマンスが向上します。

プログラムのパフォーマンスを向上させるには、ループ制御変数に対して以下のいずれかのタイプを使用します。ユーザーは、まれに必要になる場合を除き、これ以外のタイプの変数を使うべきではありません。

ゼロのスケール因数を持つ FIXED BINARY
FLOAT
ORDINAL
HANDLE
POINTER
OFFSET

また、ループ制御変数が配列、構造体、または共用体のメンバーでない場合にも、パフォーマンスは向上します。 ループ制御変数がこのようなメンバーである場合、コンパイラーは警告メッセージを出します。 AUTOMATIC であり、ほかの目的で使われないループ制御変数は、コードの生成を最適化します。

ループ制御変数が FIXED BINARY である場合は、精度が 31 で SIGNED である場合に、パフォーマンスが最高に達します。

プログラムがループ制御変数の値だけでなく、そのアドレスにも依存している場合は、パフォーマンスが低下します。 例えば、ADDR 組み込み関数が変数に適用される場合、または変数が参照 (BYADDR) によって別のルーチンに渡される場合は、パフォーマンスが低下します。

PACKAGE 対ネストされた PROCEDURE

呼び出し側のネストされたプロシージャーは、追加の隠しパラメーター (逆チェーン・ポインター) が渡されるように求めます。 結果として、アプリケーションに含まれているネストされたプロシージャーが少なくなればなるほど、実行速度は速くなります。

アプリケーションのパフォーマンスを向上させるには、ネストされたプロシージャーの母娘の対を、パッケージ内部のレベル 1 の姉妹プロシージャーに変換します。この変換は、ネストされたプロシージャーが、その親プロシージャーで宣言された自動かつ内部の静的変数に依存していない場合に可能です。

『ネストされたプロシージャーの例』にあるプロシージャー b に、a で宣言された変数が使用されていない場合は、両方のプロシージャーを再編成して、 384 ページの『パッケージ化されたプロシージャーの例』に示されているパッケージに組み込むことで、両方のプロシージャーのパフォーマンスを向上させることができます。

ネストされたプロシージャーの例

```
a: proc;

  dcl (i,j,k) fixed bin;
  dcl ib      based fixed bin;
   .
   .
   .
  call b( addr(i) );
   .
   .
   .
  b: proc( px );
```

```
    dcl px      pointer;
    display( px->ib );
  end;
end;
```

パッケージ化されたプロシージャーの例

```
p: package exports( a );

  dcl ib      based fixed bin;

  a: proc;

    dcl (i,j,k) fixed bin;
    .
    .
    .
    call b( addr(i) );
    .
    .
    .
  end;

  b: proc( px );
    dcl px      pointer;
    display( px->ib );
  end;

end p;
```

REDUCIBLE 関数

REDUCIBLE は、引数 (1 つ以上) が変更されない限り、プロシージャーまたは入り口を複数回呼び出す必要がないこと、およびプロシージャーの呼び出しに副次作用がないことを示します。

例えば、変更されないデータに基づいて結果を計算するユーザー作成の関数には、REDUCIBLE が宣言されなければなりません。 乱数や時刻などの、変更されるデータに基づいて結果を計算する関数は、IRREDUCIBLE として宣言する必要があります。

次の例では、REDUCIBLE が宣言の一部になっているため、f が一度だけ呼び出されます。宣言に IRREDUCIBLE が使われていると、f が 2 度呼び出されます。

```
dcl (f) entry options( reducible ) returns( fixed bin );

select;
  when( f(x) < 0 )
    .
    .
    .
  when( f(x) > 0 )
    .
    .
    .
  otherwise
    .
    .
    .
end;
```

REPATTERN の使用

REPATTERN 組み込み関数を使用すれば、データの日時パターンを変換するコードを作成できます。

例えば、現在日時を「YYYY-MM-DD HH:MI:SS.999999」のパターンで最も素早く取得する方法は TIMESTAMP 組み込み関数と REPATTERN 組み込み関数を使用することです。

```
repattern( timestamp(),
                'YYYY-MM-DD HH:MI:SS.999999',
                'YYYY-MM-DD-HH.MI.SS.999999' )
```

DESCLOCATOR または DESCLIST

DEFAULT(DESCLOCATOR) オプションが有効になっている場合、コンパイラーは従来のコンパイラーとほぼ同じように、記述子 (ストリングや構造体など) を必要とする引数を渡すために記述子ロケーターを使用します。

このオプションを使用すれば、エントリー・ポイントが宣言している引数をすべて渡さなくても、エントリー・ポイントを呼び出すことができます。

またこのオプションを使用すると、構造体を渡してからそれをポインターとして受け取るといった、あまり賢明ではないプログラミング方法を従来どおり使うことができます。

ただし、DEFAULT(DESCLOCATOR) を指定した場合にコンパイラーによって生成されるコードは、状況によっては DEFAULT(DESCLIST) の場合のコードよりもパフォーマンスが悪くなることがあります。

関連情報:

557 ページの『第 25 章 PL/I 記述子』
この章では、実行時の PL/I ルーチン間での PL/I パラメーターの引き渡しの規則について説明します。

DEFINED 対 UNION

UNION 属性は、DEFINED 属性よりも強力で、より多くの機能を提供します。さらにコンパイラーは、共用体参照の場合、より優れたコードを生成します。

次の例では、変数の対 b3 と b4 が、b1 および b2 と同じ機能を実行しますが、コンパイラーは共用体の対の場合に、より最適化されたコードを生成します。

```
dcl b1 bit(32);
dcl b2 bit(16) def b1;

dcl
  1 * union,
    2 b3 bit(32),
    2 b4 bit(16);
```

DEFINED 属性ではなく UNION を使うコードは、誤って解釈されることが少なくなります。 共用体の中の変数宣言は 1 個所にあるので、共用体のメンバーが変更されたり、ほかのすべてのメンバーが変更されても、それを理解しやすくなっています。 この動的変更は、DEFINED 変数を使う宣言では宣言ステートメントが複数行離れていることがあるので、認識が難しくなります。

名前付き定数対静的変数

名前付き定数は、VALUE 属性を使って変数を宣言することによって定義できます。 INITIAL 属性を指定して静的変数を使い、変数を変更しない場合は、VALUE 属性を使って変数を名前付き定数として宣言する必要があります。コンパイラーは NONASSIGNABLE のスカラー STATIC 変数を、真の名前付き定数として扱いません。

コンパイラーは、コンパイル時に式が評価される時にはいつでも、より最適化されたコードを生成するので、名前付き定数を使って、読み易さを低下させずに効果的なコードを生成できます。 例えば次の例では、VERIFY 組み込み関数の 2 通りの使用法を用いて、同一のオブジェクト・コードが生成されています。

```
dcl numeric char  value('0123456789');

jx = verify( string, numeric );

jx = verify( string, '0123456789' );
```

次の例は、VALUE 属性を使って、読み易さを低下させずに最適なコードを取得できる方法を示しています。

意味のある名前が付けられていない最適なコードの例

```
dcl  x  bit(8) aligned;

select( x );
  when( '01'b4 )
    .
    .
    .
  when( '02'b4 )
    .
    .
    .
  when( '03'b4 )
    .
    .
    .
end;
```

意味のある名前が付けられた最適でないコードの例

```
dcl (  a1  init( '01'b4)
      ,a2  init( '02'b4)
      ,a3  init( '03'b4)
      ,a4  init( '04'b4)
      ,a5  init( '05'b4)
    ) bit(8) aligned static nonassignable;

dcl  x  bit(8) aligned;

select( x );
  when( a1 )
    .
    .
    .
  when( a2 )
    .
    .
    .
  when( a3 )
```

```
   .
   .
   .
end;
```

意味のある名前が付けられた最適なコードの例

```
dcl (  a1  value( '01'b4)
      ,a2  value( '02'b4)
      ,a3  value( '03'b4)
      ,a4  value( '04'b4)
      ,a5  value( '05'b4)
    ) bit(8);

dcl  x  bit(8) aligned;

select( x );
  when( a1 )
    .
    .
    .
  when( a2 )
    .
    .
    .
  when( a3 )
    .
    .
    .
end;
```

ライブラリー・ルーチンの呼び出しの回避

ビット単位操作 (接頭部 NOT、2 項演算子 AND、2 項演算子 OR、および 2 項演算子 EXCLUSIVE OR) は、多くの場合、ライブラリー・ルーチンを呼び出すと評価されます。

ただし、これらの操作は、次のいずれかの条件が真である場合に、ライブラリーが呼び出されずに処理されます。

- 両方のオペランドが bit(1) である
- 両方のオペランドの位置が合い、同じ固定長である

特定の割り当て、式、および組み込み関数参照の場合は、コンパイラーがライブラリー・ルーチンの呼び出しを生成します。 これらの呼び出しを回避すると、一般的にコードの実行速度が速くなります。

コンパイラーがこのような呼び出しを生成する時点を判断の助けとして、ライブラリー・ルーチンを使って変換が行われるときにはいつでも、コンパイラーがメッセージを生成します。

コードが、REFER を使用して BASED 構造体の 1 つのメンバーを参照している場合、実行時にその構造体にマップするために、コンパイラーがライブラリー・ルーチンに対する 1 つ以上の呼び出しを生成することがあります。これらの呼び出しは効率を低下させる可能性があるため、コンパイラーがこれらの呼び出しを行う場合には、メッセージを出して、コード内の問題となる可能性のある場所を見つけることができるようにしています。

REFER が使用された BASED 構造体を使用するコードがあり、コンパイラーがこのメッセージを出力してフラグを立てた場合は、対応する構造体 (* エクステント付き) を宣言するサブルーチンに構造体を渡すことでパフォーマンスが向上することがあります。これにより、構造体はいったん CALL ステートメントでマップされますが、呼び出されたサブルーチンでアクセスされるときにさらに再マップされることはなくなります。

ライブラリー・ルーチンのプリロード

PL/I ライブラリーには、低レベルのシステム入出力機能に使用される RMODE 24 ルーチンが 1 つ (IBMPOIOA) 含まれています。コードでレコード入出力を行う場合、または (STDSYS オプションを指定してコンパイルせずに) SYSPRINT を STREAM OUTPUT ファイルとして使用する場合には、このルーチンをプリロードするか (E)LPA に配置することによって、パフォーマンスが大幅に改善されます。

第 4 部 他の製品に対するインターフェースの使用

© Copyright IBM Corp. 1999, 2017

第 16 章 ソート・プログラムの使用

コンパイラーは、PLISRT*x* (*x* = A、B、C、または D) という名前のインターフェースを提供しますが、このインターフェースを用いれば、IBM 提供のソート・プログラムを使用できます。

ソート・プログラムを PLISRT*x* と併用するには、次の操作を行う必要があります。

1. PLISRT*x* のいずれかのエントリー・ポイントに対する呼び出しを組み込み、ソートされるフィールドの情報をそのエントリー・ポイントに渡します。 この情報には、レコード長、使用ストレージの最大値、戻りコードとして使用する変数の名前、およびソートを実行するのに必要なその他の情報が含まれます。
2. JCL DD ステートメント内で、ソート・プログラムに必要なデータ・セットを指定します。

ソート・プログラムは、PL/I から使用されると、大量のソート・フィールド上の正常な長さのレコードすべてをソートします。 ほとんどのタイプのデータは、昇順または降順でソートできます。 ソートされるデータのソースは、データ・セットである場合と、ソートにレコードが必要になるたびにソート・プログラムによって呼び出されるユーザー作成の PL/I プロシージャーである場合とがあります。 同様に、ソートの宛先も、データ・セットでも、ソートされたレコードを処理する PL/I プロシージャーでもかまいません。

PL/I プロシージャーを使えば、ソート自体の前でも後でも処理を行えるので、1 回のソート・インターフェースの呼び出しで、ソート操作をすべて完了することができます。 入出力を処理する PL/I プロシージャーは、ソート・プログラム自身から呼び出されるため、事実上ソート・プログラムの一部となることを理解しておくことが大切です。

PL/I は、DFSORT と一緒に、または同じインターフェースを持ったプログラムと一緒に稼働することができます。DFSORT は、プログラム・プロダクト 5740-SM1 の 1 リリースです。 DFSORT はプログラム論理を書く必要をなくすために使用できる多数の組み込み機能 (例えば、INCLUDE、OMIT、OUTREC、および SUM ステートメント、プラス多数の ICETOOL オペレーター) を持っています。詳細については「*DFSORT Application Programming Guide*」を、またチュートリアルとしては「*Getting Started with DFSORT*」を参照してください。

以下の情報は DFSORT に当てはまります。DFSORT 以外のプログラムを使用することもできるため、実際の機能や制約事項はさまざまです。このような機能や制約事項に関しては、「*DFSORT Application Programming Guide*」または、ご使用になるソート製品用のこれに相当する資料を参照してください。

ソート・プログラムを使うには、ソース・プログラム内に正しい PL/I ステートメントを入れなければならず、また JCL 内で正しいデータ・セットを指定しなければなりません。

© Copyright IBM Corp. 1999, 2017

ソート・プログラムの使用準備

ソート・プログラムを使用するには、まず必要とするソートのタイプ、データ内のソート・フィールドの長さとフォーマット、データ・レコード長、ソートに使う補助記憶域と主記憶域の大きさを決めます。

使用する PLISRT*x* エントリー・ポイントを決定するには、未ソート・データのソースと、ソート済みデータの宛先を決める必要があります。データ・セットと PL/I サブルーチンのどちらかを選択します。データ・セットを使用する方がより分かりやすく、パフォーマンスも速くなります。 PL/I サブルーチンを使用すると、より高い柔軟性と機能性が得られるため、データをソートする前に処理したり印刷することができ、ソート済みの形のまま直ちにデータを使用することができます。 入力または出力処理のサブルーチンを使用する場合は、 403 ページの『データ入出力処理ルーチン』を参照してください。

エントリー・ポイント、データのソース、宛先を次に示します。

エントリー・ポイント	ソース	宛先
PLISRTA	データ・セット	データ・セット
PLISRTB	サブルーチン	データ・セット
PLISRTC	データ・セット	サブルーチン
PLISRTD	サブルーチン	サブルーチン

使用するエントリー・ポイントを決定したら、今度はデータ・セットに関して以下の事項を決定する必要があります。

- ソート・フィールドの位置。これらは 1 つのレコード全体あるいはその任意の一部 (複数の部分も可能) となります。
- これらのフィールドが表すデータのタイプ (例えば文字または 2 進数など)。
- 各フィールドでのソートを昇順にするか、降順にするか。
- 同じレコードは、入力されたとおりの順序で保持するか、またはソート時に順序を変更することができるのか。

PLISRT*x* への最初の引数である SORT ステートメント上でこれらのオプションを指定します。 次に、ソートされるレコードに関して以下の事項を決定する必要があります。

- レコード・フォーマットが固定フォーマットであるか、可変フォーマットであるか。
- レコード長。これは可変フォーマットの最大長です。

これらを PLISRT*x* への 2 番目の引数である RECORD ステートメント上で指定します。

最後に、ソート・プログラム用に確保しようとする主記憶域と補助記憶域の大きさを決めなければなりません。 詳細については、 398 ページの『ソート・プログラムに必要なストレージの決定』を参照してください。

ソート・タイプの選択

PL/I プログラム内で、ソート・インターフェース・サブルーチン PLISRT*x* に対する CALL ステートメントを使用することによって、ソートを指定します。 このサ

ブルーチンは *x*=A、B、C、および D という 4 つのエントリー・ポイントを持っています。それぞれ、未ソート・データに対して別々のソースを指定し、そのデータがソートされたときのそのデータの宛先を指定します。

例えば、PLISRTA の呼び出しは、未ソート・データ (ソートへの入力) が、ある 1 つのデータ・セット上にあることを指定し、ソート済みデータ (ソートからの出力) を別のデータ・セットに置くことを指定します。 CALL PLISRT*x* ステートメントに含めなければならないものは、ソートしようとするデータ・セットに関するソート・プログラム情報を示した引数リスト、ソートを行うフィールド、使用可能なスペースの大きさ、ソートが成功したかまたは失敗したかを示すための戻りコードをソート・プログラムが入れる変数の名前、および使用可能な出力または入力の処理プロシージャーがあればその名前です。

ソート・インターフェース・ルーチンは、ソート・プログラムの引数リストを、PLISRT*x* 引数リストが提供する情報と、選択した PLISRT*x* エントリー・ポイントから作成します。 次に、制御はソート・プログラムに移されます。 出力または入力の処理ルーチンを指定していれば、それぞれの未ソートまたはソート済みレコードを処理するのに必要な回数だけ、処理ルーチンがソート・プログラムによって呼び出されます。 ソート操作が完了するとソート・プログラムは、戻りコードでソートが成功したか失敗したかを知らせて、PL/I 呼び出しプロシージャーに戻ります。なお、その戻りコードは、インターフェース・ルーチンに渡される引数のうちの 1 つに入れられています。 次に戻りコードは、処理を継続すべきかどうかを判別するために、PL/I ルーチン中でテストすることができます。 394 ページの図 53 は、この操作を示す単純化されたフローチャートです。

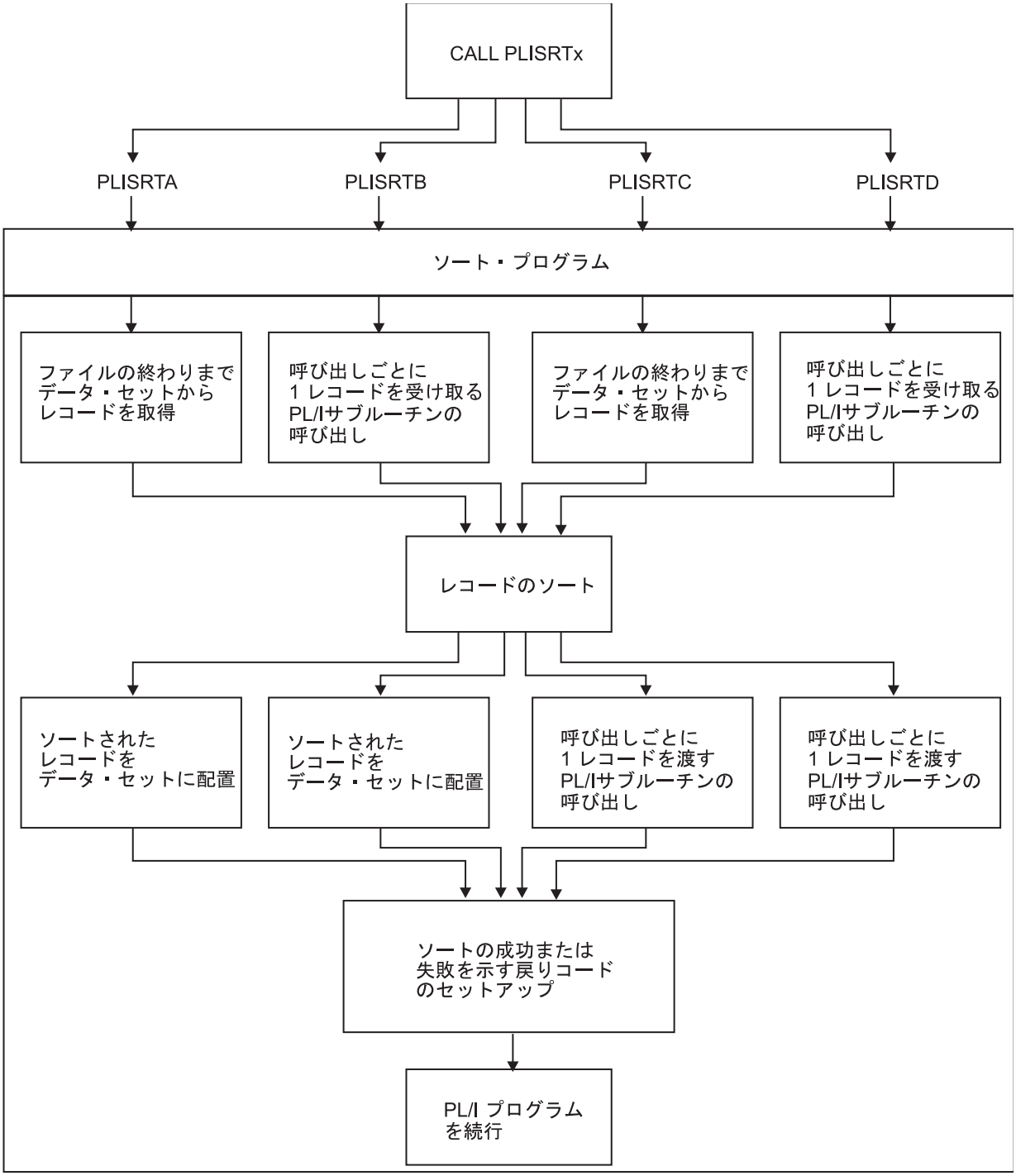


図 *53.* ソート・プログラムの制御の流れ

ソート・プログラムそのものにおいては、ソート・プログラムと入力および出力処理ルーチンとの間の制御の流れは、戻りコードで制御されます。ソート・プログラムは、その処理の途中で、適切な時点でこれらのルーチンを呼び出します。 (ソート・プログラムとそれに関連した資料では、これらのルーチンはユーザー出口 と呼ばれます。 ソートされる入力を渡すルーチンは、E15 ソート・ユーザー出口です。ソート済み出力を処理するルーチンは、E35 ソート・ユーザー出口です。) これら

のルーチンから、ソート・プログラムは、そのルーチンをもう一度呼び出すべきか、または次の処理段階に進むべきかを示す戻りコードがくることを予期します。

ソート・プログラムに関して以下の重要なポイントを覚えておく必要があります。

- ソート・プログラムは、完全なソート命令を処理する自己完結型プログラムです。
- ソート・プログラムは戻りコードを使用して、呼び出し側と通信したり、自分で呼び出したユーザー出口と通信したりします。

ソート・フィールドの指定

このトピックでは、PL/I からソート・プログラムを使用するために指定しなければならない必須の PL/I ステートメントについて説明します。 SORT ステートメントは、PLISRT*x* への最初の引数です。

SORT ステートメントの構文は、次の形式の文字ストリング式でなければなりません。

```
'bSORTbFIELDS=(start1,length1,form1,seq1,
...startn,lengthn,formn,seqn)[,other options]b'
```

次の例を参照してください。

```
' SORT FIELDS=(1,10,CH,A) '
```

b 1 つ以上のブランクを表します。ここに示されているブランクは必須です。 これ以外のブランクは認められません。

start,length,form,seq

ソート・フィールドを定義します。指定するソート・フィールドの数は任意ですが、フィールドの全長には限度があります。 複数のフィールドがソートされる場合は、レコードはまず最初のフィールドに従ってソートされ、次に、等しい値を持つレコードが 2 番目のフィールドに従ってソートされ、というようになります。 ソート値がすべて等しければ、EQUALS オプションを使わない限り、等しいレコードの順序は任意になります。 (この定義リストの後半を参照してください。) フィールドは互いにオーバーレイしてもかまいません。

DFSORT (5740-SM1) の場合、ソート・フィールドの最大全長は 4092 バイトに制限され、全ソート・フィールドはレコードの始まりから 4092 バイト以内でなくてはなりません。 他のソート製品は異なる制限を持つ場合があります。

start レコード内の開始位置を指定します。値はバイト単位で指定します (2 進数データの場合は、「バイト.ビット」表記を使うことができます)。ストリング内の最初のバイトはバイト 1 と見なされ、最初のビットはビット 0 と見なされます。(したがって、バイト 2 の 2 番目のビットは 2.1 と呼ばれます。) 可変長レコードの場合、4 バイト長の接頭部を入れて、5 をデータの最初のバイトにします。

length

ソート・フィールドの長さを指定します。値はバイト単位で指定します (ただし、2 進数データの場合は、「バイト.ビット」表記を使うことができます)。ソート・フィールドの長さには、それぞれのデータ・タイプ別に制約があります。

form データのフォーマットを指定します。これは、ソートの目的で用いられ

るフォーマットです。 PL/I ルーチンとソート・プログラム間でやり取りされるデータはすべて、文字ストリングの形式でなければなりません。 主なデータ・タイプとその長さに対する制約事項を次に示します。 これ以外のデータ・タイプも特殊目的のソート用に用意されています。 これに関しては、「*DFSORT Application Programming Guide*」または、ご使用のソート製品用の資料を参照してください。

コード	データ・タイプと長さ
CH	文字 1 から 4096 まで
ZD	ゾーン 10 進数符号付き 1 から 32 まで
PD	パック 10 進符号付き 1 から 32 まで
FI	固定小数点、符号付き 1 から 256 まで
BI	2 進数、符号なし 1 ビットから 4092 バイトまで
FL	浮動小数点、符号付き 1 から 256 まで

全フィールドの合計長は、4092 バイトを超えてはなりません。

seq データがソートされる順序を指定します。

A 昇順 (つまり、1,2,3,...)

D 降順 (つまり、...,3,2,1)

注**:** E を指定することはできません。その理由は、PL/I は、ユーザーが提供した順序を渡す手段を備えていないからです。

他のオプション

使用するソート・プログラムに応じて、ほかにいくつかのオプションを指定できます。 それらのオプションは FIELDS オペランドとの間、およびオプション同士の間をコンマで区切らなければなりません。 オペランドとオペランドの間にブランクを入れないでください。

FILSZ=y

ソートのレコード数を指定し、ソート・プログラムによる最適化を可能にします。y が単に近似値であれば、y の前に E を付けなければなりません。

SKIPREC=y

ソート・プログラムが入力ファイルの先頭から y 個のレコードを無視して残りのレコードをソートし始めるように指定します。

CKPT または **CHKPT**

チェックポイントをとることを指定します。このオプションを使用する場合は、SORTCKPT データ・セットを指定する必要があります。さらに、DFSORT をインストールする場合は、16NCKPT=NO インストール・オプションを指定する必要があります。

EQUALS|NOEQUALS

等しいレコードの順序を、入力したときと同じままにするか (EQUALS)、または任意にするか (NOEQUALS) を指定します。NOEQUALS を使えば、ソート・パフォーマンスを上げることができます。 デフォルト・オプションは、ソート・プログラムがインストールされるときに選択されます。IBM 提供のデフォルトは、NOEQUALS です。

DYNALLOC=(d,n)

(OS/VS ソート・プログラムの場合のみ) プログラムは中間記憶装置を動的に割り振ることを指定します。

d 装置タイプ (3380 など) を指定します。

n 作業域数を指定します。

ソートするレコードの指定

このトピックでは、PL/I からソート・プログラムを使用するために指定しなければならない必須の PL/I ステートメントについて説明します。RECORD ステートメントは、PLISRT*x* に対する 2 番目の引数です。

RECORD ステートメントの構文は、評価時に次の構文を使用する文字ストリング式でなければなりません。

```
'bRECORDbTYPE=rectype[,LENGTH=(L1,[,,L4,L5])]b'
```

次の例を参照してください。

```
' RECORD TYPE=F,LENGTH=(80) '
```

b 1 つ以上のブランクを表します。ここに示されているブランクは必須です。 これ以外のブランクは認められません。

TYPE

次のように、レコード・タイプを指定します。

F 固定長

V 可変長 EBCDIC

D 可変長 ASCII

未ソート・データとソート済みデータを処理するのに入力ルーチンと出力ルーチンを使用するときでも、ソート・プログラムが使用する作業データ・セットに適用されるレコード・タイプを指定しなければなりません。

可変長ストリングを入力ルーチン (E15 出口) からソート・プログラムに渡すときには、通常はレコード・フォーマットとして V を指定すべきです。 ただし、F を指定すると、最大長になるまでレコードにブランクが埋め込まれます。

LENGTH

ソートするレコードの長さを指定します。PLISRTA や PLISRTC を使う場合、レコード長は入力データ・セットからとられるので、LENGTH は省略することができます。 ソートできるレコードの最大長と最小長には制限があることに注意してください。可変長のレコードの場合は、4 バイトの接頭部を含める必要があります。

L1 ソートするレコードの長さを指定します。VSAM データ・セットを可変長レコードとしてソートする場合、この長さは最大レコード・サイズ +4 となります。

,, PL/I からの呼び出し時にソート・プログラムに適用されない 2 つの引数を表します。後続の引数を使用する場合は、コンマを組み込む必要があります。

L4 可変長レコードの使用時の最小レコード長を指定します。これを指定すると、この値は、ソート・プログラムによって最適化のために使用されます。

L5 可変長レコードの使用時の形式指定上の (最も一般的な) レコード長を指定します。これを指定すると、この値は、ソート・プログラムによって最適化のために使用されます。

最大レコード長: レコードの長さは、ユーザーが指定する最大長を超えることはありません。 最大レコード長は、可変長レコードの場合は 32756 バイトであり、固定長レコードの場合は 32760 バイトです。

ソート・プログラムに必要なストレージの決定

ソートには、主記憶域と補助記憶域の両方が必要です。

主記憶域

DFSORT の最小主記憶域は 88K バイトですが、最大限のパフォーマンスのためにはこれより多くのストレージ (1 メガバイト強程度) をお勧めします。 DFSORT は 16M より上の仮想記憶域あるいは拡張アーキテクチャー・プロセッサーを利用します。 z/OS のもとでは、DFSORT は拡張ストレージも利用できます。次のようにストレージ・パラメーターを渡せば、ソート・プログラムが使用可能なストレージを最大限使用するように指定することができます。

```
DCL MAXSTOR FIXED BINARY (31,0);
UNSPEC(MAXSTOR)='00000000'B||UNSPEC('MAX');
CALL PLISRTA
    (' SORT FIELDS=(1,80,CH,A) ',
     ' RECORD TYPE=F,LENGTH=(80) ',
       MAXSTOR,
       RETCODE,
       'TASK');
```

ファイルを E15 または E35 出口ルーチン内でオープンする場合には、ファイルを正常にオープンできるよう、十分な残余ストレージを確保しておいてください。

補助記憶域

ある特定のソート操作の最小補助記憶域を計算するのは、複雑な作業です。 補助記憶域によって最大限に効率を上げるには、できるかぎり直接アクセス記憶装置 (DASD) を使用します。 プログラム効率向上に関する詳細は、「*DFSORT Application Programming Guide*」、特に DFSORT が必要な補助記憶域を決定して割り振ることを可能とする、動的ワークスペース割り振りに関する情報を参照してください。

ソートが確実に行われるように十分なストレージを提供することだけが目的の場合は、SORTWK データ・セットの合計サイズを、ソートするレコードを 3 セット保持できる大きさにします。 (3 つのデータ・セット内に十分なスペースがあれば、3 より大きい数を指定してもとくに利益は得られません。)

ただし、この推奨値は近似値であるため、うまくいかないこともあるので、その場合には、ソートに関する資料を参照してください。 この推奨値でうまくいった場合でも、スペースを無駄に使っている可能性はあります。

ソート・プログラムの呼び出し

CALL PLISRT*x* ステートメントは慎重に作成する必要があります。このトピックでは、使用可能なエントリー・ポイントと引数をリストします。

表 *34. PLISRTx (x = A、B、C、または D)* に対するエントリー・ポイントおよび引数

エントリー・ポイント	引数
PLISRTA ソート入力: データ・セット ソート出力: データ・セット	(SORT ステートメント、RECORD ステートメント、ストレージ、戻りコード [、データ・セット接頭部、メッセージ・レベル、ソート手法])
PLISRTB ソート入力: PL/I サブルーチン ソート出力: データ・セット	(SORT ステートメント、RECORD ステートメント、ストレージ、戻りコード、 入力ルーチン [、データ・セット接頭部、メッセージ・レベル、ソート手法])
PLISRTC ソート入力: データ・セット ソート出力: PL/I サブルーチン	(SORT ステートメント、RECORD ステートメント、ストレージ、戻りコード、 出力ルーチン [、データ・セット接頭部、メッセージ・レベル、ソート手法])
PLISRTD ソート入力: PL/I サブルーチン ソート出力: PL/I サブルーチン	(SORT ステートメント、RECORD ステートメント、ストレージ、戻りコード、入力ルーチン、出力ルーチン[、データ・セット接頭部、メッセージ・レベル、ソート手法])
SORT ステートメント	ソート・プログラムの SORT ステートメントが含まれた文字ストリング式。 ソート・フィールドとフォーマットを記述する。 395 ページの『ソート・フィールドの指定』を参照してください。
RECORD ステートメント	ソート・プログラム RECORD ステートメントが含まれた文字ストリング式。 データの長さとレコード・フォーマットを記述する。 397 ページの『ソートするレコードの指定』を参照してください。
ストレージ	ソート・プログラムで使用される主記憶域の最大値を示す固定 2 進式。 DFSORT では、>88K バイトでなければならない。 398 ページの『ソート・プログラムに必要なストレージの決定』 も参照してください。
戻りコード	精度 (31,0) の固定 2 進変数。ソート・プログラムが完了するとこの中に戻りコードが入る。戻りコードの意味は次のとおり。 0= ソート・プログラムは正常に完了 16= ソート・プログラムは失敗 20= ソート・プログラム・メッセージ・データ・セットが欠落
入力ルーチン	(PLISRTB および PLISRTD の場合のみ。) ソート・プログラムにレコードをソート出口 E15 で渡すのに使用する PL/I 外部または内部プロシージャーの名前。
出力ルーチン	(PLISRTC および PLISRTD の場合のみ。) ソートがソート出口 E35 でソート済みレコードを渡す PL/I 外部または内部プロシージャーの名前。
データ・セット接頭部	データ・セット SORTIN、SORTOUT、SORTWKnn、および SORTCNTL が使用される場合に、そのデータ・セット名にあるデフォルト接頭部 'SORT' を置き換える 4 文字の文字ストリング式。 このため、引数が「TASK」であれば、データ・セット TASKIN、TASKOUT、TASKWKnn、および TASKCNTL を使用することができる。 この機能により、同一ジョブ・ステップ内でソート・プログラムを複数呼び出すことができる。 この 4 文字は英字で始まらなければならず、予約名 PEER、BALN、CRCX、OSCL、POLY、DIAG、SYSC、または LIST のいずれであってもならない。 後続引数のうちのいずれかが必要だが、この引数は必要でない場合、この引数にはヌル・ストリングをコーディングしなければならない。

表 *34. PLISRTx (x = A、B、C、または D)* に対するエントリー・ポイントおよび引数 *(続き)*

エントリー・ポイント	引数
メッセージ・レベル	ソート・プログラムの診断メッセージの処理方法を示す次のような 2 文字の文字ストリング式。 NO メッセージを SYSOUT へ出さない AP すべてのメッセージを SYSOUT へ送る CP 重大メッセージを SYSOUT へ送る SYSOUT は通常、プリンターに割り振られるため、簡略記号文字「P」を使用。 ソート・プログラムによっては、他のコードを使用することもできる。 このようなコードに関する詳細は、「*DFSORT Application Programming Guide*」を参照。後続引数が必要だが、この引数は必要でない場合、この引数にはヌル・ストリングをコーディングしなければならない。
ソート手法	(これは DFSORT では使用されない。互換性のためのみに存在。) 次のように、行おうとするソートのタイプを示す長さ 4 の文字ストリング。 PEER 対等機能ソート BALN 平衡 CRCX 交差ソート OSCL 振動 POLY 多相ソート 通常、ソート・プログラムは、ユーザーの処置を必要としないで、使用可能なスペースの大きさを分析して最も効率のよい手法を選択する。 この引数を使用するのは、ソート上の問題で迂回を行うときか、または、別の手法を使えばパフォーマンスが向上するのが確実なときに限らなければならない。 詳細については「*DFSORT Application Programming Guide*」を参照。

次の例は、CALL PLISRT*x* ステートメントが通常とる形式を示しています。

例 1

以下の例では PLISRTA に対する呼び出しが示されています。 PLISRTA では、1048576 (1 メガバイト) のストレージと、FIXED BINARY (31,0) として宣言された戻りコード RETCODE が使用され、80 バイトのレコードが SORTIN から SORTOUT にソートされます。

```
CALL PLISRTA (' SORT FIELDS=(1,80,CH,A) ',
              ' RECORD TYPE=F,LENGTH=(80) ',
                1048576,
                RETCODE);
```

例 2

この例は、入力、出力、および作業データ・セットが TASKIN、TASKOUT、TASKWK01 などという名前に変わっている点を除けば、例 1 と同じです。

ソート・プログラムを 1 つのジョブ・ステップの中で 2 回呼び出したときに、このようになることがあります。

```
CALL PLISRTA (' SORT FIELDS=(1,80,CH,A) ',
              ' RECORD TYPE=F,LENGTH=(80) ',
                1048576,
                RETCODE,
                'TASK');
```

例 3

次の例は、2 つのフィールド上でソートが行われることを除いて例 1 と同じです。最初に文字である 1 バイト目から 10 バイト目でソートされ、それらの文字が同じ場合は、2 進数フィールドである 11 バイト目と 12 バイト目でソートされます。

いずれのフィールドも、昇順でソートされます。

```
CALL PLISRTA (' SORT FIELDS=(1,10,CH,A,11,2,BI,A) ',
              ' RECORD TYPE=F,LENGTH=(80) ',
                1048576,
                RETCODE);
```

例 4

この例は、PLISRTB の呼び出しを示しています。

PL/I ルーチン PUTIN によって入力がソート・プログラムに渡されます。 80 バイト固定長レコードの 1 文字目から 10 文字目までに対してソートが実行されます。

```
CALL PLISRTB (' SORT FIELDS=(1,10,CH,A) ',
              ' RECORD TYPE=F,LENGTH=(80) ',
                1048576,
                RETCODE,
                PUTIN);
```

例 5

この例は、PLISRTD の呼び出しを示しています。

PL/I ルーチン PUTIN によって入力が提供され、PL/I ルーチンの PUTOUT に出力が渡されます。 ソートされるレコードは 84 バイト可変 (長さの接頭部を含む) です。 データのバイト 1 から 5 までを昇順でソートし、次にこれらのフィールドが等しい場合は、バイト 6 から 10 までを降順でソートします。 (4 バイト長さ接頭部が含まれているので、実際に開始点で使う値は 5 と 10 であることに注意してください。) 両方のフィールドが同じである場合は、入力の順序は保持されます。 (これは EQUALS オプションによって行われます。)

```
CALL PLISRTD (' SORT FIELDS=(5,5,CH,A,10,5,CH,D),EQUALS ',
              ' RECORD TYPE=V,LENGTH=(84) ',
                1048576,
                RETCODE,
                PUTIN,       /*input routine (sort exit E15)*/
                PUTOUT);     /*output routine (sort exit E35)*/
```

ソートが成功したかどうかの判別

ソート・プログラムはソートが完了すると、 PLISRT*x* の呼び出しの 4 番目の引数内で指定した変数内に、戻りコードを設定します。

次に、 CALL PLISRT*x* ステートメントの後のステートメントに制御を戻します。戻された値は、次のようにソートが成功または失敗のどちらであったかを示します。

0 ソート・プログラムは正常に完了した
16 ソート・プログラムは失敗した
20 ソート・メッセージ・データ・セットが欠落している

戻りコードを渡す先の変数は FIXED BINARY (31,0) と宣言する必要があります。通常の作業では、CALL PLISRT*x* ステートメントの後で戻りコードの値をテストし、操作が成功したか失敗したかに応じて適切な処置をとります。

次の例を参照してください (戻りコードが RETCODE の場合):

```
IF RETCODE¬=0 THEN DO;
  PUT DATA(RETCODE);
  SIGNAL ERROR;
END;
```

ソート後のジョブ・ステップが、ソートの成功、失敗に依存している場合は、ソート・プログラム内で戻された値を、PL/I プログラムからの戻りコードとして設定する必要があります。 これで、その戻りコードを次のジョブ・ステップに使用することができます。 PL/I 戻りコードは、PLIRETC への呼び出しで設定されます。PLIRETC を呼び出すには、ソート・プログラムから返される値を使用します。

```
CALL PLIRETC(RETCODE);
```

この PLIRETC の呼び出しを、ソート・プログラムへ制御情報を渡すのに戻りコードが使用される入出力ルーチン内での呼び出しと混同しないでください。

ソート・プログラム用のデータ・セットの確立

システムに認識されていないライブラリーに DFSORT がインストールされている場合は、JOBLIB または STEPLIB DD ステートメントで DFSORT ライブラリーを指定する必要があります。

ソート・プログラムを呼び出すときは、特定のソート・データ・セットがオープンされていてはなりません。

SYSOUT
: ソート・プログラムからのメッセージが書き込まれるデータ・セット (通常はプリンター)

ソート作業データ・セット**: SORTWK01–SORTWK32**

> 注**:** 32 より多いソート作業データ・セットを指定した場合、DFSORT は最初の 32 個しか使用しません。
>
> ******WK01–****WK32**
> : ソート処理で使用される 1 から 32 個までのデータ・セット。
>
> これらは直接アクセスでなければなりません。 必要なスペースとデータ・セットの数については、 398 ページの『ソート・プログラムに必要なストレージの決定』を参照してください。
>
> ******** は、 PLISRT*x* への呼び出しにおいてデータ・セット接頭部引数として指定できる 4 文字を表します。 この文字を使えば、SORT 以外の接頭部を使ってデータ・セットを使用することもできます。 これは英字で始まらなければならず、名前は
> PEER、BALN、CRCX、OSCL、POLY、SYSC、LIST、または DIAG
> であってはなりません。

入力データ・セット**: SORTIN**

******IN**
PLISRTA および PLISRTC を呼び出すときに使用する入力データ・セット

詳しくは、****WK01–****WK32 を参照してください。

出力データ・セット**: SORTOUT**

******OUT**
PLISRTA および PLISRTB を呼び出すときに使用する出力データ・セット

詳しくは、****WK01–****WK32 を参照してください。

チェックポイント・データ・セット**: SORTCKPT**
チェックポイント・データを保持するために使用されるデータ・セット (CKPT オプションまたは CHKPT オプションが SORT ステートメント引数で使用されていて DFSORT 16NCKPT=NO インストール・オプションが指定されている場合)

このプログラム DD ステートメントに関する詳細は、「*DFSORT Application Programming Guide*」を参照してください。

DFSPARM SORTCNTL
追加の制御ステートメントまたは変更された制御ステートメントが入った (オプショナルの) データ・セット

このプログラム DD ステートメントに関する詳細は、「*DFSORT Application Programming Guide*」を参照してください。

詳しくは、****WK01–****WK32 を参照してください。

ソート・データの入出力

ソートするデータのソースは、データ・セットから直接提供される場合と、ユーザーが作成したルーチン (ソート出口 E15) によって間接的に提供される場合とがあります。 同様に、ソートされた出力の宛先も、データ・セットである場合と、ユーザー提供のルーチン (ソート出口 E35) である場合とがあります。

PLISRTA は、データ・セットからデータ・セットにソートするものであるため、すべてのインターフェースの中で最も単純なインターフェースです。 PLISRTA プログラムの例については、 409 ページの図 57 を参照してください。他のインターフェースには、入力処理ルーチンと出力処理ルーチンの一方または両方が必要です。

データ入出力処理ルーチン

PLISRTB、PLISRTC、または PLISRTD を使用するとき、ソート・プログラムはいくつかの入力処理ルーチンと出力処理ルーチンを呼び出します。

これらのルーチンは、PL/I で作成する必要があり、内部プロシージャーであっても外部プロシージャーであってもかまいません。入出力処理ルーチンが、PLISRT*x* を呼び出すルーチンに対して内部であれば、名前の有効範囲については、通常の内部プロシージャーと同様に動作します。入出力プロシージャー名自体が、PLISRT*x* を呼び出すプロシージャーで認識されていなければなりません。

これらのルーチンは、各レコードが、ソート・プログラムで必要になるか、ソート・プログラムから渡されるたびに個別に呼び出されます。 したがって、各ルーチンは一度に 1 つのレコードを処理できるように作成しなければなりません。 プロシージャー内で AUTOMATIC と宣言される変数は、呼び出しから次の呼び出しの間ではその値を保持しません。 したがって、1 つの呼び出しから次の呼び出しへ保持されなければならないカウンターのような項目は、STATIC として宣言するか、収容ブロック内で宣言する必要があります。

E15 と E35 のソート出口は、MAIN プロシージャーであってはなりません。

E15 — 入力処理ルーチン (ソート出口 E15)

入力ルーチンは、通常、データがソートされる前に、データに対して何らかの処理を加えるのに使用されます。

入力ルーチンは、 410 ページの図 58 および 412 ページの図 60 に示してあるとおり、データを印刷するのに使用するか、正しい結果を出すためにソート・フィールドを生成または操作するのに使用することができます。

入力処理ルーチンは、PLISRTB または PLISRTD を呼び出すときにソート・プログラムが使用します。 ソート・プログラムはレコードが必要な場合は、文字ストリング・フォーマットでレコードを返す入力ルーチンを呼び出して、戻りコード 12 を出力します。この戻りコードの意味は、渡したレコードはソートの対象になるということです。 ソート・プログラムは、戻りコード 8 が渡されるまで、このルーチンを呼び出し続けます。戻りコード 8 の意味は、すべてのレコードが既に渡されていて、ソート・プログラムがそのルーチンを再び呼び出す必要がないということです。戻りコードが 8 のときにレコードが戻された場合、そのレコードはソート・プログラムによって無視されます。

ルーチンによって戻されるデータは、文字ストリングでなければなりません。この文字ストリングは、固定でも可変でも構いません。 可変の場合、PLISRT*x* への呼び出し内の 2 番目の引数である RECORD ステートメント内のレコード・フォーマットとして、通常は、V を指定する必要があります。しかし、F を指定することも可能で、その場合は、ストリングが最大長になるようブランクが埋め込まれます。レコードは RETURN ステートメントを使って戻されるため、PROCEDURE ステートメント内で RETURNS 属性を指定しなければなりません。戻りコードは、PLIRETC の呼び出し内で設定されます。 代表的な入力ルーチンのフローチャートを、『E15 — 入力処理ルーチン (ソート出口 E15)』 に示します。

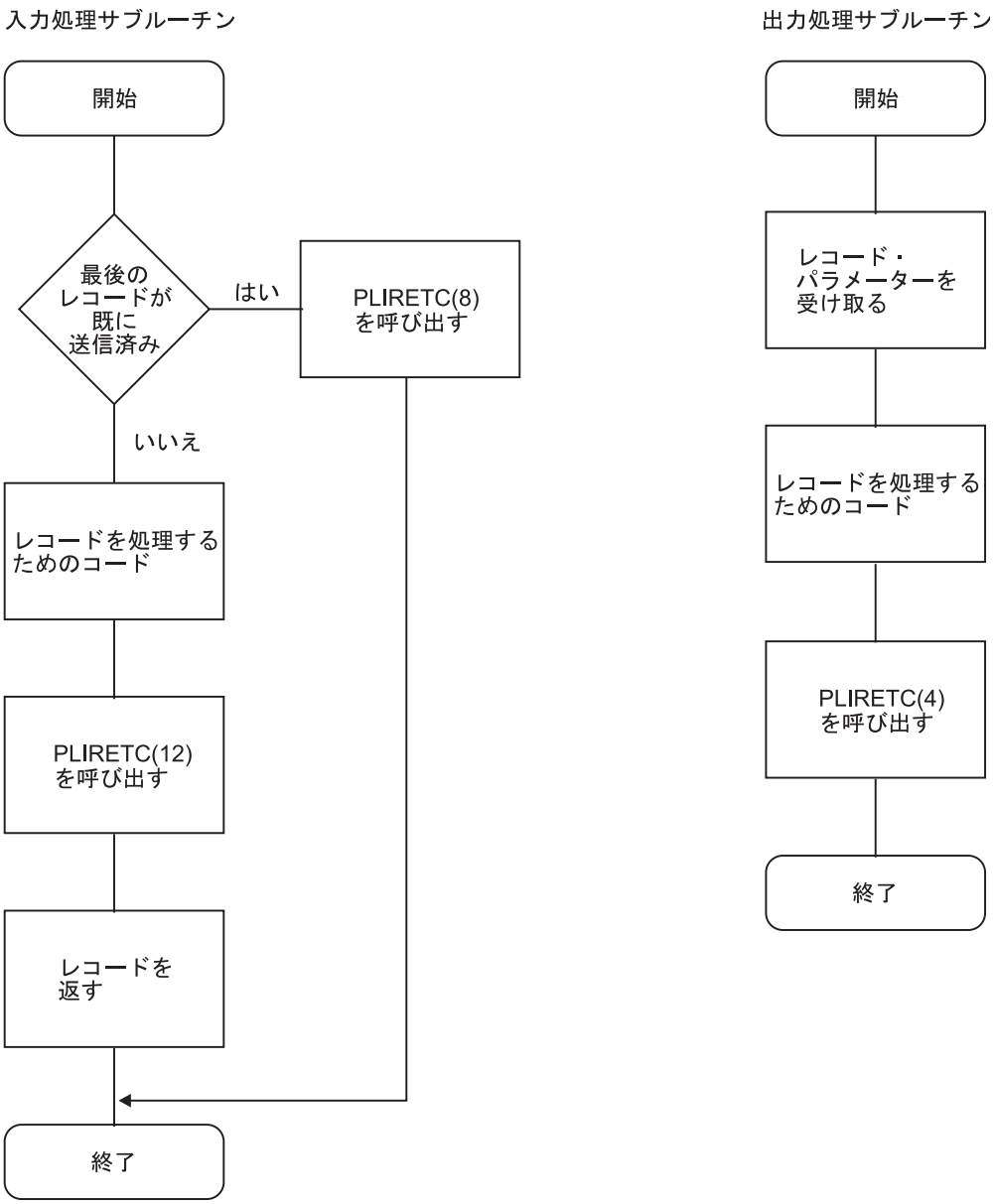


図 *54.* 入力および出力処理サブルーチンのフローチャート

代表的な入力ルーチンの骨組みコードを、 406 ページの図 55 に示します。

```
E15: PROC RETURNS (CHAR(80));
     /*---------------------------------------------------------------*/
     /*RETURNS attribute must be used specifying length of data to be */
     /* sorted, maximum length if varying strings are passed to Sort. */
     /*---------------------------------------------------------------*/
   DCL STRING CHAR(80); /*-----------------------------------------*/
                        /*A character string variable will normally be*/
                        /* required to return the data to Sort       */
                        /*-----------------------------------------*/

   IF LAST_RECORD_SENT THEN
     DO;
      /*---------------------------------------------------------------*/
      /*A test must be made to see if all the records have been sent,  */
      /*if they have, a return code of 8 is set up and control returned*/
      /*to Sort                                                        */
      /*---------------------------------------------------------------*/

        CALL PLIRETC(8);  /*-----------------------------------------*/
                          /* Set return code of 8, meaning last record */
                          /* already sent.                           */
                          /*-----------------------------------------*/
        RETURN('');
     END;

   ELSE
     DO;
      /*-------------------------------------------------*/
      /* If another record is to be sent to Sort, do the*/
      /* necessary processing, set a return code of 12  */
      /* by calling PLIRETC, and return the data as a   */
      /* character string to Sort                       */
      /*-------------------------------------------------*/

   ****(The code to do your processing goes here)

       CALL PLIRETC (12);  /*--------------------------------------*/
                           /* Set return code of 12, meaning this  */
                           /* record is to be included in the sort */
                           /*--------------------------------------*/
      RETURN (STRING);  /*Return data with RETURN statement*/
     END;

  END;   /*End of the input procedure*/
```

図 *55.* 入力プロシージャー用の骨組みコード

入力ルーチンの例については、 410 ページの図 58 および 412 ページの図 60 を参照してください。

ソート・プログラムでは、戻りコード 12 (ソートに現行レコードを組み込む) と戻りコード 8 (すべてのレコードを送信済み) 以外に戻りコード 16 (ソート・プログラム失敗) が使用されます。戻りコード 16 が発生すると、ソートは終了し、ソート・プログラムは制御を PL/I プログラムに返します。

注: PLIRETC への呼び出しは、PL/I プログラムから渡され、それ以降の任意のジョブ・ステップで使用することができる戻りコードを設定します。 出力処理ルーチンを使用した後は、 PLISRT*x* を呼び出してから PLIRETC を呼び出し、戻りコードをリセットして、ゼロ以外の完了コードが出ないようにすることをお勧めします。ソート・プログラムからの戻りコードを引数として使用して PLIRETC を呼び

出せば、PL/I 戻りコードにソートの成功または失敗を反映させることができます。この方法は、 411 ページの図 59 に示してあります。

E35 — 出力処理ルーチン (ソート出口 E35)

出力処理ルーチンは通常、ソート後に必要なすべての処理に使われます。

例えば、出力処理ルーチンは、 411 ページの図 59 や 412 ページの図 60 にあるように、ソートされたデータを印刷したり、ソートされたデータで詳細情報を生成したりする場合に使用できます。出力処理ルーチンは、ソート・プログラムが PLISRTC または PLISRTD を呼び出すときに使用します。レコードのソートが終われば、ソート・プログラムはそれを一度に 1 つずつ、出力処理ルーチンに渡します。 次に、出力ルーチンは、必要に応じてそれを処理します。 すべてのレコードを渡し終えると、ソート・プログラムはその戻りコードをセットアップし、 CALL PLISRT*x* ステートメントの後のステートメントに戻ります。ソート・プログラムは、最終レコードに達したことを出力処理ルーチンに示しません。したがって、データ終了処理は、PLISRT*x* を呼び出すプロシージャーで行わなければなりません。

レコードはソート・プログラムから出力ルーチンへ文字ストリングとして渡されるため、そのデータを受け取るには、出力処理サブルーチン内で文字ストリング・パラメーターを宣言しなければなりません。 また、出力処理サブルーチンはソート・プログラムに戻りコード 4 を渡して、別のレコードを処理する準備ができたことを示す必要があります。この戻りコードは、PLIRETC への呼び出しによって設定します。

ソートを停止するには、ソート・プログラムに戻りコード 16 を渡します。この場合、ソート・プログラムは戻りコード 16 (ソート失敗) を出力して呼び出し側プログラムに制御を返します。

ソート・プログラムからルーチンに渡されるレコードは、文字ストリング・パラメーターです。 PLISRT*x* に対する呼び出しにおいて 2 番目の引数でレコード・タイプを F として指定した場合は、レコード長を指定してこのパラメーターを宣言する必要があります。レコード・タイプを V として指定する場合は、次の例にあるように、パラメーターを調整可能なパラメーターとして宣言する必要があります。

```
DCL STRING CHAR(*);
```

413 ページの図 61 は、可変長レコードをソートするためのプログラムを示しています。

典型的な出力処理ルーチンのフローチャートを、 404 ページの『E15 — 入力処理ルーチン (ソート出口 E15)』 に示してあります。 典型的な出力処理ルーチンの骨組みコードは、 408 ページの図 56 に示してあります。

```
E35: PROC(STRING);      /*The procedure must have a character string
                          parameter to receive the record from Sort*/

  DCL STRING CHAR(80);  /*Declaration of parameter*/

 (Your code goes here)

  CALL PLIRETC(4);    /*Pass return code to Sort indicating that the next
                        sorted record is to be passed to this procedure.*/
   END E35;          /*End of procedure returns control to Sort*/
```

図 *56.* 出力処理プロシージャー用の骨組みコード

PLIRETC を呼び出すと、PL/I プログラムから渡され、その後の任意のジョブ・ステップで使用できる戻りコードが設定されることに注意しなければなりません。 出力処理ルーチンを使用したときは、PLISRT*x* を呼び出してから PLIRETC を呼び出して戻りコードをリセットし、ゼロ以外の完了コードが発生しないようにすることをお勧めします。ソート・プログラムからの戻りコードを引数として使用して PLIRETC を呼び出せば、PL/I 戻りコードにソートの成功または失敗を反映させることができます。この方法は、このトピックに続く例で示されています。

PLISRTA の呼び出し例

このトピックでは、PLISRTA プログラムの例を示します。

PL/I の入力処理ルーチンおよび出力処理ルーチンが、戻りコード情報をソート・プログラムに伝達するたびに、戻りコード・フィールドがゼロにリセットされます。そのため、そのような戻りコードは、ソート・プログラムの特定用途以外の通常の戻りコードとしては使用されません。

処理条件に関する詳細、特に入力および出力処理ルーチン時に生じる条件に関しては、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

```
//OPT14#7  JOB ...
//STEP1 EXEC IBMZCBG
//PLI.SYSIN DD *
 EX106: PROC OPTIONS(MAIN);
     DCL RETURN_CODE FIXED BIN(31,0);

     CALL PLISRTA (' SORT FIELDS=(7,74,CH,A) ',
                   ' RECORD TYPE=F,LENGTH=(80) ',
                     1048576
                     RETURN_CODE);
     SELECT (RETURN_CODE);
       WHEN(0) PUT SKIP EDIT
           ('SORT COMPLETE RETURN_CODE 0') (A);
       WHEN(16) PUT SKIP EDIT
           ('SORT FAILED, RETURN_CODE 16') (A);
       WHEN(20) PUT SKIP EDIT
           ('SORT MESSAGE DATASET MISSING ') (A);
       OTHER    PUT SKIP EDIT (
            'INVALID SORT RETURN_CODE = ', RETURN_CODE) (A,F(2));
       END /* select */;
       CALL PLIRETC(RETURN_CODE);
      /*set PL/I return code to reflect success of sort*/
       END EX106;
//GO.SORTIN DD *
003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD
002886BOOKER R.R. ROTORUA, LINKEDGE LANE, TOBLEY
003077ROOKER & SON, LITTLETON NURSERIES, SHOLTSPAR
059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON
073872HOME TAVERN, WESTLEIGH
000931FOREST, IVER, BUCKS
/*
//GO.SYSPRINT DD SYSOUT=A
//GO.SORTOUT DD SYSOUT=A
//GO.SYSOUT DD SYSOUT=A
//GO.SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,2)
/*
```

図 *57. PLISRTA* - 入力データ・セットから出力データ・セットへのソート

PLISRTB の呼び出し例

このトピックでは、PLISRTB プログラムの例を示します。このプログラムは、入力ルーチンを呼び出してデータを取得したり、ソート済みレコードをデータ・セット上に配置したりします。

```
//OPT14#8  JOB ...
//STEP1 EXEC IBMZCBG
//PLI.SYSIN DD *
 EX107:  PROC OPTIONS(MAIN);
     DCL RETURN_CODE FIXED BIN(31,0);

     CALL PLISRTB (' SORT FIELDS=(7,74,CH,A) ',
                   ' RECORD TYPE=F,LENGTH=(80) ',
                   1048576
                   RETURN_CODE,
                   E15X);
     SELECT(RETURN_CODE);
       WHEN(0)  PUT SKIP EDIT
           ('SORT COMPLETE RETURN_CODE 0') (A);
       WHEN(16) PUT SKIP EDIT
           ('SORT FAILED, RETURN_CODE 16') (A);
       WHEN(20) PUT SKIP EDIT
           ('SORT MESSAGE DATASET MISSING ') (A);
       OTHER   PUT SKIP EDIT
           ('INVALID RETURN_CODE = ',RETURN_CODE)(A,F(2));
     END /* select */;
     CALL PLIRETC(RETURN_CODE);
      /*set PL/I return code to reflect success of sort*/

 E15X:   /* INPUT HANDLING ROUTINE GETS RECORDS FROM THE INPUT
            STREAM AND PUTS THEM BEFORE THEY ARE SORTED*/
     PROC RETURNS (CHAR(80));
          DCL SYSIN FILE RECORD INPUT,
              INFIELD CHAR(80);

          ON ENDFILE(SYSIN) BEGIN;
             PUT SKIP(3) EDIT ('END OF SORT PROGRAM INPUT')(A);
             CALL PLIRETC(8);  /* signal that last record has
                                  already been sent to sort*/
             INFIELD = '';
             GOTO ENDE15;
             END;

           READ FILE (SYSIN) INTO (INFIELD);
           PUT SKIP EDIT (INFIELD)(A(80)); /*PRINT INPUT*/
           CALL PLIRETC(12);  /* request sort to include current
                                 record and return for more*/
    ENDE15:
          RETURN(INFIELD);
          END E15X;
  END EX107;
/*
//GO.SYSIN DD *
003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD
002886BOOKER R.R. ROTORUA, LINKEDGE LANE, TOBLEY
003077ROOKER & SON, LITTLETON NURSERIES, SHOLTSPAR
059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON
073872HOME TAVERN, WESTLEIGH
000931FOREST, IVER, BUCKS
/*
//GO.SYSPRINT DD SYSOUT=A
//GO.SORTOUT  DD SYSOUT=A
//GO.SYSOUT DD SYSOUT=A
//*
//GO.SORTCNTL DD *
   OPTION DYNALLOC=(3380,2),SKIPREC=2
/*
```

図 *58. PLISRTB* - 入力処理ルーチンから出力データ・セットへのソート

PLISRTC の呼び出し例

このトピックでは、PLISRTC プログラムの例を示します。このプログラムは、入力データ・セット内のレコードをソートしたり、出力処理ルーチンを呼び出してソート済みデータを出力したりします。

```
//OPT14#9  JOB ...
//STEP1 EXEC IBMZCBG
//PLI.SYSIN DD *
 EX108:  PROC OPTIONS(MAIN);

     DCL RETURN_CODE FIXED BIN(31,0);

     CALL PLISRTC (' SORT FIELDS=(7,74,CH,A) ',
                  ' RECORD TYPE=F,LENGTH=(80) ',
                  1048576
                  RETURN_CODE,
                  E35X);
        SELECT(RETURN_CODE);
          WHEN(0)  PUT SKIP EDIT
              ('SORT COMPLETE RETURN_CODE 0') (A);
          WHEN(16) PUT SKIP EDIT
              ('SORT FAILED, RETURN_CODE 16') (A);
          WHEN(20) PUT SKIP EDIT
              ('SORT MESSAGE DATASET MISSING ') (A);
          OTHER    PUT SKIP EDIT
              ('INVALID RETURN_CODE = ', RETURN_CODE) (A,F(2));
        END /* select */;
     CALL PLIRETC (RETURN_CODE);
      /*set PL/I return code to reflect success of sort*/

 E35X:   /* output handling routine prints sorted records*/
       PROC (INREC);
          DCL INREC CHAR(80);
          PUT SKIP EDIT (INREC) (A);
          CALL PLIRETC(4); /*request next record from sort*/
       END E35X;
    END EX108;
/*
//GO.STEPLIB DD DSN=SYS1.SORTLINK,DISP=SHR
//GO.SYSPRINT DD SYSOUT=A
//GO.SYSOUT   DD SYSOUT=A
//GO.SORTIN DD *
003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD
002886BOOKER R.R. ROTORUA, LINKEDGE LANE, TOBLEY
003077ROOKER & SON, LITTLETON NURSERIES, SHOLTSPAR
059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON
073872HOME TAVERN, WESTLEIGH
000931FOREST, IVER, BUCKS
/*
//GO.SORTCNTL DD *
   OPTION DYNALLOC=(3380,2),SKIPREC=2
/*
```

図 *59. PLISRTC* - 入力データ・セットから出力処理ルーチンへのソート

PLISRTD の呼び出し例

このトピックでは、PLISRTD プログラムの例を示します。このプログラムは、入力処理ルーチンを呼び出して未ソート・データを出力したり、出力処理ルーチンを呼び出してソート済みデータを出力したりします。

```
//OPT14#10  JOB ...
//STEP1 EXEC IBMZCBG
//PLI.SYSIN DD *
 EX109:  PROC OPTIONS(MAIN);
     DCL RETURN_CODE FIXED BIN(31,0);
     CALL PLISRTD (' SORT FIELDS=(7,74,CH,A) ',
                  ' RECORD TYPE=F,LENGTH=(80) ',
                  1048576
                  RETURN_CODE,
                  E15X,
                  E35X);

     SELECT(RETURN_CODE);
       WHEN(0)  PUT SKIP EDIT
           ('SORT COMPLETE RETURN_CODE 0') (A);
       WHEN(20) PUT SKIP EDIT
           ('SORT MESSAGE DATASET MISSING ') (A);
       OTHER    PUT SKIP EDIT
           ('INVALID RETURN_CODE = ', RETURN_CODE) (A,F(2));
       END /* select */;

     CALL PLIRETC(RETURN_CODE);
      /*set PL/I return code to reflect success of sort*/

  E15X:   /* Input handling routine prints input before sorting*/
        PROC RETURNS(CHAR(80));
             DCL INFIELD CHAR(80);

             ON ENDFILE(SYSIN) BEGIN;
                PUT SKIP(3) EDIT ('END OF SORT PROGRAM INPUT.  ',
                    'SORTED OUTPUT SHOULD FOLLOW')(A);
                CALL PLIRETC(8);  /* Signal end of input to sort*/
                INFIELD = '';
                GOTO ENDE15;
             END;

             GET FILE (SYSIN) EDIT (INFIELD) (A(80));
             PUT SKIP EDIT (INFIELD)(A);
             CALL PLIRETC(12);  /*Input to sort continues*/
   ENDE15:
            RETURN(INFIELD);
             END E15X;

   E35X:   /* Output handling routine prints the sorted records*/
        PROC (INREC);

             DCL INREC CHAR(80);
             PUT SKIP EDIT (INREC) (A);
      NEXT:  CALL PLIRETC(4); /* Request next record from sort*/
             END E35X;
  END EX109;
 /*
//GO.SYSOUT DD SYSOUT=A
//GO.SYSPRINT DD SYSOUT=A
//GO.SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,1)
//GO.SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,1)
//GO.SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,1)
//GO.SYSIN DD *
003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD
002886BOOKER R.R. ROTORUA, LINKEDGE LANE, TOBLEY
003077ROOKER & SON, LITTLETON NURSERIES, SHOLTSPAR
059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON
073872HOME TAVERN, WESTLEIGH
000931FOREST, IVER, BUCKS
/*
```

図 *60. PLISRTD* - 入力処理ルーチンから出力処理ルーチンへのソート

可変長レコードのソートの例

以下の例では、可変長レコードをソートするためのプログラムが示されています。

```
//OPT14#11 JOB ...
//STEP1 EXEC IBMZCBG
//PLI.SYSIN DD *
  /* PL/I EXAMPLE USING PLISRTD TO SORT VARIABLE-LENGTH
     RECORDS */

  EX1306:  PROC OPTIONS(MAIN);
          DCL RETURN_CODE FIXED BIN(31,0);
          CALL PLISRTD (' SORT FIELDS=(11,14,CH,A) ',
                       ' RECORD TYPE=V,LENGTH=(84,,,24,44) ',
                       /*NOTE THAT LENGTH IS MAX AND INCLUDES
                         4 BYTE LENGTH PREFIX*/
                       1048576
                       RETURN_CODE,
                       PUTIN,
                       PUTOUT);

               SELECT(RETURN_CODE);
                 WHEN(0)  PUT SKIP EDIT (
                        'SORT COMPLETE RETURN_CODE 0') (A);
                 WHEN(16) PUT SKIP EDIT (
                        'SORT FAILED, RETURN_CODE 16') (A);
                 WHEN(20) PUT SKIP EDIT (
                        'SORT MESSAGE DATASET MISSING ') (A);
                 OTHER    PUT SKIP EDIT (
                      'INVALID RETURN_CODE = ', RETURN_CODE)
                         (A,F(2));
               END /* SELECT */;

     CALL PLIRETC(RETURN_CODE);
      /*SET PL/I RETURN CODE TO REFLECT SUCCESS OF SORT*/
      PUTIN: PROC RETURNS (CHAR(80) VARYING);
        /*OUTPUT HANDLING ROUTINE*/
        /*NOTE THAT VARYING MUST BE USED ON RETURNS ATTRIBUTE
          WHEN USING VARYING LENGTH RECORDS*/
          DCL STRING CHAR(80) VAR;

          ON ENDFILE(SYSIN) BEGIN;
               PUT SKIP EDIT ('END OF INPUT')(A);
               CALL PLIRETC(8);
               STRING = '';
               GOTO ENDPUT;
               END;

          GET EDIT(STRING)(A(80));
          I=INDEX(STRING||'  ','  ')-1;/*RESET LENGTH OF THE*/
          STRING = SUBSTR(STRING,1,I); /* STRING FROM 80 TO */
                                       /* LENGTH OF TEXT IN */
                                       /* EACH INPUT RECORD.*/
```

図 *61.* 入出力処理ルーチンを使った可変長レコードのソート

```
          PUT SKIP EDIT(I,STRING) (F(2),X(3),A);
          CALL PLIRETC(12);
 ENDPUT:  RETURN(STRING);
          END;
 PUTOUT:PROC(STRING);
         /*OUTPUT HANDLING ROUTINE OUTPUT SORTED RECORDS*/
          DCL STRING CHAR (*);
           /*NOTE THAT FOR VARYING RECORDS THE STRING
             PARAMETER FOR THE OUTPUT-HANDLING ROUTINE
             SHOULD BE DECLARED ADJUSTABLE BUT CANNOT BE
             DECLARED VARYING*/
          PUT SKIP EDIT(STRING)(A);  /*PRINT THE SORTED DATA*/
          CALL PLIRETC(4);
          END;  /*ENDS PUTOUT*/
         END;
/*
//GO.SYSIN DD *
003329HOOKER S.W. RIVERDALE, SATCHWELL LANE, BACONSFIELD
002886BOOKER R.R. ROTORUA, LINKEDGE LANE, TOBLEY
003077ROOKER & SON, LITTLETON NURSERIES, SHOLTSPAR
059334HOOK E.H. 109 ELMTREE ROAD, GANNET PARK, NORTHAMPTON
073872HOME TAVERN, WESTLEIGH
000931FOREST, IVER, BUCKS
/*
//GO.SYSPRINT DD SYSOUT=A
//GO.SORTOUT DD SYSOUT=A
//GO.SYSOUT DD SYSOUT=A
//GO.SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,1)
//GO.SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,1)
//*
```

入出力処理ルーチンを使った可変長レコードのソート *(*続き*)*

第 17 章 C との ILC

この章では、PL/I と C の間で行われる言語間通信 (ILC) の局面について、いくつか説明します。ここでは、両方の言語に共通する多くのデータ型を使用する方法について例を交えて説明しています。また、その例は、C を呼び出したり C に呼び出されたりする PL/I コードを作成するときに役立ちます。

同等なデータ・タイプ

このトピックは、C と PL/I に共通する同等なデータ・タイプの一覧です。

表 *35. C と PL/I* の同等なタイプ

C のタイプ	適合する **PL/I** タイプ
char[...]	char(...) varyingz
wchar[...]	wchar(...) varyingz
signed char	fixed bin(7)
unsigned char	unsigned fixed bin(8)
short	fixed bin(15)
unsigned short	unsigned fixed bin(16)
int	fixed bin(31)
unsigned int	unsigned fixed bin(32)
long long	fixed bin(63)
unsigned long long	unsigned fixed bin(64)
float	float bin(21)
double	float bin(53)
long double	float bin(p) (p >= 54)
enum	ordinal
typedef	define alias
struct	define struct
union	define union
struct *	handle

単純なタイプの一致

次の例は、C ヘッダー・ファイル `time.h` から抜粋した、単純な `time_t` の `typedef` の変換を示しています。

© Copyright IBM Corp. 1999, 2017

```
typedef long time_t;

define alias time_t fixed bin(31);
```

図 *62.* 単純なタイプの一致

struct タイプの一致

次の例は、C ヘッダー・ファイル time.h から抜粋した、単純な tm の struct の変換を示しています。

```
struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};

define structure
  1 tm
     ,2 tm_sec   fixed bin(31)
     ,2 tm_min   fixed bin(31)
     ,2 tm_hour  fixed bin(31)
     ,2 tm_mday  fixed bin(31)
     ,2 tm_mon   fixed bin(31)
     ,2 tm_year  fixed bin(31)
     ,2 tm_wday  fixed bin(31)
     ,2 tm_yday  fixed bin(31)
     ,2 tm_isdst fixed bin(31)
   ;
```

図 *63. struct* タイプの一致の例

enum タイプの一致

以下の例は、C ヘッダー・ファイル stdio.h にある単純な enum __device_t を変換する方法を示したものです。

```
typedef enum {
   __disk       =  0,
   __terminal   =  1,
   __printer    =  2,
   __tape       =  3,
   __tdq        =  5,
   __dummy      =  6,
   __memory     =  8,
   __hfs        =  9,
   __hiperspace = 10
} __device_t;

define ordinal __device_t (
        __disk       value(0)
      , __terminal   value(1)
      , __printer    value(2)
      , __tape       value(3)
      , __tdq        value(4)
      , __dummy      value(5)
      , __memory     value(8)
      , __hfs        value(9)
      , __hiperspace value(10)
   );
```

図 *64. enum* タイプの一致の例

ファイル・タイプの一致

C のファイル宣言はプラットフォームによって異なりますが、通常は次のように始まります。

```
struct __file {
    unsigned char  *__bufPtr;
 ... } FILE;
```

図 *65. FILE* タイプの *C* 宣言の開始

必要なものはファイルのポインター (トークン) です。そのため、この変換は次のようにうまく解決できます。

```
define struct    1 file;
define alias     file_Handle  handle file;
```

図 *66. C* ファイルと一致する *PL/I*

C 関数を使用する

C 関数 **fopen** と **fread** を使用して、ファイルを読み取ってフォーマット付き 16 進数としてダンプするプログラムをプログラマーが作成するとします。

このプログラムのコードは、次のように簡単なものです。

```
filedump:
  proc(fn) options(noexecops main);

  dcl fn             char(*) var;

  %include filedump;

  file = fopen( fn, 'rb' );

  if file = sysnull() then
    do;
      display( 'file could not be opened' );
      return;
    end;

  do forever;
    unspec(buffer) = ''b;

    read_In = fread( addr(buffer), 1, stg(buffer), file );

    if read_In = 0 then
      leave;

    display(    heximage(addr(buffer),16,' ')  ||  '  '
             || translate(buffer,(32)'.',unprintable)  );

    if read_In < stg(buffer) then
      leave;
  end;

  call fclose( file );
end filedump;
```

図 *67. fopen* と *fread* を使用してファイルをダンプするコードの例

次のように、INCLUDE ファイル `filedump` の宣言のほとんどは自明なものです。

```
  define struct      1 file;
  define alias       file_Handle  handle file;

  define alias       size_t unsigned fixed bin(32);
  define alias       int signed fixed bin(31);

  dcl file           type(file_Handle);
  dcl read_In        fixed bin(31);
  dcl buffer         char(16);

  dcl unprintable    char(32) value( substr(collate(),1,32) );
```

図 *68. filedump* プログラムの宣言

一致する単純パラメーター・タイプ

C 関数の宣言の変換は間違いを起こしやすいかもしれません。 例えば、 419 ページの図 69 に示されている C 関数 **fread** の宣言は、 419 ページの図 70 に示され

ている宣言に変換できます。

```
size_t   fread(  void *,
                 size_t,
                 size_t,
                 FILE *);
```

図 *69. fread* の C 宣言

```
dcl fread     ext
              entry( pointer,
                     type size_t,
                     type size_t,
                     type file_Handle )
              returns( type size_t );
```

図 *70. fread* の誤った宣言 *(*その *1)*

プラットフォームによっては、C の名前に大文字小文字の区別があるため、これではリンクが正常に行われない場合があります。 この種のリンカーの問題を防ぐために最適な方法は、external 属性の拡張形式を使用して、大文字小文字混合の名前を指定することです。したがって、例えば **fread** の宣言は次のように改良できます。

```
dcl fread     ext('fread')
              entry( pointer,
                     type size_t,
                     type size_t,
                     type file_Handle )
              returns( type size_t );
```

図 *71. fread* の誤った宣言 *(*その *2)*

ただし、これは正しく実行されません。 PL/I パラメーターはデフォルトで byaddr ですが、C パラメーターはデフォルトで byvalue であるためです。この問題を修正するには、byvalue 属性をパラメーターに追加します。

```
dcl fread     ext('fread')
              entry( pointer byvalue,
                     type size_t byvalue,
                     type size_t byvalue,
                     type file_Handle byvalue )
              returns( type size_t );
```

図 *72. fread* の誤った宣言 *(*その *3)*

ただし、 420 ページの図 73 で戻り値がどのように設定されているかに注意してください。4 番目のパラメーター (一時 _temp5 のアドレス) が関数 **fread** に渡され、この関数はそのアドレスに整数の戻りコードを置くことになります。これは、returns に byaddr 属性が適用されている場合に値を戻す方法の規則で、PL/I はデ

フォルトでこの規則を使用します。

```
*      read_In = fread( addr(buffer), 1, stg(buffer), file );
*
       L     r4,FILE(,r13,176)
       L     r1,fread(,r5,12)
       LA    r2,_temp5(,r13,420)
       LA    r8,BUFFER(,r13,184)
       L     r15,&EPA_&WSA(,r1,8)
       L     r0,&EPA_&WSA(,r1,12)
       ST    r0,_CEECAA_(,r12,500)
       LA    r1,#MX_TEMP1(,r13,152)
       ST    r8,#MX_TEMP1(,r13,152)
       LA    r8,1
       ST    r8,#MX_TEMP1(,r13,156)
       ST    r7,#MX_TEMP1(,r13,160)
       ST    r4,#MX_TEMP1(,r13,164)
       ST    r2,#MX_TEMP1(,r13,168)
       BALR  r14,r15
       L     r0,_temp5(,r13,420)
       ST    r0,READ_IN(,r13,180)
```

図 *73. RETURNS BYADDR* に対して生成されるコード

C の戻り値は byvalue なので、これは正しく実行されません。このエラーを修正するには、byvalue 属性をもう 1 つ追加します。

```
dcl fread     ext('fread')
              entry( pointer byvalue,
                     type size_t byvalue,
                     type size_t byvalue,
                     type file_Handle byvalue )
              returns( type size_t byvalue );
```

図 *74. fread* の正しい宣言

421 ページの図 75 で戻り値がどのように設定されているかに注目してください。追加のアドレスは渡されず、戻り値がレジスター 15 に戻されるだけです。

```
*       read_In = fread( addr(buffer), 1, stg(buffer), file );
*
        L     r2,FILE(,r13,176)
        L     r1,fread(,r5,12)
        LA    r7,BUFFER(,r13,184)
        L     r15,&EPA_&WSA(,r1,8)
        L     r0,&EPA_&WSA(,r1,12)
        ST    r0,_CEECAA_(,r12,500)
        LA    r1,#MX_TEMP1(,r13,152)
        ST    r7,#MX_TEMP1(,r13,152)
        LA    r7,1
        ST    r7,#MX_TEMP1(,r13,156)
        ST    r4,#MX_TEMP1(,r13,160)
        ST    r2,#MX_TEMP1(,r13,164)
        BALR  r14,r15
        LR    r0,r15
        ST    r0,READ_IN(,r13,180)
```

図 *75. RETURNS BYVALUE* に対して生成されるコード

一致するストリング・パラメーター・タイプ

現在、**fread** は正しく変換されるため、プログラマーがこの変換を **fopen** に対して試すとします。

```
dcl fopen    ext('fopen')
             entry( char(*) varyingz byvalue,
                    char(*) varyingz byvalue )
             returns( byvalue type file_handle );
```

図 *76. fopen* の誤った宣言 *(その 1)*

しかし、C にストリングはなく、ポインターのみがあります。そのポインターは値 (byvalue) で渡されます。そのため、ストリングは参照 (byaddr) によって渡されなければなりません。

```
dcl fopen    ext('fopen')
             entry( char(*) varyingz byaddr,
                    char(*) varyingz byaddr )
             returns( byvalue type file_handle );
```

図 *77. fopen* の誤った宣言 *(その 2)*

しかし、PL/I はストリングとともに記述子を渡しますが、C は記述子を解釈できないので、記述子は抑止する必要があります。この宣言を抑止するには、options(nodescriptor) を宣言に追加します。

```
dcl fopen    ext('fopen')
             entry( char(*) varyingz byaddr,
                    char(*) varyingz byaddr )
             returns( byvalue type file_handle )
             options ( nodescriptor );
```

図 *78. fopen* の正しい宣言

これは正しく機能しますが最適ではありません (パラメーターが入力専用であるため)。パラメーターが定数の場合、nonassignable 属性を指定すれば、コピーが作成されて渡されることはありません。したがって、**fopen** の宣言の最適な変換は次のとおりです。

```
dcl fopen    ext('fopen')
             entry( char(*) varyingz nonasgn byaddr,
                    char(*) varyingz nonasgn byaddr )
             returns( byvalue type file_handle )
             options ( nodescriptor );
```

図 *79. fopen* の正しく最適な宣言

この時点で、**fclose** 関数の宣言には、おそらく returns 指定の optional 属性を除けば、意外なものはほとんどありません。この属性を指定すれば、CALL ステートメントを使用して **fclose** 関数を呼び出すことができるようになり、戻りコードを処理する必要がなくなります。ただし、ファイルが出力ファイルだった場合は、**fclose** の戻りコードを必ず確認する必要があることに注意してください。これは、ファイルがクローズされるときにのみ最終バッファーが書き出されるが、その書き出しは、スペース不足が原因で失敗する可能性があるためです。

```
dcl fclose   ext('fclose')
             entry( type file_handle byvalue )
             returns( optional type int byvalue )
             options ( nodescriptor );
```

図 *80. fclose* の宣言

これで、z/OS UNIX において次のコマンドでプログラムをコンパイルして実行できるようになりました。

```
pli -qdisplay=std filedump.pli

filedump filedump.pli
```

図 *81. filedump* をコンパイルして実行するためのコマンド

この結果、次の出力が生成されます。

```
15408689 938584A4 94977A40 97999683  . filedump: proc
4D86955D 409697A3 899695A2 4D959685  (fn) options(noe
A7858396 97A24094 8189955D 5E151540  xecops main);..
```

図 *82. filedump* の実行結果の出力

ENTRY を戻す関数

C のクイック・ソート関数 **qsort** は、比較ルーチンを使用します。例えば、整数の配列をソートするには、次の関数 (byvalue 属性が 2 回使用される) を使用できます。

```
comp2:
  proc( key, element )
  returns( fixed bin(31) byvalue );

  dcl (key, element) pointer byvalue;
  dcl word based fixed bin(31);

  select;
    when( key->word < element->word )
      return( -1 );
    when( key->word = element->word )
      return( 0 );
    otherwise
      return( +1 );
  end;
end;
```

図 *83. C qsort* 関数の比較ルーチンの例

次のコード・フラグメントに示すように、 C **qsort** 関数とこの比較ルーチンを組み合わせて使用して、整数の配列をソートできます。

```
dcl a(1:4) fixed bin(31) init(19,17,13,11);

put skip data( a );

call qsort( addr(a), dim(a), stg(a(1)), comp2 );

put skip data( a );
```

図 *84. C qsort* 関数を使用するためのコード例

ただし、C 関数ポインターは PL/I ENTRY 変数と同じではないため、C **qsort** 関数を単純に次のように宣言してはなりません。

```
dcl qsort    ext('qsort')
             entry( pointer,
                    fixed bin(31),
                    fixed bin(31),
                    entry returns( byvalue fixed bin(31) )
                  )
             options( byvalue nodescriptor );
```

図 *85. qsort* の誤った宣言

PL/I ENTRY 変数がネスト関数を指す場合があることに注意してください (このため、エントリー・ポイント・アドレスだけでなく逆チェーン・アドレスが必要です)。しかし、C 関数ポインターが指す先は非ネスト関数だけに限定されるので、PL/I ENTRY 変数と C 関数ポインターはストレージ容量を使用することすらありません。

ただし、C 関数ポインターは PL/I タイプ LIMITED ENTRY と同等です。このため、C **qsort** 関数は次のように宣言できます。

```
dcl qsort    ext('qsort')
             entry( pointer,
                    fixed bin(31),
                    fixed bin(31),
                    limited entry
                    returns( byvalue fixed bin(31) )
                  )
             options( byvalue nodescriptor );
```

図 *86. qsort* の正しい宣言

リンケージ

z/OS 上では、リンケージに関して次の 2 つの重要な事実があります。

- IBM C、JAVA、および Enterprise PL/I は、デフォルトで同じリンケージを使用します。
- このリンケージはシステム・リンケージではありません。

パラメーターがすべて byaddr である従来の PL/I アプリケーションの場合、関数にデフォルト・リンケージが使用される場合に生成されるコードと、関数にシステム・リンケージが使用される場合に生成されるコードの違いは、通常は問題になりません。ただし、パラメーターが byvalue である場合 (C および JAVA ではこれが通常) は、この違いによってコードが使えなくなる可能性があります。

実際には、パラメーターが byaddr である場合の違いはほんの小さなものです。425 ページの図 87 では、デフォルト・リンケージを使用する関数に対して生成されたコードと、システム・リンケージを使用する関数に対して生成されたコードの違いは、システム・リンケージ呼び出しの最終パラメーターの高位ビットがオンになっていることだけです。

この違いは多くのプログラムには透過的です。

```
dcl dfta  ext entry( fixed bin(31) byaddr
                    ,fixed bin(31) byaddr );
dcl sysa  ext entry( fixed bin(31) byaddr
                    ,fixed bin(31) byaddr )
           options( linkage(system) );

*      call dfta( n, m );
*
          LA    r0,M(,r13,172)
          LA    r2,N(,r13,168)
          L     r15,=V(DFTV)(,r3,126)
          LA    r1,#MX_TEMP1(,r13,152)
          ST    r2,#MX_TEMP1(,r13,152)
          ST    r0,#MX_TEMP1(,r13,156)
          BALR  r14,r15
*
*      call sysa( n, m );
*
          LA    r0,M(,r13,172)
          LA    r2,N(,r13,168)
          O     r0,=X'80000000'
          L     r15,=V(SYSV)(,r3,130)
          LA    r1,#MX_TEMP1(,r13,152)
          ST    r2,#MX_TEMP1(,r13,152)
          ST    r0,#MX_TEMP1(,r13,156)
          BALR  r14,r15
```

図 *87.* パラメーターが *BYADDR* である場合のコード

ただし、パラメーターが byaddr でなく byvalue の場合は、大きな違いがあります。 426 ページの図 88 で、デフォルト・リンケージを使用する関数の場合は、レジスター 1 は渡される整数の値を指します。一方、システム・リンケージを使用する関数の場合は、レジスター 1 はこれらの値のアドレスを指します。

この違いは多くのプログラムには透過的ではありません。

```
dcl dftv  ext entry( fixed bin(31) byvalue
                    ,fixed bin(31) byvalue );
dcl sysv  ext entry( fixed bin(31) byvalue
                    ,fixed bin(31) byvalue )
           options( linkage(system) );

*      call dftv( n, m );
*
          L     r2,N(,r13,168)
          L     r0,M(,r13,172)
          L     r15,=V(DFTV)(,r3,174)
          LA    r1,#MX_TEMP1(,r13,152)
          ST    r2,#MX_TEMP1(,r13,152)
          ST    r0,#MX_TEMP1(,r13,156)
          BALR  r14,r15
*
*      call sysv( n, m );
*
          L     r1,N(,r13,168)
          L     r0,M(,r13,172)
          ST    r0,#wtemp_1(,r13,176)
          LA    r0,#wtemp_1(,r13,176)
          ST    r1,#wtemp_2(,r13,180)
          LA    r2,#wtemp_2(,r13,180)
          O     r0,=X'80000000'
          L     r15,=V(SYSV)(,r3,178)
          LA    r1,#MX_TEMP1(,r13,152)
          ST    r2,#MX_TEMP1(,r13,152)
          ST    r0,#MX_TEMP1(,r13,156)
          BALR  r14,r15
```

図 *88.* パラメーターが *BYVALUE* である場合のコード

出力および入力の共有

このセクションでは、STDSYS オプションを指定することによって入出力を C プログラムと共用する方法について説明します。

STDSYS オプションの詳細および制限については、以下の情報を参照してください。

- 3 ページの『第 1 章 コンパイラー・オプションと機能の使用』の 97 ページの『STDSYS』。
- 「*Enterprise PL/I for z/OS* コンパイラーおよびランタイム 移行ガイド」の『印刷不能な文字が含まれた STREAM 入出力』。

出力の共用

C プログラムと SYSPRINT を共用したい場合は、STDSYS オプションを指定して PL/I コードをコンパイルする必要があります。

デフォルトでは、DISPLAY ステートメントは、出力を表示するために WTO を使用します。 DISPLAY(STD) コンパイラー・オプションを指定すると、DISPLAY ステートメントは、C の **puts** 関数を使用して、出力を表示します。これは、z/OS UNIX 環境では特に便利です。

MVS バッチ、TSO バッチ、IMS バッチ、および IMS インタラクティブで出力を共有するための標準 C ストリームの動作は、以下のとおりです。

1. stdout は最初に DD:SYSPRINT へ送られます。
2. DD:SYSPRINT が存在しない場合、stdout は DD:SYSTERM を検索します。
3. DD:SYSTERM も DD:SYSERR も存在しない場合、ライブラリーでは、DD SYSPRINT が使用されて sysout=* データ・セットがオープンし、stdout ストリームがそのデータ・セットに送信されます。

入力の共有

SYSIN を C プログラムと共用するには、STDSYS オプションを指定してアプリケーションをコンパイルし、入力ストリーム・ファイルとして SYSIN をオープンする必要があります。 C ライブラリーで予約されている DD 名は使用しないでください。

また、前のジョブ・ステップで SYSIN を一時データ・セットにコピーし、それを PL/I ジョブ・ステップで SYSIN として使用することもできます。これは、インストリーム・ファイルに割り振られない場合は共有が可能です。

入力ストリーム・ファイルは、JCL 内の SYSIN DD への割り振り時に一度だけ開くことができます。

MVS バッチ、TSO バッチ、IMS バッチ、および IMS インタラクティブで入力を共有するための標準 C ストリームの動作は、以下のとおりです。

1. stdin は DD:SYSIN に送られます。
2. DD:SYSIN が存在しない場合、stdin からの読み取り操作はすべて失敗します。

ATTACH ステートメントの使用

PL/I プログラム内の ATTACH ステートメントは、基礎となる pthread ライブラリーを使用してスレッド管理を実行します。 ATTACH ステートメントが実行されると、pthread ライブラリーは標準 C ストリーム・ファイルの割り振りを試行します。

SYSIN が、JCL 内の SYSIN DD (//SYSIN DD * や //SYSIN DD DATA など) に割り振られているインストリーム・ファイルである場合は、ATTACH ステートメントの発行後に、アプリケーションが SYSIN を開こうとしても失敗します。SYSIN OPEN FAILED エラーが出されます。

C 標準ストリームのリダイレクト

PL/I と C の混合アプリケーションすべてと、すべてのマルチスレッド化アプリケーションでは、 STDSYS オプションを使用します。

NOSTDSYS オプションを指定してプログラムをコンパイルすると、 SYSIN DD 名の使用と SYSPRINT DD 名の使用が C で競合する可能性があります。

例えば、プログラムに ATTACH ステートメントが含まれていると、 C 環境が直接始動します。 C 環境が始動すると、 SYSIN ストリームと SYSPRINT ストリーム

が、 PL/I アプリケーション・プログラムとは無関係に開いたり閉じたりします。場合によっては、SYSIN が開けなくなったり、SYSPRINT データ・セットが上書きされたりすることがあります。

要約

このトピックでは、このセクションで説明されているキーポイントを要約します。

- C は大/小文字の区別がある。
- パラメーターは BYVALUE にする必要がある。
- 戻り値は BYVALUE にする必要がある。
- ストリング・パラメーターは BYADDR にする必要がある。
- 配列と構造体も BYADDR にする必要がある。
- 記述子を渡してはならない。
- 入力専用のストリングは NONASSIGNABLE にする必要がある。
- C 関数ポインターは LIMITED ENTRY にマップする。
- IBM C コンパイラーと IBM PL/I コンパイラーは、同じデフォルト・リンケージを使用する (このことが問題になる)。

第 18 章 Java とのインターフェース

この章では、Java™ と Java Native Interface (JNI) の概要を示し、JNI を PL/I と組み合わせて使用する場合の利点について説明します。

この章では、簡単な Java - PL/I アプリケーションについて説明します。また、2 つの言語間の互換性についても説明します。 Java - PL/I サンプル・アプリケーションを作成して実行する方法の説明では、z/OS の z/OS UNIX システム・サービス 環境で作業を行うことを前提にしています。

PL/I から Java とのやり取りを行うには、事前に z/OS システムに Java をインストールしておく必要があります。z/OS Java 環境のセットアップ方法の詳細については、該当のシステム管理者に問い合わせてください。

このサンプル・プログラムは、Java JRE バージョン 1.6.0 でコンパイルされ、テストされました。ご使用の z/OS UNIX システム・サービス 環境における Java のレベルを判別するには、コマンド行から次のコマンドを入力します。

```
java -version
```

次の例にあるようにアクティブ Java バージョンが表示されます。

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build pmz3160_26sr1-20111114_01(SR1))
IBM J9 VM (build 2.6, JRE 1.6.0 z/OS s390-31 20111113_94967 (JIT enabled, AOT enabled)
```

Java Native Interface (JNI)

Java は、Sun Microsystems によって開発されたオブジェクト指向プログラミング言語で、インターネット文書を対話式に作成するための強力な手段です。 Java Native Interface (JNI) は、ネイティブ・プログラミング言語に対する Java インターフェースで、Java Development Kit の一部です。

JNI を使用するプログラムを作成すれば、多くのプラットフォームにわたってコードを移植できるようになります。

JNI によって、Java 仮想マシン (JVM) 内で稼働する Java コードは、PL/I などの他言語で書かれたアプリケーションやライブラリーと相互運用できます。さらに、*Invocation API* を使用すれば、Java 仮想マシンを PL/I アプリケーションに組み込むことができます。

Java は完成度の高いプログラム言語ですが、状況によっては他のプログラミング言語で書かれたプログラムを呼び出す必要も生じます。 Java からこれを行うには、ネイティブ言語に対するメソッド呼び出し (ネイティブ・メソッド と呼ばれる) を使用します。

ネイティブ・メソッドを使用する理由をいくつか示します。

- アプリケーションのニーズを満たす Java クラス・ライブラリーにはない特殊な機能がネイティブ言語に備わっている。

© Copyright IBM Corp. 1999, 2017

- ネイティブ言語で書かれたアプリケーションが既に多数存在し、 Java アプリケーションからこれらにアクセスできるようにしたい。
- ネイティブ言語で一連の複雑な計算を集中的にインプリメントし、これらの関数を Java アプリケーションから呼び出したい。
- ユーザーまたはプログラマーがネイティブ言語の幅広いスキルを持っていて、この利点を失いたくない。

JNI を使用したプログラミングでは、ネイティブ・メソッドを使用して多種多様な操作を実行できます。

- ネイティブ・メソッドでは、Java メソッドでオブジェクトが使用される方法と同じ方法で Java オブジェクトを使用できます。
- ネイティブ・メソッドでは、Java オブジェクト (配列やストリングなど) を作成して検査し、そのオブジェクトを使用してタスクを実行できます。
- ネイティブ・メソッドでは、Java アプリケーション・コードによって作成されたオブジェクトを検査して使用できます。
- ネイティブ・メソッドでは、ネイティブ・メソッドで作成されたりネイティブ・メソッドに渡されたりした Java オブジェクトを更新できます。更新されたオブジェクトは、Java アプリケーションで使用できるようにすることが可能です。

最後に、ネイティブ・メソッドは Java プログラミング・フレームワークに既に組み込まれている機能を利用して、既存の Java メソッドを呼び出すことも簡単にできます。 このように、アプリケーションのネイティブ言語側と Java 側の両方で Java オブジェクトを作成、更新、および使用でき、さらにこれらのオブジェクトを相互間で共用できます。

Java からの PL/I プログラムの呼び出し

Java プログラムから PL/I プログラムを呼び出す場合、PL/I で開いたファイルはすべて、PL/I が最終的に Java に制御を返す前に、明示的に PL/I で閉じる必要があります。

同様に、 PL/I プログラムでフェッチされたモジュールはすべて、解放する必要があります。

JNI サンプル・プログラム #1 -「Hello World」

最初のサンプル・プログラムは「Hello World!」プログラムの一種です。"Hello World!" プログラムには、1 つの Java クラス `callingPLI.java` があります。PL/I で書かれたネイティブ・メソッドは、`hiFromPLI.pli` に含まれています。

このサンプル・プログラムを作成するための手順を簡単に概説します。

1. ネイティブ・メソッドを含むクラスを定義し、ネイティブ・ロード・ライブラリーをロードし、ネイティブ・メソッドを呼び出す Java プログラムを作成する。
2. Java プログラムをコンパイルして Java クラスを作成する。
3. ネイティブ・メソッドをインプリメントして "Hello!" テキストを表示する PL/I プログラムを作成する。
4. PL/I プログラムをコンパイルしてリンクする。

5. PL/I プログラム内のネイティブ・メソッドを呼び出す Java プログラムを実行する。

ステップ 1: Java プログラムの作成

手順

1. ネイティブ・メソッドを宣言します。

   Java メソッドであるかネイティブ・メソッドであるかに関係なく、メソッドはすべて Java クラス内で宣言する必要があります。 Java メソッドとネイティブ・メソッドの宣言の違いは、キーワード **native** だけです。 **native** キーワードは、このメソッドのインプリメンテーションのある場所が、プログラムの実行時にロードされるネイティブ・ライブラリー内であることを Java に指示します。 ネイティブ・メソッドは次のように宣言できます。

   ```
   public native void callToPLI();
   ```

   上記のステートメントの中で、void はこのネイティブ・メソッド呼び出しから予期される戻り値がないことを示しています。 メソッド名 callToPLI( ) の空括弧は、ネイティブ・メソッドの呼び出し時に渡すパラメーターがないことを示しています。

2. ネイティブ・ライブラリーが実行時にロードされるようにネイティブ・ライブラリーをロードします。

   以下の Java ステートメントを使用すれば、ネイティブ・ライブラリーをロードできます。

   ```
   static {
     System.loadLibrary("hiFromPLI");
   }
   ```

   上記のステートメントでは、ネイティブ・ライブラリーを検索してロードするために、Java システム・メソッド System.loadLibrary(...) が呼び出されています。 PL/I プログラムをコンパイルしてリンクするステップの実行中に、PL/I 共用ライブラリー libhiFromPLI.so が作成されます。

3. Java main メソッドの作成

   callingPLI クラスには、クラスのインスタンスを生成してネイティブ・メソッドを呼び出す main メソッドも含まれています。 main メソッドは callingPLI のインスタンスを生成し、 callToPLI() ネイティブ・メソッドを呼び出します。

   このトピックで前述した点をすべて含む callingPLI クラスの完全な定義は次のようになります。

   ```
   public class callingPLI {
     public native void callToPLI();
     static {
        System.loadLibrary("hiFromPLI");
     }
     public static void main(String[] argv) {
        callingPLI callPLI = new callingPLI();
   ```

```
        callPLI.callToPLI();
        System.out.println("And Hello from Java, too!");
    }
}
```

ステップ 2: Java プログラムのコンパイル

手順

Java コンパイラーを使用して callingPLI クラスをコンパイルし、実行可能形式にします。 以下のコマンドを使用できます。

```
javac callingPLI.java
```

ステップ 3: PL/I プログラムの作成

ネイティブ・メソッドの PL/I インプリメンテーションは、他の PL/I サブルーチンとほぼ同じようなものです。

便利な PL/I コンパイラー・オプション

サンプル・プログラムには、重要なコンパイラー・オプションを定義する一連の *PROCESS ステートメントが含まれています。

```
*Process Limits( Extname( 100 ) ) Margins( 1, 100 ) ;
*Process Display(Std) Dllinit Extrn(Short);
*Process Rent Default( ASCII IEEE );
```

次に、これらのオプションの概要と利点を説明します。

Extname(100)
Java スタイルの長い外部名を許可します。

Margins(1,100)
マージンを拡張して、Java スタイルの名前と ID が入る場所を確保します。

Display(Std)
WTO を介さずに、"Hello World" テキストを stdout に書き込みます。z/OS UNIX 環境では、WTO はユーザーによって意識されることはありません。

Dllinit
DLL の作成に必要な初期化コードをインクルードします。

Extrn(Short)
EXTRN は、参照対象の定数に対してのみ発行されます。このオプションは、Enterprise PL/I V3R3 以上で必要です。

Default(ASCII IEEE);
ASCII は、CHARACTER と PICTURE のデータを ASCII 形式 (JAVA での保持形式) で保持するように指定します。

IEEE は、FLOAT データを IEEE フォーマット (JAVA での保持形式) で保持するように指定します。

RENT
コードが静的変数に対して書き込みを行う場合でもコードを再入可能にします。

PL/I プロシージャー名とプロシージャー・ステートメントの正しい形式

PL/I プロシージャー名は、実行時に Java クラス・ローダーによって検出されるために、Java 命名規則に準拠している必要があります。Java 命名体系は 3 つの部分で構成されます。 最初の部分は Java 環境に対してルーチンを識別し、 2 番目の部分はネイティブ・メソッドを定義する Java クラスの名前、 3 番目の部分はネイティブ・メソッド自体の名前です。

次に、サンプル・プログラムにある PL/I プロシージャー名 `Java_callingPLI_callToPLI` について詳しく説明します。

Java
動的ライブラリー内にあるネイティブ・メソッドはすべて、`Java` を最初に指定する必要があります。

_callingPLI
ネイティブ・メソッドを宣言する Java クラスの名前。

_callToPLI
ネイティブ・メソッド自体の名前。

注: PL/I と C とでは、ネイティブ・メソッドのコーディングに重要な違いがあります。JDK に付属する **javah** ツールは、C プログラムに必要となる外部参照の形式を生成します。ネイティブ・メソッドを PL/I で書き、前述した PL/I 外部参照の命名規則に準拠する場合は、 PL/I ネイティブ・メソッドに対して **javah** ステップを実行する必要はありません。

また、PROCEDURE ステートメントの OPTIONS オプションに、以下のオプションを指定する必要があります。

- FromAlien
- NoDescriptor
- ByValue

サンプル・プログラムの完全なプロシージャー・ステートメントは次のとおりです。

```
Java_callingPLI_callToPLI:
Proc( JNIEnv , MyJObject )
  External( "Java_callingPLI_callToPLI" )
  Options( FromAlien NoDescriptor ByValue );
```

JNI インクルード・ファイル

Java ネイティブ・インターフェースの PL/I 定義を含む 2 つの PL/I 組み込みファイルは、`ibmzjni.inc` と、そこに含まれる `ibmzjnim.inc` です。これらのインクルード・ファイルは、次のステートメントとともに組み込まれています。

```
%include ibmzjni;
```

`ibmzjni` および `ibmzjnim` インクルード・ファイルは、PL/I SIBMZSAM データ・セットの中に提供されています。

完全な PL/I プロシージャー

最後にまとめとして、ネイティブ・メソッドを定義する PL/I プログラム全体を示します。

```
*Process Limits( Extname( 100 ) ) Margins( 1, 100 ) ;
*Process Display(Std) Dllinit Extrn(Short);
*Process Rent Default( ASCII IEEE );
PliJava_Demo: Package Exports(*);

 Java_callingPLI_callToPLI:
 Proc( JNIEnv , MyJObject )
   External( "Java_callingPLI_callToPLI" )
   Options( FromAlien NoDescriptor ByValue );

  %include ibmzjni;
  Dcl myJObject        Type jObject;

  Display('Hello from Enterprise PL/I!');

End;
```

ステップ 4: PL/I プログラムのコンパイルとリンク

手順

1. 次のコマンドを使用して、PL/I サンプル・プログラムをコンパイルします。

   ```
   pli -c hiFromPLI.pli
   ```

2. 次のコマンドを使用して、生成された PL/I オブジェクト・デックを共用ライブラリーにリンクします。

   ```
   c89 -o libhiFromPLI.so hiFromPLI.o
   ```

   PL/I 共用ライブラリーの名前には必ず `lib` 接頭部を付けてください。そうしないと、Java クラス・ローダーはそのライブラリーを検出できません。

ステップ 5: サンプル・プログラムの実行

手順

次のコマンドを使用して、Java - PL/I サンプル・プログラムを実行します。

```
java callingPLI
```

サンプル・プログラムの出力は以下のとおりです。

```
Hello from Enterprise PL/I!
And Hello from Java, too!
```

最初の行は PL/I ネイティブ・メソッドから書き込まれたものです。 2 行目は、PL/I ネイティブ・メソッド呼び出しから戻った後、呼び出し側の Java クラスから書き込まれたものです。

JNI サンプル・プログラム #2 - ストリングを渡す

このサンプル・プログラムは、Java と PL/I の間で双方向にストリングの受け渡しを行います。

jPassString.java プログラムの完全なリストについては、 436 ページの図 89 を参照してください。 Java 部分には、1 つの Java クラス jPassString.java があります。 PL/I で書かれたネイティブ・メソッドは、passString.pli に含まれています。 430 ページの『JNI サンプル・プログラム #1 -「Hello World」』に記載されている内容の多くが、このサンプル・プログラムにも当てはまります。以下のトピックでは、このサンプル・プログラムの新しい面または異なる面についてのみ説明します。

ステップ 1: Java プログラムの作成

手順

1. ネイティブ・メソッドを宣言します。

   ```
   public native void pliShowString();
   ```

2. ネイティブ・ライブラリーをロードします。

   ```
   static {
     System.loadLibrary("passString");
   }
   ```

3. Java main メソッドの作成

   jPassString クラスには、クラスのインスタンスを生成してネイティブ・メソッドを呼び出す main メソッドも含まれています。 main メソッドは jPassString のインスタンスを生成し、pliShowString() ネイティブ・メソッドを呼び出します。

   このサンプル・プログラムは、ストリングの入力をユーザーに促し、コマンド行からその値を読み込みます。 この作業は、 436 ページの図 89 に示す try/catch ステートメント内で行われます。

```
// Read a string, call PL/I, display new string upon return
import java.io.*;

public class jPassString{

 /* Field to hold Java string                          */
 String myString;

 /* Load the PL/I native library                       */
 static {
        System.loadLibrary("passString");
 }

 /* Declare the PL/I native method                     */
 public native void pliShowString();

 /* Main Java class                                    */
 public static void main(String[] arg) {

   System.out.println(" ");

   /* Instantiate Java class and initialize string     */
   jPassString myPassString = new jPassString();
   myPassString.myString = " ";

   /* Prompt user for a string                         */
   try {
        BufferedReader in = new BufferedReader(
          new InputStreamReader(System.in));

        /* Process until 'quit' received               */
        while (!myPassString.myString.equalsIgnoreCase("quit")) {
          System.out.println(
            "From Java: Enter a string or 'quit' to quit.");
          System.out.print("Java Prompt > ");
          /* Get string from command line              */
          myPassString.myString = in.readLine();
          if (!myPassString.myString.equalsIgnoreCase("quit"))
            {
             /* Call PL/I native method                */
             myPassString.pliShowString();
             /* Return from PL/I and display new string */
             System.out.println(" ");
             System.out.println(
                 "From Java: String set by PL/I is: "
               + myPassString.myString );
            }
        }
        }  catch (IOException e) {
        }
 }
}
```

図 *89. Java* サンプル・プログラム *#2* - ストリングの引き渡し

ステップ 2: Java プログラムのコンパイル

手順

Java コンパイラーを使用して、Java コードをコンパイルします。 以下のコマンドを使用できます。

```
javac jPassString.java
```

ステップ 3: PL/I プログラムの作成

432 ページの『ステップ 3: PL/I プログラムの作成』で説明されている PL/I の「Hello World」サンプル・プログラムを作成する方法に関するすべての情報は、このプログラムにも当てはまります。

PL/I プロシージャー名とプロシージャー・ステートメントの正しい形式

このプログラムの PL/I プロシージャー名は Java_jPassString_pliShowString です。

サンプル・プログラムの完全なプロシージャー・ステートメントは次のとおりです。

```
Java_jPassString_pliShowString:
Proc( JNIEnv , myjobject )
  external( "Java_jPassString_pliShowString" )
  Options( FromAlien NoDescriptor ByValue );
```

JNI インクルード・ファイル

Java ネイティブ・インターフェースの PL/I 定義を含む 2 つの PL/I 組み込みファイルは、ibmzjni.inc と、そこに含まれる ibmzjnim.inc です。これらの組み込みファイルは、次のステートメントを使用して組み込まれます。

```
 %include ibmzjni;
```

ibmzjni および ibmzjnim インクルード・ファイルは、PL/I SIBMZSAM データ・セットの中に提供されています。

完全な PL/I プロシージャー

完全な PL/I プログラムは、 439 ページの図 90 に示してあります。 このサンプル PL/I プログラムは、JNI を介していくつかの呼び出しを行います。

開始時に、呼び出し側 Java オブジェクト myObject への参照が PL/I プロシージャーに渡されます。PL/I プログラムはこの参照を使用して、呼び出し側からの情報を取得します。最初の情報は、**GetObjectClass** JNI 関数を使用して検索される呼び出し側オブジェクトのクラスです。このクラス値は、Java オブジェクト内の Java ストリング・フィールドの ID を取得するために、**GetFieldID** JNI 関数によって使用されます。この Java フィールドは、フィールド名 myString、および JNI フィールド記述子 Ljava/lang/String; (これによりフィールドは Java ストリング・フィールドとみなされる) によってさらに詳細に識別されます。その後、Java ストリング・フィールドの値が **GetObjectField** JNI 関数を使用して検索されます。PL/I

が Java ストリング値を使用するには、事前にこの値をアンパックして PL/I が解釈できる形式にする必要があります。**GetStringUTFChars** JNI 関数により Java ストリングが PL/I varyingz ストリングに変換されます。その後で、このストリングが PL/I プログラムによって表示されます。

取得した Java ストリングを表示した後、PL/I プログラムは、呼び出し側 Java オブジェクト内のストリング・フィールドの更新に使用する PL/I ストリングの入力をユーザーに促します。PL/I ストリングの値は、**NewString** JNI 関数を使用して Java ストリングに変換されます。この新しい Java ストリングを使用して、**SetObjectField** JNI 関数によって呼び出し側 Java オブジェクト内のストリング・フィールドが更新されます。

PL/I プログラムが終了すると Java に制御が戻され、新しく更新された Java ストリングが Java プログラムによって表示されます。

```
*Process Limits( Extname( 100 ) ) Margins( 1, 100 ) ;
*Process Display(Std) Dllinit Extrn(Short);
*Process Rent Default( ASCII IEEE );
plijava_demo: package exports(*);

 Java_passString_pliShowString:
 Proc( JNIEnv , myJObject )
   external( "Java_jPassString_pliShowString" )
   Options( FromAlien NoDescriptor ByValue );

  %include ibmzjni;

  Dcl myBool           Type jBoolean;
  Dcl myClazz          Type jClass;
  Dcl myFID            Type jFieldID;
  Dcl myJObject        Type jObject;
  Dcl myJString        Type jString;
  Dcl newJString       Type jString;
  Dcl myID             Char(9)   Varz static init( 'myString' );
  Dcl mySig            Char(18)  Varz static
                       init( 'Ljava/lang/String;' );
  Dcl pliStr           Char(132) Varz Based(pliStrPtr);
  Dcl pliReply         Char(132) Varz;
  Dcl pliStrPtr        Pointer;
  Dcl nullPtr          Pointer;

  Display('  ');

  /* Get information about the calling Class             */
  myClazz = GetObjectClass(JNIEnv, myJObject);

  /* Get Field ID for String field from Java             */
  myFID = GetFieldID(JNIEnv, myClazz, myID, mySig );

  /* Get the Java String in the string field             */
  myJString = GetObjectField(JNIEnv, myJObject, myFID );

  /* Convert the Java String to a PL/I string            */
  pliStrPtr = GetStringUTFChars(JNIEnv, myJString, myBool );

  Display('From  PLI: String retrieved from Java is: ' || pliStr );
  Display('From  PLI: Enter a string to be returned to Java:' )
         reply(pliReply);

  /* Convert the new PL/I string to a Java String        */
  newJString = NewString(JNIEnv, trim(pliReply), length(pliReply) );

  /* Change the Java String field to the new string value */
  nullPtr = SetObjectField(JNIEnv, myJObject, myFID, newJString);

 End;

end;
```

図 *90. PL/I* サンプル・プログラム *#2* - ストリングの引き渡し

ステップ 4: PL/I プログラムのコンパイルとリンク

手順

1. 次のコマンドを使用して、PL/I サンプル・プログラムをコンパイルします。

   ```
   pli -c passString.pli
   ```

2. 次のコマンドを使用して、生成された PL/I オブジェクト・デックを共用ライブラリーにリンクします。

   ```
   c89 -o libpassString.so passString.o
   ```

   名前には必ず lib 接頭部を付けてください。そうしないと、PL/I 共用ライブラリーも Java クラス・ローダーも名前を検出できません。

ステップ 5: サンプル・プログラムの実行

手順

次のコマンドを使用して、Java - PL/I サンプル・プログラムを実行します。

```
java jPassString
```

サンプル・プログラムの出力 (Java と PL/I の両方からユーザー入力を求める要求を含む) は次のようになります。

```
>java jPassString

From Java: Enter a string or 'quit' to quit.
Java Prompt > A string entered in Java

From  PLI: String retrieved from Java is: A string entered in Java
From  PLI: Enter a string to be returned to Java:
A string entered in PL/I

From Java: String set by PL/I is: A string entered in PL/I
From Java: Enter a string or 'quit' to quit.
Java Prompt > quit
>
```

JNI サンプル・プログラム #3 - 整数の引き渡し

このサンプル・プログラムは、Java と PL/I の間で双方向に整数の受け渡しを行います。

jPassInt.java プログラムの完全なリストについては、 442 ページの図 91 を参照してください。 Java 部分には、1 つの Java クラス jPassInt.java があります。PL/I で書かれたネイティブ・メソッドは、passInt.pli に含まれています。 430 ページの『JNI サンプル・プログラム #1 -「Hello World」』に記載されている内容の多くが、このサンプル・プログラムにも当てはまります。以下のトピックでは、このサンプル・プログラムの新しい面または異なる面についてのみ説明します。

ステップ 1: Java プログラムの作成

手順

1. ネイティブ・メソッドを宣言します。

   ```
   public native void pliShowInt();
   ```

2. ネイティブ・ライブラリーをロードします。

   ```
   static {
     System.loadLibrary("passInt");
   }
   ```

3. Java main メソッドの作成

   jPassInt クラスには、クラスのインスタンスを生成してネイティブ・メソッドを呼び出す main メソッドも含まれています。main メソッドは jPassInt のインスタンスを生成し、pliShowInt() ネイティブ・メソッドを呼び出します。

   このサンプル・プログラムは、整数の入力をユーザーに促し、コマンド行からその値を読み込みます。この作業は、 442 ページの図 91 に示す try/catch ステートメント内で行われます。

```
// Read an integer, call PL/I, display new integer upon return
import java.io.*;
import java.lang.*;

public class jPassInt{

 /* Fields to hold Java string and int                  */
 int myInt;
 String myString;

 /* Load the PL/I native library                        */
 static {
         System.loadLibrary("passInt");
 }

 /* Declare the PL/I native method                      */
 public native void pliShowInt();

 /* Main Java class                                     */
 public static void main(String[] arg) {

   System.out.println(" ");

   /* Instantiate Java class and initialize string      */
   jPassInt pInt = new jPassInt();
   pInt.myInt = 1024;
   pInt.myString = " ";

   /* Prompt user for an integer                        */
   try {
        BufferedReader in = new BufferedReader(
          new InputStreamReader(System.in));

        /* Process until 'quit' received                */
        while (!pInt.myString.equalsIgnoreCase("quit")) {
          System.out.println
            ("From Java: Enter an Integer or 'quit' to quit.");
          System.out.print("Java Prompt > ");
          /* Get string from command line               */
          pInt.myString = in.readLine();
          if (!pInt.myString.equalsIgnoreCase("quit"))
            {
             /* Set int to integer value of String      */
             pInt.myInt = Integer.parseInt( pInt.myString );
             /* Call PL/I native method                 */
             pInt.pliShowInt();
             /* Return from PL/I and display new string */
             System.out.println(" ");
             System.out.println
               ("From Java: Integer set by PL/I is: " + pInt.myInt );
            }
        }
        }  catch (IOException e) {
        }
 }

}
```

図 *91. Java* サンプル・プログラム *#3* - 整数の引き渡し

ステップ 2: Java プログラムのコンパイル

手順

Java コンパイラーを使用して、Java コードをコンパイルします。 以下のコマンドを使用できます。

```
javac jPassInt.java
```

ステップ 3: PL/I プログラムの作成

432 ページの『ステップ 3: PL/I プログラムの作成』で説明されている PL/I の「Hello World」サンプル・プログラムを作成する方法に関するすべての情報は、このプログラムにも当てはまります。

PL/I プロシージャー名とプロシージャー・ステートメントの正しい形式

このプログラムの PL/I プロシージャー名は `Java_jPassInt_pliShowInt` です。

サンプル・プログラムの完全なプロシージャー・ステートメントは次のとおりです。

```
Java_passNum_pliShowInt:
Proc( JNIEnv , myjobject )
  external( "Java_jPassInt_pliShowInt" )
  Options( FromAlien NoDescriptor ByValue );
```

JNI インクルード・ファイル

Java ネイティブ・インターフェースの PL/I 定義を含む 2 つの PL/I 組み込みファイルは、`ibmzjni.inc` と、そこに含まれる `ibmzjnim.inc` です。これらのインクルード・ファイルは、次のステートメントとともに組み込まれています。

```
%include ibmzjni;
```

`ibmzjni` および `ibmzjnim` インクルード・ファイルは、PL/I SIBMZSAM データ・セットの中に提供されています。

完全な PL/I プロシージャー

完全な PL/I プログラムは、 444 ページの図 92 に示してあります。 このサンプル PL/I プログラムは、JNI を介していくつかの呼び出しを行います。

開始時に、呼び出し側 Java オブジェクト `myObject` への参照が PL/I プロシージャーに渡されます。 PL/I プログラムはこの参照を使用して、呼び出し側からの情報を取得します。最初の情報は、**GetObjectClass** JNI 関数を使用して検索される呼び出し側オブジェクトのクラスです。このクラス値は、Java オブジェクト内の Java 整数フィールドの ID を取得するために、**GetFieldID** JNI 関数によって使用されます。この Java フィールドは、フィールド名 `myInt` および JNI フィールド記述子 I (これによりフィールドは整数フィールドとみなされる) によってさらに詳細に識別されます。その後、Java 整数フィールドの値が **GetIntField** JNI 関数を使用して検索され、PL/I プログラムによって表示されます。

取得した Java 整数を表示した後、PL/I プログラムは、呼び出し側 Java オブジェクト内の整数フィールドの更新に使用する PL/I 整数の入力をユーザーに促します。この PL/I 整数値を使用して、**SetIntField** JNI 関数によって呼び出し側 Java オブジェクトの整数フィールドが更新されます。

PL/I プログラムが終了すると Java に制御が戻され、新しく更新された Java 整数が Java プログラムによって表示されます。

```
*Process Limits( Extname( 100 ) ) Margins( 1, 100 ) ;
*Process Display(Std) Dllinit Extrn(Short);
*Process Rent Default( ASCII IEEE );
plijava_demo: package exports(*);

 Java_passNum_pliShowInt:
 Proc( JNIEnv , myjobject )
   external( "Java_jPassInt_pliShowInt" )
   Options( FromAlien NoDescriptor ByValue );

  %include ibmzjni;

  Dcl myClazz          Type jClass;
  Dcl myFID            Type jFieldID;
  Dcl myJInt           Type jInt;
  dcl rtnJInt          Type jInt;
  Dcl myJObject        Type jObject;
  Dcl pliReply         Char(132) Varz;
  Dcl nullPtr          Pointer;

  Display('  ');

  /* Get information about the calling Class             */
  myClazz = GetObjectClass(JNIEnv, myJObject);

  /* Get Field ID for int field from Java                */
  myFID = GetFieldID(JNIEnv, myClazz, "myInt", "I");

  /* Get Integer value from Java                         */
  myJInt = GetIntField(JNIEnv, myJObject, myFID);

  display('From  PLI: Integer retrieved from Java is: ' || trim(myJInt) );
  display('From  PLI: Enter an integer to be returned to Java:' )
         reply(pliReply);

  rtnJInt = pliReply;

  /* Set Integer value in Java from PL/I                 */
  nullPtr = SetIntField(JNIEnv, myJObject, myFID, rtnJInt);

 End;

end;
```

図 *92. PL/I* サンプル・プログラム *#3* - 整数の引き渡し

ステップ 4: PL/I プログラムのコンパイルとリンク

手順

1. 次のコマンドを使用して、PL/I サンプル・プログラムをコンパイルします。

   ```
   pli -c passInt.pli
   ```

2. 次のコマンドを使用して、生成された PL/I オブジェクト・デックを共用ライブラリーにリンクします。

   ```
   c89 -o libpassInt.so passInt.o
   ```

   名前には必ず lib 接頭部を付けてください。そうしないと、PL/I 共用ライブラリーも Java クラス・ローダーも名前を検出できません。

ステップ 5: サンプル・プログラムの実行

手順

次のコマンドを使用して、Java - PL/I サンプル・プログラムを実行します。

```
java jPassInt
```

サンプル・プログラムの出力 (Java と PL/I の両方からユーザー入力を求める要求を含む) は次のようになります。

```
>java jPassInt

From Java: Enter an Integer or 'quit' to quit.
Java Prompt > 12345

From  PLI: Integer retrieved from Java is: 12345
From  PLI: Enter an integer to be returned to Java:
54321

From Java: Integer set by PL/I is: 54321
From Java: Enter an Integer or 'quit' to quit.
Java Prompt > quit
>
```

JNI サンプル・プログラム #4 - Java 呼び出し API

このサンプル・プログラムは、今までのサンプルと若干異なります。このサンプルでは、最初に PL/I が Java 呼び出し API を使用して Java を呼び出して、組み込み Java 仮想マシン (JVM) を作成します。次に PL/I が Java メソッドを呼び出します。その後で、PL/I から Java メソッドに渡されたストリングが表示されます。

PL/I サンプル・プログラムは createJVM.pli という名前で、このサンプル・プログラムによって呼び出される Java メソッドは javaPart.java に含まれています。

ステップ 1: Java プログラムの作成

このタスクについて

このサンプルは PL/I ネイティブ・メソッドを使用しないため、それを宣言する必要がありません。その代わり、このサンプルの Java 部分には単純な Java メソッドが 1 つ含まれています。

手順

Java main メソッドの作成
javaPart クラスには、ステートメントが 1 つだけ含まれます。 このステートメントは、Java から「Hello World...」という短いテキストを出力し、PL/I プログラムから Java に渡されたストリングを追加します。クラス全体は、図 93 に示されています。

```
// Receive a string from PL/I then display it after saying "Hello"
public class javaPart {
   public static void main(String[] args) {
      System.out.println("From Java  -  Hello World... " + args[0]);
   }
}
```

図 *93. Java* サンプル・プログラム *#4* - ストリングの受け取りおよび出力

ステップ 2: Java プログラムのコンパイル

手順

Java コンパイラーを使用して、Java コードをコンパイルします。 以下のコマンドを使用できます。

```
javac javaPart.java
```

ステップ 3: PL/I プログラムの作成

432 ページの『ステップ 3: PL/I プログラムの作成』で説明されている PL/I の「Hello World」サンプル・プログラムを作成する方法に関する多くの情報は、このプログラムにも当てはまります。しかし、このサンプルでは PL/I は Java を呼び出すため、考慮する必要のある点が他にもあります。

PL/I プロシージャー名とプロシージャー・ステートメントの正しい形式

このサンプルでは PL/I プログラムが Java を呼び出すため、PL/I プログラムは MAIN になります。この PL/I プログラムの外部名は参照されないため、考慮する必要はありません。

サンプル・プログラムの完全なプロシージャー・ステートメントは次のとおりです。

```
createJVM: Proc Options(Main);
```

JNI インクルード・ファイル

Java ネイティブ・インターフェースの PL/I 定義を含む 2 つの PL/I 組み込みファイルは、ibmzjni.inc と、そこに含まれる ibmzjnim.inc です。このサンプルで

は PL/I が Java を呼び出しているとしても、これらの組み込みファイルは必要です。これらのインクルード・ファイルは、次のステートメントとともに組み込まれています。

```
%include ibmzjni;
```

ibmzjni および ibmzjnim インクルード・ファイルは、PL/I SIBMZSAM データ・セットの中に提供されています。

PL/I プログラムと Java ライブラリーとのリンク

この PL/I サンプル・プログラムは Java を呼び出すため、Java ライブラリーにリンクする必要があります。 Java ライブラリーは XPLINK とリンクしていますが、PL/I モジュールはリンクしていません。PL/I は現在でも、XPLINK ライブラリーとリンクしてそれを呼び出すことができますが、PLIXOPT 変数を使用して、**XPLINK=ON** ランタイム・オプションを指定する必要があります。PLIXOPT 変数は次のように宣言できます。

```
Dcl PLIXOPT     Char(40) Varying Ext Static Init( 'XPLINK(ON)'e );
```

PLIXOPT については、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。

Java 呼び出し API の使用

この PL/I サンプル・プログラムは、Java 呼び出し API **JNI_CreateJavaVM** を呼び出して、自身の組み込み JVM を作成します。この API には、 449 ページの図 94 に示されているような、セットアップおよび初期化を正しく行うための特定の構造体が必要です。

1. **JNI_GetDefaultJavaVMInitArgs** が、デフォルトの初期化オプションを取得するために呼び出されます。
2. これらのデフォルト・オプションが、java.class.path 情報の追加により変更されます。
3. **JNI_CreateJavaVM** が組み込み JVM を作成するために呼び出されます。

完全な PL/I プログラム

完全な PL/I プログラムは、 449 ページの図 94 に示してあります。 このサンプル PL/I プログラムは、JNI を介していくつかの呼び出しを行います。

このサンプルでは、Java オブジェクト (この場合は新しく作成された JVM) への参照は渡されませんが、代わりに **JNI_CreateJavaVM** API への呼び出しから戻されます。PL/I プログラムはこの参照を使用して、JVM から情報を取得します。最初の情報部分は、呼び出される Java メソッドが含まれたクラスです。このクラスは、**FindClass** JNI 関数を使用して検出されます。次に、クラス値が **GetStaticMethodID** JNI 関数によって使用され、呼び出される Java メソッドの ID が取得されます。

この Java メソッドを呼び出す前に、Java で認識されるフォーマットに PL/I ストリングを変換します。 PL/I プログラムでは、ストリングは ASCII フォーマットで保持されます。 Java ストリングは UTF フォーマットで保管されます。 さらに、Java ストリングは、PL/I プログラマーが認識するような意味では真にストリ

ングではありませんが、それ自体 Java クラスであり、メソッドを通じてのみ変更できます。**NewStringUTF** JNI 関数を使用して Java ストリングを作成します。この関数は、UTF に変換された PL/I ストリングを含む myJString という Java オブジェクトを戻します。次に、**myJString** オブジェクトに対する参照を渡して NewObjectArray JNI 関数を呼び出すことで、Java オブジェクト配列を作成します。この関数は、Java メソッドで表示されるストリングを含む Java オブジェクト配列に対する参照を返します。

これで、**CallStaticVoidMethod** JNI 関数で Java メソッドを呼び出せるようになりました。また、このメソッドにより、その関数に渡されるストリングが表示されるようになります。ストリングを表示した後、PL/I プログラムは、**DestroyJavaVM** JNI 関数を使用して組み込み JVM を破棄し、PL/I プログラムが完了します。

449 ページの図 94 に、PL/I プログラムの完全なソースを示します。

```
*Process Limits( Extname( 100 ) )  Margins( 1, 100 );
*Process Margins( 1, 100 ) ;
*Process Display(STD) Rent;
*Process Default( ASCII ) Or('|');
createJVM: Proc Options(Main);

  %include ibmzjni;

  Dcl myJObjArray Type jobjectArray;
  Dcl myClass     Type jclass;
  Dcl myMethodID  Type jmethodID;
  Dcl myJString   Type jstring;
  Dcl myRC        Fixed Bin(31)  Init(0);
  Dcl myPLIStr    Char(50) Varz
                  Init('... a PLI string in a Java Virtual Machine!');
  Dcl OptStr1 char(1024) varz;
  Dcl OptStr2 char(1024) varz;
  Dcl myNull      Pointer;
  Dcl VM_Args     Like JavaVMInitArgs;
  Dcl myOptions   Like JavaVMOption;
  Dcl PLIXOPT     Char(40) Varying Ext Static Init( 'XPLINK(ON)'e );

  Display('From PL/I  - Beginning execution of createJVM...');
  VM_Args.version = JNI_VERSION_1_6;

  myRC =  JNI_GetDefaultJavaVMInitArgs( addr(VM_Args) );
  OptStr1 = "-Djava.class.path=.:";
  OptStr2 = "-Djava.compiler=NONE";

  myOptions(1).theOptions = addr(OptStr1);
  myOptions(2).theOptions = addr(OptStr2);
  VM_Args.nOptions = 2;
  VM_Args.JavaVMOption = addr(myOptions);

  /* Create the Java VM                                        */
  myrc = JNI_CreateJavaVM(
             addr(jvm_ptr),
             addr(JNIEnv),
             addr(VM_Args) );

  /* Get the Java Class for the javaPart class                 */
  myClass = FindClass(JNIEnv, "javaPart");
  /* Get static method ID                                      */
  myMethodID = GetStaticMethodID(JNIEnv, myClass,"main",
               "([Ljava/lang/String;)V" );
  /* Create a Java String Object from the PL/I string.         */
  myJString = NewStringUTF(JNIenv, myPLIStr);
  myJObjArray = NewObjectArray(JNIEnv, 1,
                FindClass(JNIEnv,"java/lang/String"), myJString);
  Display('From PL/I  - Calling Java method in new JVM from PL/I...');
  Display('  ');
  myNull = CallStaticVoidMethod(JNIEnv, myClass,
             myMethodID, myJObjArray );
  /* destroy the Java VM                                       */
  Display('  ');
  Display('From PL/I  - Destroying the new JVM from PL/I...');
  myRC = DestroyJavaVM( JavaVM );
end;
```

図 *94. PL/I* サンプル・プログラム *#4 - Java* 呼び出し *API* の呼び出し

ステップ 4: PL/I プログラムのコンパイルとリンク

手順

1. 次のコマンドを使用して、PL/I サンプル・プログラムをコンパイルします。

   ```
   pli -c createJVM.pli
   ```

2. 次のコマンドを使用して、生成された PL/I オブジェクト・デックを共用ライブラリーにリンクします。

   ```
   c89 -o createJVM createJVM.o $JAVA_HOME/bin/classic/libjvm.x
   ```

   $JAVA_HOME 環境変数への参照に注意してください。この変数は、ご使用の Java 1.4 製品がインストールされているディレクトリーを指していなければなりません。例えば、ご使用の環境でこの変数をセットアップするには、次のコマンドを使用します。

   ```
   export JAVA_HOME="/usr/lpp/java/J6.0"
   ```

   このケースでは、Java 1.4 製品が `/usr/lpp/java/J6.0` にインストールされていることが前提となっています。

ステップ 5: サンプル・プログラムの実行

手順

次のコマンドを使用して、Java - PL/I サンプル・プログラムを実行します。

```
createJVM
```

サンプル・プログラムの出力は以下のとおりです。

```
From PL/I  - Beginning execution of createJVM...
From PL/I  - Calling Java method in new JVM from PL/I...
From Java  - Hello World... ... a PLI string in a Java Virtual Machine!
From PL/I  - Destroying the new JVM from PL/I...
```

既存の Java VM へのプログラムの接続

Java 呼び出し API を使用すれば、ご使用のプログラムを、そのプログラムによって作成されたものではない既存の Java VM に接続できます。

このタスクについて

PL/I アプリケーションが IMS JMP (Java メッセージ処理) 領域内で実行されていれば、Java VM は IMS によって既に作成されています。以下の手順を使用すれば、ご使用のプログラムを Java VM に接続できます。 その後で、Java Native Interface を使用して、必要な関数を呼び出すことができます。

手順

1. **JNI_GetCreatedJavaVMs** API を使用して、プログラムの接続先となる Java VM インスタンスを見つけます。 この IMS 環境では、作成される Java VM が 1 つのみであるため、Java VM ポインターが 1 つのみ返されるように要求します。 この呼び出しは次のようにコーディングできます。

   ```
   rc = JNI_GetCreatedJavaVMs( jvm_ptr, 1, nVMs );
   ```

jvm_ptr
Java VM へのポインターです。IBMZJNI インクルード・ファイル内で宣言されます。 呼び出しが正常終了して戻ってくると、この変数内の Java VM のアドレスが Java から返されます。

nVMs 固定の bin(31) 変数です。Java が戻ってくると、Java によって Java VM アドレスの数でこの変数が更新されます。 呼び出しが成功終了した場合、*nVMs* は 1 です。

2. Java VM 用の JNI 環境ポインターを取得します。 IBMZJNI インクルード・ファイル内で宣言された **JNIInvokeInterface_ structure** 内で **JGetEnv** 関数を使用できます。

```
rc = JGetEnv( jvm_ptr, JNIEnv, JNI_VERSION_1_6 );
```

jvm_ptr
Java VM インスタンスへのポインターです。

JNIEnv
Java VM インスタンスの JNI 環境ポインターに対して Java によって設定されるポインターです。

JNI_VERSION_1_6
インターフェース・バージョン番号の値 (これも IBMZJNI インクルード・ファイル内で宣言されます) を保持する定数です。

タスクの結果

これで、Java VM の JNI 環境ポインターが設定されたため、Java Native Interface を使用して任意の JNI 関数を呼び出すことができます。

Java および PL/I の同等なデータ・タイプの判別

PL/I から Java とのやり取りを行う際には、2 つのプログラム言語間でデータ・タイプを一致させる必要があります。

次の表に、Java の基本的なタイプと PL/I の同等なタイプを示します。

表 *36. Java* 基本タイプと、同等の *PL/I* ネイティブ・タイプ

Java のタイプ	**PL/I** タイプ	サイズ **(**ビット**)**
Boolean	jboolean	8、符号なし
byte	jbyte	8
char	jchar	16、符号なし
short	jshort	16
int	jint	32
long	jlong	64
float	jfloat	21
double	jdouble	53
void	jvoid	n/a

第 5 部 特殊プログラミング・タスク

© Copyright IBM Corp. 1999, 2017

第 19 章 PLISAXA および PLISAXB XML パーサーの使用

PLISAXx (x = A または B) 組み込みサブルーチンは、プログラムがインバウンド XML 文書を取り込んで、適格性を検査して、その内容を処理できるようにする、基本的な XML 構文解析機能を提供します。

これらのサブルーチンでは XML の生成は提供しておらず、代わりに、PL/I プログラム・ロジックによって行うか、または XMLCHAR 組み込み関数を使用する必要があります。

PLISAXA および PLISAXB には、特別な環境要件はありません。
CICS、IMS、MQ Series、z/OS バッチ、および TSO を含め、主要なすべてのランタイム環境で実行できます。

PLISAXA および PLISAXB には、重要な制限がいくつかあります。

- XML 名前空間はサポートされていません。
- ユニコード UTF-8 文書はサポートされていません。
- XML 文書の構文解析前に、その文書全体を (バッファーまたはファイルで) 渡す必要があります。

これらの制限は PLISAXC 組み込みサブルーチンおよび PLISAXD 組み込みサブルーチンにはありません。

関連情報:

485 ページの『第 20 章 PLISAXC および PLISAXD XML パーサーの使用』
PLISAXC および PLISAXD 組み込みサブルーチンは、基本的な XML 構文解析機能を提供します。 これによりプログラムは、インバウンド XML 文書を取り込み、適格性を検査し、その内容を処理できます。

概説

XML 構文解析用のインターフェースには、大きく分けてイベント・ベースとツリー・ベースの 2 種類があります。

イベント・ベース API の場合、パーサーはコールバックによってアプリケーションにイベントを報告します。 このようなイベントには、文書の開始、エレメントの開始などがあります。 アプリケーションは、パーサーから報告されたイベントを処理するためのハンドラーを備えています。 Simple API for XML (SAX) は、業界標準のイベント・ベース API の一例です。

ツリー・ベース API (文書オブジェクト・モデル (DOM) など) の場合、パーサーは XML をツリー・ベースの内部表現に変換します。ツリーをナビゲートするためのインターフェースが提供されています。

© Copyright IBM Corp. 1999, 2017

IBM PL/I には、XML 文書の構文解析用に SAX のようなイベント・ベースのインターフェースが備わっています。パーサーは、対応する文書フラグメントへの参照を渡して、アプリケーション提供のパーサー・イベント用のハンドラーを呼び出します。

パーサーには次の特性があります。

- 高性能であるが非標準のインターフェースを提供します。
- ユニコード UTF-16、または 463 ページの『XML 文書のコード化文字セット』で示されたいずれかの 1 バイト・コード・ページでエンコードされた XML ファイルをサポートします。
- このパーサーは妥当性検査を行いませんが、整形式であるかどうかを部分的に検査し、何かを検出すると例外イベントを生成します。

XML 文書の準拠レベルには適格性と有効性の 2 つがあり、どちらのレベルも XML 標準に定義されています。 XML 標準は、http://www.w3c.org/XML/ に掲載されています。 これらの定義を要約すると、XML 文書が基本的な XML 文法と、いくつかの特定の規則 (開始エレメントと終了エレメントのタグが一致していることなどの要件) に準拠していれば、 XML 文書は整形式です。 さらに、整形式 XML 文書に文書タイプ宣言 (DTD) が関連していて、文書が DTD に表された制約に準拠している場合、その文書は有効です。

それぞれのパーサー・イベントごとに、 458 ページの図 95 のコード例に示すように、適切なパラメーターを受け入れて適切な戻り値を戻す PL/I 関数を用意する必要があります。特に、戻り値は値によって (BYVALUE) 返されなければならないことに注意してください。また、これらの関数は、すべて OPTLINK リンケージを使用する必要があります。 DEFAULT(LINKAGE(OPTLINK)) オプションを使用してこのリンケージを指定するか、または OPTIONS(LINKAGE(OPTLINK)) 属性を使用して、個々の PROCEDURE および ENTRY でこのリンケージを指定することができます。

PLISAXA 組み込みサブルーチン

PLISAXA 組み込みサブルーチンを使用すれば、プログラムのバッファーにある XML 文書に対して XML パーサーを呼び出すことができます。

e イベント構造体

p パーサーがイベント関数に戻すポインター値または「トークン」

x 入力 XML が入っているバッファーのアドレス

n そのバッファーにあるデータのバイト数

c その XML のコード・ページの名称を指定する数値表現

注:

- XML が CHARACTER VARYING ストリングまたは WIDECHAR VARYING ストリングに含まれている場合は、ADDRDATA 組み込み関数を使用して最初のデータ・バイトのアドレスを取得する必要があります。
- XML が WIDECHAR ストリングに含まれている場合、バイト数の値は LENGTH 組み込み関数から返される値の 2 倍になります。

PLISAXB 組み込みサブルーチン

PLISAXB 組み込みサブルーチンを使用すれば、ファイルに記載されている XML 文書に対して XML パーサーを呼び出すことができます。

e イベント構造体

p パーサーがイベント関数に戻すポインター値または「トークン」

x 入力ファイルを指定する文字ストリング式

c その XML のコード・ページの名称を指定する数値表現

バッチのもとでは、入力ファイルを指定する文字ストリングは `file://dd:`*ddname* のフォームであり、*ddname* は、そのファイルを指定している DD ステートメントの名前です。

z/OS UNIX では、入力ファイルを指定する文字ストリングは `file://`*filename* の形式になっていなければなりません。 *filename* は z/OS UNIX ファイルの名前です。

バッチと z/OS UNIX の両方の環境とも、入力ファイルを指定する文字ストリングには、先行ブランクも末尾ブランクも含めません。

入力 XML ファイルのサイズは、2G 未満でなくてはなりません。 また、パーサーはファイル全体をメモリーに読み取るため、ご使用のプログラムの REGION は、文書すべてを含めることができるストレージの部分をパーサーが取得できるほどに大きくなければなりません。

SAX イベント構造体

イベント構造体は、24 個の LIMITED ENTRY 変数で構成される構造体です。これらの変数は、さまざまな「イベント」に対してパーサーが起動する機能を指しています。

これらすべての ENTRY は、OPTLINK リンケージを使用する必要があります。

XML パーサーをサポートするライブラリー・ルーチンは、いずれかの PLISAX イベント構造体におけるイベントが (NULLENTRY 組み込み関数または UNSPEC を使用して) ヌルに設定されているかどうかを認識し、そのイベントがヌルに設定されている場合は、そのイベントを呼び出しません。これにより、関心のあるイベントのみに XML 構文解析コードを制限することができ、同時に構文解析全体のパフ

ォーマンスを向上させることができます。

```
xmlDocument =
   '<?xml version="1.0" standalone="yes"?>'
|| '<!--This document is just an example-->'
|| '<sandwich>'
|| '<bread type="baker&quot;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';
```

図 *95.* サンプル *XML* 文書

この構造体での出現順に、パーサーは次のイベントを認識することができます。

注**:** 各イベントの説明は、図 95 の XML 文書の例に対応しています。この説明にある *XML* テキスト という用語は、イベントに渡されるポインターと長さに基づくストリングを意味しています。

start_of_document

このイベントは、文書の構文解析が開始されるときに 1 回発生します。パーサーは、LF (改行) や NL (改行) などの行制御文字を含む、文書全体のアドレスと長さを渡します。 図 95 で示されている例の場合、文書の長さは 305 文字です。

version_information

このイベントは、オプショナルの XML 宣言内のバージョン情報に対して発生します。 パーサーは、バージョン値 (図 95 に示されている例では 1.0) を含むテキストのアドレスと長さを渡します。

encoding_declaration

このイベントは、XML 宣言内でオプショナルのエンコード宣言に対して発生します。 パーサーは、エンコード方式値が入ったテキストのアドレスと長さを渡します。

standalone_declaration

このイベントは、XML 宣言内でオプショナルのスタンドアロン宣言に対して発生します。 パーサーは、スタンドアロン値 (図 95 に示されている例では yes) を含むテキストのアドレスと長さを渡します。

document_type_declaration

このイベントは、パーサーが文書タイプ宣言を検出したときに発生します。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。その間には、かなり複雑な文法規則で記述された内容が含まれます。パーサーは、開始と終了の文字シーケンスを含む宣言全体が含まれるテキストのアドレスと長さを渡します。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。 図 95 に示されている例には、文書タイプ宣言が含まれていません。

end_of_document

このイベントは、文書の構文解析が完了したときに 1 回発生します。

start_of_element

このイベントは、エレメント開始タグ、または空エレメント・タグごとに 1 回発生します。 パーサーは、エレメント名が入ったテキストのアドレスと長さを渡します。 458 ページの図 95 に示されている例において構文解析中に最初に発生する start_of_element イベントの場合、このストリングは `sandwich` です。

attribute_name

このイベントは、有効な名前をパーサーが認識した後で、エレメント開始タグまたは空エレメント・タグにおける属性ごとに発生します。パーサーは、属性名が入ったテキストのアドレスと長さを渡します。 458 ページの図 95 に示されている例における唯一の属性名は `type` です。

attribute_characters

このイベントは、属性値のフラグメントごとに発生します。 パーサーは、フラグメントが入ったテキストのアドレスと長さを渡します。属性値は通常、次のように複数の行に分割されている場合でもただ 1 つのストリングで構成されます。

```
<element attribute="This attribute value is
split across two lines"/>
```

ただし、属性値は複数の部分で構成されている場合があります。 例えば、 458 ページの図 95 に示されている例にある `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ています。パーサーは、これらのフラグメントを 3 つの別々のイベントとして渡します。各ストリング (`baker` および `s best`) は attribute_characters イベントとして渡され、単一文字 `'` は attribute_predefined_reference イベントとして渡されます。

関連情報:

『attribute_predefined_reference』

attribute_predefined_reference

このイベントは、5 つの定義済みエンティティー参照 `&`、`'`、`>`、`<`、および `"` について属性値において発生します。パーサーは、`&`、`'`、`>`、`<`、または `"` のいずれか 1 つがそれぞれに含まれる CHAR(1) 値または WIDECHAR(1) 値を渡します。

attribute_character_reference

このイベントは、`&#dd;` または `&#xhh;` という形式の数字参照 (ユニコード・コード・ポイントまたは「スカラー値」) について、属性値において発生します。 *d* は 10 進数字を表します。 *h* は 16 進数字を表します。パーサーは、対応する整数値が入った FIXED BIN(31) 値を渡します。

end_of_element

このイベントは、エレメント終了タグ、または空エレメント・タグごとに、パーサーがタグの終了不等号括弧を認識したときに 1 回発生します。パーサーは、エレメント名が入ったテキストのアドレスと長さを渡します。

start_of_CDATA_section

このイベントは、CDATA セクションが開始されると発生します。 CDATA セクションはストリング `<![CDATA[` で始まりストリング `]]>` で終わります。 CDATA セクションは、CDATA セクションでなければ XML マークアップとして認識される文字が含まれているテキストのブロックを「エスケープ」するために使用されます。パーサーは、開始文字 `<![CDATA[` が含まれているテキストのアドレスと長さを渡します。パーサーは、これらの区切り文字間にある 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。 458 ページの図 95 で示されている例に関しては、content-characters イベントとしてテキスト `We should add a <relish> element in future!` が渡されています。

end_of_CDATA_section

このイベントは、パーサーが CDATA セクションの終了を認識したときに発生します。 パーサーは、終了文字シーケンス `]]>` を含むテキストのアドレスと長さを渡します。

content_characters

このイベントは、XML 文書の「本体」である、エレメントの開始タグと終了タグの間にある文字データを表します。 パーサーは、データを含むテキストのアドレスおよび長さを渡します。データは通常、次のように複数の行に分割されている場合でも、単一のストリングのみで構成されます。

```
<element1>This character content is
split across two lines</element1>
```

エレメント内容に参照や他のエレメントが含まれている場合、内容全体が複数のセグメントで構成されることがあります。例えば、 458 ページの図 95 に示されている例にある `meat` エレメントの内容は、ストリング `Ham`、文字 `&`、およびストリング `turkey` で構成されています。これら 2 つのストリング・フラグメントのそれぞれ先頭と末尾にあるスペースに注意してください。パーサーは、これら 3 つの内容フラグメントを別々のイベントとして渡します。ストリングの内容フラグメント (`Ham` および `turkey`) は content_characters イベントとして渡され、単一文字 `&` は content_predefined_reference イベントとして渡されます。またパーサーは、content_characters イベントを使用して CDATA セクションのテキストをアプリケーションに渡します。

content_predefined_reference

このイベントは、5 つの定義済みエンティティー参照 `&`、`'`、`>`、`<`、および `"` についてエレメント内容において発生します。パーサーは、`&`、`'`、`>`、`<`、または `"` のいずれか 1 つがそれぞれに含まれる CHAR(1) 値または WIDECHAR(1) 値を渡します。

content_character_reference

このイベントは、&#dd; または &#xhh; という形式の数字参照 (ユニコード・コード・ポイントまたは「スカラー値」) について、エレメント内容において発生します。 *d* は 10 進数字を表します。 *h* は 16 進数字を表します。パーサーは、対応する整数値が入った FIXED BIN(31) 値を渡します。

processing_instruction

処理命令 (PI) を使用すると、XML 文書にアプリケーション用の特別な命令を含めることができます。 このイベントは、パーサーが PI 開始文字シーケンス <? に続く名前を認識したときに発生します。さらにこのイベントは、処理命令 (PI) ターゲットに続く、PI 終了文字シーケンス ?> の直前までのデータを対象とします。データの末尾にある空白文字は含まれますが、先頭にある空白文字は含まれません。パーサーは、ターゲットが含まれているテキスト (458 ページの図 95 に示されている例では spread) のアドレスと長さ、およびデータが含まれているテキスト (例では please use real mayonnaise) のアドレスと長さを渡します。

comment

このイベントは、XML 文書内のコメントに対して発生します。 パーサーは、開始コメント区切り文字 <!-- と終了コメント区切り文字 --> の間にあるテキストのアドレスと長さを渡します。 458 ページの図 95 に示されている例では、This document is just an example が唯一のコメントのテキストです。

unknown_attribute_reference

このイベントは、属性値の中で、5 つの定義済みエンティティー参照 (イベント attribute_predefined_character の項に示した) 以外のエンティティー参照に対して発生します。パーサーは、エンティティー名が入ったテキストのアドレスと長さを渡します。

unknown_content_reference

このイベントは、content_predefined_character イベントについてリストされている 5 つの定義済みエンティティー参照以外のエンティティー参照についてエレメント内容の内側で発生します。パーサーは、エンティティー名が入ったテキストのアドレスと長さを渡します。

start_of_prefix_mapping

このイベントは、現在は生成されません。

end_of_prefix_mapping

このイベントは、現在は生成されません。

exception

XML 文書の処理中にエラーを検出すると、パーサーはこのイベントを生成します。

イベント関数に渡されるパラメーター

イベント関数はすべて、パーサーへの戻りコードである BYVALUE FIXED BIN(31) 値を戻す必要があります。 パーサーを正常に継続するためには、この値がゼロでなければなりません。

これらの関数すべてに、最初の引数として BYVALUE POINTER が渡されます。この値は、元は組み込み関数への 2 番目の引数として渡されたトークン値です。

次に示す例外を除き、イベントのテキスト・エレメントのアドレスと長さを提供する BYVALUE POINTER と BYVALUE FIXED BIN(31) も、すべての関数に渡されます。 以下の関数とイベントは例外です。

end_of_document
ユーザー・トークン以外の引数は渡されません。

attribute_predefined_reference
ユーザー・トークンのほかに、もう 1 つの引数が渡されます。それは、定義済み文字の値を保持する BYVALUE CHAR(1) (UTF-16 文書の場合は BYVALUE WIDECHAR(1)) です。

content_predefined_reference
ユーザー・トークンのほかに、もう 1 つの引数が渡されます。それは、定義済み文字の値を保持する BYVALUE CHAR(1) (UTF-16 文書の場合は BYVALUE WIDECHAR(1)) です。

attribute_character_reference
ユーザー・トークンに加えて、数値参照の値を保持する BYVALUE FIXED BIN(31) がもう 1 つの引数として渡されます。

content_character_reference
ユーザー・トークンに加えて、数値参照の値を保持する BYVALUE FIXED BIN(31) がもう 1 つの引数として渡されます。

processing_instruction
ユーザー・トークンに加えて、次の 4 つの引数が渡されます。

1. ターゲット・テキストのアドレスを示す BYVALUE POINTER
2. ターゲット・テキストの長さを示す BYVALUE FIXED BIN(31)
3. データ・テキストのアドレスを示す BYVALUE POINTER
4. データ・テキストの長さを示す BYVALUE FIXED BIN(31)

exception
ユーザー・トークンに加えて、次の 3 つの引数が渡されます。

1. 問題のテキストのアドレスを示す BYVALUE POINTER
2. 文書内での問題のテキストのバイト・オフセットを示す BYVALUE FIXED BIN(31)
3. 例外コードの値を示す BYVALUE FIXED BIN(31)

XML 文書のコード化文字セット

PLISAX 組み込みサブルーチンがサポートする XML 文書は、ユニコード UTF-16 でエンコードされた WIDECHAR の文書、またはこのセクションでリストされていて明示的にサポートされているいずれかの 1 バイト文字セットでエンコードされた CHARACTER の文書のみです。

パーサーは、XML 文書のエンコード方式に関する情報ソースを 3 つまで使用し、これらのソース間で矛盾を検出した場合は、次のように例外 XML イベントをシグナル通知します。

1. パーサーは、文書の最初の文字を検査することによって文書の基本エンコードを判別します。
2. ステップ 1 が正常に完了した場合、パーサーはエンコード宣言を検索します。
3. 最後に、パーサーは PLISAX 組み込みサブルーチン呼び出しのコード・ページ値を参照します。 このパラメーターが省略された場合、デフォルトで使用される値は、明示指定またはデフォルトの CODEPAGE コンパイラー・オプションの値です。

XML 文書の最初に XML 宣言があり、その宣言に、このセクションでリストされているいずれかのサポート対象コード・ページが指定されているエンコード宣言が含まれている場合に、そのエンコード宣言が、基本文書エンコード方式や、PLISAX 組み込みサブルーチンからのエンコード情報と矛盾しなければ、パーサーはそのエンコード宣言を受け入れます。 XML 文書に XML 宣言自体がない場合、または XML 宣言がエンコード宣言を省略している場合は、基本文書エンコードと矛盾しなければ、パーサーは PLISAX 組み込みサブルーチンからのエンコード情報を使用して文書を処理します。

サポートされる EBCDIC コード・ページ

次の表で、最初の番号はユーロ国別拡張コード・ページ (ECECP)、2 番目の番号は国別拡張コード・ページ (CECP) のものです。

CCSID	説明
01047	Latin 1/オープン・システム
01140、00037	米国、カナダなど
01141、00273	オーストリア、ドイツ
01142、00277	デンマーク、ノルウェー
01143、00278	フィンランド、スウェーデン
01144、00280	イタリア
01145、00284	スペイン、ラテンアメリカ (スペイン語)
01146、00285	英国
01147、00297	フランス
01148、00500	国際
01149、00871	アイスランド

サポートされる ASCII コード・ページ

CCSID	説明
00813	ISO 8859-7 ギリシャ語/ラテン語
00819	ISO 8859-1 Latin 1/オープン・システム
00920	ISO 8859-9 Latin 5 (ECMA-128、トルコ TS-5881)

コード・ページの指定

文書の XML 宣言にエンコード宣言が含まれていない場合、または XML 宣言自体が存在しない場合、パーサーは、PLISAX 組み込みサブルーチン呼び出しで提供されるエンコード情報を文書の基本エンコードと併せて使用します。

ほとんどの XML 文書の最初にある XML 宣言の中で、文書のエンコード情報を指定することもできます。 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"?>
```

XML 文書にエンコード宣言がある場合は、 PLISAX 組み込みサブルーチンから提供されるエンコード情報、および文書の基本エンコードと、エンコード宣言が整合していることを確認してください。 エンコード宣言、PLISAX 組み込みサブルーチンから提供されるエンコード情報、および文書の基本エンコードの間に矛盾がある場合、パーサーは例外 XML イベントをシグナル通知します。

エンコード宣言は、番号または別名を使用して指定できます。

番号を使用してエンコード宣言を指定

次のいずれかの接頭部 (大文字と小文字の組み合わせは自由) を付けて CCSID 番号を (先行ゼロなしで、または任意の数の先行ゼロを付けて) 指定できます。

IBM_	CP	CCSID_
IBM-	CP_	CCSID-
	CP-	

別名を使用してエンコード宣言を指定

以下のサポート対象の別名 (大文字と小文字の組み合わせは自由) を任意に使用できます。

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

例外

ほとんどの例外の場合、XML テキストには、文書の、例外が検出されたポイントまで (そのポイントを含む) の構文解析済み部分が入ります。構文解析の開始前にシグナル通知されるエンコード対立例外に関しては、XML テキストの長さはゼロになるか、または文書からのエンコード宣言値のみが XML テキストに含まれます。

458 ページの図 95 に示されている例には、例外イベントを引き起こす項目が 1 つあります。それは、`sandwich` エレメント終了タグの後ろにある余分な `junk` です。

例外には次の 2 種類があります。

1. 構文解析を任意で継続できる例外

   継続可能な例外の例外コードの範囲は 1 から 99 まで、100001 から 165535 まで、または 200001 から 265535 までです。例に含まれる例外イベントの例外番号は 1 であるため、その例外イベントは継続可能です。
2. 継続が可能でない致命的な例外

   致命的な例外の例外コードは、99 より大きい (ただし 100000 より小さい) 値です。

非ゼロの戻りコードを出した例外イベント関数から戻ると、通常パーサーは文書の処理を停止し、PLISAXA または PLISAXB 組み込みサブルーチンを呼び出したプログラムに制御を戻します。

継続可能な例外の場合は、ゼロの戻りコードを指定して例外イベント関数から戻ることにより、パーサーに文書の処理継続を要求します。ただし、その後でさらに例外が発生する場合もあります。継続を要求したときにパーサーが行う処置について詳しくは、477 ページの『継続可能な例外コード』を参照してください。

範囲 100001 から 165535 まで、および 200001 から 265535 までの例外番号の例外については、特殊なケースが適用されます。これらの範囲の例外コードは、文書の CCSID (エンコード宣言など、文書の先頭を検査することによって決定される) が、PLISAXA または PLISAXB 組み込みサブルーチンによって指定 (明示的または暗黙的に) された CCSID 値と同一でないことを示しています。このことは、両方の CCSID が同じ基本エンコード (EBCDIC または ASCII) を示すものであっても起こります。

これらの例外の場合、例外イベントに渡される例外コードは、EBCDIC CCSID の場合は文書の CCSID に 100000 を加算した値、ASCII CCSID の場合は 200000 を加算した値になります。例えば、例外コードが 101140 の場合、文書の CCSID は 01140 です。 PLISAXA 組み込みサブルーチンまたは PLISAXB 組み込みサブルーチンによって提供される CCSID 値は、呼び出しの最終引数として明示的に設定されるか、または最終引数が省略されて CODEPAGE コンパイラー・オプションの値が使用されるときは暗黙的に設定されます。

このような CCSID の矛盾によって起こった例外に対する例外イベント関数から戻った後、戻りコードの値に応じて、パーサーは次の 3 つのうちいずれかの処置を実行します。

1. 戻りコードがゼロの場合、パーサーは組み込みサブルーチンによって提供された CCSID を使用して処理を続行します。
2. 戻りコードが文書の CCSID (つまり、元の例外コード値から 100000 または 200000 を減算した値) の場合、パーサーは文書の CCSID を使用して処理を続行します。これは、構文解析イベントのいずれかから非ゼロの値が戻された後、パーサーが処理を継続する唯一のケースです。
3. そうでなければ、パーサーは文書の処理を停止し、制御を PLISAXA または PLISAXB 組み込みサブルーチンに戻します。サブルーチンは ERROR 条件を発生させます。

例

ここでは、PLISAXA 組み込みサブルーチンの使用について例を交えて説明します。

次の例では、 458 ページの図 95 に示されているサンプル XML 文書が使用されています。

```
saxtest: package exports(saxtest);

  define alias event
      limited entry( pointer, pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_end_of_document
      limited entry( pointer )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_predefined_ref
      limited entry( pointer, char(1) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) nodescriptor );

  define alias event_character_ref
      limited entry( pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_pi
      limited entry( pointer, pointer, fixed bin(31),
                     pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_exception
      limited entry( pointer, pointer, fixed bin(31),
                     fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );
```

図 *96. PLISAXA* のコーディング例 - 型宣言

```
saxtest: proc options( main );

  dcl
    1 eventHandler static

      ,2 e01 type event
             init( start_of_document )
      ,2 e02 type event
             init( version_information )
      ,2 e03 type event
             init( encoding_declaration )
      ,2 e04 type event
             init( standalone_declaration )
      ,2 e05 type event
             init( document_type_declaration )
      ,2 e06 type event_end_of_document
             init( end_of_document )
      ,2 e07 type event
             init( start_of_element )
      ,2 e08 type event
             init( attribute_name )
      ,2 e09 type event
             init( attribute_characters )
      ,2 e10 type event_predefined_ref
             init( attribute_predefined_reference )
      ,2 e11 type event_character_ref
             init( attribute_character_reference )
      ,2 e12 type event
             init( end_of_element )
      ,2 e13 type event
             init( start_of_CDATA )
      ,2 e14 type event
             init( end_of_CDATA )
      ,2 e15 type event
             init( content_characters )
      ,2 e16 type event_predefined_ref
             init( content_predefined_reference )
      ,2 e17 type event_character_ref
             init( content_character_reference )
      ,2 e18 type event_pi
             init( processing_instruction )
      ,2 e19 type event
             init( comment )
      ,2 e20 type event
             init( unknown_attribute_reference )
      ,2 e21 type event
             init( unknown_content_reference )
      ,2 e22 type event
             init( start_of_prefix_mapping )
      ,2 e23 type event
             init( end_of_prefix_mapping )
      ,2 e24 type event_exception
             init( exception )
      ;
```

図 *97. PLISAXA* のコーディング例 - イベント構造体

```
   dcl token       char(8);

   dcl xmlDocument char(4000) var;

   xmlDocument =
      '<?xml version="1.0" standalone="yes"?>'
   || '<!--This document is just an example-->'
   || '<sandwich>'
   || '<bread type="baker&quot;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';


   call plisaxa( eventHandler,
                 addr(token),
                 addrdata(xmlDocument),
                 length(xmlDocument) );

end;
```

図 *98. PLISAXA* のコーディング例 - メインルーチン

```
dcl chars char(32000) based;

start_of_document:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' length=' || tokenlength );

  return(0);
end;

version_information:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

encoding_declaration:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;
```

図 *99. PLISAXA* のコーディング例 - イベント・ルーチン

```
standalone_declaration:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

document_type_declaration:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

end_of_document:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;

  put skip list( lowercase( procname() ) );

  return(0);
end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
  start_of_element:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  attribute_name:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  attribute_characters:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
  attribute_predefined_reference:
    proc( userToken, reference )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) nodescriptor );

    dcl userToken      pointer;
    dcl reference   char(1);

    put skip list( lowercase( procname() )
     || ' ' || hex(reference ) );

    return(0);
  end;

  attribute_character_reference:
    proc( userToken, reference )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl reference      fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || hex(reference ) );

    return(0);
  end;

  end_of_element:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
start_of_CDATA:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

end_of_CDATA:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

content_characters:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
  content_predefined_reference:
    proc( userToken, reference )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) nodescriptor );

    dcl userToken      pointer;
    dcl reference   char(1);

    put skip list( lowercase( procname() )
     || ' ' || hex(reference ) );

    return(0);
  end;

  content_character_reference:
    proc( userToken, reference )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl reference      fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || hex(reference ) );

    return(0);
  end;

  processing_instruction:
    proc( userToken, piTarget, piTargetLength,
                             piData, piDataLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl piTarget       pointer;
    dcl piTargetLength fixed bin(31);
    dcl piData         pointer;
    dcl piDataLength   fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(piTarget->chars,1,piTargetLength ) || '>' );

    return(0);
  end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
  comment:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  unknown_attribute_reference:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  unknown_content_reference:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(続き)*

```
  start_of_prefix_mapping:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  end_of_prefix_mapping:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  exception:
    proc( userToken, xmlToken, currentOffset, errorID )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl currentOffset  fixed bin(31);
    dcl errorID        fixed bin(31);

    put skip list( lowercase( procname() )
     || ' errorid =' || errorid );

    return(0);
  end;
end;
```

PLISAXA のコーディング例 - イベント・ルーチン *(*続き*)*

前のプログラムで生成される出力は次のとおりです。

```
start_of_dcoument length=           305
version_information <1.0>
standalone_declaration <yes>
comment <This document is just an example>
start_of_element <sandwich>
start_of_element <bread>
attribute_name <type>
attribute_characters <baker>
attribute_predefined_reference 7D
attribute_characters <s best>
end_of_element <bread>
processing_instruction <spread>
start_of_element <meat>
content_characters <Ham >
content_predefined_reference 50
content_characters < turkey>
end_of_element <meat>
start_of_element <filling>
content_characters <Cheese, lettuce, tomato, etc.>
end_of_element <filling>
start_of_cdata <<![CDATA[>
content_characters <We should add a <relish> element in future!>
end_of_cdata <]]>
end_of_element <sandwich>
exception errorid =             1
content_characters <j>
exception errorid =             1
content_characters <u>
exception errorid =             1
content_characters <n>
exception errorid =             1
content_characters <k>
end_of_document
```

図 *100. PLISAXA* のコーディング例 - プログラム出力

継続可能な例外コード

このトピックでは、例外コードについて説明します。また、例外が発生した後で処理を続行するようにパーサーに要求したときにパーサーが行うアクションについても説明します。

次の表では、例外イベントに渡される例外コード・パラメーターの各値は「番号」列にリストされています。例外コードごとに説明があります。また、例外が発生した後で処理を続行するようにパーサーに要求したときにパーサーが行うアクションも示されています。この説明にある *XML* テキスト という用語は、イベントに渡されるポインターと長さに基づくストリングを意味しています。

表 *37.* 継続可能な例外

番号	説明	継続時のパーサーの処置
1	エレメント内容の外側で、空白文字をスキャン中にパーサーが無効文字を検出した。	パーサーは content_characters イベントを生成し、XML テキストには (単一の) 無効文字が入る。構文解析は無効文字の後の文字から継続する。

表 37. 継続可能な例外 (続き)

番号	説明	継続時のパーサーの処置
2	エレメント内容の外側で、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をパーサーが検出した。	パーサーは content_characters イベントを生成し、XML テキストには 2 から 3 文字までの無効な先頭文字シーケンスが入る。 構文解析は無効シーケンスの後の文字から継続する。
3	属性名の重複をパーサーが検出した。	パーサーは attribute_name イベントを生成し、XML テキストには重複した属性名が入る。
4	属性値の中にマークアップ文字 < をパーサーが検出した。	パーサーは例外イベントを生成する前に、< 文字の前にある属性値の部分に対して attribute_characters イベントを生成する。例外イベントの後で、パーサーは attribute_characters イベントを生成し、< を XML テキストに含める。構文解析は < の後の文字から継続する。
5	エレメントの開始タグと終了タグの名前が一致しない。	パーサーは end_of_element イベントを生成し、XML テキストには一致しなかった終了名が入る。
6	エレメント内容の中で無効文字をパーサーが検出した。	パーサーは、後続の content_characters イベントの XML テキストに無効文字を入れる。
7	エレメント内容の中で、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をパーサーが検出した。	例外イベントを生成する前に、< マークアップ文字の前にある内容の部分に対して、パーサーは content_characters イベントを生成する。例外イベントの後、パーサーは content_characters イベントを生成し、XML テキストには < と無効文字の 2 つの文字が入る。構文解析は無効文字の後の文字から継続する。
8	エレメント内容の中で、対になる CDATA 開始文字シーケンス <![CDATA[がない終了文字シーケンス]]> をパーサーが検出した。	パーサーは例外イベントを生成する前に、]]> 文字シーケンスの前にある内容の部分に対して content_characters イベントを生成する。パーサーは、例外イベントの後で、content_characters イベントを生成し、3 文字のシーケンス]]> を XML テキストに含める。構文解析はこのシーケンスの後の文字から継続する。
9	コメントの中で無効文字をパーサーが検出した。	パーサーは、後続の comment イベントの XML テキストに無効文字を入れる。
10	コメントの中で、文字シーケンス -- の後に > が付いていないことをパーサーが検出した。	パーサーは -- 文字シーケンスによってコメントが終了したと想定し、comment イベントを生成する。構文解析は -- シーケンスの後の文字から継続する。
11	処理命令データ・セグメントの中で、無効文字をパーサーが検出した。	パーサーは、後続の processing_instruction イベントに対して XML テキストに無効文字を組み込む。
12	処理命令ターゲット名が、小文字、大文字、または大/小文字混合の xml だった。	パーサーは processing_instruction イベントを生成し、元のとおりの大/小文字で xml を XML テキストに含める。

表 37. 継続可能な例外 (続き)

番号	説明	継続時のパーサーの処置
13	16 進文字参照 (形式 &#xdddd;) の中で、無効数字をパーサーが検出した。	パーサーは attribute_characters イベントまたは content_characters イベントを生成し、XML テキストには無効数字が入る。参照の構文解析は、この無効数字の後から継続する。
14	10 進文字参照 (形式 &#dddd;) の中で、無効数字をパーサーが検出した。	パーサーは attribute_characters イベントまたは content_characters イベントを生成し、XML テキストには無効数字が入る。参照の構文解析は、この無効数字の後から継続する。
15	XML 宣言におけるエンコード宣言値が、小文字大文字に関係なく A から Z までの文字で始まっていなかった。	パーサーは encoding イベントを生成し、XML テキストには指定されたとおりのエンコード宣言値が入る。
16	文字参照が正しい XML 文字を参照していない。	パーサーは、attribute_character_reference イベントまたは content_character_reference イベントを生成し、文字参照で指定された単一ユニコード文字を XML-NTEXT に含める。
17	エンティティー参照名の中で、無効文字をパーサーが検出した。	パーサーは、後続の unknown_attribute_reference イベント、または unknown_content_reference イベントの XML テキストに無効文字を入れる。
18	属性値の中で、無効文字をパーサーが検出した。	パーサーは、後続の attribute_characters イベントの XML テキストに無効文字を入れる。 **注:** PL/I XML パーサーは標準からはずれて、属性ストリングにおいて「<」を有効なものとして受け入れます。
50	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書のエンコード宣言に認識可能なエンコード方式が指定されていない。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
51	文書は EBCDIC でエンコードされ、文書のエンコード宣言にはサポートされる EBCDIC エンコード方式が指定されているが、CODEPAGE コンパイラー・オプションに指定されたコード・ページをパーサーがサポートしていない。	パーサーは、文書のエンコード宣言に指定されたエンコード方式を使用する。
52	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書のエンコード宣言には ASCII エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
53	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書のエンコード宣言にはサポートされるユニコード・エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。

表 37. 継続可能な例外 (続き)

番号	説明	継続時のパーサーの処置
54	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書のエンコード宣言には、パーサーがサポートしないユニコード・エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
55	文書は EBCDIC でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書のエンコード宣言には、パーサーがサポートしないエンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
56	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書のエンコード宣言に認識可能なエンコード方式が指定されていない。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
57	文書は ASCII でエンコードされ、文書のエンコード宣言にはサポートされる ASCII エンコード方式が指定されているが、CODEPAGE コンパイラー・オプションに指定されたコード・ページをパーサーがサポートしていない。	パーサーは、文書のエンコード宣言に指定されたエンコード方式を使用する。
58	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書のエンコード宣言にはサポートされる EBCDIC エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
59	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書のエンコード宣言にはサポートされるユニコード・エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
60	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書のエンコード宣言には、パーサーがサポートしないユニコード・エンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。
61	文書は ASCII でエンコードされ、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書のエンコード宣言には、パーサーがサポートしないエンコード方式が指定されている。	パーサーは、CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。

表 37. 継続可能な例外 (続き)

番号	説明	継続時のパーサーの処置
100001 から 165535 まで	文書は EBCDIC でエンコードされた。CODEPAGE コンパイラー・オプションで指定されたエンコード方式と、文書エンコード宣言で指定されたエンコード方式はどちらも、サポート対象の EBCDIC コード・ページであるが同じではない。例外コードには、エンコード宣言の CCSID に 100000 を加算した値が入る。	ユーザーが例外イベントからゼロを戻した場合、パーサーは CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。 文書のエンコード宣言からの CCSID を戻した場合 (例外コードから 100000 を減算して)、パーサーはこのエンコード方式を使用する。
200001 から 265535 まで	文書は ASCII でエンコードされた。CODEPAGE コンパイラー・オプションで指定されたエンコード方式と、文書エンコード宣言で指定されたエンコード方式はどちらも、サポート対象の ASCII コード・ページであるが同じではない。例外コードには、エンコード宣言の CCSID に 200000 を加算した値が入る。	ユーザーが例外イベントからゼロを戻した場合、パーサーは CODEPAGE コンパイラー・オプションに指定されたエンコード方式を使用する。 文書のエンコード宣言からの CCSID を戻した場合 (例外コードから 200000 を減算して)、パーサーはこのエンコード方式を使用する。

例外コードの終了

このトピックでは、終了例外コードについて説明します。

表 38. 終了例外

番号	説明
100	XML 宣言の開始のスキャン中に、パーサーが文書の終わりに達した。
101	XML 宣言の終了の検索中に、パーサーが文書の終わりに達した。
102	ルート・エレメントの検索中に、パーサーが文書の終わりに達した。
103	XML 宣言のバージョン情報の検索中に、パーサーが文書の終わりに達した。
104	XML 宣言のバージョン情報値の検索中に、パーサーが文書の終わりに達した。
106	XML 宣言のエンコード宣言値の検索中に、パーサーが文書の終わりに達した。
108	XML 宣言のスタンドアロン宣言値の検索中に、パーサーが文書の終わりに達した。
109	属性名のスキャン中に、パーサーが文書の終わりに達した。
110	属性値のスキャン中に、パーサーが文書の終わりに達した。
111	属性値の文字参照またはエンティティー参照のスキャン中に、パーサーが文書の終わりに達した。
112	空エレメント・タグのスキャン中に、パーサーが文書の終わりに達した。
113	ルート・エレメント名のスキャン中に、パーサーが文書の終わりに達した。
114	エレメント名のスキャン中に、パーサーが文書の終わりに達した。
115	エレメント内容の文字データのスキャン中に、パーサーが文書の終わりに達した。
116	エレメント内容の処理命令のスキャン中に、パーサーが文書の終わりに達した。
117	エレメント内容のコメントまたは CDATA セクションのスキャン中に、パーサーが文書の終わりに達した。
118	エレメント内容のコメントのスキャン中に、パーサーが文書の終わりに達した。
119	エレメント内容の CDATA セクションのスキャン中に、パーサーが文書の終わりに達した。

表 38. 終了例外 (続き)

番号	説明
120	エレメント内容の文字参照またはエンティティー参照のスキャン中に、パーサーが文書の終わりに達した。
121	ルート・エレメントの終了後のスキャン中に、パーサーが文書の終わりに達した。
122	文書タイプ宣言の開始が無効である可能性があることをパーサーが検出した。
123	2 番目の文書タイプ宣言をパーサーが検出した。
124	ルート・エレメント名の先頭文字が、文字、_、または : でない。
125	エレメントの最初の属性名の先頭文字が、文字、_、または : でない。
126	エレメント名の内部または後に、パーサーが無効文字を検出した。
127	属性名の後に = 以外の文字が続いていることをパーサーが検出した。
128	無効な属性値区切り文字をパーサーが検出した。
130	属性名の先頭文字が、文字、_、: のいずれでもなかった。
131	属性名の内部または後に無効文字をパーサーが検出した。
132	空エレメント・タグが、> とそれに続く / で終わっていなかった。
133	エレメント終了タグ名の先頭文字が、文字、_、: のいずれでもなかった。
134	エレメント終了タグ名が > で終わっていない。
135	エレメント名の先頭文字が、文字、_、または : でない。
136	エレメント内容の中で、コメントまたは CDATA セクションの無効な開始をパーサーが検出した。
137	コメントの無効な開始をパーサーが検出した。
138	処理命令の先頭文字が、文字、_、または : でない。
139	処理命令ターゲット名の内部または後に、無効文字をパーサーが検出した。
140	処理命令が終了文字シーケンス ?> で終わっていない。
141	文字参照またはエンティティー参照名の中で、& の後に無効文字をパーサーが検出した。
142	XML 宣言の中にバージョン情報がない。
143	XML 宣言の中で、'version' の後に = が付いていない。
144	XML 宣言の中で、バージョン宣言値が欠落しているか、誤って区切られている。
145	XML 宣言の中で、バージョン情報値に不正な文字が指定されているか、または開始と終了の区切り文字が一致しない。
146	XML 宣言の中で、バージョン情報値の終了区切り文字の後に無効文字をパーサーが検出した。
147	XML 宣言の中で、オプショナルのエンコード宣言があるべき個所に無効な属性をパーサーが検出した。
148	XML 宣言の中で、encoding の後に = が付いていない。
149	XML 宣言の中で、エンコード宣言値が欠落しているか、誤って区切られている。
150	XML 宣言の中で、エンコード宣言値に不正な文字が指定されているか、または開始と終了の区切り文字が一致しない。
151	XML 宣言の中で、エンコード宣言値の終了区切り文字の後に無効文字をパーサーが検出した。
152	XML 宣言の中で、オプショナルのスタンドアロン宣言があるべき個所に無効な属性をパーサーが検出した。
153	XML 宣言の中で、'standalone' の後に = が付いていない。

表 *38.* 終了例外 *(*続き*)*

番号	説明
154	XML 宣言の中で、スタンドアロン宣言値が欠落しているか、誤って区切られている。
155	スタンドアロン宣言値が yes または no のどちらでもない。
156	XML 宣言の中で、スタンドアロン宣言値に不正な文字が指定されているか、または開始と終了の区切り文字が一致しない。
157	XML 宣言の中で、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をパーサーが検出した。
158	XML 宣言が正しい文字シーケンス ?> で終わっていないか、無効な属性を含んでいる。
159	ルート・エレメントの終了後、文書タイプ宣言の開始をパーサーが検出した。
160	ルート・エレメントの終了後、エレメントの開始をパーサーが検出した。
300	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されている。
301	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションにはユニコードが指定されている。
302	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションにはサポートされないコード・ページが指定されている。
303	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションがサポートされておらず、文書エンコード宣言は空であるか、サポートされない英字のエンコード方式の別名が指定されている。
304	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションが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ない。
305	文書は EBCDIC でエンコードされているが、CODEPAGE コンパイラー・オプションがサポートされておらず、文書のエンコード宣言にはサポートされる EBCDIC コード方式が指定されていない。
306	文書は ASCII でエンコードされているが、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されている。
307	文書は ASCII でエンコードされているが、CODEPAGE コンパイラー・オプションにはユニコードが指定されている。
308	文書は ASCII でエンコードされているが、CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページ、ASCII、またはユニコードが指定されていない。
309	CODEPAGE コンパイラー・オプションにはサポートされる ASCII コード・ページが指定されているが、文書はユニコードでエンコードされている。
310	CODEPAGE コンパイラー・オプションにはサポートされる EBCDIC コード・ページが指定されているが、文書はユニコードでエンコードされている。
311	CODEPAGE コンパイラー・オプションにはサポートされないコード・ページが指定されており、文書はユニコードでエンコードされている。
312	文書は ASCII でエンコードされていたが、外部指定されたエンコード方式と、文書のエンコード宣言で指定されたエンコード方式が両方ともサポートされていなかった。
313	文書は ASCII でエンコードされているが、CODEPAGE コンパイラー・オプションが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ない。
314	文書は ASCII でエンコードされているが、CODEPAGE コンパイラー・オプションがサポートされておらず、文書のエンコード宣言にはサポートされる ASCII コード方式が指定されていない。

表 38. 終了例外 (続き)

番号	説明
315	文書は UTF-16 リトル・エンディアンでエンコードされているが、パーサーはこのプラットフォーム上でその方式をサポートしない。
316	文書は UCS4 でエンコードされているが、パーサーはその方式をサポートしない。
317	文書のエンコード方式をパーサーが判別できない。 文書は損傷している可能性がある。
318	文書は UTF-8 でエンコードされているが、パーサーはその方式をサポートしない。
319	文書は UTF-16 ビッグ・エンディアンでエンコードされているが、パーサーはこのプラットフォーム上でその方式をサポートしない。
501, 502, 503	PLISAX(A|B) で内部エラーが発生した。 IBM サポートにお問い合わせください。
500	PLISAXA 内部データ構造へのメモリー割り振りが失敗した。 アプリケーション・プログラムが使用できるストレージ量を増やしてください。
520	PLISAXB 内部データ構造へのメモリー割り振りが失敗した。 アプリケーション・プログラムが使用できるストレージ量を増やしてください。
521	PLISAX(A|B) で内部エラーが発生した。 IBM サポートにお問い合わせください。
523	PLISAXB がファイル入出力エラーを検出した。
524	ファイル・システムから XML 文書をキャッシュに入れようとして PLISAXB でメモリー割り振りに失敗した。 アプリケーション・プログラムが使用できるストレージ量を増やしてください。
525	サポートされない URI スキームが PLISAXB に指定された。
526	PLISAXB に提供された XML 文書の文字数が、最小限の 4 文字より少ないか多すぎた。
527, 560	PLISAX(A|B) で内部エラーが発生した。 IBM サポートにお問い合わせください。
561	PLISAX(A|B) のどちらにもイベント・ハンドラーが指定されていない。
562, 563, 580, 581	PLISAX(A|B) で内部エラーが発生した。 IBM サポートにお問い合わせください。
600 から 99999 まで	内部エラーです。 このエラーをサービス担当員に報告してください。

第 20 章 PLISAXC および PLISAXD XML パーサーの使用

PLISAXC および PLISAXD 組み込みサブルーチンは、基本的な XML 構文解析機能を提供します。これによりプログラムは、インバウンド XML 文書を取り込み、適格性を検査し、その内容を処理できます。

PLISAXC で使用される XML パーサーは有効性検査を行いませんが、適格性のエラーを部分的に検査し、エラーを発見した場合は例外イベントを生成します。

PLISAXD 組み込みサブルーチンは、妥当性検査機能を備えた XML 構文解析を提供します。 これにより、インバウンド XML 文書がインバウンド XML スキーマで指定された一連の規則に従っているかどうかが判別されます。

PLISAXC および PLISAXD サブルーチンは XML 生成を行いません。XML 生成は代わりに、PL/I プログラム・ロジックによって、または XMLCHAR 組み込み関数を使用して行う必要があります。

PLISAXC および PLISAXD には、AMODE 24 でサポートされないことを除き、特別な環境要件はありません。 CICS、IMS、MQ Series、z/OS バッチ、および TSO を含め、主要なすべてのランタイム環境で実行できます。

PLISAXC 組み込みサブルーチンと PLISAXD 組み込みサブルーチン、および PLISAXA 組み込みサブルーチンおよび PLISAXB 組み込みサブルーチンには多くの類似点があるため、このセクションの情報の一部は、 455 ページの『第 19 章 PLISAXA および PLISAXB XML パーサーの使用』に記載されている情報の繰り返しになります。

概説

XML 構文解析用のインターフェースには、大きく分けてイベント・ベースとツリー・ベースの 2 種類があります。

イベント・ベース API の場合、パーサーはコールバックによってアプリケーションにイベントを報告します。 このようなイベントには、文書の開始、エレメントの開始などがあります。 アプリケーションは、パーサーから報告されたイベントを処理するためのハンドラーを備えています。 Simple API for XML (SAX) は、業界標準のイベント・ベース API の一例です。

ツリー・ベース API (文書オブジェクト・モデル (DOM) など) の場合、パーサーは XML をツリー・ベースの内部表現に変換します。ツリーをナビゲートするためのインターフェースが提供されています。

IBM PL/I コンパイラーは、PLISAXC または PLISAXD を使用して、XML 文書の構文解析用に SAX のようなイベント・ベースのインターフェースを提供します。パーサーは、対応する文書フラグメントへの参照を渡して、アプリケーション提供のパーサー・イベント用のハンドラーを呼び出します。

パーサーには次の特性があります。

© Copyright IBM Corp. 1999, 2017

- 高性能であるが非標準のインターフェースを提供します。
- ユニコード UTF-16、UTF-8、または 495 ページの『XML 文書のコード化文字セット』で示されたいずれかの 1 バイト・コード・ページでエンコードされた XML ファイルをサポートします。

XML 文書の準拠レベルには適格性と有効性の 2 つがあり、どちらのレベルも XML 標準に定義されています。 XML 標準は、http://www.w3c.org/XML/ に掲載されています。 これらの定義を要約すると、XML 文書が基本的な XML 文法と、いくつかの特定の規則 (開始エレメントと終了エレメントのタグが一致していることなどの要件) に準拠していれば、 XML 文書は整形式です。 さらに、整形式 XML 文書に文書タイプ宣言 (DTD) が関連していて、文書が DTD に表された制約に準拠している場合、その文書は有効です。

それぞれのパーサー・イベントごとに、 488 ページの図 101 のコード例に示すように、適切なパラメーターを受け入れて適切な戻り値を戻す PL/I 関数を用意する必要があります。

注:

- この関数に関しては、戻り値は値 BYVALUE によって返されなければなりません。
- この関数に関しては、使用されるリンケージは OPTLINK リンケージでなければなりません。 DEFAULT(LINKAGE(OPTLINK)) オプションを使用してこのリンケージを指定するか、OPTIONS(LINKAGE(OPTLINK)) 属性を使用して個々の PROCEDURE および ENTRY でこのリンケージを指定できます。

PLISAXC 組み込みサブルーチン

PLISAXC 組み込みサブルーチンを使用すると、プログラムの 1 つ以上のバッファー内にある XML 文書に対して XML パーサーを起動することができます。

e イベント構造体

p パーサーがイベント関数に戻すポインター値または「トークン」

x 入力 XML が入っている初期バッファーのアドレス

n そのバッファーにあるデータのバイト数

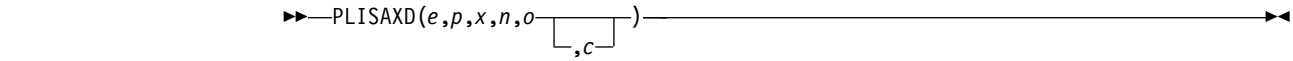
c その XML のコード・ページを指定する数値表現

注:

- XML が CHARACTER VARYING ストリングまたは WIDECHAR VARYING ストリングに含まれている場合は、ADDRDATA 組み込み関数を使用して最初のデータ・バイトのアドレスを取得する必要があります。
- XML が WIDECHAR ストリングに含まれている場合、バイト数の値は LENGTH 組み込み関数から返される値の 2 倍になります。

PLISAXD 組み込みサブルーチン

PLISAXD 組み込みサブルーチンを使用すると、妥当性検査機能を備えた XML パーサーを呼び出すことができます。 XML 文書と Optimized Schema Representation (OSR) ファイルの両方が、プログラム内の 1 つ以上のバッファー内にあります。



e イベント構造体

p パーサーがイベント関数に戻すポインター値または「トークン」

x 入力 XML が入っている初期バッファーのアドレス

n そのバッファーにあるデータのバイト数

o 入力 OSR が入っているバッファーのアドレス

c XML 文書のコード・ページを指定する数値表現

注:

- XML が CHARACTER VARYING ストリングまたは WIDECHAR VARYING ストリングに含まれている場合は、ADDRDATA 組み込み関数を使用して最初のデータ・バイトのアドレスを取得する必要があります。
- XML が WIDECHAR ストリングに含まれている場合、バイト数の値は LENGTH 組み込み関数から返される値の 2 倍になります。
- OSR は、プリプロセス版のスキーマです。 OSR について詳しくは、「*XML System Services* ユーザーズ・ガイドおよび解説書」を参照してください。

SAX イベント構造体

イベント構造体は、19 個の LIMITED ENTRY 変数で構成される構造体です。これらの変数は、さまざまな「イベント」に対してパーサーが起動する機能を指しています。

これらすべての ENTRY は、OPTLINK リンケージを使用する必要があります。

XML パーサーをサポートするライブラリー・ルーチンは、いずれかの PLISAX イベント構造体におけるイベントが (NULLENTRY 組み込み関数または UNSPEC を使用して) ヌルに設定されているかどうかを認識し、そのイベントがヌルに設定されている場合は、そのイベントを呼び出しません。これにより、関心のあるイベントのみに XML 構文解析コードを制限することができ、同時に構文解析全体のパフォーマンスを向上させることができます。

これらのすべての ENTRY には、最初の (場合によっては唯一の) パラメーター (プログラムによって PLISAXC および PLISAXD に渡されるユーザー・トークン) があります。

19 個のイベントの説明は、 488 ページの図 101 の XML 文書の例に対応しています。この説明にある *XML テキスト* という用語は、イベントに渡されるポインター

と長さに基づくストリングを意味しています。

```
xmlDocument =
   '<?xml version="1.0" standalone="yes"?>'
|| '<!--This document is just an example-->'
|| '<sandwich>'
|| '<bread type="baker&quot;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>';
```

図 *101.* サンプル *XML* 文書

XML 文書の内容に応じて、パーサーは以下のイベントを認識します。

start_of_document

このイベントは、文書の構文解析が開始されるときに 1 回発生します。パーサーは、このイベントには (ユーザー・トークンを除いて) パラメーターを渡しません。

version_information

このイベントは、オプショナルの XML 宣言内のバージョン情報に対して発生します。 パーサーは、バージョン値 (図 101 に示されている例では 1.0) を含むテキストのアドレスと長さを渡します。

encoding_declaration

このイベントは、XML 宣言内でオプショナルのエンコード宣言に対して発生します。 パーサーは、エンコード方式値が入ったテキストのアドレスと長さを渡します。

standalone_declaration

このイベントは、XML 宣言内でオプショナルのスタンドアロン宣言に対して発生します。 パーサーは、スタンドアロン値 (図 101 に示されている例では yes) を含むテキストのアドレスと長さを渡します。

document_type_declaration

このイベントは、パーサーが文書タイプ宣言を検出したときに発生します。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。その間には、かなり複雑な文法規則で記述された内容が含まれます。パーサーは、開始と終了の文字シーケンスを含む宣言全体が含まれるテキストのアドレスと長さを渡します。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。 図 101 に示されている例には、文書タイプ宣言が含まれていません。

end_of_document

このイベントは、文書の構文解析が完了したときに 1 回発生します。 パーサーは、このイベントには (ユーザー・トークンを除いて) パラメーターを渡しません。

start_of_element

このイベントは、エレメント開始タグ、または空エレメント・タグごとに 1 回発生します。 パーサーは、エレメント名が入ったテキストのアドレスと長さ、および適用される名前空間情報を渡します。 488 ページの図 101 に示されている例において構文解析中に最初に発生する start_of_element イベントの場合、このストリングは `sandwich` です。

attribute_name

このイベントは、有効な名前をパーサーが認識した後で、エレメント開始タグまたは空エレメント・タグにおける属性ごとに発生します。パーサーは、属性名が入ったテキストのアドレスと長さ、および適用される名前空間情報を渡します。 488 ページの図 101 に示されている例における唯一の属性名は `type` です。

attribute_characters

このイベントは、属性値ごとに発生します。 パーサーは、フラグメントが入ったテキストのアドレスと長さを渡します。 属性値は通常、次のように複数の行に分割されている場合でもただ 1 つのストリングで構成されます。

```
<element attribute="This attribute value is
split across two lines"/>
```

また、パーサーはフラグ・バイト (内容の一部を形成する追加文字を次のイベントが提供するのかどうかを示す) も渡します。これは、開始タグと終了タグの間に多くのデータが存在する場合に当てはまります。

end_of_element

このイベントは、エレメント終了タグ、または空エレメント・タグごとに、パーサーがタグの終了不等号括弧を認識したときに 1 回発生します。パーサーは、エレメント名が入ったテキストのアドレスと長さ、および適用される名前空間情報を渡します。

start_of_CDATA_section

このイベントは、CDATA セクションが開始されると発生します。 CDATA セクションはストリング `<![CDATA[` で始まりストリング `]]>` で終わります。 CDATA セクションは、CDATA セクションでなければ XML マークアップとして認識される文字が含まれているテキストのブロックを「エスケープ」するために使用されます。パーサーは、このイベントには (ユーザー・トークンを除いて) パラメーターを渡しません。 このイベントの後、パーサーは、これらの区切り文字間にある CDATA セクションの内容を 1 つ以上の content-characters イベントとして渡します。 488 ページの図 101 で示されている例に関しては、content-characters イベントとしてテキスト `We should add a <relish> element in future!` が渡されています。

end_of_CDATA_section

このイベントは、パーサーが CDATA セクションの終了を認識したときに発生します。 パーサーは、このイベントには (ユーザー・トークンを除いて) パラメーターを渡しません。

content_characters

このイベントは、XML 文書の「本体」である、エレメントの開始タグと終了タグの間にある文字データを表します。 パーサーは、データを含むテキストのアドレスおよび長さを渡します。データは通常、次のように複数の行に分割されている場合でも、単一のストリングのみで構成されます。

```
<element1>This character content is
split across two lines</element1>
```

また、パーサーはフラグ・バイト (内容の一部を形成する追加文字を次のイベントが提供するのかどうかを示す) も渡します。これは、開始タグと終了タグの間に多くのデータが存在する場合に当てはまります。

またパーサーは、content_characters イベントを使用して CDATA セクションのテキストをアプリケーションに渡します。

processing_instruction

処理命令 (PI) を使用すると、XML 文書にアプリケーション用の特別な命令を含めることができます。 このイベントは、パーサーが PI 開始文字シーケンス `<?` に続く名前を認識したときに発生します。さらにこのイベントは、処理命令 (PI) ターゲットに続く、PI 終了文字シーケンス `?>` の直前までのデータを対象とします。データの末尾にある空白文字は含まれますが、先頭にある空白文字は含まれません。パーサーは、ターゲットが含まれているテキスト (488 ページの図 101 に示されている例では `spread`) のアドレスと長さ、およびデータが含まれているテキスト (例では `please use real mayonnaise`) のアドレスと長さを渡します。

また、パーサーはフラグ・バイト (内容の一部を形成する追加文字を次のイベントが提供するのかどうかを示す) も渡します。これは、開始タグと終了タグの間に多くのデータが存在する場合に当てはまります。

comment

このイベントは、XML 文書内のコメントに対して発生します。 パーサーは、開始コメント区切り文字 `<!--` と終了コメント区切り文字 `-->` の間にあるテキストのアドレスと長さを渡します。 488 ページの図 101 に示されている例では、`This document is just an example` が唯一のコメントのテキストです。

また、パーサーはフラグ・バイト (内容の一部を形成する追加文字を次のイベントが提供するのかどうかを示す) も渡します。これは、開始タグと終了タグの間に多くのデータが存在する場合に当てはまります。

namespace_declare

このイベントは、XML 文書内の名前空間宣言に対して発生します。パーサーは、名前空間接頭部のアドレスと長さ (存在する場合)、および名前空間 URI のアドレスと長さを渡します。 名前空間接頭部が存在しない場合には、渡される長さはゼロであり、アドレスの値は使用されません。 PLIXSAXA および PLISAXB 組み込みサブルーチンには、対応するイベントはありません。

end_of_input

このイベントは、パーサーが現在の入力バッファーの最後に到達すると常に発生します。 パーサーは、(BYVALUE ユーザー・トークンとともに) 2 つの BYADDR パラメーター (処理対象の次のバッファーのアドレスおよび長さ) を渡します。 なお、BYADDR パラメーターをとるイベントは、このイベントおよび content-characters イベントのみですが、このイベントのみが、呼び出されるイベントが変更する必要があるパラメーターをとります。 PLIXSAXA および PLISAXB 組み込みサブルーチンには、対応するイベントはありません。また、このイベントによって、 PLISAXC および PLISAXD は任意のサイズの XML 文書の構文解析を行うことができます。

unresolved_reference

このイベントは、XML 文書内の未解決の参照に対して発生します。パーサーは、未解決の参照のアドレスおよび長さを渡します。

exception

XML 文書の処理中にエラーを検出すると、パーサーはこのイベントを生成します。

イベント関数に渡されるパラメーター

イベント関数はすべて、パーサーへの戻りコードである BYVALUE FIXED BIN(31) 値を戻す必要があります。 ゼロ以外の値が返された場合は、パーサーは終了します。

これらすべての関数に最初の引数として BYVALUE POINTER が渡されます。このポインターは本来、組み込み関数に 2 番目の引数として渡されるトークン値です。

次に示す例外を除き、イベントのテキスト・エレメントのアドレスと長さを提供する BYVALUE POINTER と BYVALUE FIXED BIN(31) も、すべての関数に渡されます。 以下の関数とイベントは例外です。

start_of_document
ユーザー・トークン以外の引数は渡されません。

end_of_document
ユーザー・トークン以外の引数は渡されません。

start_of_CDATA
ユーザー・トークン以外の引数は渡されません。

end_of_CDATA
ユーザー・トークン以外の引数は渡されません。

start_of_element
通常の 3 つのパラメーターに加えて、次の 4 つの引数が渡されます。

1. 名前空間接頭部のアドレスを示す BYVALUE POINTER
2. 名前空間接頭部の長さを示す BYVALUE FIXED BIN(31)
3. 名前空間 URI のアドレスを示す BYVALUE POINTER
4. 名前空間 URI の長さを示す BYVALUE FIXED BIN(31)

end_of_element

通常の 3 つのパラメーターに加えて、次の 4 つの引数が渡されます。

1. 名前空間接頭部のアドレスを示す BYVALUE POINTER
2. 名前空間接頭部の長さを示す BYVALUE FIXED BIN(31)
3. 名前空間 URI のアドレスを示す BYVALUE POINTER
4. 名前空間 URI の長さを示す BYVALUE FIXED BIN(31)

attribute_name

通常の 3 つのパラメーターに加えて、次の 4 つの引数が渡されます。

1. 名前空間接頭部のアドレスを示す BYVALUE POINTER
2. 名前空間接頭部の長さを示す BYVALUE FIXED BIN(31)
3. 名前空間 URI のアドレスを示す BYVALUE POINTER
4. 名前空間 URI の長さを示す BYVALUE FIXED BIN(31)

attribute_characters

通常の 3 つのパラメーターに加えて、次の 1 つの引数が渡されます。

- 以下の情報を示す BYVALUE ALIGNED BIT(8) フラグ・バイト:
  - 次のイベントに content-charcaters がさらに提供されるかどうか。これは、最初のビットがオンである場合は真です。つまり、このフィールドと '80'BX との AND 演算の結果が非ヌルである場合、これは真です。
  - XML に再び変換するときにエスケープしなければならない文字が存在しないかどうか。これは、2 番目のビットがオンの場合は真です。つまり、このフィールドと '40'BX との AND 演算の結果が非ヌルである場合、これは真です。

なお、このエントリーは、OPTIONS(NODESCRIPTOR) で宣言する必要もあります。

namespace_declare

ユーザー・トークンに加えて、次の 4 つの引数が渡されます。

1. 名前空間接頭部のアドレスを示す BYVALUE POINTER
2. 名前空間接頭部の長さを示す BYVALUE FIXED BIN(31)
3. 名前空間 URI のアドレスを示す BYVALUE POINTER
4. 名前空間 URI の長さを示す BYVALUE FIXED BIN(31)

content_characters

通常の 3 つのパラメーターに加えて、次の 1 つの引数が渡されます。

- 以下の情報を示す BYVALUE ALIGNED BIT(8) フラグ・バイト:
  - 次のイベントに content-charcaters がさらに提供されるかどうか。これは、最初のビットがオンである場合は真です。つまり、このフィールドと '80'BX との AND 演算の結果が非ヌルである場合、これは真です。
  - XML に再び変換するときにエスケープしなければならない文字が存在しないかどうか。これは、2 番目のビットがオンの場合は真です。つまり、このフィールドと '40'BX との AND 演算の結果が非ヌルである場合、これは真です。

なお、このエントリーは、OPTIONS(NODESCRIPTOR) で宣言する必要もあります。

end_of_input
ユーザー・トークンに加えて、次の 2 つの引数が渡されます。

1. 次の入力バッファーのアドレスを示す BYADDR POINTER
2. 次の入力バッファーの長さを示す BYADDR FIXED BIN(31)

processing_instruction
ユーザー・トークンに加えて、次の 5 つの追加引数が渡されます。

1. ターゲット・テキストのアドレスを示す BYVALUE POINTER
2. ターゲット・テキストの長さを示す BYVALUE FIXED BIN(31)
3. データ・テキストのアドレスを示す BYVALUE POINTER
4. データ・テキストの長さを示す BYVALUE FIXED BIN(31)
5. 以下の情報を示す BYVALUE ALIGNED BIT(8) フラグ・バイト:
   - 次のイベントに content-charcaters がさらに提供されるかどうか。これは、最初のビットがオンである場合は真です。つまり、このフィールドと '80'BX との AND 演算の結果が非ヌルである場合、これは真です。
   - XML に再び変換するときにエスケープしなければならない文字が存在しないかどうか。これは、2 番目のビットがオンの場合は真です。つまり、このフィールドと '40'BX との AND 演算の結果が非ヌルである場合、これは真です。

   なお、このエントリーは、OPTIONS(NODESCRIPTOR) で宣言する必要もあります。

comment
通常の 3 つのパラメーターに加えて、次の 1 つの引数が渡されます。

- 以下の情報を示す BYVALUE ALIGNED BIT(8) フラグ・バイト:
  - 次のイベントに content-charcaters がさらに提供されるかどうか。これは、最初のビットがオンである場合は真です。つまり、このフィールドと '80'BX との AND 演算の結果が非ヌルである場合、これは真です。
  - XML に再び変換するときにエスケープしなければならない文字が存在しないかどうか。これは、2 番目のビットがオンの場合は真です。つまり、このフィールドと '40'BX との AND 演算の結果が非ヌルである場合、これは真です。

なお、このエントリーは、OPTIONS(NODESCRIPTOR) で宣言する必要もあります。

exception
ユーザー・トークンに加えて、次の 3 つの引数が渡されます。

1. 文書内での問題のテキストのバイト・オフセットを示す BYVALUE FIXED BIN(31)
2. 例外の戻りコードを示す BYVALUE FIXED BIN(31)
3. 例外の理由コードを示す BYVALUE FIXED BIN(31)

イベントにおける差異

以下のイベントは、PLISAXA および PLISAXB イベント構造体の一部ですが、PLISAXC および PLISAXD には存在しません。

- attribute_predefined_reference
- attribute_character_reference
- content_predefined_reference
- content_character_reference
- unknown_attribute_reference
- unknown_content_reference
- start_of_prefix_mapping
- end_of_prefix_mapping

以下のイベントは、PLISAXA および PLISAXB イベント構造体の一部ですが、PLISAXC および PLISAXD には存在しません。

- namespace_declare
- unresolved_reference
- end_of_input

PLISAXA および PLISAXB、PLISAXC および PLISAXD に共通するイベントの一部では、(通常のユーザー・トークンを除いて) 異なるパラメーターが渡されます。

start_of_document
パラメーターが渡されない

start_of_element
名前空間データも渡される

end_of_element
名前空間データも渡される

attribute_name
名前空間データも渡される

attribute_characters
フラグ・バイトも渡される

start_of_cdata
パラメーターが渡されない

end_of_cdata
パラメーターが渡されない

content_characters
フラグ・バイトも渡される

processing_instruction
フラグ・バイトも渡される

comment
フラグ・バイトも渡される

exception
エラー ID の代わりに戻りコードおよび理由コードが渡される

XML 文書のコード化文字セット

PLISAXC 組み込みサブルーチンおよび PLISAXD 組み込みサブルーチンがサポートする XML 文書は、ユニコード UTF-16 でエンコードされた WIDECHAR の文書、または UTF-8 でエンコードされた CHARACTER の文書、またはこのセクションでリストされていて明示的にサポートされているいずれかの 1 バイト文字セットでエンコードされた CHARACTER の文書のみです。

パーサーは、XML 文書のエンコード方式に関する情報ソースを 3 つまで使用し、これらのソース間で矛盾を検出した場合は、次のように例外 XML イベントをシグナル通知します。

1. パーサーは、文書の最初の文字を検査することによって文書の基本エンコードを判別します。
2. ステップ 1 が正常に完了した場合、パーサーはエンコード宣言を検索します。
3. 最後に、パーサーは PLISAXC または PLISAXD 組み込みサブルーチン呼び出しのコード・ページ値を参照します。このパラメーターが省略された場合、デフォルトで使用される値は、明示指定またはデフォルトの CODEPAGE コンパイラー・オプションの値です。

XML 文書の最初に XML 宣言があり、その宣言に、いずれかのサポート対象コード・ページが指定されているエンコード宣言が含まれている場合に、そのエンコード宣言が、基本文書エンコード方式や、PLISAXC/PLISAXD 組み込みサブルーチンからのエンコード情報と矛盾しなければ、パーサーはそのエンコード宣言を受け入れます。 XML 文書に XML 宣言自体がない場合、または XML 宣言でエンコード宣言が省略されている場合、PLISAXC または PLISAXD 組み込みサブルーチンからのエンコード情報が基本文書エンコードと矛盾していなければ、パーサーはそのエンコード情報を使用して文書を処理します。

サポートされるコード・ページ

次の表で、最初の番号はユーロ国別拡張コード・ページ (ECECP)、2 番目の番号は国別拡張コード・ページ (CECP) のものです。

CCSID	説明
01208	ユニコード UTF-8
01047	Latin 1/オープン・システム
01140、00037	米国、カナダなど
01141、00273	オーストリア、ドイツ
01142、00277	デンマーク、ノルウェー
01143、00278	フィンランド、スウェーデン
01144、00280	イタリア
01145、00284	スペイン、ラテンアメリカ (スペイン語)
01146、00285	英国
01147、00297	フランス
01148、00500	国際
01149、00871	アイスランド

コード・ページの指定

文書の XML 宣言にエンコード宣言が含まれていない場合、または XML 宣言自体が存在しない場合、パーサーは、PLISAXC または PLISAXD 組み込みサブルーチン呼び出しで提供されるエンコード情報を文書の基本エンコードと併せて使用します。

ほとんどの XML 文書の最初にある XML 宣言の中で、文書のエンコード情報を指定することもできます。 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"?>
```

XML 文書にエンコード宣言がある場合は、そのエンコード宣言が、PLISAXC または PLISAXD 組み込みサブルーチンから提供されるエンコード情報および文書の基本エンコードと整合していることを確認してください。エンコード宣言、PLISAXC または PLISAXD 組み込みサブルーチンから提供されるエンコード情報、および文書の基本エンコードの間に矛盾がある場合、パーサーは例外 XML イベントをシグナル通知します。

エンコード宣言は、番号または別名を使用して指定できます。

番号を使用してエンコード宣言を指定

次のいずれかの接頭部 (大文字と小文字の組み合わせは自由) を付けて CCSID 番号を (先行ゼロなしで、または任意の数の先行ゼロを付けて) 指定できます。

IBM_	CP	CCSID_
IBM-	CP_	CCSID-
	CP-	

別名を使用してエンコード宣言を指定

以下のサポート対象の別名 (大文字と小文字の組み合わせは自由) を任意に使用できます。

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

例外

例外イベントが発生した場合、そのイベントに渡される理由コードおよび戻りコードは、XML System Services パーサーからのものです。これらの戻りコードおよび理由コードの意味については、このパーサーに付属の資料に説明があります。

妥当性検査を伴う XML 文書の構文解析

PLISAXD 組み込みサブルーチンは、PLISAXC と同じ方法で XML 文書を解析するだけでなく、インバウンド XML 文書がインバウンド XML スキーマで指定されている一連の規則に従っているかどうかの判別も行います。

PLISAXD 組み込みサブルーチンを使用する場合、XML 妥当性検査に使用されるインバウンド・スキーマは、Optimized Schema Representation (OSR) と呼ばれるプリプロセス済みフォーマットでなければなりません。

以下のトピックでは、XML スキーマについて、および XML スキーマに対して OSR を構築する方法について説明します。XML 文書を解析および妥当性検査するための PLISAXD 組み込みサブルーチンの使用例については、 509 ページの『PLISAXD 組み込みサブルーチンの使用例』を参照してください。

XML スキーマ

XML スキーマは、W3C により定義された、XML 文書の構造と内容を記述および制約するためのメカニズムです。

データ型および名前空間のサポートを通じて、XML スキーマは、XML エレメントおよび属性の標準構造を提供することができます。そのため、XML スキーマ (それ自体が XML で表される) は、指定されたタイプ (在庫品目など) の XML 文書のクラスを効率的に定義できます。

次の XML 文書の例では、在庫管理の目的で品目を記述しています。

```
      '<?xml version="1.0" standalone="yes"?>'
   || '<!--Document for stock keeping example-->'
   || '<stockItem itemNumber="453-SR">'
   || '<itemName>Stainless steel rope thimbles</itemName>'
   || '<quantityOnHand>23</quantityOnHand>'
   || '<stockItem>';
```

在庫管理の文書例は、適切な形式であり、stock.xsd という次のスキーマに従って有効です。(各行の前にある番号はスキーマの一部ではなく、スキーマの後にある説明で使用します。)

```
1. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
2.
3. <xsd:element name="stockItem" type="stockItemType"/>
4.
5. <xsd:complexType name="stockItemType">
6.  <xsd:sequence>
7.   <xsd:element name="itemName" type="xsd:string" minOccurs="0"/>
8.   <xsd:element name="quantityOnHand">
9.    <xsd:simpleType>
10.    <xsd:restriction base="xsd:nonNegativeInteger">
11.      <xsd:maxExclusive value="100"/>
12.     </xsd:restriction>
13.    </xsd:simpleType>
14.   </xsd:element>
15.  </xsd:sequence>
16.  <xsd:attribute name="itemNumber" type="SKU" use="required"/>
17. </xsd:complexType>
18.
19. <xsd:simpleType name="SKU">
20.  <xsd:restriction base="xsd:string">
21.   <xsd:pattern value="¥d{3}-[A-Z]{2}"/>
```

```
22.  </xsd:restriction>
23. </xsd:simpleType>
24.
25. </xsd:schema>
```

スキーマは、ルート・エレメントが stockItem であることを宣言しています (行 3)。このルート・エレメントには、SKU 型の必須属性 itemNumber があり (行 16)、次のような他のエレメントのシーケンスが含まれています (行 6 から 15)。

- オプションのストリング型の itemName エレメント (行 7)
- nonNegativeInteger 型に基づいた 1 から 99 までに制約された範囲内の必須エレメント quantityOnHand (行 8 から 14)

型宣言はインライン化し、無名化することができます (行 9 から 13)。これには、quantityOnHand エレメントに正しい値を指定するための maxExclusive ファセットが含まれています。

一方、itemNumber 属性については、名前付き型 SKU が別途宣言されています (行 19 から 23)。これには、正規表現構文を使用して、その型の正しい値が 3 桁の数字、ハイフン (負符号)、2 つの大文字がこの順序で構成されるように指定する、パターン・ファセットが含まれています。

OSR の作成

テキスト形式のスキーマから OSR 形式のスキーマを生成するには、z/OS UNIX コマンド **xsdosrg** を使用します。このコマンドは、z/OS UNIX システム・サービスで提供される OSR ジェネレーターを呼び出します。

例えば、stock.xsd ファイル内のテキスト形式のスキーマを stock.osr ファイル内のプリプロセス済み形式のスキーマに変換するには、次の z/OS UNIX コマンドを使用します。

```
xsdosrg -v –o /u/HLQ/xml/stock.osr  /u/HLQ/xml/stock.xsd
```

/u/HLQ/xml/ は、stock.osr ファイルおよび stock.xsd ファイルがあるディレクトリーです。

生成された OSR ファイルを PDS にコピーする場合は、z/OS UNIX の **cp** コマンドを使用します。 MVS データ・セット名を指定するには、名前の前にダブルスラッシュ (//) を付けます。 例えば、HFS ファイル stock.osr を、HLQ.XML.OSR という名前の固定長ブロック・レコード・フォーマットの PDS (レコード長 80) にコピーするには、次のコマンドを使用します。

```
cp –p /u/HLQ/xml/stock.osr "//'HLQ.XML.OSR(STOCK)'"
```

PDS の完全修飾名を省略するには、次のように cp ステートメントにおける単一引用符を省きます。

```
cp –p /u/HLQ/xml/stock.osr "//XML.OSR(STOCK)"
```

詳しくは、「*XML System Services* ユーザーズ・ガイドおよび解説書」を参照してください。

単純な文書での例

このセクションには、PLISAXC 組み込みサブルーチンおよび PLISAXD 組み込みサブルーチンの使用方法を示す 2 つの例が含まれています。

PLISAXC 組み込みサブルーチンの使用例

ここでは、PLISAXC 組み込みサブルーチンの使用について例を交えて説明します。

次の例では、 488 ページの図 101 に示されているサンプル XML 文書が使用されています。この例では名前空間は使用されていません。 PLISAXC が初めて呼び出されたときにすべての入力が渡されます (その結果、end_of_input イベントは呼び出されません)。

```
saxtest: package exports(saxtest);

  define alias event
      limited entry( pointer, pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_with_flag
      limited entry( pointer, pointer, fixed bin(31),
                     bit(8) aligned )
      returns( byvalue fixed bin(31) )
      options( nodescriptor byvalue linkage(optlink) );

  define alias event_with_namespace
      limited entry( pointer, pointer, fixed bin(31),
                     pointer, fixed bin(31),
                     pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_without_data
      limited entry( pointer )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_pi
      limited entry( pointer, pointer, fixed bin(31),
                     pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_namespace_dcl
      limited entry( pointer, pointer, fixed bin(31),
                     pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_exception
      limited entry( pointer, fixed bin(31),
                     fixed bin(31),
                     fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_end_of_input
      limited entry( pointer,
                     pointer byaddr,
                     fixed bin(31) byaddr )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );
```

図 *102. PLISAXC* のコーディング例 - 型宣言

```
saxtest: proc options( main );


  dcl
    1 eventHandler static

      ,2 e01 type event_without_data
             init( start_of_document )

      ,2 e02 type event
             init( version_information )

      ,2 e03 type event
             init( encoding_declaration )

      ,2 e04 type event
             init( standalone_declaration )

      ,2 e05 type event
             init( document_type_declaration )

      ,2 e06 type event_without_data
             init( end_of_document )

      ,2 e07 type event_with_namespace
             init( start_of_element )

      ,2 e08 type event_with_namespace
             init( attribute_name )

      ,2 e09 type event_with_flag
             init( attribute_characters )

      ,2 e10 type event_with_namespace
             init( end_of_element )

      ,2 e11 type event_without_data
             init( start_of_CDATA )

      ,2 e12 type event_without_data
             init( end_of_CDATA )

      ,2 e13 type event_with_flag
             init( content_characters )

      ,2 e14 type event_pi
             init( processing_instruction )

      ,2 e15 type event_with_flag
             init( comment )

      ,2 e16 type event_namespace_dcl
             init( namespace_declare )

      ,2 e17 type event_end_of_input
             init( end_of_input )

      ,2 e18 type event
             init( unresolved_reference )

      ,2 e19 type event_exception
             init( exception )
      ;
```

図 *103. PLISAXC* のコーディング例 - イベント構造体

```
   dcl token       char(8);

   dcl xmlDocument char(4000) var;

   xmlDocument =
      '<?xml version="1.0" standalone="yes"?>'
   || '<!--This document is just an example-->'
   || '<sandwich>'
   || '<baker type="baker"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>'
   || ' ';


   call plisaxc( eventHandler,
                 addr(token),
                 addrdata(xmlDocument),
                 length(xmlDocument) );

end;
```

図 *104. PLISAXC* のコーディング例 - メインルーチン

```
dcl chars char(32000) based;

start_of_document:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' length=' || tokenlength );

  return(0);
end;

version_information:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

encoding_declaration:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;

standalone_declaration:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  return(0);
end;
```

図 *105. PLISAXC* のコーディング例 - イベント・ルーチン

```
  document_type_declaration:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  namespace_declare:
    proc( userToken, nsPrefix, nsPrefixLength,
                     nsUri, nsUriLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl nsPrefix       pointer;
    dcl nsPrefixLength fixed bin(31);
    dcl nsUri          pointer;
    dcl nsUriLength    fixed bin(31);

    put skip list( lowercase( procname() ) );
    put skip list( 'prefix = '
     || ' <' || substr(nsPrefix->chars,1,nsPrefixlength ) || '>' );
    put skip list( 'Uri = '
     || ' <' || substr(nsUri->chars,1,nsUrilength ) || '>' );

    return(0);
  end;

  end_of_document:
    proc( userToken )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;

    put skip list( lowercase( procname() ) );

    return(0);
  end;
```

PLISAXC のコーディング例 - イベント・ルーチン *(続き)*

```
  start_of_element:
    proc( userToken, xmlToken, TokenLength,
          nsPrefix, nsPrefixLength,
          nsUri, nsUriLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);
    dcl nsPrefix       pointer;
    dcl nsPrefixLength fixed bin(31);
    dcl nsUri          pointer;
    dcl nsUriLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  attribute_name:
    proc( userToken, xmlToken, TokenLength,
          nsPrefix, nsPrefixLength,
          nsUri, nsUriLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);
    dcl nsPrefix       pointer;
    dcl nsPrefixLength fixed bin(31);
    dcl nsUri          pointer;
    dcl nsUriLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;

  end_of_element:
    proc( userToken, xmlToken, TokenLength,
          nsPrefix, nsPrefixLength,
          nsUri, nsUriLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);
    dcl nsPrefix       pointer;
    dcl nsPrefixLength fixed bin(31);
    dcl nsUri          pointer;
    dcl nsUriLength    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;
```

PLISAXC のコーディング例 - イベント・ルーチン *(続き)*

```
content_characters:
  proc( userToken, xmlToken, TokenLength, flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl flags          bit(8) aligned;

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  if flags = ''b then;
  else
    put skip list( '!!flags = ' || flags );

  return(0);
end;

attribute_characters:
  proc( userToken, xmlToken, TokenLength, flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl flags          bit(8) aligned;

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  if flags = ''b then;
  else
    put skip list( '!!flags = ' || flags );

  return(0);
end;

start_of_CDATA:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;

  put skip list( lowercase( procname() ) );

  return(0);
end;

end_of_CDATA:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;

  put skip list( lowercase( procname() ) );

  return(0);
end;
```

PLISAXC のコーディング例 - イベント・ルーチン *(続き)*

```
  processing_instruction:
    proc( userToken,
          piTarget, piTargetLength,
          piData, piDataLength,
          flags )
    returns( byvalue fixed bin(31) )
    options( nodescriptor, byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl piTarget       pointer;
    dcl piTargetLength fixed bin(31);
    dcl piData         pointer;
    dcl piDataLength   fixed bin(31);
    dcl flags          bit(8) aligned;

    put skip list( lowercase( procname() )
     || ' <' || substr(piTarget->chars,1,piTargetLength ) || '>' );

    if flags = ''b then;
    else
      put skip list( '!!flags = ' || flags );

    return(0);
  end;

  comment:
    proc( userToken, xmlToken, TokenLength, flags )
    returns( byvalue fixed bin(31) )
    options( nodescriptor, byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);
    dcl flags          bit(8) aligned;

    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

if flags = ''b then;
    else
      put skip list( '!!flags = ' || flags );

    return(0);
  end;

  unresolved_reference:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( nodescriptor, byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);


    put skip list( lowercase( procname() )
     || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

    return(0);
  end;
```

PLISAXC のコーディング例 - イベント・ルーチン *(続き)*

```
  exception:
    proc( userToken, currentOffset, return_code, reason_code )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl currentOffset  fixed bin(31);
    dcl return_code    fixed bin(31);
    dcl reason_code    fixed bin(31);

    put skip list( lowercase( procname() )
     || ' return_code =' || hex(return_code)
     || ', reason_code =' || hex(reason_code)
     || ', offset =' || currentOffset );

    return(0);
  end;

  end_of_input:
    proc( userToken, addr_xml, length_xml )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl addr_xml       byaddr pointer;
    dcl length_xml     byaddr fixed bin(31);

    return(1);
  end;
end;
```

PLISAXC のコーディング例 - イベント・ルーチン *(*続き*)*

前のプログラムで生成される出力は次のとおりです。

```
start_of_document
version_information <1.0>
standalone_declaration <yes>
comment <This document is just an example>
start_of_element <sandwich>
prefix =  <>
Uri =  <>
start_of_element <bread>
prefix =  <>
Uri =  <>
attribute_name <type>
prefix =  <>
Uri =  <>
attribute_characters <baker"s best>
end_of_element <bread>
prefix =  <>
Uri =  <>
processing_instruction <spread>
piData =  <please use real mayonnaise >
start_of_element <meat>
prefix =  <>
Uri =  <>
content_characters <Ham & turkey>
end_of_element <meat>
prefix =  <>
Uri =  <>
start_of_element <filling>
prefix =  <>
Uri =  <>
content_characters <Cheese, lettuce, tomato, etc.>
!!flags = 01000000
end_of_element <filling>
prefix =  <>
Uri =  <>
start_of_cdata
content_characters <We should add a <relish> element in future >
end_of_cdata
end_of_element <sandwich>
prefix =  <>
Uri =  <>
end_of_document
```

図 *106. PLISAXC* のコーディング例 - プログラム出力

PLISAXD 組み込みサブルーチンの使用例

ここでは、PLISAXD 組み込みサブルーチンの使用について例を交えて説明します。

次の例では、 488 ページの図 101 に示されているサンプル XML 文書と、 499 ページの『PLISAXC 組み込みサブルーチンの使用例』 にある例に記述されている XML スキーマが使用されています。

この例には、同じ `stock.osr` スキーマに対する 8 個の XML ファイルの妥当性検査が含まれています。 次の出力で、スキーマに対する妥当性検査が失敗した `saxdtest` プログラム内の XML 文書を確認できます。

PLISAXD 組み込みサブルーチンでは、XML スキーマ・ファイルをバッファーに読み取る必要があります。次の例の OSR ファイルは PDS 内にあります。 OSR バ

ッファーの初期サイズは 4096 に設定されています。それより大きいサイズの OSR ファイルがある場合は、それに応じて OSR バッファーの初期サイズを大きくすることができます。

インバウンド・スキーマ・ファイルが HFS ファイル内にある場合は、次のコードを使用して、OSR ファイルをバッファーへ読み取ることができます。

```
dcl osrin file input stream environment(u);
dcl fileddint builtin;
dcl fileread builtin;

/* Read the HFS OSR file into buffer*/

open file(osrin);
osr_length = fileddint( osrin, 'filesize');
osr_ptr = allocate(osr_length);
rc = fileread(osrin,osr_ptr,osr_length);
```

PDS 内の OSR を使用してプログラムを実行するには、JCL で次の DD ステートメントを指定します。

```
//OSRIN DD DSN=HLQ.XML.OSR(STOCK),DISP=SHR
```

関連付けられた ddname OSRIN が HFS ファイルである場合は、代わりに次の JCL ステートメントを使用します。

```
//OSRIN DD PATH="/u/HLQ/xml/stock.osr"
```

```
saxdtest: package exports(saxdtest);
  /*************************************************************/
  /* saxdtest: Test PL/I XML validation support                */
  /* expected output:                                          */
  /*                                                           */
  /*   SAXDTEST: PL/I XML Validation sample                    */
  /*   SAXDTEST: Document Successfully parsed                  */
  /*   SAXDTEST: Document Successfully parsed                  */
  /*   Invalid: missing attribute itemNumber.                  */
  /*   exception return_code =00000018, reason_code =8613      */
  /*   Invalid: unexpected attribute warehouse.                */
  /*   exception return_code =00000018, reason_code =8612      */
  /*   Invalid: illegal attribute value 123-Ab.                */
  /*   exception return_code =00000018, reason_code =8809      */
  /*   Invalid: missing element quantityOnHand.                */
  /*   exception return_code =00000018, reason_code =8611      */
  /*   Invalid: unexpected element comment.                    */
  /*   exception return_code =00000018, reason_code =8607      */
  /*   Invalid: out-of-range element value 100                 */
  /*   exception return_code =00000018, reason_code =8803      */
  /*                                                           */
  /*************************************************************/

  define alias event
      limited entry( pointer, pointer, fixed bin(31) )
      returns( byvalue fixed bin(31) )
      options( byvalue linkage(optlink) );

  define alias event_with_flag
      limited entry( pointer, pointer, fixed bin(31),
                     bit(8) aligned )
      returns( byvalue fixed bin(31) )
      options( nodescriptor byvalue linkage(optlink) );
```

図 *107. PLISAXD* のコーディング例 - イベント・ルーチン

```
    define alias event_with_namespace
        limited entry( pointer, pointer, fixed bin(31),
                       pointer, fixed bin(31),
                       pointer, fixed bin(31) )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );

    define alias event_without_data
        limited entry( pointer )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );

    define alias event_pi
        limited entry( pointer, pointer, fixed bin(31),
                       pointer, fixed bin(31) )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );

    define alias event_namespace_dcl
        limited entry( pointer, pointer, fixed bin(31),
                       pointer, fixed bin(31) )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );

    define alias event_exception
        limited entry( pointer, fixed bin(31),
                       fixed bin(31),
                       fixed bin(31) )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );

    define alias event_end_of_input
        limited entry( pointer,
                       pointer byaddr,
                       fixed bin(31) byaddr )
        returns( byvalue fixed bin(31) )
        options( byvalue linkage(optlink) );
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
saxdtest: proc options( main );

  dcl
    1 eventHandler static

      ,2 e01 type event_without_data
             init( start_of_document )

      ,2 e02 type event
             init( version_information )

      ,2 e03 type event
             init( encoding_declaration )

      ,2 e04 type event
             init( standalone_declaration )

      ,2 e05 type event
             init( document_type_declaration )

      ,2 e06 type event_without_data
             init( end_of_document )

      ,2 e07 type event_with_namespace
             init( start_of_element )

      ,2 e08 type event_with_namespace
             init( attribute_name )

      ,2 e09 type event_with_flag
             init( attribute_characters )

      ,2 e10 type event_with_namespace
             init( end_of_element )

      ,2 e11 type event_without_data
             init( start_of_CDATA )

      ,2 e12 type event_without_data
             init( end_of_CDATA )

      ,2 e13 type event_with_flag
             init( content_characters )

      ,2 e14 type event_pi
             init( processing_instruction )

      ,2 e15 type event_with_flag
             init( comment )

      ,2 e16 type event_namespace_dcl
             init( namespace_declare )

      ,2 e17 type event_end_of_input
             init( end_of_input )

      ,2 e18 type event
             init( unresolved_reference )

      ,2 e19 type event_exception
             init( exception )
      ;
```

PLISAXD のコーディング例 - イベント・ルーチン *(*続き*)*

```
dcl token       char(45);

dcl rc              fixed bin(31);
dcl i               fixed bin(31);
dcl xml_document(8) char(300) var;
dcl xml_valid_msg(8) char(45) var;
dcl osr_ptr         pointer;
dcl record          char(80);
dcl osr_index       fixed bin;
dcl osr_buf_tail    fixed bin;
dcl temp_osr        pointer;
dcl buf_size        fixed bin(31) init(4096);
dcl rec_size        fixed bin(31);
dcl osr_length      fixed bin(31);
dcl buf_length      fixed bin(31);
dcl eof fixed       bin(31) init(0);
dcl osrin           file input;

on endfile(osrin) begin;
  eof = 1;
end;

/* read the entire PDS osr file into the buffer */

put skip list ('SAXDTEST: PL/I XML Validation sample ');

osr_length = buf_size;
osr_index = 0;
osr_buf_tail = 0;
rec_size = length(record);

osr_ptr = allocate(osr_length);

do while (eof = 0 );

  read file(osrin) into(record);
  osr_buf_tail += rec_size;
  if osr_buf_tail > buf_size then
    do;
      buf_length = osr_length;
      osr_length +=buf_size;
      temp_osr = allocate(osr_length);

      call plimove(temp_osr, osr_ptr, buf_length);
      call plifree(osr_ptr);
      osr_ptr = temp_osr;
      osr_buf_tail = rec_size;

      call plimove(osr_ptr+osr_index, addr(record), rec_size);
      osr_index += rec_size;
    end;
  else
    do;
      call plimove(osr_ptr+osr_index, addr(record), rec_size);
      osr_index  +=rec_size;
    end;

end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
    /* Valid XMLFILE */
    xml_document(1) = '<stockItem itemNumber="453-SR">'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '<quantityOnHand>23</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg(1) = 'Valid XMLFILE ';
    /* Valid: the ITEMNAME element can be omitted */
    xml_document(2) = '<stockItem  itemNumber="453-SR">'
    || '<quantityOnHand>23</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg(2) = 'Valid: the ITEMNAME element can be omitted.';

    /* Invalid: missing attribute itemNumber */
    xml_document(3) = '<stockItem>'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '<quantityOnHand>23</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg(3) = 'Invalid: missing attribute itemNumber.';

    /* Invalid: unexpected attribute warehouse */
    xml_document(4) = '<stockItem itemNumber="453-SR" warehouse="NY">'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '<quantityOnHand>23</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg4) = 'Invalid: unexpected attribute warehouse.';

    /* Invalid: illegal attribute value 123-Ab */
    xml_document(5) = '<stockItem itemNumber="123-Ab">'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '<quantityOnHand>23</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg(5) = 'Invalid: illegal attribute value 123-Ab.';

    /* Invalid: missing element quantityOnHand */
    xml_document(6) = '<stockItem itemNumber="074-UN">'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '</stockItem>';

    xml_valid_msg(6) = 'Invalid: missing element quantityOnHand.';

    /* Invalid: unexpected element comment */
    xml_document(7) = '<stockItem itemNumber="453-SR">'
    || '<itemName>Stainless steel rope thimbles</itemName>'
    || '<quantityOnHand>1</quantityOnHand>'
    || '<commnet>Nylon bristles</comment>'
    || '</stockItem>';

    xml_valid_msg(7) = 'Invalid: unexpected element comment.';

    /* Invalid: out-of-range element value 100 */
    xml_document(8) = '<stockItem itemNumber="123-AB">'
    || '<itemName>Paintbrush</itemName>'
    || '<quantityOnHand>100</quantityOnHand>'
    || '</stockItem>';

    xml_valid_msg(8) = 'Invalid: out-of-range element value 100';
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
    do i = 1 to hbound(xml_document);;
      token = xml_valid_msg(i);
      call plisaxd( eventHandler,
                    addr(token),
                    addrdata(xml_document(i)),
                    length(xml_document(i)),
                    osr_ptr,
                    37 );
    end;

    close file(osrin);
    call plifree(osr_ptr);

  end;

  start_of_document:
    proc( userToken )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;

    return(0);
  end;

  version_information:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);


    return(0);
  end;

  encoding_declaration:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);


    return(0);
  end;

  standalone_declaration:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);

    return(0);
  end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
  document_type_declaration:
    proc( userToken, xmlToken, TokenLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);


    return(0);
  end;

  end_of_document:
    proc( userToken )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;

    put skip list( 'SAXDTEST: Document Successfully parsed ');

    return(0);
  end;

  start_of_element:
    proc( userToken, xmlToken, TokenLength,
          nsPrefix, nsPrefixLength,
          nsUri, nsUriLength )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl xmlToken       pointer;
    dcl tokenLength    fixed bin(31);
    dcl nsPrefix       pointer;
    dcl nsPrefixLength fixed bin(31);
    dcl nsUri          pointer;
    dcl nsUriLength    fixed bin(31);

    return(0);
  end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
attribute_name:
  proc( userToken, xmlToken, tokenLength,
        nsPrefix, nsPrefixLength,
        nsUri, nsUriLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl nsPrefix       pointer;
  dcl nsPrefixLength fixed bin(31);
  dcl nsUri          pointer;
  dcl nsUriLength    fixed bin(31);

  return(0);
end;

attribute_characters:
  proc( userToken, xmlToken, TokenLength, flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl flags          bit(8) aligned;

  put skip list( lowercase( procname() )
   ||' <' || substr(xmltoken->chars,1,tokenlength) || '>' );

  if flags = ''b then;
  else
    put skip list( '!!flags = ' || flags );

  return(0);
end;

end_of_element:
  proc( userToken, xmlToken, TokenLength,
        nsPrefix, nsPrefixLength,
        nsUri, nsUriLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl nsPrefix       pointer;
  dcl nsPrefixLength fixed bin(31);
  dcl nsUri          pointer;
  dcl nsUriLength    fixed bin(31);

  return(0);
end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
start_of_CDATA:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;


  return(0);
end;

end_of_CDATA:
  proc( userToken )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;

  return(0);
end;

content_characters:
  proc( userToken, xmlToken, TokenLength, flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl flags          bit(8) aligned;

  if flags = ''b then;
  else

  return(0);
end;

processing_instruction:
  proc( userToken,
        piTarget, piTargetLength,
        piData, piDataLength,
        flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl piTarget       pointer;
  dcl piTargetLength fixed bin(31);
  dcl piData         pointer;
  dcl piDataLength   fixed bin(31);
  dcl flags          bit(8) aligned;

  put skip list( lowercase( procname() )
   || ' <' || substr(piTarget->chars,1,piTargetLength ) || '>' );

  if flags = ''b then;
  else
    put skip list( '!!flags = ' || flags );

  return(0);
end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
comment:
  proc( userToken, xmlToken, TokenLength, flags )
  returns( byvalue fixed bin(31) )
  options( nodescriptor, byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);
  dcl flags          bit(8) aligned;

  put skip list( lowercase( procname() )
   || ' <' || substr(xmltoken->chars,1,tokenlength ) || '>' );

  if flags = ''b then;
  else
    put skip list( '!!flags = ' || flags );    return(0);
end;

namespace_declare:
  proc( userToken, nsPrefix, nsPrefixLength,
                   nsUri, nsUriLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl nsPrefix       pointer;
  dcl nsPrefixLength fixed bin(31);
  dcl nsUri          pointer;
  dcl nsUriLength    fixed bin(31);

  return(0);
end;

unresolved_reference:
  proc( userToken, xmlToken, TokenLength )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl xmlToken       pointer;
  dcl tokenLength    fixed bin(31);

  return(0);
end;

exception:
  proc( userToken, currentOffset, return_code, reason_code )
  returns( byvalue fixed bin(31) )
  options( byvalue linkage(optlink) );

  dcl userToken      pointer;
  dcl currentOffset  fixed bin(31);
  dcl return_code    fixed bin(31);
  dcl reason_code    fixed bin(31);
  dcl validmsg       char(45) based;

  put skip list( userToken -> validmsg);
  put skip list( lowercase( procname() )
   || ' return_code =' || hex(return_code)
   || ', reason_code =' || substr(hex(reason_code),5,4));

  return(0);
end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(続き)*

```
  end_of_input:
    proc( userToken, addr_xml, length_xml )
    returns( byvalue fixed bin(31) )
    options( byvalue linkage(optlink) );

    dcl userToken      pointer;
    dcl addr_xml       byaddr pointer;
    dcl length_xml     byaddr fixed bin(31);

    return(0);
  end;
```

PLISAXD のコーディング例 - イベント・ルーチン *(*続き*)*

次の出力に、サンプル・プログラムの結果が示されています。無効な文書の場合は、PLISAXD 組み込みサブルーチンが、リストされている戻りコードおよび理由コードを持つ XML 例外イベントを呼び出します。それぞれの戻りコードおよび理由コードについて詳しくは、「*XML System Services* ユーザーズ・ガイドおよび解説書」を参照してください。

```
SAXDTEST: PL/I XML Validation sample
SAXDTEST: Document Successfully parsed
SAXDTEST: Document Successfully parsed
Invalid: missing attribute itemNumber.
exception return_code =00000018, reason_code =8613
Invalid: unexpected attribute warehouse.
exception return_code =00000018, reason_code =8612
Invalid: illegal attribute value 123-Ab.
exception return_code =00000018, reason_code =8809
Invalid: missing element quantityOnHand.
exception return_code =00000018, reason_code =8611
Invalid: unexpected element comment.
exception return_code =00000018, reason_code =8607
Invalid: out-of-range element value 100
exception return_code =00000018, reason_code =8803
```

図 *108. PLISAXD* サンプルからの出力

第 21 章 PLIDUMP の用法

このセクションでは、PLIDUMP を呼び出すのに使用されるダンプ・オプションと構文を記載し、ダンプに含まれていてユーザーがユーザー・ルーチンをデバッグする際に役立つ PL/I に固有の情報について説明します。

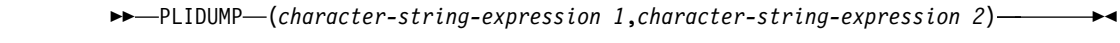
注: PLIDUMP は各国語サポートの標準に準拠しています。

図 109 に、z/OS 言語環境プログラム・ダンプを作成するために PLIDUMP を呼び出す PL/I ルーチンの例を示します。 この例では、メインルーチン PLIDMP が PLIDMPA を呼び出し、次にそれが PLIDMPB を呼び出します。 PLIDUMP の呼び出しは、ルーチン PLIDMPB 内で行います。

```
%PROCESS MAP SOURCE STG LIST OFFSET LC(101);
 PLIDMP: PROC  OPTIONS(MAIN) ;

   Declare   (H,I) Fixed bin(31) Auto;
   Declare   Names Char(17) Static init('Bob Teri Bo Jason');
   H = 5;  I = 9;
   Put skip list('PLIDMP Starting');
   Call PLIDMPA;

     PLIDMPA:  PROC;
       Declare  (a,b)  Fixed bin(31)   Auto;
       a = 1;  b = 3;
       Put skip list('PLIDMPA Starting');
       Call PLIDMPB;

         PLIDMPB:  PROC;
           Declare  1 Name  auto,
                      2  First   Char(12) Varying,
                      2  Last    Char(12) Varying;
           First = 'Teri';
           Last  = 'Gillispy';
           Put skip list('PLIDMPB Starting');
           Call PLIDUMP('TBFC','PLIDUMP called from procedure PLIDMPB');
           Put Data;
         End PLIDMPB;

     End PLIDMPA;

 End PLIDMP;
```

図 *109. PLIDUMP* を呼び出す *PL/I* ルーチンの例

PLIDUMP の構文とオプションは次のとおりです。



character-string-expression 1
次の 1 つ以上のオプションからなるダンプ・オプション文字ストリング。

A マルチタスキング・プログラムのすべてのタスクに関連する要求情報。

© Copyright IBM Corp. 1999, 2017

B BLOCKS (PL/I 16 進ダンプ)。

C 続行する。 ルーチンはダンプの完了後、続行されます。

E マルチタスキング・プログラムの現行タスクからの出口。 そのプログラムは、要求されたダンプの完了後も実行し続けます。

F FILES。

H STORAGE。

これには、すべての言語環境プログラム・ストレージが含まれます。したがって、ALLOCATE ステートメントを介して獲得されるすべての ALLOCATE ストレージおよび CONTROLLED ストレージが含まれます。

注: CEESNAP の DD 名は、PL/I ルーチンのスナップ・ダンプを作成できるよう、H オプションを使用して指定してください。ただし、これが省略されても、言語環境プログラムはメッセージを出しますが、極めて有用な情報を含むダンプを生成します。

K BLOCKS (CICS 下での実行時)。トランザクション作業域が組み込まれます。

NB NOBLOCKS。

NF NOFILES。

NH NOSTORAGE。

NK NOBLOCKS (CICS 下での実行時)。

NT NOTRACEBACK。

O マルチタスキング・プログラムの現行タスクに関連する唯一の情報。

S 停止する。 エンクレーブはダンプで終了します。

T TRACEBACK。

T、F、および C がデフォルト・オプションです。

character-string-expression 2
ダンプ・ヘッダーとして印刷される最長 80 文字のユーザー識別文字ストリング。

PLIDUMP の使用上の注意

PLIDUMP を使用する場合は、次の考慮事項が適用されます。

- ルーチンが PLIDUMP を何回か呼び出すような場合には、PLIDUMP 呼び出しごとに固有のユーザー ID を使ってください。 そうすると、各ダンプの開始を簡単に識別できます。
- DD 名が PLIDUMP、PL1DUMP、または CEEDUMP である DD ステートメントは、ダンプ用のデータ・セットの定義に使うことができます。
- PLIDUMP、PL1DUMP、または CEEDUMP DD ステートメントで定義されたデータ・セットでは、ダンプ・レコードの折り返しを防ぐために 133 以上の論理レコード長 (LRECL) を指定する必要があります。 これらのいずれかの DD 内でターゲットとして SYSOUT を使用する場合は、 MSGFILE

(SYSOUT,FBA,133,0) または MSGFILE(SYSOUT,VBA,137,0) を指定して、行が折り返されないようにする必要があります。

- PLIDUMP への呼び出し内に H オプションを指定するときには、 PL/I ライブラリーは、仮想記憶域のダンプを入手するため、OS SNAP マクロを出します。PLIDUMP の最初の起動で SNAP ID は 0 になります。それ以降の各呼び出しでは、ID が 1 ずつ、最大 256 まで増え、それから 0 にリセットされます。
- PLIDUMP を使ったスナップ・ダンプは、z/OS 下でのみサポートされます。スナップ・ダンプは、CICS 環境では作成されません。
  - SNAP が正常に完了しない場合は、CEE3DMP DUMP ファイルに次のメッセージが表示されます。

    ```
    Snap was unsuccessful
    ```

  - SNAP が正常に完了した場合は、CEE3DMP に次のメッセージが表示されます。

    ```
    Snap was successful; snap ID = nnn
    ```

    この *nnn* は、上記の SNAP ID に対応します。 SNAP が失敗した場合は、ID はインクリメントされません。
- プログラム単位名、プログラム単位アドレス、およびプログラム単位オフセットをダンプ・トレースバック・テーブルにおいて正しくリストしたい場合は、TEST(NONE,NOSYM) 以外のコンパイル時オプションを指定して PL/I プログラム単位をコンパイルします。例えば、TEST(NOSYM,NOHOOK,BLOCK) をオプションとして指定することができます。

各システム・プラットフォーム間での移植性を確保するには、PLIDUMP を使って PL/I ルーチンのダンプを生成します。

PLIDUMP 出力内の変数の検出

PLIDUMP 出力内で変数を検出するには、MAP オプションを指定してプログラムをコンパイルする必要があります。 MAP オプションを指定すると、AUTOMATIC または STATIC であるすべてのレベル 1 変数の AUTOMATIC および STATIC ストレージ内でのオフセットを示すテーブルが、コンパイラーによってリストに追加されます。

構造体内の 1 つのエレメントである変数を検出することは、AGGREGATE 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にも便利です。 このオプションを指定すると、プログラム内のすべての構造体のすべてのエレメントのオフセットを示すテーブルが、コンパイラーによってリストに追加されます。

AUTOMATIC 変数の検出

ダンプ内で AUTOMATIC 変数を検出するには、MAP オプション (および、必要なら AGGREGATE オプション) を指定した出力を使用して、AUTOMATIC 内でのオフセットを検出する必要があります。

PLIDUMP が B オプションで呼び出された場合、ダンプ出力には、ブロックごとに動的保存域 (DSA) の 16 進ダンプが含まれます。これが、そのブロックの自動ストレージになります。

例えば、以下の簡単なプログラムがあるとします。

```
Compiler Source
  Line.File
     2.0      test: proc options(main);
     3.0
     4.0        dcl a fixed bin(31);
     5.0        dcl b fixed bin(31);
     6.0
     7.0        on error
     8.0          begin;
     9.0            call plidump('TFBC');
    10.0          end;
    11.0
    12.0        a = 0;
    13.0        b = 29;
    14.0        b = 17 / a;
```

このプログラムに対するコンパイラーの MAP オプションの結果は、実際には右側にもう 1 つ列があること、列の間のスペースはもっと離れていることを除き、以下のようになります。

```
* * * * *  S T O R A G E  O F F S E T  L I S T I N G  * * * * *
 IDENTIFIER DEFINITION ATTRIBUTES

 A          1-0:4      Class = automatic,  Location = 160 : 0xA0(r13),
 B          1-0:5      Class = automatic,  Location = 164 : 0xA4(r13),
```

したがって、A はレジスター 13 の 16 進 A0 のオフセット位置にあり、B はレジスター 13 の 16 進 A4 のオフセット位置にあることになります (ここで、レジスター 13 は DSA を指しています)。

このプログラムでは PLIDUMP が B オプションを指定して呼び出されているため、現行の呼び出しチェーン内のそれぞれのブロックごとに、自動ストレージの 16 進ダンプが含まれています。これは以下のようになります (これも、右側の列が省略されています)。

```
Dynamic save area (TEST): 0AD963C8
      +000000 0AD963C8  10000000 0AD96188 00000000 00000000
      +000020 0AD963E8  00000000 00000000 00000000 00000000
      +000040 0AD96408  00000000 00000000 00000000 0AD96518
      +000060 0AD96428  00000000 00000000 00000000 00000000
      +000080 0AD96448 - +00009F 0AD96467   same as above
      +0000A0 0AD96468  00000000 0000001D 00100000 00000000
      +0000C0 0AD96488  0B300000 0A700930 0AD963C8 00000000
      +0000E0 0AD964A8  00000000 00000000 00000000 00000000
      +000100 0AD964C8  0AA47810 0A70E6D0 0AD96540 0AD960F0
      +000120 0AD964E8  00000001 0A70F4F8 0AD96318 00000000
      +000140 0AD96508  00000000 00000000 00000000 00000000
```

AUTOMATIC 内で、A は 16 進オフセット A0 のところにあり、B は 16 進オフセット A4 のところにあるので、ダンプでは、A と B がそれぞれ (予想どおりの) 16 進値である 00000000 と 0000001D になっていることを示しています。

コンパイラー・オプション OPT(2) および OPT(3) では、一部の変数 (特に FIXED BIN スカラー変数および POINTER スカラー変数) はストレージに割り振られないため、ダンプ出力に表示されないことがあります。

STATIC 変数の検出

RENT オプションを指定してコードをコンパイルした場合、静的変数は、現行ロード・モジュールの書き込み可能静的区域 (WSA) に配置されます。

WSA 内での変数のオフセットは、MAP オプションを指定した出力から検出することができ、WSA は CAA という言語環境プログラム制御ブロックの中に保持されます。 WSA の値は、言語環境プログラム・ダンプにもリストされます。

ただし、NORENT オプションを指定してコードをコンパイルした場合、EXTERNAL STATIC は通常どおり (リンカー・リストおよびコンパイラーの MAP オプションの出力を使用して) 検出されます。 INTERNAL STATIC は、言語環境プログラム・ダンプの一部としてダンプされます (PLIDUMP が B オプションを指定して呼び出された場合)。

旧 PL/I コンパイラーとは異なり、STATIC のアドレスがいずれか 1 つのレジスターを占有することはありません。

例えば、上記のプログラムで、変数を STATIC に変更したとします。

```
Compiler Source
  Line.File
     2.0      test: proc options(main);
     3.0
     4.0        dcl a fixed bin(31) static;
     5.0        dcl b fixed bin(31) static;
     6.0
     7.0        on error
     8.0          begin;
     9.0            call plidump('TFBC');
    10.0          end;
    11.0
    12.0        a = 0;
    13.0        b = 29;
    14.0        b = 17 / a;
```

プログラムが NORENT オプションでコンパイルされると、そのプログラムに対するコンパイラー MAP オプションの結果は以下のようになります。ただし、実際には、右側にもう 1 つの列があり、列と列の間はさらに離れています。

```
* * * * *  S T O R A G E  O F F S E T  L I S T I N G  * * * * *
 IDENTIFIER DEFINITION ATTRIBUTES

 A          1-0:4      Class = static,  Location = 0 : 0x0 + CSECT ***TEST2
 B          1-0:5      Class = static,  Location = 4 : 0x4 + CSECT ***TEST2
```

したがって、A はコンパイル単位 TEST の静的 CSECT の中の 16 進オフセット 00 の位置にあり、B は 16 進オフセット 04 の位置にあることになります。

このプログラムでは PLIDUMP が B オプションを指定して呼び出されているため、現行の呼び出しチェーン内のそれぞれのコンパイルごとに、静的ストレージの 16 進ダンプが含まれています。これは、以下のようになります (これも、右側の列が省略されています)。

```
Static for procedure TEST    Timestamp: 2004.08.12
      +000000 0FC00AA0  00000000 0000001D 0FC00DC8 0FC00AC0
      +000020 0FC00AC0  0FC00AA8 00444042 00A3AE01 0FC009C8
      +000040 0FC00AE0  6E3BFFE0 00000000 00000000 00000000
      +000060 0FC00B00  00000000 00000000 00000000 00000000
      +000000 0AD963C8  10000000 0AD96188 00000000 00000000
```

したがって、16 進オフセット 00 にある A は (予想どおりの) 16 進値 00000000 を持ち、16 進オフセット 04 にある B は (これも予想どおりの) 16 進値 0000001D、つまり 10 進値 29 を持つことになります。

CONTROLLED 変数の検出

CONTROLLED 変数は本質的に LIFO スタックであり、それぞれの CONTROLLED 変数は、そのスタックの先頭を指す「アンカー」を持っています。 CONTROLLED 変数を検出する秘訣はこのアンカーを検出することです。その場所はコンパイラー・オプションによって異なります。

CONTROLLED 変数に関するこの説明の残りの部分では、プログラム・ソースは 521 ページの図 109 にあるプログラムと同じプログラムですが、ストレージ・クラスが CONTROLLED に変更されています。

```
Compiler Source
  Line.File
     2.0      test: proc options(main);
     3.0
     4.0        dcl a fixed bin(31) controlled;
     5.0        dcl b fixed bin(31) controlled;
     6.0
     7.0        on error
     8.0          begin;
     9.0            call plidump('TFBHC');
    10.0          end;
    11.0
    12.0        allocate a, b;
    13.0        a = 0;
    14.0        b = 29;
    15.0        b = 17 / a;
```

NORENT WRITABLE を指定した場合

コンパイラー MAP オプションの結果は、この場合も、実際には右側にもう 1 つ列があること、列の間のスペースはもっと離れていることを除き、以下のようになります。

```
* * * * *  S T O R A G E  O F F S E T  L I S T I N G  * * * * *
 IDENTIFIER DEFINITION ATTRIBUTES

 A          1-0:4      Class = static,  Location = 8 : 0x8 + CSECT ***TEST2
 B          1-0:5      Class = static,  Location = 12 : 0xC + CSECT ***TEST2
```

これらの行は、(A や B 自体の場所ではなく) A および B のアンカーの場所を記述していることに注意してください。 そのため、A のアンカーはコンパイル単位 TEST の静的 CSECT 内の 16 進 08 の位置にあり、B のアンカーは 16 進 0C の位置にあります。

PLIDUMP が B オプションを指定して呼び出されている場合、現行の呼び出しチェーン内のそれぞれのコンパイルごとに、静的ストレージの 16 進ダンプが含まれています。 これは、以下のようになります (これも、右側の列が省略されています)。

```
Static for procedure TEST    Timestamp: . . ..

  +000000 0FC00A88  0FC00DB0 0FC00AA8 102B8A30 102B8A50
  +000020 0FC00AA8  0FC00A88 00444042 00A3AE01 0FC009B0
  +000040 0FC00AC8  6E3BFFE0 00000000 00000000 00000000
```

そのため、A のアンカーは 102B8A30 にあり、B のアンカーは 102B8A50 にあります。しかし、CONTROLLED 変数があるため、これらのストレージは ALLOCATE ステートメントを使用して取得されており、そのため、これらのアドレスはヒープ・ストレージ内を指しています。しかし、PLIDUMP が H オプションを指定して呼び出されている場合は、ヒープ・ストレージの 16 進ダンプが含まれることになります。これは、以下のようになります (これも、右側の列が省略されています)。

```
Enclave Storage:
  Initial (User) Heap
    +000000 102B7018  C8C1D5C3 0FC0F990 0FC0F990 00000000
      . . .
    +001A00 102B8A18  102B7018 00000020 0FC00A90 00000014
                      00000000 00000000 00000000 00000000
    +001A20 102B8A38  102B7018 00000020 0FC00A94 00000014
                      00000000 00000000 0000001D 00000000
```

A のアンカーが 102B8A30 にあったため、A は 16 進値 00000000 を持ちます。B のアンカーが 102B8A50 にあったため、B は (予想どおりの) 16 進値 0000001D を持ちます。

NORENT NOWRITABLE(FWS) を指定した場合

これらのオプションを指定したコンパイラー MAP オプションの結果は、この場合も、実際には右側にもう 1 つ列があること、列の間のスペースはもっと離れていることを除き、以下のようになります。

```
 * * * * *  S T O R A G E  O F F S E T  L I S T I N G  * * * * *
 IDENTIFIER DEFINITION ATTRIBUTES

 A          1-0:4      Class = automatic,  Location = 236 : 0xEC(r13)
 B          1-0:5      Class = automatic,  Location = 240 : 0xF0(r13)
```

注: これらのオプションを指定した場合は、CONTROLLED 変数の位置を指定するときに追加の間接指示のレベルがあるため、上記の行は A および B のアンカーのアドレスの位置を記しています。そのため、A のアンカーのアドレスはブロック TEST の AUTOMATIC 内の 16 進 EC の位置にあります。一方、B のアンカーは 16 進 F0 の位置にあります。

PLIDUMP が B オプションを指定して呼び出されているため、現行の呼び出しチェーン内のそれぞれのブロックごとに、自動ストレージの 16 進ダンプが含まれています。これは、以下のようになります (これも、右側の列が省略されています)。

```
 Dynamic save area (TEST): 102973C8
  +000000 102973C8  10000000 10297188 00000000 8FC007DA
          ....
  +0000E0 102974A8  0FC00998 00000000 00000000 102B8A40
                    102B8A28 10297030 102977D0 8FDF3D7E
```

したがって、A のアンカーのアドレスは 102B8A40 であり、B のアンカーのアドレスは 102B8A28 です。

PLIDUMP が H オプションも指定して呼び出されているので、ヒープ・ストレージの 16 進ダンプが含まれることになります。これは、以下のようになります (これも、右側の列が省略されています)。

```
Enclave Storage:
   Initial (User) Heap
     +000000 102B7018  C8C1D5C3 0FC0F990 0FC0F990 00000000
       . . .
     +001A00 102B8A18  102B7018 00000018 00000000 0FC00A78
                       102B8A80 00000000 102B7018 00000018
     +001A20 102B8A38  102B8A20 0FC00A74 102B8A60 00000000
                       102B7018 00000020 102B8A40 00000014
     +001A40 102B8A58  00000000 00000000 00000000 00000000
                       102B7018 00000020 102B8A28 00000014
     +001A60 102B8A78  00000000 00000000 0000001D 00000000
                       00000000 00000000 00000000 00000000
```

B のアンカーのアドレスが以前は 102B8A28 にあって、B のアンカーは現在は 102B8A80 にあるため、B は予想どおり 16 進値 0000001D (つまり 10 進値 29) を持ちます。

NORENT NOWRITABLE(PRV) を指定した場合

プログラムがこのオプションでコンパイルされたときの MAP リストは次のようになります。

```
* * * * *  S T O R A G E  O F F S E T  L I S T I N G  * * * * *
 IDENTIFIER DEFINITION ATTRIBUTES

 ***TEST3     1-0:4  Class = ext def,  Location = CSECT ***TEST3
 ***TEST4     1-0:5  Class = ext def,  Location = CSECT ***TEST4
_PRV_OFFSETS  1-0:1  Class = static,   Location = 8 : 0x8 + CSECT ***TEST2
```

ここでのキーは、この出力の最後の行です。 PRV_OFFSETS は、それぞれの CONTROLLED 変数についての PRV テーブル内でのオフセットを保持する静的テーブルです。 この静的テーブルは、MAP オプションが指定された場合にのみ生成されます。

このテーブルを解釈するために、コンパイラーは、ブロック名テーブルのすぐ後に、通常は小さな別のリストも生成します。このリストは、このサンプルでは、以下のようになります。

```
          PRV Offsets

   Number   Offset Name
        1        8 A
        1        C B
```

このテーブルには、名前が示された CONTROLLED 変数ごとに、ランタイム _PRV_OFFSETS テーブルでの 16 進オフセットがリストされます。 ブロック番号 (最初の列) を使用して、同じ名前であるが、異なるブロックに宣言されている変数を区別することができます。

_PRV_OFFSETS テーブルは静的ストレージ内 (16 進オフセット 8) にあり、PLIDUMP が B オプションを指定して呼び出されたため、以下に示すようなダンプ出力が表示されます。

```
Static for procedure TEST    Timestamp: . . ..
  +000000 10908EC8  02020240 00000005 6DD7D9E5 6DD6C6C6
                    00000000 00000004 D00000A0 00100000
  +000020 10908EE8  6E3BFFE0 00000000 00000000 00000000
                    00000000 90010000 00000000 00000000
```

したがって、PRV テーブル内の A のオフセットは 0 になり、PRV テーブル内の B のオフセットは 4 になります。_PRV_OFFSETS テーブルの最初の 8 バイトを占めている目印「_PRV_OFF」にも注目してください。

PRV テーブルは常に CAA 内のオフセット 4 の位置にあります。 PLIDUMP が H オプションで呼び出されたため、このテーブルはダンプ出力に含まれます。 CAA は次のようになります。

```
Control Blocks Associated with the Thread:
  CAA: 0A7107D0
   +000000 0A7107D0  00000800 0ADB7DE0 0AD97018 0ADB7018
                     00000000 00000000 00000000 00000000
```

したがって、PRV テーブルのアドレスは 0ADB7DE0 になり、ヒープ・ストレージ内のダンプ出力にも表示されます。

```
 Enclave Storage:
    Initial (User) Heap
      +000000 102B7018  C8C1D5C3 0FC0F990 0FC0F990 00000000
        . . .
      +000DC0 0ADB7DD8  00000000 00000000 0ADB8A38 0ADB8A58
                        0ADB7018 00000488 00000000 00000000
```

そのため、PRV テーブルには 0ADB8A38 0ADB8A58 などが含まれます。 _PRV_OFFSETS テーブルから導き出されるように、PRV テーブルまでの A のオフセットは 0 であり、B のオフセットは 4 であるため、これらはそれぞれ A と B のアドレスでもあります。

これらのアドレスは、ダンプ内のヒープ・ストレージにも表示されます。

```
  Enclave Storage:
    Initial (User) Heap
      +000000 102B7018  C8C1D5C3 0FC0F990 0FC0F990 00000000
        . . .
      +001A00 0ADB8A18  00000000 00000000 0ADB7018 00000020
                        00000000 00000014 0A7107D4 00000000
      +001A20 0ADB8A38  00000000 00000000 0ADB7018 00000020
                        00000004 00000014 0A7107D4 00000000
      +001A40 0ADB8A58  0000001D 00000000 00000000 00000000
                        00000000 00000000 00000000 00000000
```

したがって、A のアドレスは 0ADB8A38 であるため A の 16 進値は予想どおり 00000000 となり、B のアドレスは 0ADB8A58 であるため B の 16 進値も予想どおり 0000001D となります。

保存されたコンパイル・データ

コンパイルの間、コンパイラーはロード・モジュールにコンパイルに関するさまざまな情報を保存します。 この情報は、デバッグや将来のマイグレーション作業のときに大きな助けになることがあります。 このセクションでは、この保存された情報について述べます。

Copyright

COPYRIGHT コンパイラー・オプションを指定すると、コンパイラーは COPYRIGHT ストリングを、オブジェクト内のタイム・スタンプ・データの直前に配置される CHARACTER VARYING ストリングとして保存します。

このストリングの後にはブランクが必要なだけ続いて、ストリングにこれらのブランクを加えたものが 4 バイトの倍数を占めるようになります。

タイム・スタンプ

コンパイラーは各ロード・モジュールにタイム・スタンプ を保存します。タイム・スタンプは *YYYYMMDDHHMISSVNRNML* 形式の 20 バイト文字ストリングです。これにより、コンパイル日時のほか、このストリングを生成したコンパイラーのバージョンが記録されます。

ストリングのエレメントの意味は次のとおりです。

YYYY
コンパイルされた年

MM コンパイルされた月

DD コンパイルされた日

HH コンパイルされた時刻 (時)

MI コンパイルされた時刻 (分)

SS コンパイルされた時刻 (秒)

VN コンパイラーのバージョン番号

RN コンパイラーのリリース番号

ML コンパイラーの保守レベル

タイム・スタンプは、PPA2 で見付けることができます。PPA2 の中のオフセット 12 は 4 バイト整数で、PPA2 のアドレスからタイム・スタンプにオフセット (負の数のことがあります) を与えます。

次に、PPA2 は、PPA1 で見付けることができます。PPA1 の中のオフセット 4 は、4 バイト整数で、PPA1 に対応するエントリー・ポイント・アドレスから PPA2 にオフセット (負の数のことがあります) を与えます。

PPA1 は次のように位置指定できます。

- コードが LP(32) でコンパイルされた場合、PPA1 はブロックのエントリー・ポイント・アドレスから位置指定できます。エントリー・ポイント・アドレスからのオフセット 12 の位置には、そのエントリー・ポイント・アドレスから PPA1 までのオフセットを示す 4 バイト整数 (負の数の場合もある) があります。
- コードが LP(64) でコンパイルされた場合、PPA1 までのオフセットは、各エントリー・ポイント・アドレスの前にフルワードの 8 バイトで表されます。

保存されたオプション・ストリング

コンパイラーは、ロード・モジュールのビルドに使用されたコンパイラー・オプションが記録された 32 バイト・ストリングをロード・モジュールに格納します。

保存されたオプション・ストリングに対する宣言は、サンプル・データ・セット SIBMZSAM において組み込みファイル `ibmvsos` で指定されます。

構造体の中のほとんどのフィールドにおいて、そのフィールドの意味は名前によって明白ですが、いくつかのフィールドについてはいくらかの説明が必要です。

- sos_words は、構造体のバイト数を 4 で割った数値を保持しています。
- sos_version は、この構造体のバージョン番号です。 コンパイラーのバージョン番号ではありません。
- 構造体のサイズ、および設定されているフィールドは、バージョン番号によって異なります。

保存されたオプション・ストリングは、次の 2 つのどちらかの方法でタイム・スタンプの後に位置指定されます。

1. サービス・オプションが指定されている場合、サービス・オプションの中に指定されたストリングは、文字がストリングを変更すると即時にタイム・スタンプに続きます。次に、保存されたオプション・ストリングが、2 番目の文字がストリングを変更するとサービス・ストリングに続きます。
2. サービス・オプションが指定されていない場合、保存されたオプション・ストリングは、文字がストリングを変更すると即時にタイム・スタンプに続きます。

保存されたオプション・ストリングを保持する可変ストリングの長さは、保存されたオプション・ストリングそれ自体のサイズより長くなります。

サービス・ストリングの存在 (または不在) は、PPA2 内に、PPA2 の 10 進数のオフセット 20 にフラグ・バイトで示されます。このバイトと '20'BX の AND 演算結果がゼロでなければ、サービス・ストリングが存在します。

以前にリリースされた PL/I コンパイラーには、保存されたオプション・ストリングを、コンパイラーがロード・モジュールに置かないものがありました。 保存されたオプション・ストリングの存在 (または不在) は、PPA2 内に、10 進数のオフセット 20 にフラグ・バイトで示されます。このバイトと '02'BX の AND 演算結果がゼロでなければ、保存されたオプション・ストリングが存在します。

第 22 章 割り込みとアテンションの処理

PL/I プログラムにアテンション割り込みを認識させるには、次の 2 つの操作が可能でなければなりません。

- 割り込みを作成できなければなりません。 それを行う方法は、使用する端末とオペレーティング・システムによって異なります。
- ユーザーのプログラムで、割り込みに対応する準備が整っていなければなりません。 ATTENTION 条件が生じたときにユーザーのプログラムが制御を得るようにするために、ユーザーのプログラム内に ON ATTENTION ステートメントを作成しておくことができます。

  注: 呼び出したい ATTENTION ON ユニットがプログラムにある場合は、次のいずれかのオプションを使用してプログラムをコンパイルしなければなりません。

  – INTERRUPT オプション (TSO でのみサポートされる)
  – NOTEST または TEST(NONE,NOSYM) 以外の TEST オプション

    この方法でコンパイルすると、PLIXOPT で INTERRUPT(OFF) を明示的に指定していない限り、INTERRUPT(ON) が有効になります。

割り込みを作成する手順は、使用しているオペレーティング・システムと端末に関する IBM マニュアルに記載されています。

割り込み (オペレーティング・システムがユーザーの要求を認識すること) と、ATTENTION 条件の発生とは異なります。

割り込み とは、オペレーティング・システムが実行中のプログラムに通知するユーザーの要求のことです。 INTERRUPT コンパイル時オプションが指定されて PL/I プログラムがコンパイルされた場合は、プログラム内の個別の位置に内部割り込みスイッチをテストする命令が組み込まれます。 この内部割り込みスイッチは、ロード・モジュール内の任意のプログラムが INTERRUPT コンパイル時オプションを指定してコンパイルされた場合に設定することができます。

内部スイッチは、割り込み要求が出されたことをオペレーティング・システムが認識した時に設定されます。 特殊テスト命令 (ポーリング) の実行によって、ATTENTION 条件が発生します。 ポーリングが起きる前にデバッグ・ツール・フック (つまり CALL PLITEST) が検出されると、ATTENTION 条件の処理が始まる前にデバッグ・ツールに制御が与えられるようにすることができます。

ポーリングにより、ATTENTION 条件は、PL/I ステートメントの内部ではなく、ステートメントとステートメントの間で発生します。

534 ページの図 110 に、骨組みプログラム、ATTENTION ON ユニット、およびポーリング命令が生成されるいくつかの状況を示してあります。 プログラム内では、次の位置でポーリングが行われます。

- LABEL1
- DO の各反復

© Copyright IBM Corp. 1999, 2017

- ELSE PUT SKIP ... ステートメント
- ブロックの END ステートメント

```
%PROCESS INTERRUPT;
     .
     .
     .
 ON ATTENTION
   BEGIN;
     DCL X FIXED BINARY(15);
     PUT SKIP LIST ('Enter 1 to terminate, 0 to continue.');
     GET SKIP LIST (X);
     IF X = 1 THEN
       STOP;
     ELSE
       PUT SKIP LIST ('Attention was ignored');
   END;
     .
     .
     .
  LABEL1:
   IF EMPNO ...
     .
     .
     .
   DO I = 1 TO 10;
     .
     .
     .
   END;
     .
     .
     .
```

図 *110. ATTENTION ON* ユニットの使用

ATTENTION ON ユニットの使用

ATTENTION ON ユニット内の処理を使って、プログラム内の潜在的なエンドレス・ループを終了させることができます。

ATTENTION ON ユニットに制御が与えられるのは、割り込みが発生したことをポーリング命令が認識したときです。 通常、ON ユニットからの戻りは、ポーリング・コードに続くステートメントに対して行われます。

デバッグ・ツールとの対話

プログラム内で TEST(ALL) または TEST(ERROR) ランタイム・オプションが有効であるときに割り込みが生じると、次にフックが検出されたときに、デバッグ・ツールが制御を受け取ることになります。これは、割り込みが発生したことをプログラムのポーリング・コードが認識する前である可能性があります。

あとから ATTENTION 条件が発生すると、デバッグ・ツールは条件の処理のために再び制御を受け取ります。

第 23 章 チェックポイント/再始動機能の使用

この章では、バッチ環境で長時間稼働するプログラムの実行途中でチェックポイントをとるために役立つ手段である PL/I チェックポイント/再始動機能について説明します。

プログラム内の指定されたポイントで、プログラムの現状についての情報が、データ・セットにレコードとして書き込まれます。 システム障害が原因でプログラムが終了したときに、この情報を使えば、プログラムを全部実行し直す必要がなく、障害の発生地点の近くからプログラムを再始動することができます。

この再始動は、自動再始動または据え置き再始動のどちらもあり得ます。 自動再始動は、即時に実行される再始動です (ただし、システム・メッセージで要求されたときに、オペレーターがそれを許可することを前提とします)。 据え置き再始動は、新しいジョブとして、あとから実行される再始動です。

システム障害が起きていなくても、プログラム内から自動再始動を要求することができます。

PL/I チェックポイント/再始動は、オペレーティング・システムの拡張チェックポイント/再始動機能を使用します。 この機能については、 589 ページの『参考文献』に掲載したマニュアルを参照してください。

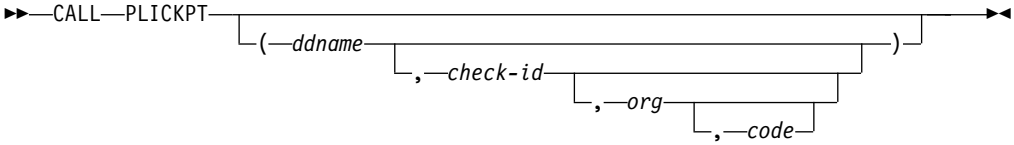
チェックポイント/再始動を使用するには、以下の操作を行う必要があります。

- ユーザーのプログラム内の適切なポイントで、チェックポイント・レコードを書き込むように要求します。 これは、組み込みサブルーチン PLICKPT を使って行います。
- チェックポイント・レコードを書き込めるデータ・セットを提供します。
- 期待どおりに再始動アクティビティーが行われるようにするには、EXEC ステートメントまたは JOB ステートメントで RD パラメーターを指定しなければならない場合があります (「*z/OS JCL Reference*」を参照してください)。

注: プログラムで使用するデータ・セットに関連した制約事項に注意する必要があります。 この制約事項の詳細については、 589 ページの『参考文献』を参照してください。

チェックポイント・レコードの要求

チェックポイント・レコードを書かせたいときには、そのたびに、 PL/I プログラムから組み込みサブルーチン PLICKPT を呼び出す必要があります。



© Copyright IBM Corp. 1999, 2017

4 つの引数はすべてオプショナルです。 引数を使わない場合は、所定の順序でその引数に続く別の引数を指定するとき以外は、引数を指定する必要はありません。 引数を指定する場合は、未使用引数をヌル・ストリング ('') として指定します。

ddname
文字ストリングの定数または変数で、チェックポイント・レコードに使用するデータ・セットを定義する DD ステートメントの名前を指定します。この引数を省略すると、システムはデフォルトの DD 名 SYSCHK を使用します。

check-id
あとからチェックポイント・レコードを識別できるようにするためにチェックポイント・レコードに割り当てる名前を指定する文字ストリングの定数または変数です。この引数を省略すると、システムは固有の ID を提供し、それをオペレーターのコンソールに印刷します。

org
値がオペレーティング・システム用語でチェックポイント・データ・セットの編成を示す属性 CHARACTER(2) を持つ文字ストリング定数または変数です。指定できる値は以下のとおりです。

PS 順次 (すなわち、CONSECUTIVE) 編成を指示します。

PO 区分編成を指示します。

この引数を省略すると、PS がとられます。

code
PLICKPT から戻りコードを受け取ることができる、属性 FIXED BINARY (31) を持つ変数です。戻りコードには、次の値があります。

0 チェックポイントは正常にとられた。

4 再始動は正常に行われた。

8 チェックポイントはとられていない。 PLICKPT ステートメントを調べる必要があります。

12 チェックポイントはとられていない。 欠落した DD ステートメント、ハードウェア・エラーがないか、またはデータ・セットのスペースが不十分でなかったかを調べてください。 REPLY オプションを指定した DISPLAY ステートメントが完了する前にチェックポイントをとろうとしても、失敗します。

16 チェックポイントはとられたが、ENQ マクロ呼び出しが未解決のままで、再始動時に復元されない。 このような事態は、通常、PL/I プログラムの場合は生じません。

チェックポイント・データ・セットの定義

チェックポイント・レコードを入れる先のデータ・セットを定義するには、ジョブ制御プロシージャー内に DD ステートメントを入れなければなりません。

このデータ・セットは、CONSECUTIVE 編成と区分編成のいずれでもかまいません。 任意の有効 DD 名を使用することができます。 DD 名 SYSCHK を使う場合は、PLICKPT を呼び出すときに DD 名を指定する必要はありません。

データ・セット名を指定しなければならないのは、据え置き再始動用にデータ・セットをとっておきたい場合だけです。 入出力装置は、直接アクセス装置であれば使用できます。

最後のチェックポイント・レコードだけを取得するには、状況を NEW (または、データ・セットが既に存在する場合は OLD) と指定します。これにより、各チェックポイント・レコードが直前のレコードを上書きします。

複数のチェックポイント・レコードを保存するには、状況を MOD と指定します。これで、各チェックポイント・レコードが直前のレコードのあとに続けて追加されます。

チェックポイント・データ・セットがライブラリーであれば、メンバー名として『check-id』が使用されます。 したがって、チェックポイントは、それ以前にとられた同一名を持つすべてのチェックポイントを削除します。

直接アクセス・ストレージの場合、保持しようとする最大数のチェックポイント・レコードを保管するのに十分な 1 次スペースを割り振らなければなりません。 増分スペース割り振りを指定することはできますが、ここでは使用されません。 チェックポイント・レコードはこのステップで割り振られる主記憶域の大きさより約 5000 バイト長くなっています。

DCB 情報は必要ありませんが、適切であれば次のどれでも組み込むことができます。

```
OPTCD=W, OPTCD=C, RECFM=UT
```

これらのサブパラメーターについては、「*z/OS MVS JCL* ユーザーズ・ガイド」を参照してください。

再始動の要求

再始動には、自動再始動または据え置き再始動があります。

自動再始動にできるのは、システム障害のあとか、またはプログラム内から行われる場合です。システム・オペレーターは、システムから要求があれば、すべての自動再始動を許可しなければなりません。

システム障害後の自動再始動

チェックポイントをとり終わってからシステム障害が生じた場合、EXEC ステートメントまたは JOB ステートメント内で RD=R を指定していた (または RD パラメーターを省略していた) ならば、最後にとられたチェックポイントで自動再始動が行われます。

チェックポイントをまだとっていないときにシステム障害が生じた場合に、EXEC ステートメントまたは JOB ステートメント内で RD=R を指定していたのであれば、その場合にも自動再始動が、ジョブ・ステップの初めから行われます。

システム障害が起きてからでも、EXEC ステートメントまたは JOB ステートメント内で RD=RNC を指定すれば、ジョブ・ステップの初めから自動再始動を強制実

行させることもできます。 別のシステム障害が生じても、RD=RNC を指定すれば、チェックポイントを処理しないで自動ステップ再始動を要求することになります。

プログラム内の自動再始動

再始動は、プログラム内の任意の点で要求できます。

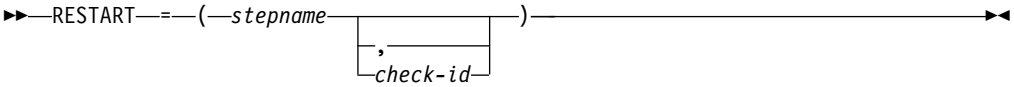
この再始動に関する規則は、システム障害後の再始動の場合と同じです。再始動を要求するには、次のステートメントを実行する必要があります。

```
CALL PLIREST;
```

再始動を行うために、コンパイラーは、システム完了コード 4092 でプログラムを異常終了させます。 したがって、この機能を使用するには、システム生成時に適格コードのテーブルからシステム完了コード 4092 を削除してはなりません。

据え置き再始動

自動再始動活動を取り消して、しかもそのチェックポイントを据え置き再始動に使えるように確保しておくには、プログラムの最初の実行時に、EXEC ステートメントまたは JOB ステートメント内で RD=NR を指定します。



あとから据え置き再始動が必要になった場合は、JOB ステートメントに RESTART パラメーターを指定して、そのプログラムを新しいジョブとして実行依頼する必要があります。 RESTART パラメーターを使って、再始動を開始するジョブ・ステップを指定します。また、チェックポイントから再始動を行うにはチェックポイント・レコード名を指定します。

チェックポイントからの再始動の場合、チェックポイント・レコードが入ったデータ・セットを定義する DD ステートメントも提供する必要があります。 この DD ステートメントには SYSCHK という名前でなければなりません。 DD ステートメントは、ジョブ・ステップの EXEC ステートメントの直前に挿入しなければなりません。

チェックポイント/再始動活動の変更

プログラム内に設定されているチェックポイントからの自動再始動アクティビティーを取り消すことができます。

自動再始動を取り消すには、次のステートメントを実行します。

```
CALL PLICANC;
```

ただし、JOB ステートメントや EXEC ステートメント内で RD=R または RD=RNC を指定していたのであれば、やはり自動再始動がジョブ・ステップの先頭から行われます。

また、以前にとられているチェックポイントがあれば、それをそのまま据え置き再始動に使用することができます。

JOB ステートメントまたは EXEC ステートメント内で RD=NC を指定すれば、プログラム内で要求したものであっても、任意の自動再始動およびチェックポイントを取る予定を取り消すことができます。

第 24 章 ユーザー出口の用法

PL/I にはユーザー出口がいくつか備わっています。そのユーザー出口を使用すれば、自分のニーズに合うように PL/I 製品をカスタマイズできます。

PL/I 製品は、デフォルト出口と、関連するソース・ファイルを提供します。

デフォルト出口で提供される機能とは異なる機能を出口で実行したい場合は、提供されているソース・ファイルを適宜変更することをお勧めします。

場合によっては、ユーザーの組織の要件を満たすようにコンパイラーを調整できることが役立つこともあります。 例えば、特定のメッセージを抑止したり、ほかのメッセージの重大度を変更したりする必要が生じることがあります。 コンパイルについての統計情報のログをファイルに書き込むなど、各コンパイルごとに特定の機能を実行するように指定する必要が生じることがあります。 コンパイラー・ユーザー出口は、この種の機能を処理します。

PL/I を使用すれば、独自のユーザー出口を作成することもできますし、製品に付属の出口を「そのまま」使用することも、変更して使用することもできます。 製品提供のユーザー出口ソース・コードを、 204 ページの図 21 に示しています。

この章で提供される情報は以下のとおりです。

- コンパイラー・ユーザー出口がサポートするプロシージャー
- コンパイラー・ユーザー出口を活動化する方法
- IBMUEXIT、IBM 提供のコンパイラー・ユーザー出口
- 独自のコンパイラー・ユーザー出口作成の要件

コンパイラー・ユーザー出口によって実行されるプロシージャー

コンパイラー・ユーザー出口は、3 つの特定のプロシージャー (初期化、コンパイラー・メッセージの代行受信およびフィルター操作、終了) を実行します。

542 ページの図 111 に示してあるように、コンパイラーは、初期化プロシージャー、メッセージ・フィルター・プロシージャー、および終了プロシージャーへ制御を渡します。 これらの 3 つのプロシージャーは、それぞれ、要求されたプロシージャーが完了したならば、制御をコンパイラーへ戻します。

© Copyright IBM Corp. 1999, 2017

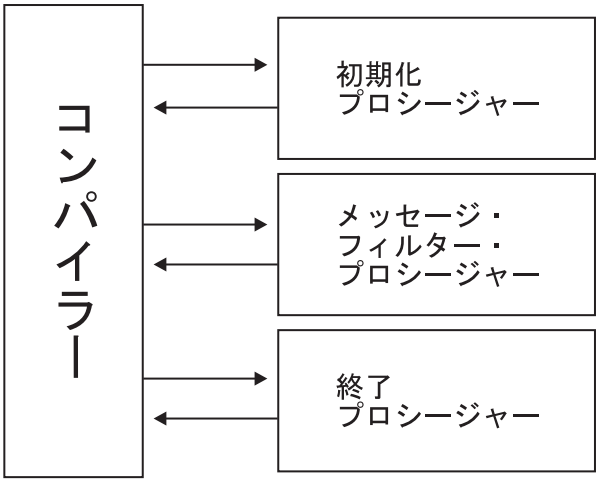


図 *111. PL/I* コンパイラー・ユーザー出口のプロシージャー

これらの各プロシージャーには、次の 2 つの制御ブロックが渡されます。

- コンパイルについての情報が含まれているグローバル制御ブロック。これは、最初のパラメーターとして渡されます。
- 2 番目のパラメーターとして渡される機能専用の制御ブロック。この制御ブロックの内容は、どのプロシージャーが呼び出されているかによって異なります。詳細については、 545 ページの『初期化プロシージャーの作成』、 546 ページの『メッセージ・フィルター操作プロシージャーの作成』、および 548 ページの『終了プロシージャーの作成』を参照してください。

関連情報:

『グローバル制御ブロックの構造』
グローバル制御ブロックは、3 つのユーザー出口プロシージャー (初期化、 フィルター操作、および終了) が呼び出されるたびに、それぞれに渡されます。

グローバル制御ブロックの構造

グローバル制御ブロックは、3 つのユーザー出口プロシージャー (初期化、フィルター操作、および終了) が呼び出されるたびに、それぞれに渡されます。

次のコードと説明は、グローバル制御ブロック内の各フィールドの内容を示すものです。

```
Dcl
  1 Uex_UIB           native based,
    2 Uex_UIB_Length         fixed bin(31),

    2 Uex_UIB_Exit_token     pointer,        /* for user exit's use */

    2 Uex_UIB_User_char_str  pointer,        /* to exit option str  */
    2 Uex_UIB_User_char_len  fixed bin(31),

    2 Uex_UIB_Filename_str   pointer,        /* to source filename  */
    2 Uex_UIB_Filename_len   fixed bin(31),

    2 Uex_UIB_return_code fixed bin(31),     /* set by exit procs   */
    2 Uex_UIB_reason_code fixed bin(31),     /* set by exit procs   */
```

```
2 Uex_UIB_Exit_Routs,                    /* exit entries set at
                                            initialization      */
  3 ( Uex_UIB_Termination,
      Uex_UIB_Message_Filter,           /* call for each msg   */
      *, *, *, * )
    limited entry (
        *,                               /* to Uex_UIB          */
        *,                               /* to a request area   */
     );
```

データ入力フィールド

Uex_UIB_ Length
バイト単位の制御ブロックの長さ。値は storage (Uex_UIB) です。

Uex_UIB_Exit_token
ユーザー出口プロシージャーによって使用されます。例えば、初期化では、メッセージ・フィルター・プロシージャーと終了プロシージャーの両方によって使われるデータ構造に設定できます。

Uex_UIB_User_char_str
オプショナルの文字ストリングを指す (ただし、指定した場合のみ)。例えば、pli filename (EXIT ('string'))...fn では、31 文字までの文字ストリングにすることができます。

Uex_UIB_char_len
User_char_str が指すストリングの長さ。この値はコンパイラーによって設定されます。

Uex_UIB_Filename_str
コンパイルするソース・ファイルの名前で、ファイル名のほかにドライブとサブディレクトリーも含まれます。この値はコンパイラーによって設定されます。

Uex_UIB_Filename_len
Filename_str が指すソース・ファイルの名前の長さ。この値はコンパイラーによって設定されます。

Uex_UIB_return_code
ユーザー出口プロシージャーからの戻りコード。この値はユーザーによって設定されます。

Uex__UIB_reason_code
プロシージャー理由コード。この値はユーザーによって設定されます。

Uex_UIB_Exit_Routs
初期化プロシージャーが設定する出口項目。

Uex_UIB_Termination
終了時にコンパイラーが呼び出す項目。この値はユーザーによって設定されます。

Uex_UIB_Message_Filter
メッセージを生成する必要があるときにコンパイラーが呼び出す項目。この値はユーザーによって設定されます。

IBM 提供のコンパイラー出口、IBMUEXIT

IBM は、自動的にメッセージのフィルター操作を行うサンプル・コンパイラー・ユーザー出口 IBMUEXIT を提供しています。

このコンパイラー出口は、メッセージをモニターし、指定されたメッセージ番号に基づいてメッセージを抑止したりメッセージの重大度を変更したりします。

204 ページの図 21 にある IBMUEXIT のソースを参照してください。

コンパイラー・ユーザー出口の活動化

コンパイラー・ユーザー出口を活動化するには、EXIT コンパイル時オプションを指定する必要があります。

EXIT コンパイル時オプションを使用すれば、ユーザー出口入力ファイルの DD 名を指定するユーザー・オプション・ストリングを指定できます。ストリングを指定しない場合は、SYSUEXIT がユーザー出口入力ファイルの DD 名として使用されます。

ユーザー・オプション・ストリングは、グローバル制御ブロックにおいてユーザー出口関数に渡されます。詳しくは、 542 ページの『グローバル制御ブロックの構造』の「Uex_UIB_User_char_str」を参照してください。

関連情報:

37 ページの『EXIT』
EXIT オプションにより、コンパイラー・ユーザー出口を呼び出す ことができます。

コンパイラー・ユーザー出口のカスタマイズ

独自のコンパイラー・ユーザー出口を作成することも、コンパイラーに付属の出口を単に使用することもできます。いずれの場合も、コンパイラー・ユーザー出口用のフェッチ可能ファイルの名前は IBMUEXIT でなければなりません。

このセクションでは、以下の作業を行う方法について説明します。

- カスタマイズされたメッセージ・フィルター操作用にユーザー出口入力ファイルを変更する
- 独自のコンパイラー・ユーザー出口を作成する

SYSUEXIT の変更

まったく新しいコンパイラー・ユーザー出口を作成するのではなく、簡単にユーザー出口入力ファイルを変更できます。

ファイルを編集して、抑止するメッセージ番号と、変更するメッセージ番号重大度レベルを指示します。 545 ページの図 112 に、サンプル・ファイルを示します。

```
  Fac Id   Msg No   Severity   Suppress   Comment
+--------+--------+----------+----------+-------------------------------
  'IBM'     1042       -1          1      String spans multiple lines
  'IBM'     1044       -1          1      FIXED BIN 7 mapped to 1 byte
  'IBM'     1047        8          0      Order inhibits optimization
  'IBM'     1052       -1          1      Nodescriptor with * extent arg
  'IBM'     1059        0          0      Select without OTHERWISE
  'IBM'     1169        0          1      Precision of result determined
```

図 *112.* ユーザー出口入力ファイルの例

最初の 2 行はヘッダー行で、IBMUEXIT では無視されます。残りの行には、可変数のブランクで区切られた入力データが含まれています。

ファイルの各列は、コンパイラー・ユーザー出口に次のように関係しています。

- 出口が適用されるすべてのコンパイラー・メッセージについて、単一引用符で囲まれた 'IBM' という文字が最初の列に含まれていなければなりません。
- 2 番目の列は、4 桁のメッセージ番号です。
- 3 番目の列は、新しいメッセージ重大度です。 重大度 -1 は、重大度がデフォルト値のままにすることを表します。
- 4 番目の列は、メッセージを抑止するかどうかを示します。以下のいずれかの値を指定します。

  1　　メッセージを抑止します。

  0　　メッセージを出力します。
- 最後の列はコメント欄で、通知の目的の列であり、IBMUEXIT では無視されます。

独自のコンパイラー出口の作成

独自のユーザー出口を作成するには、モデルとして IBMUEXIT を使用できます。出口を作成するときは、初期化、メッセージ・フィルター操作、および終了の領域を必ずカバーするようにしてください。

204 ページの図 21 にある IBMUEXIT のソースを参照してください。

コンパイラー・ユーザー出口は、RENT オプションを指定してコンパイルし、DLL としてリンクする必要があります。

初期化プロシージャーの作成

初期化プロシージャーは、出口が必要とする初期化、例えば、ファイルのオープンやストレージの割り振りなどを実行する必要があります。

初期化プロシージャー固有の制御ブロックは、以下のようにコーディングします。

```
Dcl 1 Uex_ISA native based,
    2 Uex_ISA_Length_fixed bin(31);   /* storage(Uex_ISA) *  /
```

初期化プロシージャーのグローバル制御ブロック構文については、 542 ページの『グローバル制御ブロックの構造』を参照してください。

初期化プロシージャーの完了時には、戻りコード/理由コードを次のように設定する必要があります。

0/0
コンパイルを続行する

4/n
将来の利用のために予約済み

8/n
将来の利用のために予約済み

12/n
将来の利用のために予約済み

16/n
コンパイルを打ち切る

メッセージ・フィルター操作プロシージャーの作成

メッセージ・フィルター操作プロシージャーを使うと、メッセージを抑止したり、メッセージの重大度を変更したりすることができます。

どのメッセージの重大度も高くすることはできますが、低くすることができるのは、**ERROR** (重大度コード 8) メッセージまたは、**WARNING** (重大度コード 4) メッセージだけです。

プロシージャー特有の制御ブロックには、メッセージについての情報が含まれています。 これは、特定のメッセージの取り扱い方法を示す情報をコンパイラーに渡すために使われます。

プロシージャー固有のメッセージ・フィルター制御ブロックを以下に例示します。

```
Dcl 1 Uex_MFX native based,
      2 Uex_MFX_Length   fixed bin(31),

      2 Uex_MFX_Facility_Id  char(3),        /* of component writing
                                               message             */
      2 *                    char(1),
      2 Uex_MFX_Message_no   fixed bin(31),
      2 Uex_MFX_Severity     fixed bin(15),
      2 Uex_MFX_New_Severity fixed bin(15),  /* set by exit proc    */
      2 Uex_MFX_Inserts                fixed bin(15),
      2 Uex_MFX_Inserts_Data( 6 refer(Uex_MFX_Inserts) ),
        3 Uex_MFX_Ins_Type             fixed bin(7),
        3 Uex_MFX_Ins_Type_Data union unaligned,
          4 *                          char(8),
          4 Uex_MFX_Ins_Bin8           fixed bin(63),
          4 Uex_MFX_Ins_Bin            fixed bin(31),
          4 Uex_MFX_Ins_Str,
            5 Uex_MFX_Ins_Str_Len      fixed bin(15),
            5 Uex_MFX_Ins_Str_Addr     pointer,
          4 Uex_MFX_Ins_Series,
            5 Uex_MFX_Ins_Series_Sep   char(1),
            5 Uex_MFX_Ins_Series_Addr  pointer;
```

データ入力フィールド

Uex_MFX_Length
バイト単位の制御ブロックの長さ。値は storage (Uex_MFX) です。

Uex_MFX_Facility_Id
機能の ID。コンパイラーの場合、ID は IBM です。この値はコンパイラーによって設定されます。

Uex_MFX_Message_no
コンパイラーが生成する予定のメッセージ番号。この値はコンパイラーによって設定されます。

Uex_MFX_Severity
メッセージの重大度レベルで、長さは 1 から 15 文字まで。この値はコンパイラーによって設定されます。

Uex_MFX_New_Severity
メッセージの新しい重大度レベルで、長さは 1 から 15 文字まで。この値はユーザーによって設定されます。

Uex_MFX_Inserts
メッセージでの挿入の数。範囲はゼロから 6 までです。この値はコンパイラーによって設定されます。

Uex_MFX_Inserts_Data
各挿入が記述されたフィールド。これらの値はコンパイラーによって設定されます。

Uex_MFX_Ins_Type
挿入のタイプ。可能な挿入タイプは以下のとおりです。

Uex_Ins_Type_Xb31
FIXED BIN(31) に使用され、値は 1 です。

Uex_Ins_Type_Char
CHAR ストリングに使用され、値は 2 です。

Uex_Ins_Type_Series
一連の CHAR ストリングに使用され、値は 3 です。

Uex_Ins_Type_Xb63
FIXED BIN(63) に使用され、値は 4 です。

この値はコンパイラーによって設定されます。

Uex_MFX_Ins_Bin
整数タイプの挿入の場合の整数値。この値はコンパイラーによって設定されます。

Uex_MFX_Ins_Str_Len
文字タイプを持つ挿入の場合の長さ (バイト)。この値はコンパイラーによって設定されます。

Uex_MFX_Ins_Str_Addr
文字タイプを持つ挿入の文字ストリングのアドレス。この値はコンパイラーによって設定されます。

Uex_MFX_Ins_Series_Sep
シリーズ・タイプを持つ挿入の各エレメント間に挿入されるべき文字。通常、これはブランク、ピリオド、またはコンマです。この値はコンパイラーによって設定されます。

Uex_MFX_Ins_Series_Addr
シリーズ・タイプを持つ挿入の各種文字ストリングのシリーズのアドレス。このアドレスは FIXED BIN(31) フィールドを指します。このフィールドには、連結されるストリングの数と、その後にそれらのストリングのアドレスが保持されます。この値はコンパイラーによって設定されます。

メッセージ・フィルター操作プロシージャーの完了時に、戻りコード/理由コードを次のいずれかに設定してください。

0/0
コンパイルを続行し、メッセージを出力する

0/1
コンパイルを続行し、メッセージを出力しない

4/n
将来の利用のために予約済み

8/n
将来の利用のために予約済み

16/n
コンパイルを打ち切る

終了プロシージャーの作成

ファイルのクローズのような、必要なクリーンアップを実行するには、終了プロシージャーを使います。 また、エラー・メッセージ・フィルター・プロシージャーと初期化プロシージャーの実行時に収集される情報に基づいて、最終統計レポートを作成することもできます。

終了プロシージャー固有の制御ブロックは、以下のようにコーディングします。

```
Dcl 1 Uex_ISA native based,
    2 Uex_ISA_Length_fixed bin(31); /* storage(Uex_ISA)       */
```

終了プロシージャーのグローバル制御ブロック構文については、 542 ページの『グローバル制御ブロックの構造』を参照してください。終了プロシージャーの完了時に、戻りコード/理由コードを次のいずれかに設定してください。

0/0
コンパイルを続行する

4/n
将来の利用のために予約済み

8/n
将来の利用のために予約済み

12/n
将来の利用のために予約済み

16/n
コンパイルを打ち切る

SQL メッセージの抑止例

この例は、メッセージの挿入について調べ、2 つの SQL 情報メッセージと 1 つの SQL 警告メッセージを抑止するようにユーザー出口を変更する方法を示しています。

```
*Process dft(nodescriptor connected);
*Process or('|') not('!');
*Process limits(extname(31)) rent;

 /*********************************************************************/
 /*                                                                   */
 /*  NAME - IBMUEXIT.PLI                                              */
 /*                                                                   */
 /*  DESCRIPTION                                                      */
 /*    User-exit sample program.                                      */
 /*                                                                   */
 /*    Licensed Materials - Property of IBM                           */
 /*    Copyright IBM Corp. 1999, 2017 All Rights Reserved         */
 /*    All Rights Reserved.                                           */
 /*    US Government Users Restricted Rights-- Use, duplication or    */
 /*    disclosure restricted by GSA ADP Schedule Contract with        */
 /*    IBM Corp.                                                      */
 /*                                                                   */
 /*  DISCLAIMER OF WARRANTIES                                         */
 /*    The following enclosed code is sample code created by IBM      */
 /*    Corporation. This sample code is not part of any standard      */
 /*    IBM product and is provided to you solely for the purpose of   */
 /*    assisting you in the development of your applications.  The    */
 /*    code is provided "AS IS", without warranty of any kind.        */
 /*    IBM shall not be liable for any damages arising out of your    */
 /*    use of the sample code, even if IBM has been advised of the    */
 /*    possibility of such damages.                                   */
 /*                                                                   */
 /*********************************************************************/
```

図 *113. SQL* メッセージの抑止

```
/*********************************************************************/
/*                                                                   */
/*  During initialization, IBMUEXIT is called.  It reads             */
/*  information about the messages being screened from a text        */
/*  file and stores the information in a linked list. IBMUEXIT       */
/*  also sets up the entry points for the message filter service     */
/*  and termination service.                                         */
/*                                                                   */
/*  For each message generated by the compiler, the compiler         */
/*  calls the message filter registered by IBMUEXIT.  The filter     */
/*  looks the message up in the linked list previously created       */
/*  to see if it is one for which some action should be taken.       */
/*                                                                   */
/*  The termination service is called at the end of the compile      */
/*  but does nothing.  It could be enhanced to generates reports     */
/*  or do other cleanup work.                                        */
/*                                                                   */
/*********************************************************************/


pack: package exports(*);

  Dcl
    1 Uex_UIB           native based,
      2 Uex_UIB_Length         fixed bin(31),

      2 Uex_UIB_Exit_token     pointer,       /* for user exit's use */

      2 Uex_UIB_User_char_str  pointer,       /* to exit option str  */
      2 Uex_UIB_User_char_len  fixed bin(31),

      2 Uex_UIB_Filename_str   pointer,       /* to source filename  */
      2 Uex_UIB_Filename_len   fixed bin(31),

      2 Uex_UIB_return_code fixed bin(31),    /* set by exit procs   */
      2 Uex_UIB_reason_code fixed bin(31),    /* set by exit procs   */

      2 Uex_UIB_Exit_Routs,                   /* exit entries setat
                                                 initialization      */
        3 ( Uex_UIB_Termination,
            Uex_UIB_Message_Filter,           /* call for each msg   */
           *, *, *, * )
          limited entry (
           *,                                 /* to Uex_UIB          */
           *                                  /* to a request area   */
          );

  /*******************************************************************/
  /*                                                                 */
  /*   request area for initialization exit                          */
  /*                                                                 */
  /*******************************************************************/

  Dcl 1 Uex_ISA native based,
       2 Uex_ISA_Length fixed bin(31);
```

SQL メッセージの抑止 ***(***続き***)***

```
/*****************************************************************/
/*                                                               */
/*   request area for message_filter exit                        */
/*                                                               */
/*****************************************************************/

Dcl 1 Uex_MFX based,
     2 Uex_MFX_Length                fixed bin(31),
     2 Uex_MFX_Facility_Id           char(3),
     2 Uex_MFX_Version               fixed bin(7),
     2 Uex_MFX_Message_no            fixed bin(31),
     2 Uex_MFX_Severity              fixed bin(15),
     2 Uex_MFX_New_Severity          fixed bin(15),
     2 Uex_MFX_Inserts               fixed bin(15),
     2 Uex_MFX_Inserts_Data( 6 ),
       3 Uex_MFX_Ins_Type            fixed bin(7),
       3 Uex_MFX_Ins_Type_Data union unaligned,
         4 *                         char(8),
         4 Uex_MFX_Ins_Bin           fixed bin(31),
         4 Uex_MFX_Ins_Str,
           5 Uex_MFX_Ins_Str_Len     fixed bin(15),
           5 Uex_MFX_Ins_Str_Addr    pointer,
        4 Uex_MFX_Ins_Series,
           5 Uex_MFX_Ins_Series_Sep  char(1),
           5 Uex_MFX_Ins_Series_Addr pointer;

dcl uex_Ins_Type_Xb31              fixed bin(15) value(1);
dcl uex_Ins_Type_Char              fixed bin(15) value(2);
dcl uex_Ins_Type_Series            fixed bin(15) value(3);

/*****************************************************************/
/*                                                               */
/*   request area for terminate exit                             */
/*                                                               */
/*****************************************************************/

Dcl 1 Uex_TSA native based,
      2 Uex_TSA_Length fixed bin(31);

/*****************************************************************/
/*                                                               */
/*   severity codes                                              */
/*                                                               */
/*****************************************************************/

dcl uex_Severity_Normal            fixed bin(15) value(0);
dcl uex_Severity_Warning           fixed bin(15) value(4);
dcl uex_Severity_Error             fixed bin(15) value(8);
dcl uex_Severity_Severe            fixed bin(15) value(12);
dcl uex_Severity_Unrecoverable     fixed bin(15) value(16);

/*****************************************************************/
/*                                                               */
/*   return codes                                                */
/*                                                               */
/*****************************************************************/

dcl uex_Return_Normal              fixed bin(15) value(0);
dcl uex_Return_Warning             fixed bin(15) value(4);
dcl uex_Return_Error               fixed bin(15) value(8);
dcl uex_Return_Severe              fixed bin(15) value(12);
dcl uex_Return_Unrecoverable       fixed bin(15) value(16);
```

SQL メッセージの抑止 ***(***続き***)***

```
  /*******************************************************************/
  /*                                                                 */
  /*   reason codes                                                  */
  /*                                                                 */
  /*******************************************************************/

  dcl uex_Reason_Output               fixed bin(15) value(0);
  dcl uex_Reason_Suppress             fixed bin(15) value(1);

  dcl header  pointer;

  dcl
    1 message_item native based,
      2 message_Info,
        3 facid     char(3),
        3 msgno     fixed bin(31),
        3 newsev    fixed bin(15),
        3 reason    fixed bin(31),
        3 variable  char(31) var,
      2 link pointer;

ibmuexit: proc ( ue, ia ) options( fetchable );

  dcl 1 ue like uex_Uib byaddr;
  dcl 1 ia like uex_Isa byaddr;

  dcl sysuexit     file stream input env(recsize(80));
  dcl next         pointer;
  dcl based_Chars  char(31) based;
  dcl title_Str    char(31) var;
  dcl eof          bit(1);


  on error
    begin;
      on error system;
      call plidump('TFBHS' );
    end;

  on undefinedfile(sysuexit)
    begin;
      put edit ('** User exit unable to open exit file ')
               (A) skip;
      put skip;
      signal error;
    end;

  if ue.uex_Uib_User_Char_Len = 0 then
    do;
      open file(sysuexit);
    end;
  else
    do;
      title_Str
       = substr( ue.uex_Uib_User_Char_Str->based_Chars,
                 1, ue.uex_Uib_User_Char_Len );
      open file(sysuexit) title(title_Str);
    end;
```

SQL メッセージの抑止 ***(***続き***)***

```
/*****************************************************************/
/*                                                               */
/*  save the address of the message filter so that it will       */
/*  be invoked by the compiler                                   */
/*                                                               */
/*****************************************************************/

ue.Uex_UIB_Message_Filter = message_filter;

/*****************************************************************/
/*                                                               */
/*  set the pointer to the linked list to null                   */
/*                                                               */
/*  then allocate the first message record                       */
/*                                                               */
/*****************************************************************/

header = sysnull();
allocate message_item set(next);

/*****************************************************************/
/*                                                               */
/*  skip header lines and read the file                          */
/*                                                               */
/*  the file is expected to start with a header line and         */
/*  then a line with a scale and then the data lines, for example,*/
/*  it could look like the 5 lines below starting with "Fac Id"  */
/*                                                               */
/*   Fac Id   Msg No   Severity   Suppress   Insert              */
/* +-------+-------+---------+---------+-----------------------  */
/*   'IBM'     3259        0          1      'DSNH527'           */
/*   'IBM'     3024        0          1      'DSNH4760'          */
/*   'IBM'     3024        0          1      'DSNH050'           */
/*                                                               */
/*****************************************************************/

eof = '0'b;
on endfile(sysuexit)
  eof = '1'b;

get file(sysuexit) list(next->message_info) skip(3);

do while( eof = '0'b );

  /***************************************************************/
  /*                                                             */
  /*  put message information in linked list                     */
  /*                                                             */
  /***************************************************************/

  next->link = header;
  header = next;

  /***************************************************************/
  /*                                                             */
  /*  read next data line                                        */
  /*                                                             */
  /***************************************************************/

  allocate message_item set(next);
  get file(sysuexit) skip;
  get file(sysuexit) list(next->message_info);

end;
```

SQL メッセージの抑止 ***(続き)***

```
  /*************************************************************/
  /*                                                           */
  /*  free the last message record allocated and close the file  */
  /*                                                           */
  /*************************************************************/

  free next->message_Item;
  close file(sysuexit);

end;

message_Filter: proc ( ue, mf  );

  dcl 1 ue like uex_Uib byaddr;
  dcl 1 mf like uex_Mfx byaddr;

  dcl next         pointer;
  dcl jx           fixed bin(31);
  dcl insert       char(256) var;
  dcl based_Chars  char(256) based;

  on error
    begin;
      on error system;
      call plidump('TFBHS' );
    end;

  /*************************************************************/
  /*                                                           */
  /*  by default, leave the reason code etc unchanged          */
  /*                                                           */
  /*************************************************************/

  ue.uex_Uib_Reason_Code = uex_Reason_Output;
  ue.uex_Uib_Return_Code = 0;

  mf.uex_Mfx_New_Severity = mf.uex_Mfx_Severity;

  /*************************************************************/
  /*                                                           */
  /*  save the first insert if it has character type           */
  /*                                                           */
  /*************************************************************/

  insert = '*';
  if mf.Uex_MFX_Length < stg(mf) then;
  else
    if mf.Uex_MFX_Inserts = 0 then;
    else
      do jx = 1 to mf.Uex_MFX_Inserts;
        select( mf.Uex_MFX_Ins_Type(jx) );
          when( uex_Ins_Type_Char )
            do;
              if jx = 1 then
                insert =
                  substr( mf.Uex_MFX_Ins_Str_Addr(jx)->based_Chars,
                          1,mf.Uex_MFX_Ins_Str_Len(jx));
            end;
          otherwise;
        end;
      end;
```

SQL メッセージの抑止 ***(続き)***

```
  /*************************************************************/
  /*                                                           */
  /*  search list for matching error message                   */
  /*                                                           */
  /*************************************************************/

  search_list:
  do next = header repeat( next->link ) while( next !=sysnull() );

    if next->msgno = mf.uex_Mfx_Message_No
     & next->facid = mf.Uex_Mfx_Facility_Id then
      do;
        if next->variable = '*' then
          leave search_list;
        if next->variable
         = substr(insert,1,length(next->variable)) then
          leave search_list;
      end;
  end;

  /*************************************************************/
  /*                                                           */
  /*  if list exhausted, then                                  */
  /*    no match was found                                     */
  /*  else                                                     */
  /*    filter the message according to the match found        */
  /*                                                           */
  /*************************************************************/

  if next = sysnull() then;
  else
    do;
      /*********************************************************/
      /*                                                       */
      /*  filter error based on information in table           */
      /*                                                       */
      /*********************************************************/

      ue.uex_Uib_Reason_Code = next->reason;
      if next->newsev < 0 then;
      else
        mf.uex_Mfx_New_Severity = next->newsev;
    end;
end;

exitterm: proc ( ue, ta  );

  dcl 1 ue like uex_Uib byaddr;
  dcl 1 ta like uex_Tsa byaddr;

  ue.uex_Uib_return_Code = 0;
  ue.uex_Uib_reason_Code = 0;

end;

end;
```

SQL メッセージの抑止 ***(続き)***

第 25 章 PL/I 記述子

この章では、実行時の PL/I ルーチン間での PL/I パラメーターの引き渡しの規則について説明します。

記述子以外の 言語環境プログラムランタイム環境に関する考慮事項について詳しくは、「*z/OS Language Environment* プログラミング・ガイド」を参照してください。この資料では、ランタイム環境の規則と、その規則をサポートするアセンブラー・マクロについて説明しています。

引数の引き渡し

ストリング、配列、または構造体が引数として渡されるときには、呼び出されたルーチンが OPTIONS(NODESCRIPTOR) を使って宣言されていない限り、コンパイラーがその引数の記述子を渡します。

このような記述子を渡す方法として次の 2 つがあります。
- 記述子リストを使う
- 記述子ロケーターを使う

これら 2 つの方法それぞれについて、次の主要機能に注意してください。
- 引数が記述子リストを使って渡されるとき
  - 渡される引数の数は、いずれかの引数に記述子が必要であるときには、指定された引数の数より 1 大きい数になります。
  - 記述子を使って渡される引数は、値 (BYVALUE) で渡されるポインターとして受け取ることができます。
  - コンパイラーは、DEFAULT(DESCLIST) コンパイラー・オプションが有効なときにこのメソッドを使用します。
- 引数が記述子ロケーターによって渡されるとき
  - 渡される引数の数は常に、指定された引数の数と一致します。
  - 記述子を使用して渡される引数は、参照 (BYADDR) で渡されるポインターとして受け取ることができます。
  - コンパイラーは、DEFAULT(DESCLOCATOR) コンパイラー・オプションが有効なときにこのメソッドを使用します。

記述子リストによる引数の引き渡し

引数とその記述子が記述子リストを使って渡される場合は、1 つでも記述子を必要とする引数があれば、1 つの追加の引数が渡されます。

この追加の引数は、ポインターのリストを指すポインターです。このリストの入力項目の数は、渡される引数の数と一致します。 記述子が必要でない引数の場合は、記述子リストの中で、対応するポインターが SYSNULL に設定されます。記述子が必要である引数の場合は、記述子リストの中で、対応するポインターがその引数の記述子のアドレスに設定されます。

© Copyright IBM Corp. 1999, 2017

例えば、次のようにルーチン sample が宣言されたとします。

```
declare sample entry( fixed bin(31), varying char(*) )
               options( byaddr descriptor );
```

sample が次のようなステートメントで呼び出されたとします。

```
call sample( 1, 'test' );
```

この場合、次の 3 つの引数がルーチンに渡されます。

- 値が 1 の固定 bin(31) 一時変数のアドレス
- 値が test の可変 char(4) 一時変数のアドレス
- 次の項目で構成される記述子リストのアドレス
  - SYSNULL()
  - 可変 char(4) ストリング用の記述子のアドレス

ロケーター/記述子で引数を渡す

引数とその記述子がロケーター/記述子で渡される場合、引数に記述子が必要となるときは必ず、その引数のロケーター/記述子のアドレスが代わりに渡されます。

ストリングを除いて、ロケーター/記述子は、ポインターの対です。最初のポインターはデータのアドレスで、2 番目のポインターは記述子のアドレスです。 ストリングの場合、CMPAT(LE) では、ロケーター/記述子は同じく、ポインターの対です。ただし、他の CMPAT オプションでは、ロケーター/記述子は、ストリングのアドレスおよびストリング記述子自体から成ります。

例えば、ルーチン sample が次のように宣言されるとします。

```
declare sample entry( fixed bin(31), varying char(*) )
               options( byaddr descriptor );
```

sample が次のようなステートメントで呼び出されたとします。

```
call sample( 1, 'test' );
```

この場合、次の 2 つの引数がルーチンに渡されます。

- 値が 1 の固定 bin(31) 一時変数のアドレス
- 以下のアドレスおよび記述子で構成されるロケーター/記述子のアドレス
  - 値が test の可変 char(4) 一時変数のアドレス
  - CMPAT(LE) では、可変 char(4) ストリングの CMPAT(LE) 記述子のアドレス
  - CMPAT(V*) では、可変 char(4) ストリングの CMPAT(V*) 記述子

CMPAT(V*) 記述子

LE 記述子とは異なり、CMPAT(V*) 記述子は自己記述型ではありません。ただし、ストリング記述子は、すべての CMPAT(V*) オプションで同じであり、LE ストリング記述子と同じコード・ページ・エンコードも共用します。

ストリング記述子

ストリング記述子では、最初の 2 バイトは、ストリングの最大長を指定します。この最大長は常にネイティブ・フォーマットで保持されます。

3 番目のバイトには、(例えば、VARYING ストリングのストリング長の保持フォーマットがリトル・エンディアンなのかビッグ・エンディアンなのか、また WIDECHAR ストリングのデータ保持フォーマットがリトル・エンディアンなのかビッグ・エンディアンなのかを示すフラグなど) さまざまなフラグが含まれます。

非可変ビット・ストリングのストリング記述子では、 4 番目のバイトはビット・オフセットを表します。

CHARACTER ストリングのストリング記述子では、4 番目のバイトは、コンパイラー CODEPAGE オプションをエンコードします。

CMPAT(V1) および CMPAT(V2) の下でのストリング記述子の宣言は次のとおりです。

```
declare
  1 dso_string based,
    2 dso_string_length          fixed bin(15),
    2 dso_string_flags,
      3 dso_string_is_varying           bit(1),
      3 dso_string_is_varyingz          bit(1),
      3 dso_string_has_nonnative_len    bit(1), /* for varying     */
      3 dso_string_is_ascii             bit(1), /* for char        */
      3 dso_string_has_nonnative_data   bit(1), /* for wchar       */
      3 *                               bit(1), /* reserved, '0'b  */
      3 *                               bit(1), /* reserved, '0'b  */
      3 *                               bit(1), /* reserved, '0'b  */
    2 * union,
      3 dso_String_Codepage      ordinal ccs_Codepage_Enum,
      3 dso_string_bitofs        fixed bin(8) unsigned,
    2 dso_string_end     char(0);
```

CMPAT(V3) の下でのストリング記述子の宣言は次のとおりです。

```
declare
  1 dso_longstr based,
    2 dso_longstr_info,
      3 *                  fixed bin(8) unsigned,
      3 dso_datatype       fixed bin(8) unsigned,
      3 * union,
        4 dso_longstr_bitofs    fixed bin(8) unsigned,
        4 dso_longstr_codepage  type ccs_Codepage_Enum,
      3 dso_longstr_info2,
        4 dso_longstr_has_nonnative_len   bit(1), /* for varying    */
        4 dso_longstr_is_ebcdic           bit(1), /* for char       */
        4 dso_longstr_has_nonnative_data  bit(1), /* for wchar      */
        4 *                               bit(1),
        4 dso_longstr_is_varying          bit(1),
        4 dso_longstr_is_varyingz         bit(1),
        4 dso_longstr_is_varying4         bit(1),
        4 *                               bit(1),
    2 dso_longstr_length          fixed bin(31),
    2 dso_longstr_end     char(0);
```

コード・ページ・エンコードに使用できる値は次のように定義されます。

```
define ordinal
  ccs_Codepage_Enum
    ( ccs_Codepage_01047  value(1)
```

```
     ,ccs_Codepage_01140
     ,ccs_Codepage_01141
     ,ccs_Codepage_01142
     ,ccs_Codepage_01143
     ,ccs_Codepage_01144
     ,ccs_Codepage_01145
     ,ccs_Codepage_01146
     ,ccs_Codepage_01147
     ,ccs_Codepage_01148
     ,ccs_Codepage_01149
     ,ccs_Codepage_00819
     ,ccs_Codepage_00813
     ,ccs_Codepage_00920
     ,ccs_Codepage_00037
     ,ccs_Codepage_00273
     ,ccs_Codepage_00277
     ,ccs_Codepage_00278
     ,ccs_Codepage_00280
     ,ccs_Codepage_00284
     ,ccs_Codepage_00285
     ,ccs_Codepage_00297
     ,ccs_Codepage_00500
     ,ccs_Codepage_00871
     ,ccs_Codepage_01026
     ,ccs_Codepage_01155
    ) unsigned prec(8);
```

配列記述子

以下の宣言では、配列の上限は 15 として宣言されていますが、実際の上限は常に記述されている配列の次元数に一致しているということを理解する必要があります。

CMPAT(V1) 配列記述子の宣言は次のとおりです。

```
declare
  1 dso_v1  based,
    2 dso_v1_rvo      fixed bin(31),    /* relative virtual origin */
    2 dso_v1_data(1:15),
      3 dso_v1_stride fixed bin(31),    /*   multiplier            */
      3 dso_v1_hbound fixed bin(15),    /*   hbound                */
      3 dso_v1_lbound fixed bin(15);    /*   lbound                */
```

CMPAT(V2) 配列記述子の宣言は次のとおりです。

```
declare
  1 dso_v2  based,
    2 dso_v2_rvo      fixed bin(31),    /* relative virtual origin */
    2 dso_v2_data(1:15),
      3 dso_v2_stride fixed bin(31),    /*   multiplier            */
      3 dso_v2_hbound fixed bin(31),    /*   hbound                */
      3 dso_v2_lbound fixed bin(31);    /*   lbound                */
```

CMPAT(V3) 配列記述子の宣言は次のとおりです。

```
declare
  1 dso_v3  based,
    2 dso_v3_rvo      fixed bin(63),    /* relative virtual origin */
    2 dso_v3_data(1:15),
      3 dso_v3_stride fixed bin(63),    /*   multiplier            */
      3 dso_v3_hbound fixed bin(63),    /*   hbound                */
      3 dso_v3_lbound fixed bin(63);    /*   lbound                */
```

CMPAT(LE) 記述子

すべての LE 記述子は 4 バイト・フィールドで始まります。 最初のバイトは、記述子のタイプ (スカラー、配列、構造体、または共用体) を指定します。残りの 3 バイトは、特定の記述子タイプで設定されていない限り、ゼロになります。

記述子ヘッダーの宣言は次のとおりです。

```
declare
  1 dsc_Header based( sysnull() ),
    2 dsc_Type          fixed bin(8) unsigned,
    2 dsc_Datatype      fixed bin(8) unsigned,
    2 *                 fixed bin(8) unsigned,
    2 *                 fixed bin(8) unsigned;
```

dsc_Type フィールドに指定できる値は次のとおりです。

```
declare
  dsc_Type_Unset                fixed bin(8) value(0),
  dsc_Type_Element              fixed bin(8) value(2),
  dsc_Type_Array                fixed bin(8) value(3),
  dsc_Type_Structure            fixed bin(8) value(4),
  dsc_Type_Union                fixed bin(8) value(4);
```

ストリング記述子

ストリング記述子では、ヘッダーの 2 番目のバイトがストリング・タイプ (ビット、文字、またはグラフィックのほかに、非可変、可変、または可変 z) を表します。

非可変ビット・ストリングのストリング記述子では、ヘッダーの 3 番目のバイトがビット・オフセットを表します。

CHARACTER ストリングのストリング記述子では、ヘッダーの 3 番目のバイトは、コンパイラー CODEPAGE オプションをエンコードします。

可変ストリングのストリング記述子では、4 番目のバイトに、ストリング長がネイティブ以外のフォーマットで保持されるかどうかを表すビットが含まれます。

文字ストリングのストリング記述子でも、4 番目のバイトに、ストリング・データが EBCDIC であるかどうかを表すビットが含まれます。

ストリング記述子の宣言は次のとおりです。

```
declare
  1 dsc_String  based( sysnull() ),
    2 dsc_String_Header,
      3 *                     fixed bin(8) unsigned,
      3 dsc_String_Type       fixed bin(8) unsigned,
      3 * union,
        4 dsc_String_Codepage ordinal ccs_Codepage_Enum,
        4 dsc_String_BitOfs   fixed bin(8) unsigned,
      3 *,
        4 dsc_String_Has_Nonnative_Len    bit(1),
        4 dsc_String_Is_Ebcdic            bit(1),
        4 dsc_String_Has_Nonnative_Data   bit(1),
        4 *                               bit(5),
    2 dsc_String_Length   fixed bin(31); /* max length of string */
```

dsc_String_Type フィールドに指定できる値は次のとおりです。

```
declare
  dsc_String_Type_Unset                fixed bin(8) value(0),
  dsc_String_Type_Char_Nonvarying      fixed bin(8) value(2),
  dsc_String_Type_Char_Varyingz        fixed bin(8) value(3),
  dsc_String_Type_Char_Varying2        fixed bin(8) value(4),
  dsc_String_Type_Bit_Nonvarying       fixed bin(8) value(6),
  dsc_String_Type_Bit_Varying2         fixed bin(8) value(7),
  dsc_String_Type_Graphic_Nonvarying   fixed bin(8) value(9),
  dsc_String_Type_Graphic_Varyingz     fixed bin(8) value(10),
  dsc_String_Type_Graphic_Varying2     fixed bin(8) value(11),
  dsc_String_Type_Widechar_Nonvarying  fixed bin(8) value(13),
  dsc_String_Type_Widechar_Varyingz    fixed bin(8) value(14),
  dsc_String_Type_Widechar_Varying2    fixed bin(8) value(15);
```

配列記述子

配列記述子の宣言は次のとおりです。

```
declare
  1 dsc_Array  based( sysnull() ),
    2 dsc_Array_Header   like dsc_Header,
    2 dsc_Array_EltLen   fixed bin(31), /* Length of array element */
    2 dsc_Array_Rank     fixed bin(31), /* Count of dimensions     */
    2 dsc_Array_RVO      fixed bin(31), /* Relative virtual origin */
    2 dsc_Array_Data( 1: 1 refer(dsc_Array_Rank) ),
      3 dsc_Array_LBound fixed bin(31), /*   LBound                */
      3 dsc_Array_Extent fixed bin(31), /*   HBound - LBound + 1   */
      3 dsc_Array_Stride fixed bin(31); /*   Multiplier            */
```

第 6 部 付録

© Copyright IBM Corp. 1999, 2017

付録. SYSADATA メッセージ情報

XINFO コンパイル時オプションの MSG サブオプションを指定すると、コンパイラーは SYSADATA ファイルを生成します。

SYSADATA ファイルには以下のレコードが含まれています。

- カウンター・レコード
- リテラル・レコード
- ファイル・レコード
- メッセージ・レコード
- オプション・レコード

すべての ADATA レコードに対する宣言の完全なセットは、サンプル・データ・セット SIBMZSAM 内のメンバー `ibmwxin` および `ibmwxop` にあります。

ファイル内のレコードは、必ずしも上記にリストした順序で生成されるとはかぎらず、例えば、リテラル・レコードとファイル・レコードが交互に混ざり合う場合があることに注意してください。SYSADATA ファイルを読み取るコードを作成している場合、以下の例外を除いて、ファイル内のレコードの順序は、あてになりません。

- カウンター・レコードは、ファイル内の最初のレコードになります。
- それぞれのリテラル・レコードは、それが定義するリテラルの参照よりも前にきます。
- それぞれのファイル・レコードは、それが記述するファイルの参照よりも前にきます。

SYSADATA ファイルについて

SYSADATA ファイルは順次バイナリー・ファイルです。

z/OS バッチ環境では、コンパイラーは SYSADATA DD ステートメントで指定されたファイルに SYSADATA レコードを書き込みます。このファイルは PDS のメンバーであってはなりません。その他のすべてのシステムでは、コンパイラーは拡張子 `adt` を持つファイルに書き込みます。

ファイル内のそれぞれのレコードには、ヘッダーが入っています。 この 12 バイトのヘッダーには、以下のフィールドがあり、これはそのファイル内のすべてのレコードで同じです。

コンパイラー
データを生成したコンパイラーを表す番号。 PL/I の場合、この番号は 40 です。

エディション番号
データを生成したコンパイラーのエディション番号。 この製品の場合、この番号は 2 です。

© Copyright IBM Corp. 1999, 2017

SYSADATA レベル
このファイル・フォーマットが示す SYSADATA のレベルを表す番号。 この製品の場合、この番号は 4 です。

ヘッダーには、レコードごとに変わる、以下のフィールドもあります。

- レコード・タイプ
- レコードが次のレコードに継続するかどうか

可能なレコード・タイプは、図 114 に示すような、序数値としてエンコードされています。

```
Define ordinal xin_Rect
 (Xin_Rect_Msg       value(50),   /*  Message record              */
  Xin_Rect_Fil       value(57),   /*  File record                 */
  Xin_Rect_Opt       value(60),   /*  Options record              */
  Xin_Rect_Sum       value(61),   /*  Summary record              */
  Xin_Rect_Rep       value(62),   /*  Replace record              */
  Xin_Rect_Src       value(63),   /*  Source record               */
  Xin_Rect_Tok       value(64),   /*  Token record                */
  Xin_Rect_Sym       value(66),   /*  Symbol record               */
  Xin_Rect_Lit       value(67),   /*  Literal record              */
  Xin_Rect_Syn       value(69),   /*  Syntax record               */
  Xin_Rect_Ord_Type  value(80),   /*  Ordinal type record         */
  Xin_Rect_Ord_Elem  value(81),   /*  Ordinal element record      */
  Xin_Rect_Ctr       value(82) )  /*  Counter record              */
                     prec(15);
```

図 *114.* 序数値としてエンコードされたレコード・タイプ

レコードのヘッダー部分の宣言は、 567 ページの図 115 に示されています。

```
Dcl
  1 Xin_Hdr based,                 /*  Header portion               */
                                   /*                               */
    2 Xin_Hdr_Prod                 /*  Language code                */
           fixed bin(8) unsigned, /*                               */
                                   /*                               */
    2 Xin_Hdr_Rect                 /*  Record type                  */
        unal ordinal xin_Rect,     /*                               */
                                   /*                               */
    2 Xin_Hdr_Level                /*  SYSADATA level               */
           fixed bin(8) unsigned, /*                               */
                                   /*                               */
    2 * union,                     /*                               */
      3 xin_Hdr_Flags     bit(8),  /*  flags                        */
      3 *,                         /*                               */
        4 *               bit(6),  /*  Reserved                     */
        4 Xin_Hdr_Little_Endian    /*  ints are little endian       */
                          bit(1),  /*                               */
        4 Xin_Hdr_Cont    bit(1),  /*  Record continued in next rec */
                                   /*                               */
    2 Xin_Hdr_Edition              /*  compiler "edition"           */
           fixed bin(8) unsigned, /*                               */
                                   /*                               */
    2 Xin_Hdr_Fill        bit(32), /*  reserved                     */
                                   /*                               */
    2 Xin_Hdr_Data_Len             /*  length of data part          */
          fixed bin(16) unsigned, /*                               */
                                   /*                               */
    2 Xin_Hdr_End         char(0); /*                               */
```

図 *115.* レコードのヘッダー部分の宣言

サマリー・レコード

タイプ Xin_Rect_Sum のレコードはサマリー・レコードであり、ファイル内の先頭レコードでもあります。

宣言を 568 ページの図 116 に示します。

```
Dcl
  1 Xin_Sum     Based( ),          /* summary record               */
                                   /*                              */
    2 Xin_Sum_Hdr                  /* standard header              */
          like Xin_Hdr,            /*                              */
                                   /*                              */
    2 Xin_Sum_Max_Severity         /* max severity from compiler   */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Sum_Left_Margin          /* left margin                  */
          fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Sum_Right_Margin         /* right margin                 */
          fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 xin_Sum_Rsrvd(15)            /* reserved                     */
          fixed bin(32) unsigned; /*                              */
```

図 *116.* サマリー・レコードの宣言

オプション・レコード

タイプ Xin_Rect_Opt のレコードはオプション・レコードです。これは、コンパイル時に有効となるすべてのコンパイラー・オプションをリストします。

オプション・レコードの宣言は、サンプル・データ・セット SIBMZSAM 内のメンバー ibmwxop で指定されています。

カウンター・レコード

各カウンター・レコードは、後続のレコード・タイプについて、ファイルに入っているそのタイプのレコードの数と、それらのレコードが占有するバイト数を指定します。

```
Dcl
  1 xin_Ctr     based,             /* counter/size record          */
                                   /*                              */
    2 xin_Ctr_Hdr                  /* standard header              */
          like xin_Hdr,            /*                              */
                                   /*                              */
    2 xin_Ctr_Rect                 /* record type                  */
        unal ordinal xin_Rect,     /*                              */
                                   /*                              */
    2 *                            /*                              */
          fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 xin_Ctr_Count                /* count of that record type    */
          fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Ctr_Size                 /* size used                    */
          fixed bin(31) unsigned; /*                              */
```

図 *117.* カウンター・レコードの宣言

リテラル・レコード

それぞれのリテラル・レコードは、リテラル索引と呼ばれる 1 つの番号を割り当て、この特定のレコードに指定された文字を参照するために、後のレコードによって使用されるようにします。

```
Dcl
 1 xin_Lit     based,              /* literal record               */
                                   /*                              */
    2 xin_Lit_Hdr                  /* standard header              */
           like xin_Hdr,           /*                              */
                                   /*                              */
    2 xin_Lit_Inx                  /* adata index for literal      */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Lit_Len                  /* length of literal            */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Lit_Val      char(2000); /* literal value                */
```

図 *118.* リテラル・レコードの宣言

ファイル・レコード

それぞれのファイル・レコードは、ファイル索引と呼ばれる 1 つの番号を割り当て、このレコードに記述されたファイルを参照するために、後のレコードによって使用されるようにします。記述されたファイルは、PL/I 1 次ソース・ファイルか INCLUDE されたファイルの場合があります。それぞれのファイル・レコードは、そのファイルの完全修飾名に対するリテラル索引を指定します。

INCLUDE されたファイルの場合、それぞれのファイル・レコードには、INCLUDE 要求からのファイル索引とソース行番号も入ります。(1 次ソース・ファイルの場合、これらのフィールドはゼロになります。)

```
Dcl
  1 xin_Fil      based,            /* file record                  */
                                   /*                              */
    2 xin_Fil_Hdr                  /* standard header              */
           like xin_Hdr,           /*                              */
                                   /*                              */
    2 xin_Fil_File_Id              /* file id from whence it       */
           fixed bin(31) unsigned, /*  was INCLUDEd                */
                                   /*                              */
    2 xin_Fil_Line_No              /* line no within that file     */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Fil_Id                   /* id assigned to this file     */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Fil_Name                 /* literal index of the         */
           fixed bin(31) unsigned; /*  fully qualified file name   */
```

図 *119.* ファイル・レコードの宣言

メッセージ・レコード

それぞれのメッセージ・レコードは、コンパイル中に出されたメッセージを記述します。抑止されたメッセージに対しては、メッセージ・レコードは生成されません。

各メッセージ・レコードには以下のデータが含まれています。

- そのメッセージの起因となったファイルおよび行のファイル索引およびソース行番号。 メッセージがすべてコンパイルに関係するものである場合、このフィールドはゼロになります。
- メッセージに関連付けられている ID (例えば IBM1502) および重大度。
- そのメッセージのテキスト。

メッセージ・レコードの宣言は、以下のとおりです。

```
Dcl
  1 xin_Msg     based,             /* message record               */
                                   /*                              */
    2 xin_Msg_Hdr                  /* standard header              */
           like xin_Hdr,           /*                              */
                                   /*                              */
    2 xin_Msg_File_Id              /* file id                      */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Msg_Line_No              /* line no within file          */
           fixed bin(31) unsigned, /*                              */
                                   /*                              */
    2 xin_Msg_Id                   /* identifier (i.e. IBM1502)    */
                         char(16), /*                              */
                                   /*                              */
    2 xin_Msg_Severity             /* severity (0, 4, 8, 12 or 16) */
           fixed bin(15)   signed, /*                              */
                                   /*                              */
    2 xin_Msg_Length               /* length of message            */
           fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Msg_Text                 /* actual message               */
        char( 100 refer(xin_Msg_Length) );
```

図 *120.* メッセージ・レコードの宣言

SYSADATA シンボル情報について

XINFO コンパイル時オプションの SYM サブオプションを指定すると、コンパイラーは、MSG サブオプションに対して生成されるレコードに加えて、シンボル情報が含まれる SYSADATA ファイルも生成します。

以下のレコードにシンボル情報が含まれています。

- 序数タイプ・レコード
- 序数エレメント・レコード
- シンボル・レコード

シンボル・レコードは、組み込み関数、総称変数、または非定数エクステントを持つ変数の場合には生成されません。

序数タイプ・レコード

それぞれの序数タイプ・レコードは、序数タイプ索引と呼ばれる 1 つの番号を割り当て、このレコードに記述された序数タイプを参照するために、後のレコードによって使用されるようにします。 タイプの名前は、リテラル索引によって示されます。それぞれの序数タイプ・レコードには、その序数タイプが宣言されたファイルおよび行のファイル索引およびソース行番号が入っています。

それぞれの序数タイプ・レコードには、以下のものが入っています。

- そのタイプによって定義された値の数のカウント
- そのタイプに関連する精度
- 符号付きであるか符号なしであるかを示すビット

```
declare                              /*                              */
  1 xin_Ord_Type   based,            /*                              */
                                     /*                              */
    2 xin_Ord_Type_Hdr               /* standard header              */
          like xin_Hdr,              /*                              */
                                     /*                              */
    2 xin_Ord_Type_File_Id           /* file id                      */
          fixed bin(31) unsigned,    /*                              */
                                     /*                              */
    2 xin_Ord_Type_Line_No           /* line no within file          */
          fixed bin(31) unsigned,    /*                              */
                                     /*                              */
    2 xin_Ord_Type_Id                /* identifying number           */
                  fixed bin(31),     /*                              */
                                     /*                              */
    2 xin_Ord_Type_Count             /* count of elements            */
                  fixed bin(31),     /*                              */
                                     /*                              */
    2 xin_Ord_Type_Prec              /* precision for ordinal        */
          fixed bin(08) unsigned,    /*                              */
                                     /*                              */
    2 *,                             /*                              */
      3 xin_Ordinal_Type_Signed      /* signed attribute applies     */
                        bit(1),      /*                              */
      3 xin_Ordinal_Type_Unsigned    /* unsigned attribute applies   */
                        bit(1),      /*                              */
      3 *                            /* unused                       */
                        bit(6),      /*                              */
                                     /*                              */
    2 *                              /* unused                       */
                       char(2),      /*                              */
                                     /*                              */
    2 xin_Ord_Type_Name              /* type name                    */
                  fixed bin(31);     /*                              */
```

図 *121.* 序数タイプ・レコードの宣言

序数エレメント・レコード

それぞれの序数タイプ・レコードの直後には、その序数によって指定される値を記述する、一連のレコードが (序数タイプ・カウントで指定された数だけ) 続きます。それぞれの序数エレメント・レコードは、序数エレメント索引と呼ばれる 1 つの番号を割り当て、このレコードに記述された序数エレメントを参照するために、後のレコードによって使用されるようにします。

エレメントの名前は、リテラル索引によって示されます。 それぞれの序数エレメント・レコードには、その序数エレメントが宣言されたファイルおよび行のファイル索引およびソース行番号が入っています。

さらに、各序数エレメント・レコードには、以下のデータが含まれています。

- そのエレメントが属している序数タイプの序数タイプ索引
- そのエレメントの値

```
declare                              /*                          */
  1 xin_Ord_Elem   based,            /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Hdr               /* standard header          */
          like xin_Hdr,              /*                          */
                                     /*                          */
    2 xin_Ord_Elem_File_Id           /* file id                  */
          fixed bin(31) unsigned,    /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Line_No           /* line no within file      */
          fixed bin(31) unsigned,    /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Id                /* identifying number       */
                  fixed bin(31),     /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Type_Id           /* id of ordinal type       */
                  fixed bin(31),     /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Value             /* ordinal value            */
                  fixed bin(31),     /*                          */
                                     /*                          */
    2 xin_Ord_Elem_Name              /* ordinal name             */
                  fixed bin(31);     /*                          */
```

図 *122.* 序数エレメント・レコードの宣言

シンボル・レコード

シンボル・レコードの宣言は、サンプル・データ・セット SIBMZSAM 内のメンバー ibmwxin にあります。シンボル・レコードにはそれぞれ、シンボル索引と呼ばれる数値が割り当てられます。索引は、このレコードによって記述されるシンボルを参照するために、それ以降のレコードによって使用されます。

例えば、ユーザー変数または定数の名前に索引を使用することができます。ID の名前は、リテラル索引によって示されます。 それぞれのシンボル・レコードには、そのシンボルが宣言されたファイルおよび行のファイル索引およびソース行番号が入っています。

ID が構造体または共用体の一部である場合、シンボル・レコードには、以下のそれぞれに対するシンボル索引が入ります。

- 最初の兄弟 (存在する場合)
- 親 (存在する場合)
- 最初の子 (存在する場合)

以下の構造体を見てください。

```
dcl
  1 a
    , 3 b     fixed bin
    , 3 c     fixed bin
    , 3 d
      , 5 e   fixed bin
      , 5 f   fixed bin
;
```

上記の構造体のエレメントに割り当てられたシンボル索引は、以下のようになります。

symbol	index	sibling	parent	child
a	1	0	0	2
b	2	3	1	0
c	3	4	1	0
d	4	0	1	5
e	5	6	4	0
f	6	0	4	0

図 *123.* 構造体のエレメントに割り当てられたシンボル索引

それぞれのシンボル・レコードには、さまざまな属性がこの変数に適用されるかどうかを示す一連の bit(1) フィールドも含まれています。

それぞれのシンボル・レコードには、以下のエレメントも入っています。

ユーザー指定の構造体レベル
これは、ID に対するユーザー指定の構造体レベルです。 上記の構造体のエレメント c の場合、値は 3 です。非構造体メンバーの場合、値は 1 に設定されます。

論理構造体レベル
ID の論理構造体レベル。上記の構造体のエレメント c の場合、値は 2 です。非構造体メンバーの場合、値は 1 に設定されます。

次元

継承次元をカウントせずに、変数に宣言された次元の数。

すべての継承次元を含む、変数の次元の数。

オフセット
最外部の親構造体の中でのオフセット。

基本サイズ
基本サイズは、変数がビット位置合わせしている場合はビット単位、それ以外の場合はバイト単位です。 いずれの場合も、これは、どの次元の因数でもありません。

サイズ
その次元の因数となるバイト単位でのサイズ。

位置合わせ
以下によって、示されます。

- 0 (ビット位置合わせ)
- 7 (バイト位置合わせ)
- 15 (ハーフワード位置合わせ)
- 31 (フルワード位置合わせ)
- 63 (4 倍長ワード位置合わせ)

レコード内の共用体は、変数のストレージ・クラスに従属する情報を記述するための専用のものです。

静的変数
: 変数が、別個の外部名を持つ外部 (dcl x ext('y')) として宣言された場合、その名前のリテラル索引が指定されます。

基底付き変数
: 変数が、配列のエレメントではない別のマップ変数を基にするものとして宣言された場合、その変数のシンボル索引が指定されます。

定義済み変数
: 変数が、配列のエレメントではない別のマップされた変数で定義されたものとして宣言された場合、その変数のシンボル索引がここに指定されます。 これは、その位置属性が定数である場合にも指定されます。

変数のデータ・タイプは、図 124 に示された序数によって指定されます。

```
define
  ordinal
    xin_Data_Kind
      (  xin_Data_Kind_Unset
        ,xin_Data_Kind_Character
        ,xin_Data_Kind_Bit
        ,xin_Data_Kind_Graphic
        ,xin_Data_Kind_Fixed
        ,xin_Data_Kind_Float
        ,xin_Data_Kind_Picture
        ,xin_Data_Kind_Pointer
        ,xin_Data_Kind_Offset
        ,xin_Data_Kind_Entry
        ,xin_Data_Kind_File
        ,xin_Data_Kind_Label
        ,xin_Data_Kind_Format
        ,xin_Data_Kind_Area
        ,xin_Data_Kind_Task
        ,xin_Data_Kind_Event
        ,xin_Data_Kind_Condition
        ,xin_Data_Kind_Structure
        ,xin_Data_Kind_Union
        ,xin_Data_Kind_Descriptor
        ,xin_Data_Kind_Ordinal
        ,xin_Data_Kind_Handle
        ,xin_Data_Kind_Type
        ,xin_Data_Kind_Builtin
        ,xin_Data_Kind_Generic
        ,xin_Data_Kind_Widechar
                                  )  prec(8) unsigned;
```

図 *124.* 変数のデータ・タイプ

レコード内の共用体は、変数のデータ・タイプに従属する情報を記述するための専用のものです。 この情報 (算術型の精度など) の多くは説明不要です。ただし、以下の変数は例外と考えられます。

ピクチャー変数
: ピクチャー指定のリテラル索引が指定されます。

入り口変数
: 変数が戻り属性を持っている場合、戻り記述のシンボル索引が指定されます。

序数変数
序数タイプ索引が指定されます。

タイプ付き変数およびハンドル
基礎となるタイプのシンボル索引が指定されます。

ストリングおよび区域変数
戻り記述のシンボル索引に加えて、エクステントのタイプと値が指定されます。エクステントのタイプは、以下の値によってエンコードされます。

```
declare
  (  xin_Extent_Constant     value(01)
    ,xin_Extent_Star         value(02)
    ,xin_Extent_Nonconstant  value(04)
    ,xin_Extent_Refer        value(08)
    ,xin_Extent_In_Error     value(16)
  )
  fixed bin;
```

エレメントが何らかの次元を持つ場合、その下部と上部の境界のタイプと値が、レコードの最後に指定されます。これらのフィールドは、エレメントが次元を持たない場合には存在しません。

属性フラグには、コンパイラーがすべてのデフォルトを適用した後の属性が反映されることにも注意してください。 したがって、例えば、どの数値変数 (数値 PICTURE 変数を含む) にも、REAL または COMPLEX のいずれかの属性フラグが設定されます。

SYSADATA 構文情報について

XINFO コンパイル時オプションの SYN サブオプションを指定すると、コンパイラーは、MSG および SYM サブオプションの場合に生成されるレコードに加えて、構文情報が入った SYSADATA ファイルを生成します。

以下のレコードに構文情報が含まれています。

- ソース・レコード
- トークン・レコード
- 構文レコード

ソース・レコード

各ソース・レコードでは、ソース ID と呼ばれる 1 つの番号が割り当てられます。後のレコードでこのレコードに記述されたソース行を参照するときは、この番号が使用されます。 行は、レコード内のソース・ファイル ID フィールドと行番号フィールドで示されるように、PL/I 1 次ソース・ファイルまたは INCLUDE ファイルに含まれている可能性があります。このレコードの残りの部分は、ソース行の実データを保持します。

```
Dcl
  1 Xin_Src     based,             /* source record                */
                                   /*                              */
    2 Xin_Src_Hdr                  /* standard header              */
           like Xin_Hdr,           /*                              */
                                   /*                              */
    2 Xin_Src_File_Id              /* file id                      */
           fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Src_Line_No              /* line no within file          */
           fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Src_Id                   /* id for this source record    */
           fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Src_Length               /* length of text               */
           fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Src_Text                 /* actual text                  */
        char( 137 refer(xin_Src_Length) );
```

図 *125.* ソース・レコードの宣言

トークン・レコード

各トークン・レコードでは、トークン索引と呼ばれる 1 つの番号が割り当てられます。後のレコードで PL/I コンパイラーに認識されるトークンを参照するときは、この番号が使用されます。 また、このレコードは、トークンのタイプに加えて、トークンの開始と終了それぞれの列と行を示します。

```
Dcl
  1 Xin_Tok     based,             /* token record                 */
                                   /*                              */
    2 Xin_Tok_Hdr                  /* standard header              */
           like Xin_Hdr,           /*                              */
                                   /*                              */
    2 Xin_Tok_Inx                  /* adata index for token        */
           fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Tok_Begin_Line           /* starting line no within file */
           fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Tok_End_Line_Offset      /* offset of end line from first */
           fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Tok_Kind_Value           /* token kind                   */
           ordinal xin_Tok_Kind,   /*                              */
                                   /*                              */
    2 Xin_Tok_Rsrvd                /* reserved                     */
           fixed bin(8) unsigned,  /*                              */
                                   /*                              */
    2 Xin_Tok_Begin_Col            /* starting column              */
           fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Tok_End_Col              /* ending column                */
           fixed bin(16) unsigned; /*                              */
```

図 *126.* トークン・レコードの宣言

序数 xin_Tok_Kind は、トークン・レコードのタイプを示します。

```
Define
 ordinal
   xin_Tok_Kind
     (  xin_Tok_Kind_Unset
       ,xin_Tok_Kind_Lexeme
       ,xin_Tok_Kind_Comment
       ,xin_Tok_Kind_Literal
       ,xin_Tok_Kind_Identifier
       ,xin_Tok_Kind_Keyword
                                   )  prec(8) unsigned;
```

図 *127.* トークン・レコードの種類の宣言

構文レコード

各構文レコードでは、ノード ID と呼ばれる 1 つの番号が割り当てられます。後のレコードで他の構文レコードを参照するときは、この番号が使用されます。最初の構文レコードの種類は xin_Syn_Kind_Package です。コンパイル単位の中にプロシージャーがある場合、このレコードの子ノードが、プロシージャーのうちの最初のものを指します。 次に親ノード、兄弟ノード、および子ノードに基づいて、コンパイル単位に含まれるすべてのプロシージャーと開始ブロックの特有の関係が 1 つのマップに設定されます。

次の単純なプログラムで考えてみます。

```
a: proc;
  call b;
  call c;
  b: proc;
  end b;
  c: proc;
    call d;
    d: proc;
    end d;
  end c;
end a;
```

このプログラムのブロックにノード索引が次のように割り当てられます。

symbol	index	sibling	parent	child
-	1	0	0	2
a	2	0	1	3
b	3	4	2	0
c	4	0	2	5
d	5	0	4	0

図 *128.* プログラムのブロックに割り当てられるノード索引

```
Dcl
  1 Xin_Syn     based,             /* syntax record                */
                                   /*                              */
    2 Xin_Syn_Hdr                  /* standard header              */
          like Xin_Hdr,            /*                              */
                                   /*                              */
    2 Xin_Syn_Node_Id              /* node id                      */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Syn_Node_Kind            /* node type                    */
          ordinal xin_syn_kind,    /*                              */
                                   /*                              */
    2 Xin_Syn_Node_Exp_Kind        /* node sub type                */
          ordinal xin_exp_kind,    /*                              */
                                   /*                              */
    2 *                            /* reserved                     */
          fixed bin(16) unsigned, /*                              */
                                   /*                              */
    2 Xin_Syn_Parent_Node_Id       /* node id of parent            */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Syn_Sibling_Node_Id      /* node id of sibling           */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 Xin_Syn_Child_Node_Id        /* node id of child             */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 xin_Syn_First_Tok            /* id of first spanned token    */
          fixed bin(32) unsigned, /*                              */
                                   /*                              */
    2 xin_Syn_Last_Tok             /* id of last spanned token     */
          fixed bin(32) unsigned, /*                              */
```

図 *129.* 構文レコードの宣言

```
                                  /*                              */
  2 * union,                     /* qualifier for node            */
                                  /*                               */
    3 Xin_Syn_Int_Value          /* used if int                   */
        fixed bin(31),           /*                               */
                                  /*                               */
    3 Xin_Syn_Literal_Id         /* used if name, number, picture */
        fixed bin(31),           /*                               */
                                  /*                               */
    3 Xin_Syn_Node_Lex           /* used if lexeme, assignment,   */
        ordinal xin_Lex_kind,    /* infix_op, prefix_op           */
                                  /*                               */
    3 Xin_Syn_Node_Voc           /* used if keyword, end_for_do   */
        ordinal xin_Voc_kind,    /*                               */
                                  /*                               */
    3 Xin_Syn_Block_Node         /* used if call_begin            */
        fixed bin(31),           /*   to hold node of begin block */
                                  /*                               */
    3 Xin_Syn_Bif_Id             /* used if bif_rfrnc             */
        fixed bin(32) unsigned, /*                               */
                                  /*                               */
    3 Xin_Syn_Sym_Id             /* used if label, unsub_rfrnc,   */
        fixed bin(32) unsigned, /* subscripted_rfrnc             */
                                  /*                               */
    3 Xin_Syn_Proc_Data,         /* used if package, proc or begin*/
                                  /*                               */
      4 Xin_Syn_First_Sym        /*   id of first contained sym   */
        fixed bin(32) unsigned, /*                               */
                                  /*                               */
      4 Xin_Syn_Block_Sym        /*   id of sym for this block    */
        fixed bin(32) unsigned, /*                               */
```

構文レコードの宣言 *(続き)*

```
                                  /*                              */
    3 Xin_Syn_Number_Data,        /* used if number                */
                                  /*                               */
      4 Xin_Syn_Number_Id         /*   id of literal               */
        fixed bin(32) unsigned,   /*                               */
                                  /*                               */
      4 Xin_Syn_Number_Type       /*   type                        */
        ordinal xin_Number_Kind,  /*                               */
                                  /*                               */
      4 Xin_Syn_Number_Prec       /*   precision                   */
        fixed bin(8) unsigned,    /*                               */
                                  /*                               */
      4 Xin_Syn_Number_Scale      /*   scale factor                */
        fixed bin(7) signed,      /*                               */
                                  /*                               */
      4 Xin_Syn_Number_Bytes      /*   bytes it would occupy       */
        fixed bin(8) unsigned,    /*   in its internal form        */
                                  /*                               */
    3 Xin_Syn_String_Data,        /* used if char_string,          */
                                  /* bit_string, graphic_string    */
                                  /*                               */
      4 Xin_Syn_String_Id         /*   id of literal               */
        fixed bin(32) unsigned,   /*                               */
                                  /*                               */
      4 Xin_Syn_String_Len        /*   string length in its units  */
        fixed bin(32) unsigned,   /*                               */
                                  /*                               */
    3 Xin_Syn_Stmt_Data,          /* used if stmt                  */
                                  /*                               */
      4 Xin_Syn_File_Id           /*   file id                     */
        fixed bin(32) unsigned,   /*                               */
                                  /*                               */
      4 Xin_Syn_Line_No           /*   line no within file         */
        fixed bin(32) unsigned,   /*                               */
                                  /*                               */
  2 *               char(0);      /*                               */
```

構文レコードの宣言 *(*続き*)*

序数 xin_Syn_Kind は、構文レコードのタイプを示します。

```
Define
  ordinal
    xin_Syn_Kind
      (  xin_Syn_Kind_Unset
        ,xin_Syn_Kind_Lexeme
        ,xin_Syn_Kind_Asterisk
        ,xin_Syn_Kind_Int
        ,xin_Syn_Kind_Name
        ,xin_Syn_Kind_Expression
        ,xin_Syn_Kind_Parenthesized_Expr
        ,xin_Syn_Kind_Argument_List
        ,xin_Syn_Kind_Keyword
        ,xin_Syn_Kind_Proc_Stmt
        ,xin_Syn_Kind_Begin_Stmt
        ,xin_Syn_Kind_Stmt
        ,xin_Syn_Kind_Substmt
        ,xin_Syn_Kind_Label
        ,xin_Syn_Kind_Invoke_Begin
        ,xin_Syn_Kind_Assignment
        ,xin_Syn_Kind_Assignment_Byname
        ,xin_Syn_Kind_Do_Fragment
        ,xin_Syn_Kind_Keyed_List
        ,xin_Syn_Kind_Iteration_Factor
        ,xin_Syn_Kind_If_Clause
        ,xin_Syn_Kind_Else_Clause
        ,xin_Syn_Kind_Do_Stmt
        ,xin_Syn_Kind_Select_Stmt
        ,xin_Syn_Kind_When_Stmt
        ,xin_Syn_Kind_Otherwise_Stmt
        ,xin_Syn_Kind_Procedure
        ,xin_Syn_Kind_Package
        ,xin_Syn_Kind_Begin_Block
        ,xin_Syn_Kind_Picture
        ,xin_Syn_Kind_Raw_Rfrnc
        ,xin_Syn_Kind_Generic_Desc
                                    )  prec(8) unsigned;
```

図 *130.* 構文レコードの種類の宣言

次の単純なプログラムで考えてみます。

```
a: proc(x);
  dcl x char(8);
  x = substr(datetime(),1,8);
end;
```

このプログラムのブロックにノード索引が次のように割り当てられます。

```
node_kind      index     sibling      parent    child
-----------    -----     -------      ------    -----
 package         1          0           0         2
 procedure       2          0           1         0
 expression      3          0           0         0
 stmt            4          5           2         6
 stmt            5         10           2        11
 label           6          7           4         0
 keyword         7          8           4         0
 expression      8          9           4         0
 lexeme          9          0           4         0
 stmt           10          0           2        18
 assignment     11         12           5        13
 lexeme         12          0           5         0
 expression     13         14          11         0
 expression     14          0          11        15
 expression     15         16          14         0
 expression     16         17          14         0
 expression     17          0          14         0
 keyword        18         19          10         0
 lexeme         19          0          10         0
```

図 *131.* プログラムの構文レコードに割り当てられるノード索引

プロシージャー・レコードには、ENTRY A のシンボル・レコードの ID が含まれます (block_sym フィールド内)。このシンボル・レコードには、そのプロシージャーの最初のステートメントのノード ID が含まれます (first_stmt_id フィールド内)。

ステートメント・レコードの場合、兄弟ノード ID は次のステートメント・レコードを指します (次のステートメント・レコードが存在する場合)。子ノード ID は、そのステートメント・レコードの最初のエレメントを指します。

PROCEDURE ステートメントのレコードは、次の 4 つのレコードから成ります。

- ラベル・レコード。
- キーワード・レコード (PROCEDURE キーワード用)。
- 式レコード (パラメーター X 用)。これには、式の種類 unsub_rfrnc とシンボル X の sym_id が含まれます。
- 字句レコード (セミコロン用)。

割り当てステートメントのレコードは、次の 2 つのレコードから成ります。

- 次の 2 つの子を持つ割り当てレコード。
  - 式レコード (ターゲット X 用)。これには、式の種類 unsub_rfrnc とシンボル X の sym_id が含まれます。
  - 式レコード (ソース用)。これには、式の種類 builtin_rfrnc とシンボル SUBSTR の sym_id が含まれます。このレコード自体が、次の 3 つの子を持ちます。
    - 式レコード (最初の引数用)。これには、式の種類 builtin_rfrnc とシンボル DATETIME の sym_id が含まれます。
    - 式レコード (2 番目の引数用)。これには、式の種類 number と値 1 の literal_id が含まれます。

- 式レコード (3 番目の引数用)。これには、式の種類 number と値 8 の literal_id が含まれます。

- 字句レコード (セミコロン用)。

END ステートメントのレコードは、次の 2 つのレコードから成ります。

- キーワード・レコード (END キーワード用)。
- 字句レコード (セミコロン用)。

序数 xin_Exp_Kind は、式が記述された構文レコードの式のタイプを示します。このレコードの中には、子ノードの数がゼロでないものもあります。例えば、次のような場合です。

- 挿入演算子 (減算を表すマイナスなど) は、その左オペランドが記述された子ノードを持ちます (このオペランドの兄弟ノードに、右演算子が記述されます)。
- 接頭演算子 (否定を表すマイナスなど) は、そのオペランドが記述された子ノードを持ちます。

```
Define
 ordinal
   xin_Exp_Kind
     (  xin_Exp_Kind_Unset
       ,xin_Exp_Kind_Bit_String
       ,xin_Exp_Kind_Char_String
       ,xin_Exp_Kind_Graphic_String
       ,xin_Exp_Kind_Number
       ,xin_Exp_Kind_Infix_Op
       ,xin_Exp_Kind_Prefix_Op
       ,xin_Exp_Kind_Builtin_Rfrnc
       ,xin_Exp_Kind_Entry_Rfrnc
       ,xin_Exp_Kind_Qualified_Rfrnc
       ,xin_Exp_Kind_Unsub_Rfrnc
       ,xin_Exp_Kind_Subscripted_Rfrnc
       ,xin_Exp_Kind_Type_Func
       ,xin_Exp_Kind_Widechar_String
                                  )  prec(8) unsigned;
```

図 *132.* 式の種類の宣言

序数 xin_Number_Kind は、数値が記述された構文レコードの数値のタイプを示します。

```
Define
 ordinal
   xin_Number_Kind
     (  xin_Number_Kind_Unset
       ,xin_Number_Kind_Real_Fixed_Bin
       ,xin_Number_Kind_Real_Fixed_Dec
       ,xin_Number_Kind_Real_Float_Bin
       ,xin_Number_Kind_Real_Float_Dec
       ,xin_Number_Kind_Cplx_Fixed_Bin
       ,xin_Number_Kind_Cplx_Fixed_Dec
       ,xin_Number_Kind_Cplx_Float_Bin
       ,xin_Number_Kind_Cplx_Float_Dec
                                  )  prec(8) unsigned;
```

図 *133.* 数値の種類の宣言

序数 xin_Lex_Kind は、字句単位が記述された構文レコードの字句のタイプを示します。

- これらの序数名にある「vrule」は「垂直罫線」(「or」記号などとして使用される) を意味します。
- これらの序数名にある「dbl」は「二重」を意味します。したがって dbl_Vrule は、「連結」記号などとして使用される二重垂直罫線です。

```
Define
 ordinal
   xin_Lex_Kind
     (  xin_Lex_Undefined
       ,xin_Lex_Period
       ,xin_Lex_Colon
       ,xin_Lex_Semicolon
       ,xin_Lex_Lparen
       ,xin_Lex_Rparen
       ,xin_Lex_Comma
       ,xin_Lex_Equals
       ,xin_Lex_Gt
       ,xin_Lex_Ge
       ,xin_Lex_Lt
       ,xin_Lex_Le
       ,xin_Lex_Ne
       ,xin_Lex_Lctr
       ,xin_Lex_Star
       ,xin_Lex_Dbl_Colon
       ,xin_Lex_Not
       ,xin_Lex_Vrule
       ,xin_Lex_Dbl_Vrule
       ,xin_Lex_And
       ,xin_Lex_Dbl_Star
       ,xin_Lex_Plus
       ,xin_Lex_Minus
       ,xin_Lex_Slash
       ,xin_Lex_Equals_Gt
       ,xin_Lex_Lparen_Colon
       ,xin_Lex_Colon_Rparen
       ,xin_Lex_Plus_Equals
       ,xin_Lex_Minus_Equals
       ,xin_Lex_Star_Equals
       ,xin_Lex_Slash_Equals
       ,xin_Lex_Vrule_Equals
       ,xin_Lex_And_Equals
       ,xin_Lex_Dbl_Star_Equals
       ,xin_Lex_Dbl_Vrule_Equals
       ,xin_Lex_Dbl_Slash
     ) unsigned prec(16);
```

図 *134.* 字句の種類の宣言

序数 xin_Voc_Kind は、コンパイラーの「語彙」から項目が記述された構文レコードのキーワードを示します。

語彙のたぐいの宣言は、サンプル・データ・セット SIBMZSAM においてメンバー ibmwxin で指定されます。

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す。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 以外の製品とプログラムの操作またはサービスの評価および検証は、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。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。

本書は定期的に見直され、必要な変更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。

© Copyright IBM Corp. 1999, 2017

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtml をご覧ください。

Intel および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社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、および Windows NT は、Microsoft Corporation の米国およびその他の国における商標です。

Pentium は Intel Corporation または子会社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。

UNIX は The Open Group の米国およびその他の国における登録商標です。

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。

参考文献

PL/I 資料

Enterprise PL/I for z/OS

「プログラミング・ガイド」、GI88-4249
「言語解説書」、SA88-4235
「メッセージおよびコード」、GA88-4237
「コンパイラーおよびランタイム 移行ガイド」、GA88-4236

PL/I for MVS & VM

「*PL/I MVS&VM VA PL/I MEL* インストールとカストマイズ」、SC88-7221
「言語解説書」、SC88-7219
「コンパイル時メッセージおよびコード」、SC88-7224
「診断の手引き」、SC88-7223
「移行の手引き」、SC88-7220
「プログラミング・ガイド」、SC88-7218
「参照要約」、SX88-7011

Enterprise PL/I for AIX

「プログラミング・ガイド」、SA88-4427
「言語解説書」、SA88-4429
「メッセージおよびコード」、GA88-4430
「インストール・ガイド」、GA88-4428

関連資料

DB2 for z/OS

「管理ガイド」、SA88-4204

「アプリケーション・プログラミングおよび *SQL* 解説書」、SA88-4205

「コマンド解説書」、SA88-4208

「メッセージ」、GA88-4213

「コード」、GA88-4207

「*SQL* 解説書」、SA88-4217

インフォメーション・センター (publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc/src/alltoc/db2z_10_prodhome.htm) も参照してください。

「*LOBs with DB2 for z/OS: Stronger and Faster*」、SG24-7270

DFSORT

「アプリケーション・プログラミングの手引き」、SC88-7061

「導入およびカスタマイズ」、SC88-7163

© Copyright IBM Corp. 1999, 2017

IMS/ESA®

「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

「*Application Programming: Database Manager Summary*」、SC26-8037

「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: トランザクション管理プログラム」、SC88-7553

「*Application Programming: Transaction Manager Summary*」、SC26-8038

「アプリケーション・プログラミング: *EXEC DL/I* コマンド *(CICS* および *IMS™)*」、SC88-7554

「*Application Programming: EXEC DL/I Commands for CICS and IMS Summary*」、SC26-8036

TXSeries® for Multiplatforms

「*Encina* 管理の手引き 第 *2* 巻：サーバー管理」、SD88-7403

「*Encina SFS Programming Guide*」、SC09-4483

インフォメーション・センター (publib.boulder.ibm.com/infocenter/txformp/v7r1/index.jsp) も参照してください。

z/Architecture

「解説書」、SA88-8773

z/OS 言語環境プログラム

「概念」、SA88-8555

「デバッグ・ガイド」、GA88-8548

「ランタイム・メッセージ」、SA88-8554

「カスタマイズ」、SA88-8552

「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・アプリケーション マイグレーション・ガイド」、GA88-8553

「*ILC* (言語間通信) アプリケーションの作成」、SA88-8551

z/OS MVS

「*JCL* 解説書」、SA88-8569

「*JCL* ユーザーズ・ガイド」、SA88-8570

「システム・コマンド」、SA88-8593

z/OS TSO/E

「コマンド解説書」、SA88-8628

「ユーザーズ・ガイド」、SA88-8638

z/OS UNIX システム・サービス

「*z/OS UNIX* システム・サービス コマンド解説書」、SA88-8641

「*z/OS 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

「*z/OS UNIX* システム・サービス ユーザーズ・ガイド」、SA88-8640

Unicode および文字表現

「*z/OS Unicode* サポート*:* 変換サービスの使用法」、SD88-6163

用語集

この用語集は、PL/I のすべてのプラットフォームとリリースで使用する用語を定義したものです。このマニュアルで使用されていない用語が含まれていることがあります。 該当する用語が見つからない場合は、本書の索引を調べるか、「*IBM Dictionary of Computing*」(SC20-1699) を参照してください。

A

アクセス **(access)**
: データを参照するかまたは取り出すこと。

処置指定 **(action specification)**
: ON ステートメント内にある、ON ユニットまたは単一のキーワード SYSTEM。該当する条件が発生すれば、2 つのうちいずれかがとるべき処置を指定する。

活動化 **(**ブロックの**) (activate (a block))**
: ブロックの実行を開始すること。 プロシージャー・ブロックは、呼び出されるときに、活動化される。 開始ブロックが活動化するのは、分岐を含め、通常の制御の流れ内に現れたときである。 パッケージを活動化することはできない。

活動化 **(**プリプロセッサー変数またはプリプロセッサー・エントリー・ポイントの**) (activate (a preprocessor variable or preprocessor entry point))**
: マクロ機能 ID を、それに後続するソース・コード内で置換可能にすること。%ACTIVATE ステートメントは、プリプロセッサー変数やプリプロセッサー・エントリー・ポイントを活動化する。

アクティブ **(active)**
: 活動化から終了にいたるまでのブロックの状態。ソース・プログラム・テキスト中の対応する ID をプリプロセッサー変数やプリプロセッサー入り口名の値に置き換えることができるときの、その変数や入り口の状態。イベント変数が非同期操作に関連付けられている間に置かれている状態。タスク変数に関連するタスクが付加されるときにタスク変数が置かれている状態。タスクが終了する前に置かれている状態。

実際の起点 **(actual origin (AO))**
: 配列または構造体内の最初の項目の位置。

追加属性 **(additive attribute)**
: デフォルトを持たず、必要であれば明示的に述べるか、または、明示的に述べられた別の属性で暗黙指定しなければならないファイル記述属性。「代替属性 *(alternative attribute)*」と対比。

調節可能エクステント **(adjustable extent)**
: 関連変数の世代によって異なることのある境界 (配列の)、長さ (ストリングの)、またはサイズ (区域の)。 調節可能エクステントは、世代ごとに別々に評価される式またはアスタリスク (ただし、基底付き変数の場合は REFER オプション) で指定される。 静的変数に使用することはできない。

集合 **(aggregate)**
: 「データ集合 *(data aggregate)*」を参照。

集合式 **(aggregate expression)**
: 配列式、構造式、または共用体式のこと。

集合タイプ **(aggregate type)**
: どのデータ項目の場合も、それが構造体、共用体、または配列のいずれであるかの指定。

割り振られた変数 **(allocated variable)**
: 主記憶域が関連付けられ解放されない変数。

割り振り **(allocation)**
: 変数用の主記憶域の予約。 割り振られた変数の世代。 PL/I ファイルを、システム・データ・セット、装置、またはファイルに関連付けること。

位置合わせ **(alignment)**
: 機械に依存する特定の境界 (例えば、フルワード境界またはハーフワード境界) に関連付けて、データ項目を保管すること。

© Copyright IBM Corp. 1999, 2017

英字 **(alphabetic character)**
A から Z までの任意の英字と、#、$、@ (これらのグラフィック表記は国によって異なる場合がある) の拡張英字。

英数字 **(alphameric character)**
英字または数字。

代替属性 **(alternative attribute)**
属性グループから選択するファイル記述属性。何も指定しないと、デフォルトがとられる。 「追加属性 *(additive attribute)*」と対比。

あいまい参照 **(ambiguous reference)**
参照時点で認識されている名前をただ 1 つだけ識別するためには修飾が不十分な参照。

域 **(area)**
基底付き変数を割り振ることのできる、ストレージ中の部分。

引数 **(argument)**
サブルーチンまたは機能の呼び出しの一部である引数リスト内にある式。

引数リスト **(argument list)**
コンマで区切られ、入り口名定数、入り口名変数、総称名、または組み込み関数名に続く、括弧で囲まれたゼロまたはそれ以上の引数のリスト。 そのリストは、エントリー・ポイントのパラメーター・リストである。

算術比較 **(arithmetic comparison)**
数値の比較。 「ビット比較 *(bit comparison)*」、「文字比較 *(character comparison)*」も参照。

算術定数 **(arithmetic constant)**
固定小数点定数または浮動小数点定数。 大部分の算術定数には符号を付けることができるが、符号は定数の一部ではない。

算術変換 **(arithmetic conversion)**
ある 1 つの算術表現から別の表現に値を変換すること。

算術データ **(arithmetic data)**
基数、スケール、モード、および精度の特性を持つデータ。コード化算術データとピクチャー数字データも含まれる。

算術演算子 **(arithmetic operators)**
接頭演算子の + と -、あるいは挿入演算子 + - * / ** のうちのいずれか。

配列 **(array)**
同じ属性を持ち、1 つ以上の次元別にグループ分けされた 1 つ以上のデータ・エレメントに、名前を付けて順番に並べた集合体。

配列式 **(array expression)**
評価されると値の配列が生成される式。

構造体の配列 **(array of structures)**
次元属性を構造体名に与えて指定される、順番に並べられた同一構造体の集まり。

配列変数 **(array variable)**
同じ属性を持っていなければならないデータ項目の集合を表す変数。 「構造変数 *(structure variable)*」と対比。

ASCII
情報交換用米国標準コード (American National Standard Code for Information Interchange)。

代入 **(assignment)**
値を変数に与える処理。

非同期操作 **(asynchronous operation)**
ステートメントの実行と、入出力操作が並行して行われること。 各種タスクに複数の制御の流れを使った、プロシージャーの並行実行。

タスクの生成 **(attachment of a task)**
呼び込まれたプロシージャー (およびこれが呼び出すプロシージャー) を、呼び出しプロシージャーの実行と一緒に、非同期で実行するために、プロシージャーを呼び出して別に制御の流れを確立すること。

アテンション **(attention)**
タスクに割り込みが生じる原因となるような、タスクにとっては外部の事柄の発生。

属性 **(attribute)**
表明された特性を記述するのに名前と関連付けた記述特性。 式の計算の結果の特性を説明するために用いられる記述特性。

自動ストレージ割り振り **(automatic storage allocation)**
自動変数用のストレージ割り振り。

自動変数 **(automatic variable)**
ブロックの起動時に自動的にストレージを割り振られ、そのブロックの終了時に自動的にそれを解除される変数。

B

基数 **(base)**
算術値を表現するための数体系。

基本エレメント **(base element)**
それ自身は別の構造体や共用体ではない、構造体や共用体のメンバー。

基本項目 **(base item)**
定義変数を定義するための、自動、被制御、または静的変数、またはパラメーター。

基底付き参照 **(based reference)**
基底付きストレージ・クラスを持った参照。

基底付きストレージ割り振り **(based storage allocation)**
基底付き変数用のストレージの割り振り。

基底付き変数 **(based variable)**
ストレージ・アドレスがロケーターによって与えられる変数。 同一変数の複数の世代をアクセスすることができる。 これは、ストレージ内の固定位置を識別しない。

開始ブロック **(begin-block)**
BEGIN ステートメントと END ステートメントによって区切られ、名前有効範囲を形成するステートメントの集まり。 開始ブロックの活動化は、条件が生じたために行われる (開始ブロックが ON ユニットへの処置指定である場合) か、または GOTO ステートメントの結果の分岐を含め、通常の制御の流れを介して行われる。

2 進数 **(binary)**
0 と 1 が唯一の数表示である数体系。

2 進数字 **(binary digit)**
「ビット *(bit)*」を参照。

2 進固定小数点値 **(binary fixed-point value)**
2 進数字で構成され、オプションの 2 進小数点とオプションの符号を持った整数。「*10* 進固定小数点値 *(decimal fixed-point value)*」と対比。

2 進浮動小数点値 **(binary floating-point value)**
2 進小数部と見なせる仮数と、2 の基数に対する整数指数と見なせる指数の形式の実数の近似値。「*10* 進浮動小数点値 *(decimal floating-point value)*」と対比。

ビット **(bit)**
0 または 1。コンピューター・ストレージの最小スペース量。

ビット比較 **(bit comparison)**
2 進数字を、左から右へビットごとに比較すること。 「算術比較 *(arithmetic comparison)*」、「文字比較 *(character comparison)*」も参照。

ビット・ストリング定数 **(bit string constant)**
囲まれていて、接尾部 B が直後に付いた 2 進数字の連なり。「文字定数 *(character constant)*」と対比。単一引用符に囲まれ、後に接尾部 B4 が付いた 16 進数字の連なり。

ビット・ストリング **(bit string)**
ゼロ以上のビットで構成されたストリング。

ビット・ストリング演算子 **(bit string operators)**
論理演算子 NOT と排他 OR (¬)、 AND (&)、および OR (|)。

ビット値 **(bit value)**
ビット・タイプを表す値。

ブロック **(block)**
その中で宣言された名前の有効範囲と、その名前用のストレージ割り振りを指定する、1 つの単位として処理される一連のステートメント。 ブロックとしては、パッケージ、プロシージャー、または開始ブロックのいずれもありえる。

境界 **(bounds)**
任意の配列次元の上限と下限。

区切り文字 **(break character)**
下線記号 (_)。 ID を読みやすくするために使用することができる。 例えば、変数を OLDINVENTORYTOTAL とする代わりに OLD_INVENTORY_TOTAL と記述できる。

組み込み関数 **(built in function)**
SQRT (平方根) のような、言語が提供する定義済み関数。

組み込み関数参照 **(built-in function reference)**
オプションの引数リストを持つ組み込み関数名。

組み込み名 **(built-in name)**
組み込みサブルーチンの入り口名。

組み込みサブルーチン **(built-in subroutine)**
コンパイル時に定義され、CALL ステートメントによって呼び出される入り口名を持つサブルーチン。

バッファー **(buffer)**
レコードが入力時に読み込まれ、レコードが出力時に書き出される、入出力操作に使用する中間記憶域。

C

呼び出し **(call)**
CALL ステートメントまたは CALL オプションを使用してサブルーチンを呼び出すこと。

文字比較 **(character comparison)**
照合順序に従って、左から右へ文字単位で行われる比較。 「算術比較 *(arithmetic comparison)*」、「ビット比較 *(bit comparison)*」も参照。

文字ストリング定数 **(character string constant)**
単一引用符で囲まれた一連の文字。例えば、'Shakespeare''s 'Hamlet:'' など。

文字セット **(character set)**
あらかじめ決められた文字の集まり。「*ASCII*」と「*EBCDIC*」を参照。

文字ストリング・ピクチャー・データ **(character string picture data)**
文字値だけを持ったピクチャー・データ。このタイプのピクチャー・データは、少なくとも 1 つの A または X ピクチャー指定文字を持っていなければならない。「数値ピクチャー・データ *(numeric picture data)*」と対比。

クローズ **(ファイルの) (closing (of a file))**
ファイルをデータ・セットまたは装置と切り離すこと。

コード化算術データ **(coded arithmetic data)**
数値を表し、基数 (10 進数または 2 進数)、スケール (固定小数点または浮動小数点)、および精度 (個々に持ちうる桁数) を特徴とするデータ項目。 このデータは、変換しなくても、算術計算用に受け入れることのできる形式で保管される。

複合ネストの深さ **(combined nesting depth)**
プログラム内の PROCEDURE/BEGIN/ON、DO、SELECT、および IF...THEN...ELSE によるネストのレベルをカウントして決定される、最も深いネスト・レベル。

コメント **(comment)**
文書化のために使用され、/* および */ で区切られる、ゼロ以上の文字数の文字ストリング。

商用文字 **(commercial character)**
- CR (貸方) ピクチャー指定文字。
- DB (借方) ピクチャー指定文字。

比較演算子 **(comparison operator)**
関係内の項目を相互に比較するよう指示するための算術、ストリング・ロケーター、または論理関係で使用される演算子。 比較演算子は次のとおり。

= (に等しい)

> (より大)

< (より小)

>= (より大か等しい)

<= (より小か等しい)

¬= (等しくない)

¬> (より大ではない)

¬< (より小ではない)

コンパイル時 **(間) (compile time)**
一般に、ソース・プログラムがオブジェクト・モジュールに変換されている時間。PL/I では、変更したい場合に、ソース・プログラムを変更して、オブジェクト・プログラムに変換し終わるまでに経過する時間。

コンパイラー・オプション **(compiler options)**
コンパイルの特定の面を制御するために指定されるキーワード。例えば、生成するオ

ブジェクト・モジュールの特徴や、作成する印刷出力のタイプなどがある。

複素数データ **(complex data)**
おのおのの項目が実数部と虚数部で構成された算術データ。

合成演算子 **(composite operator)**
<=、**、および /* などの特殊文字を複数含む演算子。

複合ステートメント **(compound statement)**
他のステートメントが含まれているステートメント。 PL/I では、
IF、ON、OTHERWISE、および WHEN だけが、複合ステートメントである。
「ステートメント本体 *(statement body)*」を参照。

連結 **(concatenation)**
2 つのストリングを指定順に結合し、元の 2 つのストリングの合計長に等しい長さを持った 1 つのストリングを作成する操作。 これは、演算子 || で指定する。

条件 **(condition)**
エラー (オーバーフローなど) または予期される状況 (入力ファイルの終わりなど) のいずれかの例外的な状態。 条件が発生する (検出される) と、その条件に対する規定のアクションが処理される。 「確立された処置 *(established action)*」および「暗黙処置 *(implicit action)*」も参照。

条件名 **(condition name)**
PL/I 定義またはプログラマー定義の条件の名前。

条件接頭語 **(condition prefix)**
ステートメントの接頭部として付けられる、括弧で囲まれた 1 つ以上の条件名のリスト。 条件接頭語は、指定した条件を使用可能にするか使用不能にするかを指定する。

連結集合 **(connected aggregate)**
エレメントが、間にデータ項目の入らない連続したストレージを占有する配列または構造体。 「非連結集合 *(nonconnected aggregate)*」と対比。

連結参照 **(connected reference)**
連結ストレージに対する参照。 プログラムを実行するには、ストレージが連結されていることが明らかでなければならない。

連結ストレージ **(connected storage)**
単一名を使って参照することができる諸項目の非中断かつ線形の連なりの主記憶域。

定数 **(constant)**
名前が付いておらず、変更できない値を持った、算術またはストリング・データ項目。 VALUE 属性を指定して宣言された ID。 FILE 属性または ENTRY 属性を指定し、VARIABLE 属性を指定しないで宣言された ID。

定数参照 **(constant reference)**
対象として定数を持つ値参照。

含まれているブロック、宣言、またはソース・テキスト **(contained block, declaration, or source text)**
開始、プロシージャー、またはパッケージのブロック内のすべてのブロック、プロシージャー、ステートメント、宣言、またはソース・テキスト。 パッケージ、プロシージャー、および BEGIN ステートメントとそれに対応する END ステートメント全体は、ブロック内には含まれていない。

収容ブロック **(containing block)**
該当する宣言、ステートメント、プロシージャー、またはその他のソース・テキストを収容している、パッケージ、プロシージャー、または開始ブロック。

コンテキスト宣言 **(contextual declaration)**
DECLARE ステートメントで明示的に宣言されていないが、その使用の前後関係から、特定の属性が ID に関連付けられるような ID の存在。

制御文字 **(control character)**
特定コンテキスト内に存在することによって制御機能が指定される、文字セット内の文字。 1 つの例としてファイルの終わり (EOF) マーカーがある。

制御フォーマット項目 **(control format item)**
ストリーム内または印刷ページの内での、あるデータ項目の位置付けを指定するために、編集ディレクティブ伝送の中で使用される指定。

制御変数 **(control variable)**
DO ステートメントの反復実行を制御するのに使用する変数。

被制御パラメーター **(controlled parameter)**
DECLARE ステートメント内で CONTROLLED 属性を指定されるパラメーター。 これは、CONTROLLED 属性を持った引数としか関連付けることはできない。

被制御ストレージ割り振り **(controlled storage allocation)**
被制御変数用のストレージの割り振り。

被制御変数 **(controlled variable)**
現行世代にだけアクセスすることができ、ALLOCATE と FREE ステートメントによって割り振りと解放が制御される変数。

制御セクション **(control sections)**
オブジェクト・モジュール内のグループ化された機械命令。

型変換 **(conversion)**
ある 1 つの表現法から、一組の特定属性に合うよう別の表現法に値を変換すること。 例えば、文字ストリングを FIXED BINARY (15,0) などの算術値に変換すること。

配列のクロス・セクション **(cross section of an array)**
配列の少なくとも 1 つの次元のエクステントで表すことのできるエレメント。 配列参照内に添え字の代わりにアスタリスクがあれば、それはその次元のエクステント全体を表す。

現行世代 **(current generation)**
変数名を参照して、現在使用できる自動変数または被制御変数の世代。

D

データ **(data)**
処理に適合した形式の情報または値の表現。

データ集合 **(data aggregate)**
異なったデータ項目の集まりであるデータ項目。

データ属性 **(data attribute)**
FIXED BINARY などの、データ項目が表すデータのタイプを指定するキーワード。

データ・ディレクティブ伝送 **(data-directed transmission)**
データを伝送するための、ストリーム指向伝送のタイプ。代入ステートメントに似ていて、`name = constant` の形式をとる。

データ項目 **(data item)**
単一の名前付きデータ単位。

データ・リスト **(data list)**
ストリーム指向伝送における、GET および PUT ステートメント内で使用するデータ項目を括弧で囲んだリスト。 「フォーマット・リスト *(format list)*」と対比。

データ・セット **(data set)**
単一のファイル名の参照によってアクセスすることができる、プログラムの外部にあるデータの集まり。 参照されることが可能な装置。

データ指定 **(data specification)**
伝送モード (DATA、LIST、または EDIT) を指示し、さらにデータ・リストと、編集ディレクティブ・モードの場合はフォーマット・リストを含む、ストリーム指向伝送ステートメントの一部分。

データ・ストリーム **(data stream)**
ストリーム指向伝送でデータ・セットから、またはデータ・セットへ、文字形式のデータ・エレメントの連続ストリームとして転送されるデータ。

データ伝送 **(data transmission)**
データ・セットからプログラムへ、およびその逆に、データを転送すること。

データ・タイプ **(data type)**
一連のデータ属性。

DBCS
文字セットにおいて、それぞれの文字は 2 つの連続するバイトで表される。

非アクティブ **(deactivated)**
ある ID の値で、ソース・プログラム・テキスト内のプリプロセッサー ID を置き換えることができない状態。 「アクティブ *(active)*」と対比。

デバッグ **(debugging)**
プログラムからバグを除去する処理。

10 進 (decimal)
0 から 9 までの数字を使った数体系。

10 進ピクチャー文字 (decimal digit picture character)
ピクチャー指定文字 9 のこと。

10 進固定小数点定数 (decimal fixed-point constant)
1 つ以上の 10 進数 (および任意で小数点を付けたもの) から成る定数。

10 進固定小数点値 (decimal fixed-point value)
小数点の想定位置を持つ 10 進数の連なりで構成される有理数。 「2 進固定小数点値 *(binary fixed-point value)*」と対比。

10 進浮動小数点定数 (decimal floating-point constant)
10 進固定小数点定数から成る仮数と、3 桁を超えないオプションの符号付き整定数が後に付いた文字 E から成る指数とで構成される値。

10 進浮動小数点値 (decimal floating-point value)
10 進小数部と考えることができる仮数形式の実数、および 10 を底とする整数のべき乗と考えることができる指数の近似値。「2 進浮動小数点値 *(binary floating-point value)*」と対比。

10 進ピクチャー・データ (decimal picture data) 「数値ピクチャー・データ *(numeric picture data)*」を参照。

宣言 (declaration)
ID を名前として確立し、その ID 用に一連の属性を (部分的または全体的に) 指定すること。 特定名の属性のソース。

デフォルト (default)
指定がされていないときに、とられる値、属性、またはオプション。

定義された変数 (defined variable)
指定された基底付き変数用の一部または全部のストレージに関連付けられる変数。

区切る (delimit)
1 つ以上の項目またはステートメントの前後を、文字またはキーワードで囲むこと。

区切り文字 (delimiter)
すべてのコメントと、パーセント記号、括弧、コンマ、ピリオド、セミコロン、コロン、割り当て記号、ブランク、ポインター、アスタリスク、および単一引用符。これらは ID、定数、ピクチャー指定、iSUB、およびキーワードの限界を定めるものとなる。

記述子 (descriptor)
区域サイズ、配列境界、またはストリング長などの変数に関する情報を保持する制御ブロック。

数字 (digit)
0 から 9 までの文字の 1 つ。

次元属性 (dimension attribute)
配列の次元数を指定し、各次元の境界を示す属性。

使用不可の (disabled)
割り込みが発生せず、規定の処置も取られないような事態になった状態。

DO グループ (do-group)
DO ステートメントで区切られ、それに対応する END ステートメントで終了する、制御目的に使用される一連のステートメント。 「ブロック *(block)*」と対比。

DO ループ (do-loop)
「反復 *DO* グループ *(iterative do-group)*」を参照。

仮引数 (dummy argument)
参照によって渡すことのできない引数の値を保持するため自動的に作成される一時記憶域。

ダンプ (dump)
エラーの原因のトレースなどの、プログラムが使用するストレージの一部または全部、または他のプログラム情報の印刷出力。

E

EBCDIC
拡張 2 進化 10 進コード (Extended Binary-Coded Decimal Interchange Code)。 8 ビットのコード化文字からなるコード化文字セット。

編集ディレクティブ伝送 (edit-directed transmission)
データが連続した文字ストリームとして中

にあり、関連データ・リストに対して行いたい編集を指定するにはフォーマット・リストを必要とするようなタイプのストリーム指向伝送。

エレメント **(element)**
配列などのデータ項目の集まりとは対照的な、単一のデータ項目。スカラー項目。

要素式 **(element expression)**
評価されるとエレメント値を生じる式。

要素変数 **(element variable)**
エレメントを表す変数。スカラー変数。

エレメント名 **(elementary name)**
「基本エレメント *(base element)*」を参照。

使用可能 **(enabled)**
条件により割り込みが生じて、該当する規定 ON ユニットが呼び出される条件の状態。

end-of-step メッセージ **(end-of-step message)**
ジョブ制御ステートメントとジョブ・スケジューラー・メッセージのリストに続いており、各ステップの成功または失敗を示す戻りコードを含むメッセージ。

入り口定数 **(entry constant)**
PROCEDURE ステートメントのラベル接頭部 (入り口名)。 ENTRY 属性を指定し、VARIABLE 属性を指定しないで名前を宣言すること。

入り口データ **(entry data)**
プロシージャーへのエントリー・ポイントを表すデータ項目。

入り口式 **(entry expression)**
評価されると入り口名を生じるような式。

入り口名 **(entry name)**
ENTRY 属性を持つものとして明示的または内容に従って宣言された ID (ただし、VARIABLE 属性が与えられていない場合に限る)。または、ENTRY 属性を暗黙指定された入り口変数の値を持った ID。

エントリー・ポイント **(entry point)**
そこでプロシージャーを呼び出すことができるプロシージャー内の 1 地点。「*1 次エントリー・ポイント (primary entry point)*」および「*2 次エントリー・ポイント (secondary entry point)*」も参照。

入り口参照 **(entry reference)**
入り口値を返す入り口定数、入り口変数参照、または関数参照。

入り口変数 **(entry variable)**
入り口値を割り当てる対象となりうる変数。 これは、ENTRY 属性と VARIABLE 属性を両方とも持っている必要がある。

入り口値 **(entry value)**
入り口定数または入り口変数によって表されるエントリー・ポイント。入り口値には、その入り口定数に関連した活動化環境が含まれる。

環境 **(活動化の) (environment (of an activation))**
収容ブロック内で宣言されたデータに関して、呼び出されたブロックと関連し、そのブロック内で使用される情報。

環境 **(ラベル定数の) (environment (of a label constant))**
ステートメント・ラベル定数への参照が適用されるブロックの個々の活動化の識別情報。 この情報が決定されるのは、ステートメント・ラベル定数が、引数として渡されたり、またはステートメント・ラベル変数に割り当てられ、それが定数と一緒に渡されたり割り当てられたときである。

確立された処置 **(established action)**
条件が生じたときにとられる処置。 「暗黙の処置 *(implicit action)*」および「*ON ステートメント処置 (ON-statement action)*」も参照。

エピローグ **(epilogue)**
ブロックまたはタスクの終了時に自動的に生じる各種処理。

評価 **(evaluation)**
単一の値、値の配列、または値の構造化値へ式を換算すること。

イベント **(event)**
状況および完了を、関連したイベント変数から決定することのできるプログラムの活動。

イベント変数 **(event variable)**
: イベントと関連付けることができる EVENT 属性を持つ変数。 その値は、処置が完了したかどうか、および完了の状況を示す。

明示宣言 **(explicit declaration)**
: ラベル接頭部として DECLARE ステートメント内、またはパラメーター・リスト内に ID (名前) を出すこと。 「暗黙宣言 *(implicit declaration)*」と対比。

指数文字 **(exponent characters)**
: 以下のピクチャー指定文字のこと。

  1. K および E。指数フィールドの先頭を示すため、浮動小数点ピクチャー指定内で使用される文字。
  2. F。10 進小数点をその想定位置から右方向へ (正定数の場合) かまたは左方向へ (負定数の場合) 移動するときに、小数部の桁数を示す整定数を使って指定される位取り係数文字。

式 **(expression)**
: 値、値の配列、または一連の構造化値セットを表すのに、プログラム内で使われる表記。 単独で使用される定数または参照、あるいは、定数または参照あるいはその両方を演算子と組み合わせたもの。

拡張英字 **(extended alphabet)**
: A から Z までの大文字、小文字の英字、$、@、および #、または NAMES コンパイラー・オプションで指定されたもの。

エクステント **(extent)**
: 配列の次元の境界、ストリング長、または区域サイズによって示される範囲。 この区域がターゲット区域に割り当てられる場合は、ターゲット区域のサイズ。

外部名 **(external name)**
: 有効範囲が必ずしも 1 つのブロックとその収容ブロックだけに限定されない (EXTERNAL 属性を持つ) 名前。

外部プロシージャー **(external procedure)**
: 他のいずれのプロシージャーにも組み込まれないプロシージャー。 パッケージ内に入っていて同様にエクスポートされるレベル 2 のプロシージャー。

外部シンボル **(external symbol)**
: それ自身が定義されている制御セクションを除く制御セクション内で参照できる名前。

外部シンボル辞書 **(External Symbol Dictionary (ESD))**
: オブジェクト・モジュール内で使われるすべての外部シンボルの一覧表。

特別言語文字 **(extralingual character)**
: 英数字にも特殊文字にも分類されない文字 ($、@、および # など)。 このグループには、NAMES コンパイラー・オプションで指定された文字も含まれる。

F

属性分配 **(factoring)**
: 1 つ以上の属性を、DECLARE ステートメント内の括弧で囲まれた名前リストに対して適用して、複数の名前に共通する属性を反復する必要をなくすこと。

フィールド **(データ・ストリーム中の) (field (in the data stream))**
: 単一データまたはスペーシング・フォーマット項目によって、幅 (文字数) を定義されるデータ・ストリームの部分。

フィールド **(ピクチャー指定の) (field (of a picture specification))**
: 任意の文字ストリング・ピクチャー指定、または固定小数点数を記述した数字ピクチャー指定の部分 (または全部)。

ファイル **(file)**
: プログラムにおいて、単数または複数のデータ・セットを名前付きで表現したもの。ファイルは、オープンするごとに、単数または複数のデータ・セットに関連付けられる。

ファイル定数 **(file constant)**
: FILE 属性を指定し、VARIABLE 属性を指定しないで宣言された名前。

ファイル記述属性 **(file description attribute)**
: 各ファイル定数の個々の特性を記述したキーワード。 「代替属性 *(alternative attribute)*」と「追加属性 *(additive attribute)*」も参照。

ファイル式 **(file expression)**
評価されるとファイル・タイプを生じる式。

ファイル名 **(file name)**
ファイル用に宣言された名前。

ファイル変数 **(file variable)**
ファイル定数を割り当てることのできる変数。 この場合、ファイルは、FILE 属性と VARIABLE 属性を持っていなければならず、ファイル記述属性を持っていることはできない。

固定小数点定数 **(fixed-point constant)**
「算術定数 *(arithmetic constant)*」を参照。

修正 **(fix-up)**
コンパイル済みプログラムを実行可能にするために、コンパイル時にエラーを検出したあとでコンパイラーが実行する解決手段。

浮動小数点定数 **(floating-point constant)**
「算術定数 *(arithmetic constant)*」を参照。

制御の流れ **(flow of control)**
実行の連なり。

フォーマット **(format)**
ストリーム内のデータ項目の表現法を記述したり (データ・フォーマット項目)、またはストリーム内のデータ項目の個々の位置決めを記述する (制御フォーマット項目) ために編集指示データ伝送内で使用される仕様。

フォーマット定数 **(format constant)**
FORMAT ステートメントでのラベル接頭部。

フォーマット・データ **(format data)**
FORMAT 属性を指定された変数。

フォーマット・ラベル **(format label)**
FORMAT ステートメントでのラベル接頭部。

フォーマット・リスト **(format list)**
ストリーム指向伝送における外部メディアでのデータ項目のフォーマットを指定したリスト。 「データ・リスト *(data list)*」と対比。

完全修飾名 **(fully-qualified name)**
名前が参照するメンバーより上の階層順序内のすべての名前と、そのメンバー自身の名前が組み込まれている名前。

関数 **(**プロシージャー**) (function (procedure))**
PROCEDURE ステートメント内に RETURNS オプションのあるプロシージャー。 RETURNS 属性を指定して宣言された名前。 これは、関数参照内にその入り口名のうちの 1 つがあると呼び出され、スカラー値を参照点に返す。 「サブルーチン *(subroutine)*」と対比。

関数参照 **(function reference)**
入り口定数または入り口変数のことで、このどちらも関数を表さなければならないが、その後に空と考えられる引数リストが続く。 「サブルーチン呼び出し *(subroutine call)*」と対比。

G

世代 **(**変数の**) (generation (of a variable))**
静的変数の割り振り、被制御変数または自動変数の特定の割り振り、または基底付き変数の特定のロケーター修飾で、または定義された変数かパラメーターで指示されるストレージ。

総称記述子 **(generic descriptor)**
GENERIC 属性内で使用する記述子。

総称キー **(generic key)**
キー・クラスを識別する文字ストリング。 そのストリングで始まるキーはすべて、そのクラスのメンバーである。 例えば、「ABCD」、「ABCE」、および「ABDF」、という記録済みキーは、すべて総称キー「A」および「AB」で識別されるクラスのメンバーであり、最初の 2 つは、「ABC」というクラスのメンバーでもある。そして、これら 3 つの記録済みキーは、それぞれ「ABCD」、「ABCE」、「ABDF」というクラスの固有のメンバーであると見なすことができる。

総称名 **(generic name)**
入り口名ファミリーの名前。 総称名への参照は、呼び出し点にある引数リスト内の

引数の属性に一致するパラメーター記述子を持った入り口名によって置き換えられる。

グループ **(group)**
より大きいプログラム単位に入っているステートメントの集まり。グループは、DO グループまたは選択グループのどちらかであるが、 ON ユニットとしての場合を除き、単一ステートメントを使用できるところでは常に使用することができる。

H

16 進 **(hex)**
「*16* 進数字 *(hexadecimal digit)*」を参照。

16 進数 **(hexadecimal)**
16 の基数を持った数体系。有効数は 0 から 9 の数字と、A は 10 を、F は 15 を表す A から F までの文字。

16 進数字 **(hexadecimal digit)**
0 から 9 までと A から F までの数字のいずれか。A から F までは、それぞれ 10 進数値の 10 から 15 までを表す。

I

ID (identifier)
コメントや定数内に入ることがなく、前後に区切り文字を伴う文字のストリング。ID の先頭文字は、26 個の英字、または特別言語文字 (ある場合) でなければならない。 その他の文字がある場合には、拡張英字、数字、区切り文字を追加して入れることができる。

IEEE 米国電気電子学会 (Institute of Electrical and Electronics Engineers)。

暗黙的な **(implicit)**
明示指定のないまま取られる処置。

暗黙処置 **(implicit action)**
使用可能な条件が生じたときに、その条件用に現在確立されている ON ユニットがない場合にとられる処置。 「*ON* ステートメント処置 *(ON-statement action)*」と対比。

暗黙宣言 **(implicit declaration)**
DECLARE ステートメント内で明示的に宣言されていないか、または内容に従って宣言されていない名前。

暗黙オープン **(implicit opening)**
OPEN ステートメント以外の入力ステートメントまたは出力ステートメントが原因で、ファイルがオープンされること。

挿入演算子 **(infix operator)**
2 つのオペランド間にある演算子。

継承次元 **(inherited dimension)**
構造体、共用体、またはエレメントでの、収容構造から派生する次元。 名前が配列ではないエレメントであれば、その次元全体が継承次元で構成される。 名前が配列であるエレメントであれば、その次元は、継承次元および明示的に宣言された次元で構成される。 1 つ以上の継承次元を持つ構造体を、非結合集合と呼ぶ。 「結合集合 *(connected aggregate)*」と対比。

入出力 **(input/output)**
補助メディアと主記憶装置との間でデータを転送すること。

挿入点文字 **(insertion point character)**
関連データを文字ストリングへ割り当てるときに、指示位置に挿入されるピクチャー指定文字。入力のときに P フォーマット項目内で使用される挿入文字は、検査の目的で用いられる。

整数 **(integer)**
符号を付けるか付けないかは任意の、10 進または 2 進小数点のない一連の数字、または一連のビット。 通常は、FIXED BINARY (p,0) または FIXED DECIMAL (p,0) と記述される、符号を付けるか付けないかは任意の整数。

規定境界 **(integral boundary)**
そこでデータを位置合わせすることができる任意の 8 ビット単位のバイト・マルチアドレス。 通常はハーフワード、フルワード、またはダブルワード (2、4、または 8 バイトの長さの整倍数) 境界である。

介在配列 **(interleaved array)**
非連結ストレージを参照する配列。

介在添え字 **(interleaved subscripts)**
添え字付き修飾参照の最下位レベル以外のレベルに存在する添え字。

内部ブロック **(internal block)**
ブロックの中に組み込まれている別のブロック。

内部名 **(internal name)**
名前が宣言されたブロック内のみで認識されている名前、またそのブロック内に入っているブロックの中でも認識されている可能性もある名前。

内部プロシージャー **(internal procedure)**
ブロックの中に組み込まれている別のプロシージャー。 「外部プロシージャー *(external procedure)*」と対比。

割り込み **(interrupt)**
条件やアテンションの発生の結果として、プログラムの制御の流れを変更すること。

起動 **(invocation)**
プロシージャーの活動化。

起動する **(invoke)**
プロシージャーを活動化すること。

呼び出されたプロシージャー **(invoked procedure)**
活動化されているプロシージャー。

呼び出し側ブロック **(invoking block)**
プロシージャーを活動化するブロック。

反復因数 **(iteration factor)**
INITIAL 属性指定において、特定の値を使って初期化されることになっている配列の連続エレメント数を指定するための式。フォーマット・リストにおける、特定のフォーマット項目またはフォーマット項目のリストを連続して使用する回数を指定するための式。

反復 **DO** グループ **(iterative do-group)**
制御変数または WHILE や UNTIL オプション、またはこの両方を指定した DO ステートメントを持つ DO グループ。

K

キー **(key)**
直接アクセス・データ・セット内のレコードを識別するデータ。「ソース・キー *(source key)*」および「記録済みキー *(recorded key)*」を参照。

キーワード **(keyword)**
PL/I において定義されたコンテキスト内で使用されると特定の意味を持つ ID。

キーワード・ステートメント **(keyword statement)**
ステートメントの機能を示すキーワードで始まる単純ステートメント。

認識された **(名前に関する用語) (known (applied to a name))**
宣言された意味で認識されること。 名前は、その有効範囲内で認識される。

L

ラベル **(label)**
ステートメントの接頭部として付く名前。PROCEDURE ステートメント上の名前を、入り口定数と呼び、FORMAT ステートメント上の名前を、フォーマット定数と呼ぶ。その他の種類のステートメント上の名前を、ラベル定数と呼ぶ。 LABEL 属性を持つデータ項目。

ラベル定数 **(label constant)**
ステートメント (PROCEDURE、ENTRY、FORMAT、または PACKAGE を除く) のラベル接頭語として書き込まれる名前。実行時に、そのラベル接頭語が参照されれば、そのステートメントにプログラムの制御を渡すことができる。

ラベル・データ **(label data)**
ラベル定数または、ラベル変数の値。

ラベル接頭部 **(label prefix)**
ステートメントの接頭部として付けられたラベル。

ラベル変数 **(label variable)**
LABEL 属性を指定して宣言された変数。その値は、プログラム内でのラベル定数である。

先行ゼロ **(leading zeroes)**
算術値としては意味のないゼロ。 ある数値内で最初の非ゼロより左側にあるすべてのゼロ。

レベル番号 **(level number)**
DECLARE ステートメント中の名前の前

に付く番号で、構造体名の階層内のその相対位置を指定するもの。

レベル 1 変数 **(level-one variable)**
大構造体または共用体の名前。 構造体または共用体の中に含まれていない、添え字なし変数。

字句単位の **(lexically)**
単位を左から右への順序に扱うことに関連した用語。

ライブラリー **(library)**
メンバーと呼ばれるその他のデータ・セットを保管するのに使用できる MVS 区分データ・セット、または CMS MACLIB のこと。

リスト指示 **(list-directed)**
ストリーム内のデータがブランクやコンマで区切られた定数になり、フォーマット設定が自動的に行われるタイプのストリーム指向伝送。

ロケーター **(locator)**
変数のアドレスまたはその記述子を保持する制御ブロック。

ロケーター/記述子 **(locator/descriptor)**
その後に記述子の付いたロケーター。 ロケーターは、記述子のアドレスではなく、変数のアドレスを保持する。

ロケーター修飾 **(locator qualification)**
基底付き変数への参照において、その参照が参照している基底付き変数の世代を指定するために、基底付き変数の左側に矢印で接続されているロケーター変数または関数参照。 これは、暗黙参照であることもある。

ロケーター値 **(locator value)**
ストレージ・アドレスを識別する値か、またはストレージ・アドレスを識別するのに使用できる値。

ロケーター変数 **(locator variable)**
変数またはバッファーの主記憶域内の位置を識別する値を持った変数。 これは、POINTER 属性または OFFSET 属性を持つ。

ロック・レコード **(locked record)**
EXCLUSIVE DIRECT UPDATE ファイル内のレコードであって、1 つのタスクにだけしか使用することはできず、そのレコードを使用しているタスクによって解放されるまで、他のタスクからアクセスできないレコード。

論理レベル **(構造体または共用体メンバーの) (logical level (of a structure or union member))**
全レベル番号が直接順序になっているとき (あるレベル番号から次のレベル番号までの増分が 1 のとき) に、レベル番号で示される深さ。

論理演算子 **(logical operators)**
ビット・ストリング演算子 NOT と排他 OR (¬)、 AND (&)、および OR (|)。

ループ **(loop)**
繰り返し実行される一連の命令。

下限 **(lower bound)**
配列次元の下限。

M

主プロシージャー **(main procedure)**
OPTIONS (MAIN) 属性を持った PROCEDURE ステートメントのある外部プロシージャー。 このプロシージャーは、プログラム実行の最初のステップで自動的に呼び出される。

大構造 **(major structure)**
レベル番号 1 を指定して宣言された名前を持つ構造。

メンバー **(member)**
構造体または共用体の中の、構造体、共用体、あるいはエレメントの名前。 ライブラリー内のデータ・セット。

小構造 **(minor structure)**
別の構造体または共用体の中に組み込まれている構造体。小構造の名前は、1 よりも大きくかつ親構造体または親共用体よりも大きいレベル番号を指定して宣言される。

モード **(算術データの) (mode (of arithmetic data))** 算術データの属性。 これは、実数 または 複素数 のどちらかである。

多重宣言 **(multiple declaration)**
同一ブロックに対して内部であり、別の修

飾を持たない同一 ID の複数宣言。 同一 ID の複数外部宣言。

マルチプロセッシング **(multiprocessing)**
複数のプログラムを同時に実行するために、複数の処理装置を備えた計算機システムを使用すること。

マルチプログラミング **(multiprogramming)**
単一の処理装置を使って、複数のプログラムを並行して処理するのに、計算機システムを使用すること。

マルチタスキング **(multitasking)**
複数の PL/I プロシージャーをプログラムが同時に実行できるようにする機能。

N

名 **(name)**
変数や定数にユーザーが与える ID。コンテキスト中に現れる、キーワードではない ID。 場合によっては、ユーザー定義名とも呼ぶ。

ネスト **(nesting)**
次のものの発生。

- ブロック内にある別のブロック。
- グループ内にある別のグループ。
- THEN 文節または ELSE 文節内の IF ステートメント。
- 関数参照の引数としての関数参照。
- FORMAT ステートメントのフォーマット・リスト内のリモート・フォーマット項目。
- パラメーター記述子リスト内の別のパラメーター記述子リスト。
- 1 つ以上の属性が分配されている括弧で囲まれた名前リスト内の属性の指定。

非結合ストレージ **(nonconnected storage)**
非結合データ項目が占有するストレージ。例えば、継承次元を持つ介在配列や構造体は、非結合ストレージ内にある。

ヌル・ロケーター値 **(null locator value)**
内部記憶域内のどの位置も識別できない特殊ロケーター値。 これは、現在ロケーター変数がデータの世代を識別できないことを示すのに役立つ。

ヌル・ステートメント **(null statement)**
セミコロン記号 (;) のみの入ったステートメント。 これは、何も処置はとられないことを示す。

ヌル・ストリング **(null string)**
長さゼロの文字ストリング、漢字ストリング、またはビット・ストリング。

数字データ **(numeric-character data)**
「*10* 進ピクチャー・データ *(decimal picture data)*」を参照。

数値ピクチャー・データ **(numeric picture data)**
算術値と文字値を持ったピクチャー・データ。 このタイプのピクチャー・データは、'A' または 'X' という文字を含むことはできない。

O

オブジェクト **(object)**
単一名で参照されるデータの集まり。

オフセット変数 **(offset variable)**
OFFSET 属性を持ったロケーター変数のことであり、その値は、ストレージ内のある区域の先頭からの相対位置を識別する。

オン条件 **(ON-condition)**
PL/I プログラムにおける、プログラム割り込みの原因となりうるオカレンス。 予期しないエラーが検出されたり、予期できる出来事ではあるものの、予期しない時にそれが起きたときに発生する。

ON ステートメント処置 **(ON-statement action)**
ある条件が生じたときに処置を取れるよう、条件に対して明示的に設定された処置法。 プログラムの制御の流れ内で ON ステートメントが見つかると、とられる処置で、その条件に対する処置が設定される。この処置は、ON ユニットが設定されたままであるか、RESIGNAL ステートメントで再設定されて条件が生じたときにとられる。 「暗黙の処置 *(implicit action)*」と対比。

ON ユニット **(ON-unit)**
該当する条件が起きたときに、とられるよう指定された処置。

オープン (ファイルの) **(opening (of a file))**
ファイルをデータ・セットに関連付けること。

オペランド **(operand)**
ID、定数、または式。式には演算子が、時には他のオペランドとともに使用される。

演算式 **(operational expression)**
1 つ以上の演算子から成る式。

演算子 **(operator)**
実行する演算を指定する記号。

オプション **(option)**
ステートメントの実行や解釈に影響を及ぼすのに使われるステートメント中の指定。

P

パッケージ定数 **(package constant)**
PACKAGE ステートメントのラベル接頭語。

パック **10** 進 **(packed decimal)**
固定小数点 10 進データ項目の内部表現。

埋め込み **(padding)**
ストリングの長さを必要な長さまで拡張するために、ストリングの右側に連結される、1 つ以上の文字、漢字、またはビット。 構造体または共用体の中に挿入される、1 つ以上のバイトまたはビット。その構造、または共用体内の後続エレメントが正しい規定境界に位置合わせされるようにするためのもの。

パラメーター **(parameter)**
PROCEDURE ステートメントの後に続くパラメーター・リスト中の名前。そのプロシージャーが呼び出されれば、渡される引数を指定する。

パラメーター記述子 **(parameter descriptor)**
ENTRY 属性指定内でパラメーター用に指定される一連の属性。

パラメーター記述子リスト **(parameter descriptor list)**
ENTRY 属性指定内のすべてのパラメーター記述子のリスト。

パラメーター・リスト **(parameter list)**
コンマで区切られ、プロシージャー・ステートメント内のキーワード PROCEDURE の後に続くか、または ENTRY ステートメント内のキーワード ENTRY の後に続く、括弧で囲まれた 1 つ以上のパラメーターのリスト。 このリストは、呼び出し時に渡される引数リストと対応する。

部分修飾名 **(partially-qualified name)**
不完全な修飾名。 これには名前が参照する構造メンバーまたは共用体メンバーより上の階層順序内にある名前のうちの全部ではない 1 つ以上の名前、およびそれ自身のメンバー名が含まれる。

ピクチャー・データ **(picture data)**
文字形式で表された、数値データ、文字データ、またはそれらの混合。

ピクチャー指定 **(picture specification)**
PICTURE 属性を指定した宣言内で、または P フォーマット項目内でピクチャー文字を使用して宣言されたデータ項目。

ピクチャー指定文字 **(picture specification character)**
ピクチャー指定で使用できるすべての文字。

PL/I 文字セット **(PL/I character set)**
PL/I のプログラム・エレメントを表現するために定義されている文字セット。

PL/I プロンプター **(PL/I prompter)**
PL/I コマンドのコマンド・プロセッサー・プログラムで、オペランドを調べ、コンパイラーに必要なデータ・セットを割り振る。

呼び出し点 **(point of invocation)**
呼び込まれたプロシージャーへの参照が現れる呼び込み側ブロック内の地点。

ポインター **(pointer)**
ストレージ内の位置を識別するための変数のタイプ。

ポインター値 **(pointer value)**
ポインター型を識別する値。

ポインター変数 **(pointer variable)**
ポインター値が入った POINTER 属性を持ったロケーター変数。

精度 **(precision)**
固定小数点データ項目内にある桁数または

ビット数、または、浮動小数点データ項目での最小確保有効数字 (指数は除く) の数。

接頭部 **(prefix)**
ステートメントの先頭に付けられるラベル、または 1 つ以上の条件名の括弧で囲まれたリスト。

接頭演算子 **(prefix operator)**
オペランドの前に置かれ、そのオペランドにだけ適用される演算子。 接頭演算子には、プラス (+)、マイナス (-)、および not (¬) がある。

プリプロセッサー **(preprocessor)**
コンパイルを実行する前に、ソース・プログラムを調べるためのプログラム。

プリプロセッサー・ステートメント **(preprocessor statement)**
プリプロセッサーがとる処置を指定するために、ソース・プログラム内に入れる特殊ステートメント。 これは、プリプロセッサーによって検出されると実行される。

1 次エントリー・ポイント **(primary entry point)** PROCEDURE ステートメントのラベル・リスト内の任意の名前によって識別されるエントリー・ポイント。

優先度 **(priority)**
タスクに関連する値で、他のタスクに対するそのタスクの優先順位 (指名順位) を指定する。

問題データ **(problem data)**
コード化された算術データ、ビット・データ、文字データ、グラフィック・データ、およびピクチャー・データ。

問題状態プログラム **(problem-state program)**
オペレーティング・システムの問題プログラム状態内で稼働するプログラム。 これには、入出力指示やその他の特権命令は入らない。

プロシージャー **(procedure)**
PROCEDURE ステートメントと END ステートメントで区切られたステートメントの集まり。プロシージャーとはプログラムまたはプログラムの一部であり、名前の有効範囲を区切り、そのプロシージャーまたは入り口名の 1 つへの参照によって活動化される。 「外部プロシージャー *(external procedure)*」および「内部プロシージャー *(internal procedure)*」も参照。

プロシージャー参照 **(procedure reference)**
入り口定数または入り口変数。 この後に引数リストを続けることができる。 プロシージャー参照は、CALL ステートメントや CALL オプションに入れることも、または関数参照として使用することもできる。

プログラム **(program)**
1 つ以上の外部プロシージャーまたはパッケージのセット。 外部プロシージャーのうちの 1 つは、PROCEDURE ステートメント内に OPTIONS(MAIN) 指定を持っていなければならない。

プログラム制御データ **(program control data)**
PL/I プログラムの処理を制御するのに使用するための区域、ロケーター、ラベル、フォーマット、項目、およびファイルのデータ。

プロローグ **(prologue)**
ブロックの起動時に自動的に生じる処理。

疑似変数 **(pseudovariable)**
ターゲット変数を指定するのに使用できるすべての組み込み関数の名前。 これは通常、代入ステートメントの左側にある。

Q

修飾名 **(qualified name)**
構造メンバーまたは共用体メンバーの階層順序。ピリオドで結合されていて、構造体の中の名前を識別するのに使用される。どの名前にも添え字を付けることができる。

R

範囲 **(**デフォルト指定の**) (range (of a default specification))**
DEFAULT ステートメント内の属性を適用される ID またはパラメーター記述子のどちらか、またはこの両方のセット。

レコード **(record)**
レコード単位入力または出力の操作における、伝送の論理単位。 1 つ以上の関連データ項目の集まり。これらの項目には通

常、それぞれ異なったデータ属性があり、また通常は構造体か共用体の宣言で記述される。

記録済みキー **(recorded key)**
直接アクセス・データ・セット内でレコードを識別する文字ストリングのことであり、そこでは文字ストリングそのものもデータの一部として記録される。

レコード単位データ伝送 **(record-oriented data transmission)**
別々のレコードの形式でデータを伝送すること。 「ストリーム指向のデータ伝送 *(stream data transmission)*」と対比。

再帰的プロシージャー **(recursive procedure)**
そのプロシージャー自身からでも、または別のアクティブ・プロシージャーからでも呼び出すことのできるプロシージャー。

再入可能プロシージャー **(reentrant procedure)**
複数のタスク、スレッド、またはプロセスから同時に活動化でき、しかもこれらのタスク、スレッド、およびプロセス間で相互に干渉が生じないプロシージャー。

REFER 式 **(REFER expression)**
REFER というキーワードの前に付いた式。この式は、REFER オプションを含む基底付き変数が、ALLOCATE ステートメントまたは LOCATE ステートメントのいずれかによって割り振られるときの境界、長さ、またはサイズとして使用される。

REFER オブジェクト **(REFER object)**
REFER オプション中の変数。メンバーの現行境界、長さ、またはサイズを保持しているか、あるいは保持する予定のもの。 REFER オブジェクトは、同一構造体または共用体のメンバーでなければならない。これは、ロケーター修飾したり添え字を付けてはならず、また REFER オプションを持ったメンバーの前になければならない。

参照 **(reference)**
明示宣言を生じることになる 1 つのコンテキスト内以外の名前の出現。

相対仮想起点 **(relative virtual origin (RVO))**
配列の実際の原点から配列の仮想原点を引いたもの。

リモート・フォーマット項目 **(remote format item)**
R という文字の後に FORMAT ステートメントのラベル (括弧に囲まれている) のあるもの。 フォーマット指定ステートメントは、転送するデータのフォーマットを制御するために、編集指示データ伝送ステートメントにより使用する。

反復因数 **(repetition factor)**
以下のものを指定する、括弧に入れられた符号なし整定数。

1. 後続するストリング定数を繰り返す回数。
2. 後続するピクチャー文字を繰り返す回数。

反復指定 **(repetitive specification)**
1 つ以上のデータ項目伝送の被制御反復を指定するためのデータ・リストの 1 エレメントであり、通常は配列と同時に使用する。

制限付き式 **(restricted expression)**
コンパイル時にコンパイラーによって評価されて定数を生じる式。 このような式のオペランドは、定数、指定した定数、および制限付きの式になる。

戻り値 **(returned value)**
関数プロシージャーから返される値。

RETURNS 記述子 **(RETURNS descriptor)**
RETURNS 属性内と、PROCEDURE および ENTRY ステートメントの RETURNS オプション内で使用する記述子。

S

スカラー変数 **(scalar variable)**
構造、共用体、配列ではない変数。

スケール **(scale)**
1 つの数値表記体系であり、その算術値は固定小数点または浮動小数点で表現される。

位取り係数 **(scale factor)**
固定小数点数内の小数桁数の指定。

位取り係数 **(scaling factor)**
「位取り係数 *(scale factor)*」を参照。

有効範囲 **(条件接頭語の) (scope (of a condition prefix))**
全体にわたって特定の条件接頭語が適用される、プログラムの部分。

有効範囲 **(宣言の) (scope (of a declaration or name))**
全体にわたって特定名が認識されているプログラムの部分。

2 次エントリー・ポイント **(secondary entry point)**
入り口ステートメントのラベル・リスト内の任意の名前によって識別されるエントリー・ポイント。

選択グループ **(select-group)**
SELECT ステートメントと END ステートメントで区切られたステートメントの連なり。

選択文節 **(selection clause)**
選択グループの WHEN 文節または OTHERWISE 文節。

自己定義データ **(self-defining data)**
プログラム実行時に決定され、集合のメンバー内に保管される境界、長さ、およびサイズを持つデータ項目を含む集合。

分離文字 **(separator)**
「区切り文字 *(delimiter)*」を参照。

シフト **(shift)**
ストレージ内のデータを元の位置の左または右へ変更すること。

シフトイン **(shift-in)**
2 バイト・ストリングの終わりをコンパイラーに知らせるために使用される記号。

シフトアウト **(shift-out)**
2 バイト・ストリングの先頭でコンパイラーにシグナルを送るために使用される記号。

符号および通貨記号 **(sign and currency symbol characters)**
ピクチャー指定文字。 S、+、-、および $ (または < と > で囲まれたその他の通貨記号)。

単純パラメーター **(simple parameter)**
ストレージ・クラス属性が指定されていないパラメーター。単純パラメーターはどのストレージ・クラスの引数も表すことができるが、被制御引数の現行世代だけを表すことができる。

単純ステートメント **(simple statement)**
IF、ON、WHEN、および OTHERWISE 以外のステートメント。

ソース **(source)**
問題データに変換されるデータ項目。

ソース・キー **(source key)**
直接アクセス・データ・セット内で個々のレコードを識別するため、レコード単位伝送ステートメント内で参照されるキー。

ソース・プログラム **(source program)**
ソース・プログラム・プロセッサー、およびコンパイラーへの入力となるプログラム。

ソース変数 **(source variable)**
他の演算に使用されるが、その演算で変更されることのない変数。 「ターゲット変数 *(target variable)*」と対比。

予備ファイル **(spill file)**
一時作業ファイルとして使われる SYSUT1 という名前のデータ・セット。

標準デフォルト **(値) (standard default)**
属性またはオプションの指定がなく、適用できる DEFAULT ステートメントがない場合の、代替属性または代替オプション。

標準ファイル **(standard file)**
GET ステートメントや PUT ステートメントで FILE オプションまたは STRING オプションがない場合に、PL/I が想定するファイル。 SYSIN が標準入力ファイルであり、SYSPRINT が標準出力ファイルである。

標準システム処置 **(standard system action)**
使用可能な条件のための ON ユニットがないときにその条件が発生した場合にとられる、言語で指定された処置。

ステートメント **(statement)**
キーワード、区切り文字、ID、演算子、および定数から構成され、セミコロン (;) で終わる PL/I ステートメント。 任意で、条件接頭語リストとラベルのリストを付けることができる。 「キーワード・ステー

トメント *(keyword statement)*」と「ヌル・ステートメント *(null statement)*」も参照。

ステートメント本体 **(statement body)**
ステートメント本体は、単純ステートメントまたは複合ステートメントのどちらでもかまわない。

ステートメント・ラベル **(statement label)**
「ラベル定数 *(label constant)*」を参照。

静的ストレージ割り振り **(static storage allocation)**
静的変数用のストレージの割り振り。

静的変数 **(static variable)**
プログラム実行の開始前に割り振られ、その実行の継続時間中はその割り振りの変わらない変数。

ストリーム指向データ伝送 **(stream-oriented data transmission)**
文字形式になった個々のデータ値の連続ストリームであるものとしてデータを扱って、データを伝送すること。 「レコード単位データ伝送 *(record-oriented data transmission)*」と対比。

ストリング **(string)**
単一のデータ項目として処理される、連続した文字、グラフィックス、またはビットの列。

ストリング変数 **(string variable)**
BIT、CHARACTER、または GRAPHIC 属性を指定して宣言される変数。この変数の値は、ビット・ストリング、文字ストリング、または漢字ストリングのいずれでもかまわない。

構造体 **(structure)**
必ずしも同じ属性を持たなくても差し支えないデータ項目の集まり。「配列 *(array)*」と対比。

構造式 **(structure expression)**
評価されると構造体の値セットを生成する式。

配列の構造体 **(structure of arrays)**
次元属性を持つ構造体。

構造体メンバー **(structure member)**
「メンバー *(member)*」を参照。

構造化 **(structuring)**
メンバー数、配置順、属性、および論理レベルによって表現される構造階層。

サブルーチン **(subroutine)**
PROCEDURE ステートメント内に RETURNS オプションのないプロシージャー。 「関数 *(function)*」と対比。

サブルーチン呼び出し **(subroutine call)**
後に CALL ステートメント内にあるオプションの引数リストが付く、サブルーチンを表さなければならないエントリー参照。「関数参照 *(function reference)*」と対比。

添え字 **(subscript)**
配列の次元内の位置を指定するための要素式。 添え字がアスタリスクであれば、次元のすべてのエレメントを指定する。

添え字リスト **(subscript list)**
括弧に入れられた、1 つ以上の添え字のリスト。配列の各々の次元に対して 1 つの添え字が対応する。これらによって配列の単一エレメントまたはクロスセクションを一意的に識別する。

サブタスク **(subtask)**
特定のタスクによって生成されるタスク、または特定のタスクから最後に生成されたタスクへの直接ライン内の任意のタスク。

同期 **(synchronous)**
プログラムの順次実行での単一の制御の流れ。

T

ターゲット **(target)**
データ項目 (ソース) が変換される属性。

ターゲット参照 **(target reference)**
受取側変数 (または受取側変数の一部) を指定する参照。

ターゲット変数 **(target variable)**
値が割り当てられる変数。

タスク **(task)**
単一の制御の流れによる 1 つ以上のプロシージャーの実行。

タスク名 **(task name)**
タスク変数を参照するのに使用される ID。

タスク変数 **(task variable)**
TASK 属性を持ち、その値がタスクの相対優先順位を示す変数。

終了 **(**ブロックの**) (termination (of a block))**
ブロックの実行が終了して、RETURN ステートメントまたは END ステートメントによって、制御がその起動側ブロックに戻るか、または GO TO ステートメントによって起動側ブロックまたは他のアクティブ・ブロックに制御が渡ること。

終了 **(**タスクの**) (termination (of a task))**
タスクへの制御の流れを停止すること。

切り捨て **(truncation)**
ターゲット変数のストリング長や精度が限度を超えたときに、データ項目の片方の端から 1 つ以上の数字文字、グラフィックス、またはビットを除去すること。

タイプ **(type)**
データの世代、値、または項目に対して適用される一連のデータ属性とストレージ属性。

U

未定義 **(undefined)**
ユーザーが行ってはならないことを示す。未定義機能が使用されると、PL/I 製品の個々のインプリメンテーションによって違った結果が出る可能性がある。 このような場合、アプリケーション・プログラムはエラーとなる。

共用体 **(union)**
同一のストレージを占有し、相互にオーバーレイしたデータ・エレメントの集まり。メンバーは構造体、共用体、基本変数、または配列のいずれであっても構わない。それらは、同一の属性を持っていなくてもかまわない。

配列の共用体 **(union of arrays)**
DIMENSION 属性を持った共用体。

上限 **(upper bound)**
配列次元の上限。

V

値参照 **(value reference)**
データ項目の値を得るのに使用する参照。

変数 **(variable)**
データを参照するのに使用され、値を割り当てる対象となりうる名前付きのエンティティー。 その属性は一定のままであるが、場合に応じてそれぞれ異なる値を参照することができる。

変数参照 **(variable reference)**
変数全体またはその一部を指定する参照。

仮想起点 **(virtual origin (VO))**
すべてゼロの添え字を持った配列のエレメントを保持するための位置。 このようなエレメントが配列内になければ、仮想起点は本来それが保持されるべき場所になる。

Z

ゼロ抑止文字 **(zero-suppression characters)**
ピクチャー指定文字の Z と *。これは、対応する桁位置のゼロを抑止し、それぞれをブランクまたはアスタリスクで置き換えるのに使用する。

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]

アクセシビリティー
  Enterprise PL/I for z/OS xlvi
アクセス
  相対レコード・データ・セット 367
  ESDS 347
  REGIONAL(1) データ・セット 325
アクセス方式サービス
  領域データ・セット 327
  REGIONAL(1) データ・セット
    順次アクセス 324
    直接アクセス 324
アセンブラー・ルーチン
  FETCH 211, 217
アテンション処理
  アテンション割り込み、効果 49
  主要な説明 533
  デバッグ・ツール 534
  ATTENTION ON ユニット 534
アプリケーション・パフォーマンスの向上 373
一時作業ファイル
  SYSUT1 189
インクルード・プリプロセッサー
  構文 131
印刷
  PRINT ファイル
    行長 293
    ストリーム入出力 292
    フォーマット 303
印刷制御文字 56, 292
エラー
  エラー・コンパイルの重大度 18
エラー装置
  リダイレクト 270
オブジェクト
  モジュール
    作成と保管 65
    レコード・サイズ 189
オプション
  コメントの指定 113
  コンパイル用 118
  コンパイル用に指定する 190
  ストリングの指定 113
オプション (続き)
  領域データ・セットの作成用の 318
  PLIDUMP に保存されたオプション・ストリング 530
オプション、z/OS UNIX での
  使用
    DD 情報 241
    TITLE 241
  DD_DDNAME 環境変数
    APPEND 244
    DELAY 246
    DELIMIT 246
    LRECL 246
    LRMSKIP 246
    PROMPT 247
    PUTPAGE 247
    RECCOUNT 247
    RECSIZE 248
    SAMELINE 248
    SKIP0 249
    TYPE 249
  PL/I ENVIRONMENT 属性
    BUFSIZE 245
オプション・レコード、SYSADATA 568
オフセット
  タブ・カウントの 295
  テーブルの 121
オペレーティング・システム
  z/OS UNIX でのデータ定義 (DD) 情報 240

[カ行]

改行 (LF)
  定義 249
外部参照
  連結名 236
カウンター・レコード、SYSADATA 568
拡張 2 進化 10 進コード (EBCDIC) 251
カスタマイズ
  ユーザー出口
    グローバル制御ブロックの構造 542
    独自のコンパイラー出口の作成 545
    SYSUEXIT の変更 544
カタログ式プロシージャー
  コンパイル、バインド、および実行 172, 177
  コンパイルおよびバインド 170, 175
  コンパイルのみ 168
  コンパイルのみ - 64 ビット 174
  コンパイル用入力データ 172, 177
カタログ式プロシージャー (続き)
  説明 167
  複数呼び出し 179
  変更
    DD ステートメント 182
    EXEC ステートメント 181
  呼び出し 179
  リスト 179
  OS/390 の下での
    変更する 180
    呼び出す 179
    IBM 提供の 167
可変長レコード
  ソート・プログラム 413
  フォーマット 252
紙送り制御文字 56, 292
環境変数
  z/OS UNIX での設定 270
漢字ストリング定数のコンパイル 43
関数
  ILC で C 関数を使用する 417
キー
  代替索引
    固有 355
    非固有 356
  REGIONAL(1) データ・セット 321
    ダミー・レコード 322
  VSAM
    索引付きデータ・セット 335
    相対バイト・アドレス 335
    相対レコード 336
キー索引付き VSAM データ・セット 335
キー順データ・セット
  ステートメントとオプション 348
  ロード 350
  DIRECT ファイルを使用したアクセス 352
  SEQUENTIAL ファイルを使用したアクセス 352
記述子 557
記述子域、SQL 146
記述子リスト、引数の引き渡し 557
記述子ロケーター、引数の引き渡し 558
行
  長さ 293
  メッセージ内の数 42
記録済みキー
  領域データ・セット 321
句読点
  自動プロンプト
    指定変更 196

© Copyright IBM Corp. 1999, 2017

句読点 (続き)
自動プロンプト (続き)
使用 196
OS/390
継続文字 197
端末での ENDFILE の入力 198
GET DATA ステートメント 197
GET EDIT での自動埋め込み 198
GET LIST ステートメント 197
SKIP オプション 198
PRINT ファイルからの出力 194
組み込み
CICS ステートメント 163
SQL ステートメント 147
グラフィック・データ 281
グローバル制御ブロック
終了プロシージャーの作成 548
初期化プロシージャーの作成 545
メッセージ・フィルター操作プロシージャーの作成 546
グローバル制御ブロックの構造
終了プロシージャーの作成 548
初期化プロシージャーの作成 545
メッセージ・フィルター操作プロシージャーの作成 546
言語環境プログラム・ライブラリー xvi
言語間通信
リンケージの考慮事項 424
ATTACH ステートメントの使用 427
C との 415
構造化データ・タイプ 416
出力の共用 426
ストリング・パラメーター・タイプの一致 421
データ・タイプ 415
入力の共有 427
パラメーターの一致 418
ファイル・タイプ 417
C 関数を使用する 417
ENTRY を戻す関数 423
enum データ・タイプ 417
C 標準ストリームのリダイレクト 427
コーディング
パフォーマンスの向上 381
CICS ステートメント 163
SQL ステートメント 145
交換コード 251
更新
相対レコード・データ・セット 367
ESDS 347
REGIONAL(1) データ・セット 325
構文図の読み方 xvii
構文レコード、SYSADATA 579
固定長レコード 252
コメント
オプション内 113
混合文字ストリング定数のコンパイル 43
コンパイラー
一時作業ファイル (SYSUT1) 189
エラー条件の重大度 18
オプションの説明 3
概要 183
漢字ストリング定数 43
混合ストリング定数 43
削減、ストレージ要件の 66
呼び出し 183
リスト
印刷オプション 189
コンパイラーへの入力 118
集合長さテーブル 120
使用 117
使用されるスタック・ストレージ 98
ステートメント・オフセット・アドレス 121
ソース・プログラム 96
ソース・プログラムのインクルード 48
相互参照テーブル 120
属性テーブル 119
ファイル参照テーブル 125
プリプロセッサーへの入力 118
ブロック・レベル 119
見出し情報 117
メッセージ 125
戻りコード 126
DO レベル 119
SOURCE オプション・プログラム 119
SYSPRINT 189
DBCS ID 43
JCL ステートメントの使用 186
OS/390 バッチの下での 186
PROCESS ステートメント 113
% ステートメント 115
コンパイラー・オプション
省略語 7
説明 3
デフォルト 3
AGGREGATE 7
ARCH 8, 374
ASSERT 9
ATTRIBUTES 10
BACKREG 10
BIFPREC 10
BLANK 12
BLKOFF 12
CASERULES 13
CEESTART 14
CHECK 14
CMPAT 16
CODEPAGE 17
コンパイラー・オプション (続き)
COMPILE 18
COPYRIGHT 19
CSECT 19
CSECTCUT 20
CURRENCY 20
DBCS 21
DBRMLIB 21
DD 22
DDSQL 22
DEFAULT 25, 377
DEPRECATE 34
DEPRECATENEXT 36
DISPLAY 36
DLLINIT 37
EXIT 37
EXTRN 38
FILEREF 38
FLAG 38
FLOAT 39
FLOATINMATH 41
GOFF 41
GONUMBER 42, 374
GRAPHIC 43
IGNORE 44
INCAFTER 44
INCDIR 44
INCLUDE 45
INSOURCE 48
INTERRUPT 49
LANGLVL 50
LIMITS 51
LINECOUNT 52
LINEDIR 52
LIST 53
LISTVIEW 53
MACRO 55
MAP 55
MARGINI 56
MARGINS 56
MAXBRANCH 57
MAXGEN 58
MAXMEM 58
MAXMSG 59
MAXNEST 59
MAXSTMT 60
MAXTEMP 60
MDECK 60
MSGSUMMARY 61
NAME 62
NAMES 62
NATLANG 63
NEST 63
NOMARGINS 56
NOT 63
NUMBER 64

コンパイラー・オプション (続き)
OBJECT 65
OFFSET 65
OFFSETSIZE 65
ONSNAP 66
OPTIMIZE 66, 373
OPTIONS 67
OR 68
PP 68
PPCICS 70
PPINCLUDE 70
PPLIST 71
PPMACRO 71
PPSQL 72
PPTRACE 72
PRECTYPE 73
PREFIX 73, 376
PROCEED 74
QUOTE 75
REDUCE 75, 374
RENT 76
RESEXP 78
RESPECT 78
RTCHECK 78
RULES 79, 375
SEMANTIC 95
SERVICE 95
SOURCE 96
SPILL 96
STATIC 96
STDSYS 97
STMT 97
STORAGE 98
STRINGOFGRAPHIC 98
SYNTAX 98
SYSPARM 99
SYSTEM 99
TERMINAL 101
TEST 101
UNROLL 104
USAGE 105
WIDECHAR 106
WINDOW 107
WRITABLE 107
XINFO 108
XML 50, 111
XREF 112
コンパイラー・ユーザー出口の初期化プロシージャー 545
コンパイル
ユーザー出口
カスタマイズ 544
活動化 544
プロシージャー 541
例 549
IBMUEXIT 544
コンパイル (続き)
z/OS UNIX の下での 183
コンパイルおよびバインド用の入力データ 170, 175
コンパイル時オプション
z/OS UNIX の下での 184
コンパイル時オプションの指定
フラグの使用 185

[サ行]

再始動
要求 537
要求する
システム障害後の自動 537
据え置き再始動 538
取り消す 537
プログラム内の自動 538
変更する 538
RESTART パラメーター 538
最適コーディング
コーディング・スタイル 381
コンパイラー・オプション 373
作業データ・セット、ソート用の 402
索引付き ESDS (入力順データ・セット)
ロード 350
DIRECT ファイル 352
SEQUENTIAL ファイル 352
索引データ・セット
索引順次データ・セット 254
削減、ストレージ要件の 66
サマリー・レコード、SYSADATA 567
サンプル・プログラム、実行 434, 440, 445, 450
システム
障害 537
障害後の再始動 537
SYSTEM コンパイラー・オプション
パラメーター・リストのタイプ 99
SYSTEM(CICS) 99
SYSTEM(IMS) 99
SYSTEM(MVS) 99
SYSTEM(OS) 99
SYSTEM(TSO) 99
自動
埋め込み 198
プロンプト
指定変更 196
使用 196
RESTART
システム障害後 537
チェックポイント/再始動機能 535
プログラム内の 538
シフト・コード・コンパイル 43
集合
長さテーブル 120
終了
コンパイル 18
終了プロシージャー
構文
グローバル 542
特有の 548
コンパイラー・ユーザー出口 548
プロシージャー特有の制御ブロックの例 548
主記憶域、ソート用の 398
出力
ストリーム・ファイル用のデータ・セットの定義 282
ソート用データ 403
ソート用の骨組みコード 407
ソート・データ・セット 403
ソート・プログラム用のルーチン 403
プリプロセッサー出力の制限 60
リダイレクト 270
PLISRTA 用のデータ 408
SEQUENTIAL 310
SYSLIN 189
SYSPUNCH 189
順次アクセス
REGIONAL(1) データ・セット 324
順次データ・セット 254
使用
ホスト変数としての配列、 SQL プリプロセッサー 149
条件付きコンパイル 18
条件付きサブパラメーター 255
情報交換コード 251
初期ボリューム・ラベル 255
序数エレメント・レコード、SYSADATA 572
序数タイプ・レコード、SYSADATA 571
処理ルーチン
ソート用データ
可変長レコード 413
出力 (ソート出口 E35) 407
成功かどうかの判別 401
入力 (ソート出口 E15) 404
PLISRTB 410
PLISRTC 411
PLISRTD 412
シンボル・テーブル 101
シンボル・レコード、SYSADATA 573
据え置き再始動 538
ステートメント 115
オフセット・アドレス 121
ネスト・レベル 119
ステップの異常終了 255
ストリーム入出力
データ伝送のレコード・フォーマット 263

ストリーム入出力 (続き)
データ・セット
アクセス 290
作成 285
レコード・フォーマット 291
ファイル
定義 282
PRINT ファイル 292
SYSIN および SYSPRINT ファイル 297
連続データ・セット 281
DD ステートメント 287, 291
ENVIRONMENT オプション 282
ストリーム・ファイルおよびレコード・ファイル 300, 304
ストリング
漢字ストリング定数のコンパイル 43
ストリング記述子
ストリング記述子 559, 561
ストリングの割り当て 382
ストレージ
印刷ファイルのブロック化 293
ソート・プログラム 398
主記憶域 398
補助記憶域 398
標準データ・セット 187
要件の削減 66
ライブラリー・データ・セット 275
リスト内のレポート 98
スラッシュ (/) 242
制御
域 333
インターバル 333
CONTROL オプション
EXEC ステートメント 191
制御ブロック
機能専用 542
グローバル制御 545
制御文字
印刷 56, 292
紙送り機構 56, 292
ゼロ値 263
宣言
ホスト変数、SQL プリプロセッサー 149
OS/390 におけるファイルの 233, 235
ソース
キー
REGIONAL(1) データ・セット内の 321
プログラム
外部テキストのシフト 56
コンパイラー・リスト 119
コンパイラー・リストに組み込まれた 48
データ・セット 188
ソース (続き)
プログラム (続き)
プリプロセッサー 55
リスト 96
ID 10
リスト
位置 56
ソース・ステートメント・ライブラリー 190
ソース・レコード、SYSADATA 577
ソート
最大レコード長 398
準備 392
ストレージ
主記憶域 398
補助 398
説明 391
ソート・タイプの選択 393
ソート・フィールド 395
ソート・プログラムの呼び出し 399
データ 391
データの入出力 403
入出力ルーチンの作成 403
評価結果 401
CHKPT オプション 396
CKPT オプション 396
DYNALLOC オプション 396
E15 入力処理ルーチン 404
EQUALS オプション 396
FILSZ オプション 396
PLISRT 392
PLISRTA(x) コマンド 408, 413
RECORD ステートメント 404
RETURN ステートメント 404
SKIPREC オプション 396
SORTCKPT 403
SORTCNTL 403
SORTIN 403
SORTLIB 402
SORTOUT 403
SORTWK 398, 402
ソート用のフィールド 395
ソート用のフローチャート 404
相互参照テーブル
コンパイラー・リスト 120
XREF オプションの使用 119
相対バイト・アドレス (RBA) 335
相対レコード 336
相対レコード・データ・セット
ステートメントとオプション 363
ロード 364
DIRECT ファイルを使ったアクセス 367
SEQUENTIAL ファイルを使ったアクセス 367
属性テーブル 119

[タ行]

大量順次挿入 354
対話式プログラム
アテンション割り込み 49
タブ制御テーブル 295
ダミー・レコード
REGIONAL(1) データ・セット 322
VSAM 337
ダンプ
先頭の識別 522
定義、データ・セット
論理レコード長 522
DD ステートメント 522
PLIDUMP 組み込みサブルーチン 521
PLIDUMP の呼び出し 521
SNAP 522
z/OS 言語環境プログラム ダンプの生成 521
ダンプ・トレースバック・テーブル内のプログラム単位名 523
端末
出力 303
ストリーム・ファイルおよびレコード・ファイル 304
PRINT ファイルのフォーマット 303
PUT EDIT コマンドからの出力 305
入力 298
大文字と小文字 301
ストリーム・ファイルおよびレコード・ファイル 300
データのフォーマット 299
ファイル終わり 301
GET ステートメントの COPY オプション 303
QSAM ファイルの定義 301
チェックポイント/再始動
据え置き再始動 538
PLICANC ステートメント 538
チェックポイント/再始動機能
活動の変更 538
再始動の要求 537
説明 535
チェックポイント・データ・セット 536
チェックポイント・レコードの要求 535
戻りコード 536
CALL PLIREST ステートメント 538
PLICKPT 組み込みサブルーチン 535
RESTART パラメーター 538
チェックポイント・データ
ソート用 403

チェックポイント・データの定義、
PLICKPT 組み込みサブオプション 536
直接データ・セット 254
データ
ソート
説明 391
ソート・プログラム 403
PLISRT(x) コマンド 408
タイプ
Java と PL/I の同等な 451
SQL と PL/I の同等な 152
ファイル
z/OS UNIX での作成 243
z/OS UNIX での変換 240
データ指示入出力 381
パフォーマンスのためのコーディング 381
データ・セット
一時的な 189
区分 273
クローズ 259
索引
順次 254
順次 254
使用 233
条件付きサブパラメーターの特性 256
情報交換コード 251
ストリーム・ファイル 281
ソース・ステートメント・ライブラリー 190
ソート 402
SORTWK 398
ソート・プログラム
出力データ・セット 403
ソート作業データ・セット 402
チェックポイント・データ・セット 403
入力データ・セット 403
相対レコードの定義 365
タイプ
比較 269
PL/I レコード入出力によって使用される 269
ダンプ用の定義
論理レコード長 522
DD ステートメント 522
チェックポイント 536
直接 254
データ・セット制御ブロック (DSCB) 255
データ・セットとファイルとの関連付け 235
特性の設定 250
ファイルとの関連付け解除 259
システム決定ブロック・サイズ 258
ファイルの割り振り 233
データ・セット (続き)
複数のデータ・セットと 1 つのファイルの関連付け 237
ブロックおよびレコード 251
編成
条件付きサブパラメーター 255
タイプ 254
データ定義 (DD) ステートメント 255
ライブラリー
使用 274
情報の取り出し 279
タイプ 273
SPACE パラメーター 275
ラベル 255, 273
ラベルなし 255
ラベルの変更 257
領域の 317
レコード 251
レコード・フォーマット
可変長 252
固定長 252
不定長 253
レコード・フォーマットのデフォルト値 262
連続ストリーム指向データ 281
DD 名 187
OS/390 でのデータ・セットの定義 235
PL/I ファイルとの関連付け解除 238
PL/I ファイルの関連付け
ファイルのオープン 257
ファイルのクローズ 259
ENVIRONMENT 属性での特性の指定 259
REGIONAL(1) 321
アクセスと更新 324
作成 322
SPACE パラメーター 187
VSAM
キー 335
索引付きデータ・セット 348
大量順次挿入 354
ダミー・データ・セット 337
データ・セット・タイプ 336
定義 344
パフォーマンス・オプション 343
ファイルの定義 339
プログラムの実行 331
ブロック化 333
編成 332
ENVIRONMENT オプションの指定 339
VSAM オプション 343
データ・セット、OS/390 での
連結 238
データ・セット、OS/390 での (続き)
1 つのデータ・セットと複数のファイルの関連付け 237
HFS 239
データ・セット、z/OS UNIX での
出力での拡張 244
出力の再作成 244
デフォルトの識別 240
特性の設定
DD_DDNAME 環境変数 244
パスの設定 243
領域数 247
領域の最大値 247
DD_DDNAME 環境変数 240
PL/I ファイルとデータ・セットの関連付け
環境変数の使用 240
関連付けされていないファイルの使用 243
OPEN ステートメントの TITLE オプションの使用 241
PL/I によるデータ・セットの検索方法 243
データ・セットの定義
特性の指定 259
ファイルのオープン 257
システム決定ブロック・サイズ 258
ファイルのクローズ 259
複数のデータ・セットと 1 つのファイルの関連付け 237
複数のデータ・セットの連結 238
複数のファイルと 1 つのデータ・セットの関連付け 237
ENVIRONMENT 属性 259
ESDS 346
定義、ファイル
特性の指定 259
ファイルのオープン 257
システム決定ブロック・サイズ 258
ファイルのクローズ 259
複数のデータ・セットの連結 238
複数のファイルと 1 つのデータ・セットの関連付け 237
領域データ・セット 319
キー 321
ENV オプション 320
ENVIRONMENT 属性 259
VSAM データ・セット 339
定義、OS/390 におけるファイル
複数のデータ・セットと 1 つのファイルの関連付け 237
出口 (E15) 入力処理ルーチン 404
出口 (E35) 出力処理ルーチン 407
トークン・レコード、SYSADATA 578
通し番号ボリューム・ラベル 255

トレースバック・テーブル
内のプログラム単位名 523
トレーラー・ラベル 255

[ナ行]

長さ、レコードの
z/OS UNIX での指定 248
名前付き定数
対静的変数 386
定義 386
日時組み込み関数
パターン 385
入出力
カタログ式プロシージャーでの 168, 174
コンパイラー
データ・セット 188
コンパイルおよびバインド用のデータ 170, 175
ソート用の骨組みコード 405
ソート・データ・セット 403
OS/390、長い行の句読法 197
入力
ストリーム・ファイル用のデータ・セットの定義 282
ソート用データ 403
ソート用の骨組みコード 407
ソート・データ・セット 403
ソート・プログラム用のルーチン 403
リダイレクト 270
PLISRTA 用のデータ 408
SEQUENTIAL 310
入力順データ・セット
定義 346
VSAM 333
ステートメントとオプション 344
ESDS のロード 345
SEQUENTIAL ファイル 345

[ハ行]

配列記述子
配列記述子 560, 562
バッチ・コンパイル
OS/390 183, 186
パフォーマンス
VSAM オプション 343
パフォーマンスの向上
コンパイラー・オプションの選択
ARCH 374
DEFAULT 377
GONUMBER 374
OPTIMIZE 373
PREFIX 376
パフォーマンスの向上 (続き)
コンパイラー・オプションの選択 (続き)
REDUCE 374
RULES 375
パフォーマンスのためのコーディング
ストリングの割り当て 382
名前付き定数対静的変数 386
入力専用パラメーター 381
ライブラリー・ルーチンの呼び出しの回避 387
ライブラリー・ルーチンの呼び出しのプリロード 389
ループ制御変数 383
DATA ディレクティブ入出力 381
DEFINED 対 UNION 385
GOTO ステートメント 382
PACKAGE 対ネストされた PROCEDURE 383
REDUCIBLE 関数 384
パラメーター引き渡し
引数の引き渡し 557
CMPAT(LE) 記述子 561
CMPAT(V*) 記述子 559
引数の引き渡し 557
記述子リストによる 557
記述子ロケーターによる 558
標識変数、SQL 156
標準データ・セット 187
標準ファイル (SYSPRINT と SYSIN) 270
ファイル
クローズ 259
データ・セットとファイルとの関連付け 235
特性の設定 250
ファイルの割り振り 233
OS/390 でのデータ・セットの定義 235
ファイル・レコード、SYSADATA 569
フォーマット表記規則 xvii
複数
呼び出し
カタログ式プロシージャー 179
復帰 - 改行 (CR - LF) 249
フック
位置サブオプション 101
不定長レコード 253
負の値
ブロック・サイズ 264
レコード長 263
フラグ、コンパイル時オプションの指定 185
プリプロセス
ソース・プログラム 55
入力 118
80 バイトまでに出力を制限 60
プリプロセス (続き)
MACRO を使った 55
%INCLUDE ステートメント 115
プリプロセッサー
組み込み 131
マクロ・プリプロセッサー 132
CICS オプション 163
PL/I とともに提供される 131
SQL オプション 141
SQL プリプロセッサー 137
プリプロセッサー・オプション
CCSID0 142
CODEPAGE 142
DEPRECATE 143
EMPTYDBRM 143
HOSTCOPY 143
INCONLY 144
WARNDECP 144
プロシージャー
カタログ式、OS/390 の下での用法 167
コンパイル、バインド、および実行 (IBMZCBG) 172
コンパイル、バインド、および実行 - 64 ビット (IBMQCBG) 177
コンパイルおよびバインド (IBMZCB) 170
コンパイルおよびバインド - 64 ビット (IBMQCB) 175
コンパイルのみ (IBMZC) 168
コンパイルのみ - 64 ビット (IBMQC) 174
ブロック
およびレコード 251
サイズ
オブジェクト・モジュール 189
最大 264
指定 251
領域データ・セット 329
レコード長 265
PRINT ファイル 293
ブロック間ギャップ (IBG) 251
プロンプト
自動、指定変更 196
自動、使用 196
ヘッダー・ラベル 255
補助記憶域、ソート用の 398
ホスト
構造体 156
変数、SQL ステートメント内での使用 149
ホスト変数の使用、SQL プリプロセッサー 149
ボリューム通し番号
直接アクセス・ボリューム 255
領域データ・セット 327

［マ行］

マクロ・プリプロセッサー
  マクロ定義 132
未参照 ID 10
メッセージ
  印刷フォーマット 297
  コンパイラー・ユーザー出口での変更 544
  コンパイラー・リスト 125
  フィルター機能 546
  ランタイム・メッセージ行番号 42
メッセージのフィルター操作 544
メッセージ・レコード、SYSADATA 570
文字
  印刷制御 56, 292
  紙送り制御 56, 292
文字ストリング属性テーブル 119
モジュール
  オブジェクト・モジュールの作成と保管 65
戻りコード
  コンパイラー・リスト 126
  チェックポイント/再始動ルーチン 536
  PLIRETC 402

［ヤ行］

ユーザー出口
  カスタマイズ
    グローバル制御ブロックの構造 542
    独自のコンパイラー出口の作成 545
    SYSUEXIT の変更 544
  機能 542
  コンパイラー 541
  ソート 394
容量レコード
  REGIONAL(1) 322
呼び出し
  カタログ式プロシージャー 179
    複数呼び出し 179
    マルチタスキング・プログラムのリンク・エディット 181
予備ファイル 189

［ラ行］

ライブラリー
  概要 254
  構造体 279
  コンパイルされたオブジェクト・モジュール 277
  作成に必要な情報 274
  システム・プロシージャー (SYS1.PROCLIB) 273
ライブラリー (続き)
  使用 273
  ソース・ステートメント 190
  ソース・ステートメント・ライブラリー 183
  タイプ 273
  データ・セット・ライブラリーの作成 274
  ディレクトリー 275
  用法 274
  ライブラリー・ディレクトリーからの情報の取り出し 279
  ライブラリー・メンバーの作成と更新 275
  ロード・モジュールの配置 277
  SPACE パラメーター 275
ライブラリー・ルーチンのプリロード 389
ライブラリー・ルーチンの呼び出しの回避 387
ラベル
  データ・セットの 255
ランタイム
  メッセージ行番号 42
  OS/390 の考慮事項
    自動プロンプト 196
    長い入力行の句読法 197
    フォーマット設定規則 194
    GET EDIT ステートメント 198
    GET LIST ステートメントと GET DATA ステートメント 197
    SKIP オプション 198
  PLIXOPT の使用 194, 216
リスト
  カタログ式プロシージャー 179
  コンパイラー
    オプション 118
    集合長さテーブル 120
    ステートメントのネスト・レベル 119
    ステートメント・オフセット・アドレス 121
    ストレージ・オフセット・リスト 123
    ファイル参照テーブル 125
    プリプロセッサー入力 118
    見出し情報 117
    メッセージ 125
    戻りコード 126
    ATTRIBUTE と相互参照テーブル 119
    DD 名リスト 3
    SOURCE オプション・プログラム 119
  ステートメント・オフセット・アドレス 121
  ストレージ・オフセット・リスト 123
リスト (続き)
  ソース・プログラム 96
  OS/390 バッチ・コンパイル 183, 189
  SYSPRINT 189
リテラル・レコード、SYSADATA 569
領域
  サイズ、EXEC ステートメント 186
  REGION パラメーター 181
領域、z/OS UNIX での 247
領域データ・セット
  オペレーティング・システム要件 327
  ファイルの定義
    キーの用法 321
    領域データ・セット 319
    ENVIRONMENT オプションの指定 320
  DD ステートメント
    アクセス 329
    作成 328
  REGIONAL(1) データ・セット
    アクセスと更新 324
    作成 322
    使用 321
リンク・エディット
  説明 193, 215
ループ
  制御変数 383
例
  PLIDUMP の呼び出し 521
レコード
  コンパイラー入力の最大サイズ 188
  ソート・プログラム 397
  チェックポイント 535
    データ・セット 536
  ブロック化解除 251
  z/OS UNIX での長さ 247
レコード長
  値 263
  指定 251
  領域データ・セット 317
レコード入出力
  データ伝送 305
  データ・セット
    アクセス 312
    作成 310
    タイプ 269
    連続データ・セット 313
  フォーマット 262
  レコード・フォーマット 306
  ENVIRONMENT オプション 307
レコードのブロック化解除 251
レコード・フォーマット
  オプション 283
  可変長レコード 252
  固定長レコード 252
  指定する 306

レコード・フォーマット (続き)
ストリーム入出力 291
タイプ 251
不定長レコード 253
連結
外部参照 236
データ・セット 238
連続データ・セット
ストリーム指向データ伝送 281
データ・セットの作成 285
データ・セットへのアクセス 290
ファイルの定義 282
ENVIRONMENT オプションの指定 282
PRINT ファイルの用法 292
SYSIN ファイルおよび SYSPRINT ファイルの使用方法 297
端末からの入力 298
端末からの入力の制御
大文字と小文字 301
条件のフォーマット 298
ストリーム・ファイルおよびレコード・ファイル 300
データのフォーマット 299
ファイルの終わり 301
GET ステートメントの COPY オプション 303
QSAM ファイルの定義 301
端末への出力 303
端末への出力の制御
条件 303
ストリーム・ファイルおよびレコード・ファイル 304
PRINT ファイルのフォーマット 303
PUT EDIT コマンドの出力 305
定義と用法 281
レコード単位データ伝送
使用できるステートメントとオプション 305
データ・セットのアクセスと更新 312
データ・セットの作成 310
ファイルの定義 306
ENVIRONMENT オプションの指定 307
レコード単位入出力 305
連絡域、SQL 145
論理否定 63
論理和 68

[ワ行]

割り込み
主要な説明 533
割り込み (続き)
対話式システムでのアテンション割り込み 49
デバッグ・ツール 534
ATTENTION ON ユニット 534

A

ACCT EXEC ステートメント・パラメーター 181
AGGREGATE コンパイラー・オプション 7
ALIGNED コンパイラー・サブオプション 26
ALL オプション
フック位置サブオプション 101
ALLOCATE ステートメント 120
AMP パラメーター 331
ANS
コンパイラー・サブオプション 26
APPEND オプション、z/OS UNIX での 244
ARCH コンパイラー・オプション 8, 374
ASCII
コンパイラー・サブオプション
説明 26
ASSERT コンパイラー・オプション 9
ASSIGNABLE コンパイラー・サブオプション 27
ATTENTION ON ユニット 534
ATTRIBUTES オプション 10

B

BACKREG コンパイラー・オプション 10
BIFPREC コンパイラー・オプション 10
BIN1ARG コンパイラー・サブオプション 27
BKWD オプション 259, 340
BLANK コンパイラー・オプション 12
BLKOFF コンパイラー・オプション 12
BLKSIZE
サブパラメーター 256
ENVIRONMENT 259
レコード入出力用の 264
DCB サブパラメーターとの比較 261
ENVIRONMENT オプションの
ストリーム入出力用の 282
BRACKETS 12
BRACKETS コンパイラー・オプション 12
BUFFERS オプション
ストリーム入出力用の 282
BUFND オプション 341
BUFNI オプション 341
BUFSIZE オプション、z/OS UNIX での 245
BUFSP オプション 341
BYADDR
説明 377
パフォーマンスへの影響 378
DEFAULT オプションでの用法 28
BYVALUE
説明 377
パフォーマンスへの影響 378
DEFAULT オプションでの用法 28

C

C ルーチン
FETCH 211
CASERULES コンパイラー・オプション 13
CEESTART コンパイラー・オプション 14
CHECK コンパイラー・オプション 14
CHKPT ソート・オプション 396
CICS
サポート 162
プリプロセッサー・オプション 163
CKPT ソート・オプション 396
CMPAT コンパイラー・オプション 16
COBOL
マップ構造 120
CODE サブパラメーター 256
CODEPAGE コンパイラー・オプション 17
COMPILE コンパイラー・オプション 18
COND EXEC ステートメント・パラメーター 181
CONNECTED コンパイラー・サブオプション
説明 28
パフォーマンスへの影響 378
CONSECUTIVE
ENVIRONMENT オプションの 283, 307
COPY オプション 303
COPYRIGHT オプション 530
COPYRIGHT コンパイラー・オプション 19
CSECT コンパイラー・オプション 19
CSECTCUT コンパイラー・オプション 20
CTLASA オプションと CTL360 オプション
ENVIRONMENT オプション
連続データ・セット用の 308
SCALARVARYING 268

CURRENCY コンパイラー・オプション 20
CYLOFL サブパラメーター
  DCB パラメーター 256

D

DBCS ID コンパイル 43
DBCS コンパイラー・オプション 21
DBRMLIB コンパイラー・オプション 21
DCB サブパラメーター 261
  主な説明 256
  カタログ式プロシージャー内の一時変更 182
  同等の ENVIRONMENT オプション 262
  領域データ・セット 329
DD コンパイラー・オプション 22
DD 情報、z/OS UNIX での
  TITLE ステートメント 241
DD ステートメント 255
  カタログ式プロシージャーの変更 180, 182
  カタログ式プロシージャーへの追加 182
  チェックポイント/再始動 536
  標準データ・セット 187
    出力 (SYSLIN、SYSPUNCH) 189
    入力 (SYSIN) 188
  ライブラリーの作成 274
  領域データ・セット 328
  OS/390 バッチ・コンパイル 187
  %INCLUDE 115
DD (データ定義) 情報、z/OS UNIX での 240
DD 名
  標準データ・セット 187
  %INCLUDE 115
DDSQL コンパイラー・オプション 22
DD_DDNAME 環境変数
  APPEND 244
  DELAY 246
  DELIMIT 246
  LRECL 246
  LRMSKIP 246
  PROMPT 247
  PUTPAGE 247
  RECCOUNT 247
  RECSIZE 248
  SAMELINE 248
  SKIP0 249
  TYPE 249
  z/OS UNIX での代替 dd 名 242
  z/OS UNIX での特性の指定 243
DECOMP 24
DECOMP コンパイラー・オプション 24
DEFAULT コンパイラー・オプション
  サブオプション
    ALIGNED 26
    ASCII または EBCDIC 26
    ASSIGNABLE または NONASSIGNABLE 27
    BIN1ARG または NOBIN1ARG 27
    BYADDR または BYVALUE 28
    CONNECTED または NONCONNECTED 28
    DESCLIST または DESCLOCATOR 28
    DESCRIPTOR または NODESCRIPTOR 28
    DUMMY 29
    E 29
    EVENDEC または NOEVENDEC 29
    HEXADEC 30
    IBM または ANS 26
    INITFILL または NOINITFILL 30
    INLINE または NOINLINE 30
    LINKAGE 31
    LOWERINC | UPPERINC 31
    NATIVE または NONNATIVE 31
    NATIVEADDR または NONNATIVEADDR 32
    NULLSTRADDR または NONULLSTRADDR 32
    NULLSTRPTR 33
    NULLSYS または NULL370 32
    ORDER または REORDER 33
    ORDINAL(MIN | MAX) 33
    OVERLAP | NOOVERLAP 33
    PSEUDODUMMY または NOPSEUDODUMMY 33
    RECURSIVE または NONRECURSIVE 33
    RETCODE 34
    RETURNS 34
    SHORT 34
  説明および構文 25
DEFINED
  対 UNION 385
DELAY オプション、z/OS UNIX での
  説明 246
DELIMIT オプション、z/OS UNIX での
  説明 246
DEPRECATE コンパイラー・オプション 34
DEPRECATENEXT コンパイラー・オプション 36
DESCLIST コンパイラー・サブオプション 28
DESCLOCATOR コンパイラー・サブオプション 28
DESCRIPTOR コンパイラー・オプション
  パフォーマンスへの影響 379
DESCRIPTOR コンパイラー・サブオプション
  説明 28
DFSORT 391
DIRECT ファイル
  RRDS
    データ・セットへのアクセス 367
  VSAM での索引付き ESDS
    データ・セットの更新 354
    データ・セットへのアクセス 352
DISP パラメーター
  データ・セットを削除する 273
  連続データ・セット 313
DISPLAY コンパイラー・オプション 36
DLL
  リンクの考慮事項およびサイド・デック 193, 215
  DYNAM=DLL リンカー・オプション 202
  RENT コンパイラー・オプションおよびフェッチ 201
DLLINIT コンパイラー・オプション 37
DSA
  各ブロックごとの PLIDUMP 組み込みサブルーチンに保存された 523
DSCB (データ・セット制御ブロック) 255, 275
DSNAME パラメーター
  連続データ・セット用の 313
DSORG サブパラメーター 256
DUMMY コンパイラー・サブオプション 29
DYNALLOC ソート・オプション 396

E

E コンパイラー・サブオプション 29
E コンパイラー・メッセージ 125
E15 入力処理ルーチン 404
E35 出力処理ルーチン 407
EBCDIC
  コンパイラー・サブオプション 26
EBCDIC (拡張 2 進化 10 進コード) 251
ENDFILE
  OS/390 の下での 198
Enterprise PL/I for z/OS
  アクセシビリティー xlvi
Enterprise PL/I ライブラリー
  言語環境プログラム・ライブラリー xvi
  Enterprise PL/I for z/OS ライブラリー xv

ENVIRONMENT オプション
同等の DCB サブパラメーター 262
編成オプション 261
領域データ・セット 320
レコード・フォーマット・オプション 283
BLKSIZE オプション
DCB サブパラメーターとの比較 261
CONSECUTIVE 283, 307
CTLASA および CTL360 308
DCB サブパラメーターとの比較 261
GRAPHIC オプション 285
KEYLENGTH オプション
DCB サブパラメーターとの比較 261
LEAVE および REREAD 310
RECSIZE オプション
使用法 284
レコード・フォーマット 284
DCB サブパラメーターとの比較 261
VSAM
BKWD オプション 340
BUFND オプション 341
BUFNI オプション 341
BUFSP オプション 341
GENKEY オプション 341
PASSWORD オプション 342
REUSE オプション 342
SKIP オプション 342
VSAM オプション 343
ENVIRONMENT 属性
リスト 259
z/OS UNIX での特性の指定
BUFSIZE 245
ENVIRONMENT の F オプション
ストリーム入出力用の 282, 283
レコード入出力用の 262
ENVIRONMENT の FB オプション
ストリーム入出力用の 282, 283
レコード入出力用の 262
ENVIRONMENT の FBS オプション
ストリーム入出力用の 282
レコード入出力用の 262
ENVIRONMENT の FS オプション
ストリーム入出力用の 282
レコード入出力用の 262
ENVIRONMENT の REGIONAL オプション 320
ENVIRONMENT の U オプション
ストリーム入出力用の 282, 283
レコード入出力用の 262
ENVIRONMENT の V オプション
ストリーム入出力用の 282, 283
ENVIRONMENT の V オプション (続き)
レコード入出力用の 262
ENVIRONMENT の VB オプション
ストリーム入出力用の 282, 283
レコード入出力用の 262
ENVIRONMENT の VBS オプション
ストリーム入出力用の 282
ENVIRONMENT の VS オプション
ストリーム入出力用の 282
EQUALS ソート・オプション 396
ESDS (入力順データ・セット)
固有キー代替索引パス 355
定義 346
非固有キー代替索引パス 357
VSAM 333
ステートメントとオプション 344
ロード 345
EVENDEC コンパイラー・サブオプション 29
EXEC SQL ステートメント 137
EXEC ステートメント
オプションの指定 191
オプション・リストの最大長 191
カタログ式プロシージャーの変更 181
コンパイラー 186
最小領域サイズ 186
紹介 186
OS/390 バッチ・コンパイル 183, 186
PARM パラメーター 190
EXIT コンパイラー・オプション 37
export コマンド 243
EXPORTALL 37
EXPORTALL コンパイラー・オプション 37
EXTERNAL 属性 119
EXTRN コンパイラー・オプション 38

F

F フォーマット・レコード 252
FB フォーマット・レコード 252
FETCH
アセンブラー・ルーチン 211, 217
Enterprise PL/I ルーチン 201, 216
OS/390 C ルーチン 211
PL/I MAIN ルーチン 210, 217
FILE 属性 119
FILEREF コンパイラー・オプション 38
filespec 244
FILLERS、タブ制御テーブルの 295
FILSZ ソート・オプション 396
FIXED
z/OS UNIX での TYPE オプション 250
FLAG コンパイラー・オプション 38
FLOAT オプション 39
FLOATINMATH コンパイラー・オプション 41
FUNC サブパラメーター
使用法 256

G

GENKEY オプション
キーの分類 266
使用法 259
VSAM 339
GET DATA ステートメント 197
GET EDIT ステートメント 198
GET LIST ステートメント 197
GOFF コンパイラー・オプション 41
GONUMBER コンパイラー・オプション 374
定義 42
GOTO ステートメント 382
GRAPHIC オプション
コンパイラー 43
ストリーム入出力 282
ENVIRONMENT の 259, 285

H

HEADER 43
HEADER コンパイラー・オプション 43
HEXADEC コンパイラー・サブオプション 30

I

I コンパイラー・メッセージ 125
IBM コンパイラー・サブオプション 26
IBMQC カタログ式プロシージャー 174
IBMQCB カタログ式プロシージャー 175
IBMQCBG カタログ式プロシージャー 177
IBMUEXIT コンパイラー出口 544
IBMZC カタログ式プロシージャー 168
IBMZCB カタログ式プロシージャー 170
IBMZCBG カタログ式プロシージャー 172
ID
参照されない 10
ソース・プログラム 10
IEC225I 256, 328
IGNORE オプション 44
ILC
リンケージの考慮事項 424
C との 415
構造化データ・タイプ 416
出力の共用 426

ILC (続き)
C との (続き)
ストリング・パラメーター・タイプの一致 421
データ・タイプ 415
入力の共有 427
パラメーターの一致 418
ファイル・タイプ 417
ATTACH ステートメントの使用 427
C 標準ストリームのリダイレクト 427
ENTRY を戻す関数 423
enum データ・タイプ 417
ILC でのリンケージ考慮事項 424
INCAFTER コンパイラー・オプション 44
INCDIR コンパイラー・オプション 44
INCLUDE コンパイラー・オプション 45
INCLUDE ステートメント
コンパイラー 115
INDEXAREA オプション 259
INITFILL コンパイラー・サブオプション 30
INLINE コンパイラー・サブオプション 30
INSOURCE オプション 48
INTERNAL 属性 119
INTERRUPT コンパイラー・オプション 49

J

JAVA 429
Java 431, 432, 434, 435, 437, 440, 441, 443, 445, 446, 450
Java コード、コンパイル 432, 437, 443, 446
Java コード、作成 431, 435, 441, 445
JCL (ジョブ制御言語)
高効率化 167
コンパイル中の使用 186
jni
JNI サンプル・プログラム 430, 435, 440, 445

K

KEYLEN サブパラメーター 256
KEYLENGTH オプション 259, 268
KEYLOC オプション
使用法 259
KEYTO オプション
VSAM の下での 345
KSDS (キー順データ・セット)
更新 352
固有キー代替索引パス 358
定義とロード 350
VSAM
ロード 350
DIRECT ファイル 352
SEQUENTIAL ファイル 352

L

LANGLVL コンパイラー・オプション 50
LEAVE および REREAD オプション
ENVIRONMENT オプション
連続データ・セット用の 310
LIMCT サブパラメーター 256, 329
LIMITS コンパイラー・オプション 51
LINE オプション 283, 293
LINECOUNT コンパイラー・オプション 52
LINEDIR コンパイラー・オプション 52
LINESIZE オプション
タブ制御テーブルの 295
OPEN ステートメント 284
LINKAGE コンパイラー・サブオプション
構文 31
パフォーマンスへの影響 379
LIST コンパイラー・オプション 53
LISTVIEW コンパイラー・オプション 53
LOWERINC コンパイラー・サブオプション 31
LP 54
LP コンパイラー・オプション 54
LRECL オプション、z/OS UNIX での 246
LRECL サブパラメーター 251, 256
LRMSKIP オプション、z/OS UNIX での 246

M

MACRO オプション 55
MAP コンパイラー・オプション 55
MARGINI コンパイラー・オプション 56
MARGINS コンパイラー・オプション 56
MAXBRANCH コンパイラー・オプション 57
MAXGEN コンパイラー・オプション 58
MAXMEM コンパイラー・オプション 58
MAXMSG コンパイラー・オプション 59
MAXSTMT コンパイラー・オプション 60
MAXTEMP コンパイラー・オプション 60
MDECK コンパイラー・オプション
説明 60
MODE サブパラメーター
使用法 256
MSGSUMMARY コンパイラー・オプション 61

N

NAME コンパイラー・オプション 62
NAMES コンパイラー・オプション 62
NATIVE コンパイラー・サブオプション
説明 31
NATIVEADDR コンパイラー・サブオプション 32
NATLANG コンパイラー・オプション 63
NEST オプション 63
NOBIN1ARG コンパイラー・サブオプション 27
NODESCRIPTOR コンパイラー・サブオプション 28
NOEQUALS ソート・オプション 396
NOEVENDEC コンパイラー・サブオプション 29
NOINITFILL コンパイラー・サブオプション 30
NOINLINE コンパイラー・サブオプション 30
NOINTERRUPT コンパイラー・オプション 49
NOMAP オプション 120
NOMARGINS コンパイラー・オプション 56
NONASSIGNABLE コンパイラー・サブオプション 27
NONCONNECTED コンパイラー・サブオプション 28
NONE、フック位置サブオプション 101
NONNATIVE コンパイラー・サブオプション 31
NONNATIVEADDR コンパイラー・サブオプション 32
NONRECURSIVE コンパイラー・サブオプション 33
NONULLSTRADDR コンパイラー・サブオプション 32
NOOVERLAP コンパイラー・サブオプション 33
パフォーマンスへの影響 380
NOPSEUDODUMMY コンパイラー・サブオプション 33
NOSYNTAX コンパイラー・オプション 98
NOT コンパイラー・オプション 63
note ステートメント 125

NTM サブパラメーター
使用法 256
NULL370 コンパイラー・サブオプション 32
NULLDATE 64
NULLDATE コンパイラー・オプション 64
NULLSTRADDR コンパイラー・サブオプション 32
NULLSTRPTR コンパイラー・サブオプション 33
NULLSYS コンパイラー・サブオプション 32
NUMBER コンパイラー・オプション 64

O

OBJECT コンパイラー・オプション
定義 65
OFFSET コンパイラー・オプション 65
OFFSETSIZE コンパイラー・オプション 65
ONSNAP コンパイラー・オプション 66
OPEN ステートメント
PL/I ライブラリーのサブルーチン 257
TITLE オプション 257
OPTCD サブパラメーター 255, 256
OPTIMIZE オプション 66
OPTIMIZE コンパイラー・オプション 373
OPTIONS オプション 67
OR コンパイラー・オプション 68
ORDER コンパイラー・サブオプション
説明 33
パフォーマンスへの影響 380
ORDINAL コンパイラー・サブオプション 33
ORGANIZATION オプション 268
使用法 259
OS/390
一般コンパイル 183
長い入力行 197
バッチ・コンパイル
一時作業ファイル (SYSUT1) 189
オプションの指定 190
ソース・ステートメント・ライブラリー (SYSLIB) 190
リスト (SYSPRINT) 189
DD ステートメント 187
EXEC ステートメント 186, 191
OVERLAP コンパイラー・サブオプション 33

P

PACKAGE 対ネストされた PROCEDURE 383
PAGE オプション 283
PAGELENGTH、タブ制御テーブルの 295
PAGESIZE、タブ制御テーブルの 295
PARM パラメーター
オプションの指定 191
カタログ式プロシージャー 181
PASSWORD オプション 342
PLICANC ステートメント、およびチェックポイント/要求 538
PLICKPT 組み込みサブルーチン 535
PLIDUMP 組み込みサブルーチン
各ブロックごとの DSA に保存された 523
構文 521
出力
変数の検出 523
ダンプ・トレースバック・テーブル内のプログラム単位名 523
変数
AUTOMATIC 変数の検出 523
CONTROLLED 変数の検出 526
PLIDUMP 出力内での検出 523
STATIC 変数の検出 525
保存されたオプション・ストリング 530
保存されたロード・モジュール のタイム・スタンプ 530
ユーザー ID 522
H オプション 522
z/OS 言語環境プログラム ダンプを生成するための呼び出し 521
PLIDUMP に保存されたオプション・ストリング 530
PLIREST ステートメント 538
PLIRETC 組み込みサブルーチン
ソートでの戻りコード 402
PLISAXA 455, 456
PLISAXB 455, 457
PLISAXC 485, 486
PLISAXD 485, 487
PLISRTA インターフェース 408
PLISRTB インターフェース 410
PLISRTC インターフェース 411
PLISRTD インターフェース 412
PLITABS 外部構造
制御セクション 296
宣言 194
PLIXOPT 変数 194, 216
PL/I
コンパイラー
ユーザー出口のプロシージャー 542
ファイル
z/OS UNIX でデータ・セットと関連付ける 240
PL/I MAIN ルーチン
FETCH 210, 217
PL/I コード、コンパイル 434, 440, 445, 450
PL/I コード、作成 432, 437, 443, 446
PL/I コード、リンク 434, 440, 445, 450
PL/I 動的割り振り
データ・セットの割り振り 233
ファイルの定義
ストリーム・ファイル 282
QSAM ファイル 301
REGIONAL(1) データ・セット 319
VSAM ファイル 331
z/OS UNIX でのデータ・セットとファイルの関連付け 240
z/OS での HFS ファイルへのアクセス 239
PP コンパイラー・オプション 68
PPCICS コンパイラー・オプション 70
PPINCLUDE コンパイラー・オプション 70
PPLIST コンパイラー・オプション 71
PPMACRO コンパイラー・オプション 71
PPSQL コンパイラー・オプション 72
PPTRACE コンパイラー・オプション 72
PRECTYPE コンパイラー・オプション 73
PREFIX コンパイラー・オプション 73, 376
デフォルトのサブオプションの使用 376
PRINT ファイル
出力に句読点を付ける 194
レコード入出力 314
フォーマット設定規則 194
PROCEED コンパイラー・オプション 74
PROCESS ステートメント
オプション・デフォルトの指定変更 190
説明 113
PROMPT オプション、z/OS UNIX での 247
PRTSP サブパラメーター
使用法 256
PSEUDODUMMY コンパイラー・サブオプション 33
PUT EDIT コマンド 305

PUTPAGE オプション、z/OS UNIX での 247

Q

QUOTE コンパイラー・オプション 75

R

REAL 属性 119
RECCOUNT オプション、z/OS UNIX での 247
RECFM サブパラメーター 256
  使用法 256
  データ・セットの編成における 256
RECORD ステートメント 397
RECSIZE オプション
  ストリーム入出力用の 282, 284
  定義 263
  デフォルト値 284
  連続データ・セット 284
  z/OS UNIX での記述 248
RECURSIVE コンパイラー・サブオプション 33
REDUCE コンパイラー・オプション 75
  パフォーマンスへの影響 374
REDUCIBLE 関数 384
REGIONAL(1) データ・セット
  ファイルの定義
    領域データ・セット 319
RENT コンパイラー・オプション 76
REORDER コンパイラー・サブオプション
  説明 33
  パフォーマンスへの影響 380
RESEXP コンパイラー・オプション 78
RESPECT コンパイラー・オプション 78
RETCODE コンパイラー・サブオプション 34
RETURNS コンパイラー・サブオプション 34, 380
REUSE オプション 259, 342
RRDS (相対レコード・データ・セット)
  更新 368
  定義 365
  VSAM
    ロード 364
    DIRECT ファイル 367
    SEQUENTIAL ファイル 367
  VSAM でのロード 364
RTCHECK コンパイラー・オプション 78
RULES コンパイラー・オプション 79
  パフォーマンスへの影響 375

S

S コンパイラー・メッセージ 125
SAMELINE オプション、z/OS UNIX での 248
SAX パーサー 455, 485
SCALARVARYING オプション 267
SEMANTIC コンパイラー・オプション 95
SEQUENTIAL ファイル
  RRDS、アクセス・データ・セット 367
  VSAM での ESDS
    更新 347
    定義とロード 346
  VSAM での索引付き ESDS
    データ・セットへのアクセス 352
SERVICE コンパイラー・オプション 95
SHORT コンパイラー・サブオプション 34
SKIP オプション 342
  ストリーム入出力における 283
  OS/390 の下での 198
SKIP0 オプション、z/OS UNIX での 249
SKIPREC ソート・オプション 396
SOURCE コンパイラー・オプション 96
SPACE パラメーター
  標準データ・セット 187
  ライブラリー 275
SPILL コンパイラー・オプション 96
SQL プリプロセッサー
  オプション 141
  記述子域 146
  標識変数の使用 156
  ホスト構造体の使用 156
  ホスト変数の使用 149
  連絡域 145
  EXEC SQL ステートメント 137
  IBMUEXIT の使用 161
SQL プリプロセッサー・オプション 142, 143, 144
SQLCA 145
SQLDA 146
STACK サブパラメーター
  使用法 256
STATIC コンパイラー・オプション 96
STDSYS コンパイラー・オプション 97
STMT コンパイラー・オプション 97
STMT サブオプション、テストの 101
STORAGE コンパイラー・オプション 98
STREAM 属性 281
STRINGOFGRAPHIC コンパイラー・オプション 98
SUB 制御文字 251
SYNTAX オプション 98
SYS1.PROCLIB (システム・プロシージャー・ライブラリー) 273
SYSADATA 情報、オプション・レコード 568
SYSADATA 情報、カウンター・レコード 568
SYSADATA 情報、構文情報 576
SYSADATA 情報、構文レコード 579
SYSADATA 情報、サマリー・レコード 567
SYSADATA 情報、序数エレメント・レコード 572
SYSADATA 情報、序数タイプ・レコード 571
SYSADATA 情報、シンボル情報 571
SYSADATA 情報、シンボル・レコード 573
SYSADATA 情報、ソース・レコード 577
SYSADATA 情報、トークン・レコード 578
SYSADATA 情報、ファイル・レコード 569
SYSADATA 情報、メッセージ・レコード 570
SYSADATA 情報、リテラル・レコード 569
SYSADATA 情報の概要 565
SYSCHK デフォルト 536
SYSIN 188, 270
  C との共用 427
SYSIN および SYSPRINT ファイル 297
SYSLIB
  プリプロセス 190
  %INCLUDE 115
SYSLIN 189
SYSOUT 402
SYSPARM コンパイラー・オプション 99
SYSPRINT 270
  以前の PL/I との共有 200
  エンクレーブ間での共用 198
  および z/OS UNIX 270
  コンパイラー・リストの書き込み先 189
  必須 DD ステートメント 187
  C との共用 426, 427
  DD オプションでの指定 22
  MSGFILE(SYSPRINT) の使用 200
  PUT ステートメントでの使用 297
  STDSYS オプションの作用 97
SYSPUNCH 189
SYSUT1 コンパイラー・データ・セット 189

T

TERMINAL コンパイラー・オプション 101
TEST コンパイラー・オプション
  定義 101
TIME パラメーター 181
TIMESTAMP
  PLIDUMP 内の保存されたロード・モジュール のタイム・スタンプ 530
TITLE オプション
  使用 257
  標準 SYSPRINT ファイルの関係付け 199
  z/OS UNIX での記述 241
TITLE オプション、OS/390 での
  文字ストリング値 235
TITLE オプション、z/OS UNIX での
  データ・セットに関連付けられていないファイルの使用 243

U

U コンパイラー・メッセージ 125
U フォーマット 253
UNDEFINEDFILE 条件
  BLKSIZE エラー 264
  OPEN での行サイズの矛盾 293
  z/OS UNIX でのファイル・オープン時に発生する 250
UNDEFINEDFILE 条件、OS/390 での
  DD ステートメント・エラー 236
UNDEFINEDFILE 条件、z/OS UNIX での
  データ・セットに関連付けられていないファイルの使用 250
UNIT パラメーター
  連続データ・セット 313
UNROLL コンパイラー・オプション 104
UPPERINC コンパイラー・サブオプション 31
USAGE コンパイラー・オプション 105

V

VB フォーマット・レコード 252
VOLUME パラメーター
  連続データ・セット 313
VSAM (仮想記憶アクセス方式)
  索引付きデータ・セット
    ロード・ステートメントとオプション 348
  相対レコード・データ・セット 365
  大量順次挿入 354
  データ・セット
    キー 335
VSAM (仮想記憶アクセス方式) (続き)
  データ・セット (続き)
    キー順および索引付き入力順 348
    使用 331
    相対レコード 363
    代替索引 355
    代替索引パス 343
    タイプの選択 336
    ダミー・データ・セット 337
    定義 331, 344
    入力順 344
    パフォーマンス・オプション 343
    ファイルの定義 339
    プログラムの実行 331
    ブロック化 333
    編成 332
    ENVIRONMENT オプションの指定 339
  ファイルの定義 339
    パフォーマンス・オプション 343
    ENV オプション 339
  VSAM オプション 343
VSAM での KSDS
  VSAM での索引付き ESDS
    データ・セットの更新 354
    データ・セットへのアクセス 352
VTOC 255

W

W コンパイラー・メッセージ 125
WIDECHAR コンパイラー・オプション 106
WINDOW コンパイラー・オプション 107
WRITABLE コンパイラー・オプション 107

X

XINFO コンパイラー・オプション 108
XML
  SAX パーサーでのサポート 455, 485
XML コンパイラー・オプション 50, 111
XREF コンパイラー・オプション 112

Z

z/OS UNIX
  環境変数の設定 270
  コンパイル 183
  コンパイル時オプション
    指定 184
  コンパイル時オプションの指定
    コマンド行 184
z/OS UNIX (続き)
  フラグの使用 185
  DD_DDNAME 環境変数 243
  export コマンド 243
z/OS UNIX での TYPE オプション 249
z/OS UNIX でのデータ定義 (DD) 情報 240
z/OS UNIXでの代替 dd 名、TITLE オプションの 242

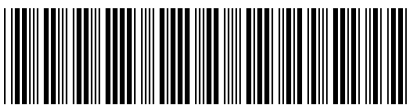
[特殊文字]

*PROCESS、内のオプションの指定 113
/ (スラッシュ) 242
% ステートメント 115
%INCLUDE ステートメント 115, 190
  制御ステートメント 115
  ソース・ステートメント・ライブラリー 190
%NOPRINT 115
  制御ステートメント 115
%NOPRINT ステートメント 115
%PAGE 115
  制御ステートメント 115
%PAGE ステートメント 115
%POP ステートメント 115
%PRINT 115
  制御ステートメント 115
%PRINT ステートメント 115
%PROCESS、内のオプションの指定 113
%PUSH ステートメント 115
%SKIP 115
  制御ステートメント 115
%SKIP ステートメント 115

プログラム番号: 5655-PL5

Printed in Japan

GC43-3419-01



日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21