# 第４次 COBOL 規格
# COBOL2002 のご紹介

2001 年６月　第一版


COBOL コンソーシアム

基礎技術分科会

http://www.cobol.gr.jp/

# 目次

概要 ..... 9

0.1 日程 ..... 9

0.2 本文解説の目的と注意事項 ..... 9

0.3 新規追加項目 ..... 10

１章 コンパイル時指示機能 ..... 11

1.1 DEFINE 指示 ..... 11

1.2 EVALUATE 指示 ..... 12

1.3 IF 指示 ..... 12

1.4 FLAG85 指示 ..... 13

1.5 SOURCE FORMAT 指示 ..... 13

1.6 PROPAGATE 指示 ..... 13

1.7 TURN 指示 ..... 14

２章 自由形式の正書法 ..... 15

2.1 固定形式のソーステキスト ..... 15

2.2 自由形式のソーステキスト ..... 15

自由形式のプログラムの例 ..... 16

３章 ビット操作機能 ..... 17

3.1 ブーリアン式 ..... 18

3.2 組込み関数 ..... 19

3.3 アライメント ..... 20

**4章 漢字等の多オクテット文字機能 ..... 22**

**4.1 文字集合 ..... 22**

4.1.1 特殊名段落（SPECIAL-NAMES） ..... 22

4.1.1.1 符号系名句（ALPHABET） ..... 23

**4.2 データ部 ..... 23**

4.2.1 国別文字定数 ..... 23

4.2.2 データ記述項 ..... 24

4.2.2.1 PICTURE 句 ..... 24

4.2.2.2 USAGE 句 ..... 24

**4.3 組込み関数 ..... 24**

4.3.1 CHAR-NATIONAL 関数 ..... 25

4.3.2 DISPLAY-OF 関数 ..... 25

4.3.3 NATIONAL-OF 関数 ..... 25

**5章 浮動小数点データ操作機能 ..... 26**

**5.1 浮動小数点数字定数 ..... 26**

**5.2 浮動小数点数字項目 ..... 26**

**5.3 USAGE 句 ..... 27**

**5.4 組込み関数 ..... 27**

**6章 ポインタ項目とアドレス付け機能 ..... 28**

**6.1 データアドレスとデータポインタ ..... 28**

**6.2 プログラムアドレスとプログラムポインタ ..... 30**

**7章 利用者定義のデータ型機能 ..... 31**

**7.1 データ型と型名 ..... 31**

7.1.1 弱く型付けされた項目 ..... 32

7.1.2 強く型付けされた項目 ..... 32

**7.2 TYPE 句と TYPEDEF 句 ..... 34**

7.1.1 TYPE 句 ..... 34
7.1.1.1 書き方 ..... 34
7.1.1.2 主な規則 ..... 34
7.1.1 TYPEDEF 句 ..... 34
7.1.1.1 書き方 ..... 34
7.1.1.2 主な規則 ..... 34

7.3 例題 ..... 35

8章 利用者定義の関数機能 ..... 37

8.1 関数一意名 ..... 37

8.2 利用者定義関数とプログラムとの違い ..... 38

8.3 利用者関数の定義 ..... 38

8.4 利用者定義関数を使うための宣言 ..... 39

9章 ファイルの共用と排他制御の機能 ..... 40

9.1 ファイル共用とレコードロック ..... 40
9.1.1 ファイル共用 ..... 40
9.1.2 レコードロック ..... 41

9.2 SHARING によるファイル共用の指定 ..... 42

9.3 LOCK 句によるレコードロックの指定 ..... 43

9.4 RETRY指定 による繰返し排他制御の指定 ..... 44

9.5 記述例 ..... 45

10章 画面処理機能 ..... 46

10.1 画面の概念 ..... 46
10.1.1 フィールド ..... 46
10.1.2 属性 ..... 46
10.1.3 機能キー ..... 47
10.1.4 CRT 状態 ..... 47

10.1.5 カーソル ..... 47

10.2 画面処理機能の環境部 ..... 47
10.2.1 CURSOR 句の書き方 ..... 47
10.2.1 CRT STATUS 句の書き方 ..... 48

10.3 画面処理機能のデータ部 ..... 48
10.3.1 画面節の書き方 ..... 48

10.4 画面記述項 ..... 49
10.4.1 画面記述項の書き方 ..... 49
【書き方 1 (集団項目)】 ..... 49
【書き方 2 (基本項目)】 ..... 50
10.4.2 属性句 ..... 51

10.5 画面処理機能の手続き部 ..... 52
10.5.1 ACCEPT 文の書き方 ..... 52
10.5.2 DISPLAY 文の書き方 ..... 53

11章 例外割り込み処理機能 ..... 54

11.1 例外割り込み処理機能の概要 ..... 54

11.2 例外名 ..... 55

11.3 TURN コンパイル指令 ..... 59

11.4 例外割り込み処理機能の手続き部 ..... 60
11.4.1 手続き部の枠組み ..... 60
11.4.2 手続き部見出し ..... 60
11.4.3 宣言部分 ..... 61
11.4.4 EXIT PROGRAM 文 ..... 61
11.4.5 RESUME 文 ..... 62
11.4.6 RAISE 文 ..... 62
11.4.7 USE 文 ..... 62
11.4.8 例外処理の組み込み関数 ..... 63

**12章 データの妥当性検査機能 ..... 64**

**12.1 妥当性検査機能の概要 ..... 64**

**12.2 妥当性検査機能のデータ部 ..... 65**

12.2.1 ALLOW 句 ..... 65
12.2.2 CLASS 句 ..... 65
12.2.3 DEFAULT 句 ..... 66
12.2.4 DESTINATION 句 ..... 66
12.2.5 ERROR 句 ..... 66
12.2.6 INVALID 句 ..... 67
12.2.7 VALID/INVALID VALUE 句 ..... 67

**12.3 VALIDATE文 ..... 68**

**13章 オブジェクト指向機能 ..... 70**

**13.1 クラスとオブジェクト ..... 70**

13.1.1 クラス ..... 71
13.1.2 インスタンスオブジェクト ..... 72
13.1.3 ファクトリオブジェクト ..... 73
13.1.4 定義済みオブジェクト ..... 73

**13.2 継承と多態 ..... 74**

13.2.1 適合 ..... 74
13.2.2 継承 ..... 74
13.2.3 多態 ..... 76
13.2.4 インタフェース ..... 76

**13.3 オブジェクト参照 ..... 79**

13.3.1 オブジェクト参照の定義 ..... 79
13.3.2 適合チェック ..... 79
13.3.3 手続き部での扱い ..... 80
13.3.4 オブジェクトビュー ..... 80

**13.4 メソッド ..... 82**

13.4.1 メソッドの一意性 ..... 83
13.4.2 メソッド呼出し ..... 83

**13.5 オブジェクトプロパティ ..... 84**
13.5.1 オブジェクトプロパティの設定/参照 ..... 84
13.5.2 PROPERTY 句 ..... 84

**13.6 オブジェクトの寿命 ..... 86**

**13.7 リポジトリ ..... 87**
13.7.1 クラス/インタフェースの情報 ..... 87
13.7.2 REPOSITORY 段落 ..... 87

**13.8 その他のオブジェクト指向機能 ..... 88**
13.8.1 パラメタ化クラス ..... 88
13.8.2 例外オブジェクト ..... 88
13.8.3 クラスライブラリ ..... 88

**13.9 オブジェクト指向関連の構文 ..... 89**
【行内メソッド呼出し】 ..... 89
【オブジェクトビュー】 ..... 89
【定義済みオブジェクト参照】 ..... 89
【オブジェクトプロパティ】 ..... 89
【クラス定義】 ..... 90
【インタフェース定義】 ..... 90
【ファクトリ定義】 ..... 91
【インスタンス定義】 ..... 91
【メソッド定義】 ..... 92
【リポジトリ段落】 ..... 92
【クラス指定子】 ..... 92
【インタフェース指定子】 ..... 92
【プロパティ指定子】 ..... 93
【PROPERTY 句】 ..... 93
【USAGE 句】 ..... 93
【手続き部の見出し】 ..... 93
【EXIT 文】 ..... 93
【GOBACK 文】 ..... 95
【RAISE 文】 ..... 95
【USE 文】 ..... 95
【SET 文】 ..... 95

**14章 言語間連絡の拡張 ..... 96**

**14.1 連絡機能の拡張 ..... 96**

**14.2 言語間連絡の拡張について ..... 99**

**15章 標準算術演算と31桁への拡張 ..... 100**

**15.1 算術演算の３１桁拡張について ..... 100**

**15.2 標準算術演算とは ..... 100**

**15.3 演算種類毎の規定概要 ..... 101**

**15.4 標準算術演算の性質 ..... 103**

**16章 その他 ..... 104**

**16.1 POSIX(注)のロケールに対応した地域・文化固有機能 ..... 104**

**16.2 国際化、他国語対応とは何か ..... 104**

**16.3 ロケール ..... 105**

**16.4 国際化の具体例 ..... 106**

**16.4 国際化のまとめ ..... 108**

**16.5 既存プログラムとの互換 ..... 109**

# 概要

現在、アメリカ COBOL 標準化委員会（NCITS/J4）で原案最終草案（FCD）の作成が進められている、第 4 次 COBOL 規格（COBOL2002）に関して、機能概要を新規機能項目ごとにご紹介します。

## 0.1 日程

第４次 COBOL 規格(COBOL2002 以降 COBOL2002 と呼びます)は､現在アメリカ COBOL 標準化委員会（(NCITS/J4）で原案最終草案（FCD）作成を完了しました。今後の規格化までのスケジュールは、次のとおりです。

1. 2002 年： 国際規格推薦最終案（FDIS）と国際規格（IS）の投票が 2002 年中
2. 2002 年 12 月： 国際規格の発行
3. 2003 年： 改定 JIS COBOL の発行

## 0.2 本文解説の目的と注意事項

本解説は第 4 次 COBOL 規格（COBOL2002）の機能面での紹介を目的としています｡2001 年 4 月現在最終規格化されていない提案草稿をベースに解説しています。このため次の点について注意が必要です。

- 本解説は、FCD 案をベースにした解説であるため 実際に発行される規格とは異なる可能性があります。
- 本解説は、次期規格草案の解説を目的としており、現在販売されている COBOL 製品、今後販売されるであろう COBOL 製品の機能を解説するものではなく、機能を規定するものでもありません。
- 本解説で使用する、用語は統一されたものではありません。改定 JIS 作成作業で用語が定義されます。本解説で使用する用語は機能を伝える目的で使用するものです。

# 0.3 新規追加項目

**COBOL2002 では、全体で約 150 の追加変更が行われています。ただし、追加変更項目の互換性の保持には十分に注意がはらわれており、従来の COBOL プログラムから外れるものではありません。大きな追加機能項目として次の項目があります。**

1.コンパイラ指示機能
2.自由形式の正書法
3.ビット操作機能
4.漢字等の多オクテット文字機能
5.浮動小数点データ操作機能
6.ポインタ項目とアドレス付け機能
7.利用者定義のデータ型機能
8.利用者定義の関数機能
9.ファイルの共用と排他制御の機能
10.画面処理機能
11.例外割り込み処理機能
12.データの妥当性検査機能
13.オブジェクト指向機能
14.言語間連絡の拡張
15.標準算術演算と３１桁への拡張
16.その他（POSIXのロケールに対応した地域・文化固有機能、
既存プログラムとの互換）

# １章 コンパイル時指示機能

従来の COBOL では、プログラムテキストに関しての操作や条件翻訳（条件によってプログラムテキストを有効/無効する）機能は、REPLACE 指定の COPY 文、REPLACE 文等がありました。ただ、C 等の他言語に比較すると、条件により変更ができない等、扱いずらい面があり、COBOL 向けのプリプロセッサを利用して代替する場合もありました。
COBOL2002 では、この点を強化しコンパイル時指示機能として新たな機能を提供しています。
コンパイル時指示機能では、コンパイル時の選択機能やコンパイル時に利用する変数（翻訳変数）の定義を行うことが可能です

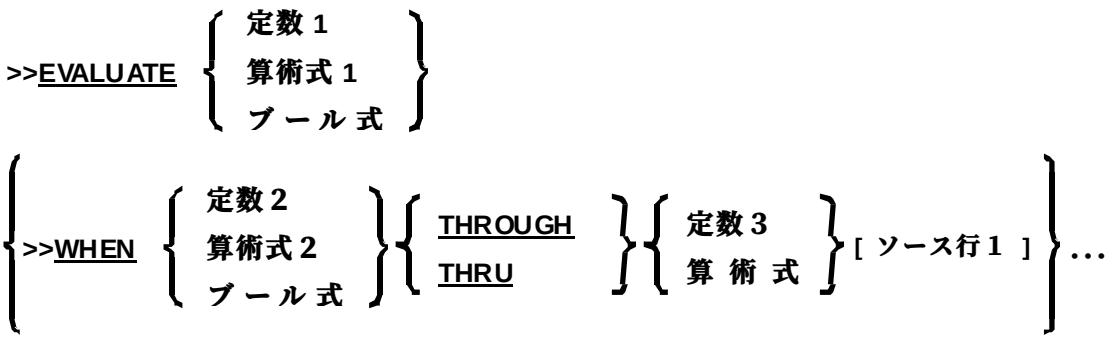
- 条件翻訳：コンパイル時に翻訳の対象または、省略の対象になるプログラムソースの行の選択を行う機能をいいます。DEFINE 指示、EVALUATE 指示、IF 指示があります。

## 1.1 DEFINE 指示

コンパイル時変数の定義を行い、コンパイル時変数へ値を設定します。

## 1.2 EVALUATE 指示

多分岐の条件コンパイルを行います。




[ >>WHEN OTHER [ ソース行２ ] ]

>>END-EVALUATE

## 1.3 IF 指示

単分岐の条件コンパイルを行います。

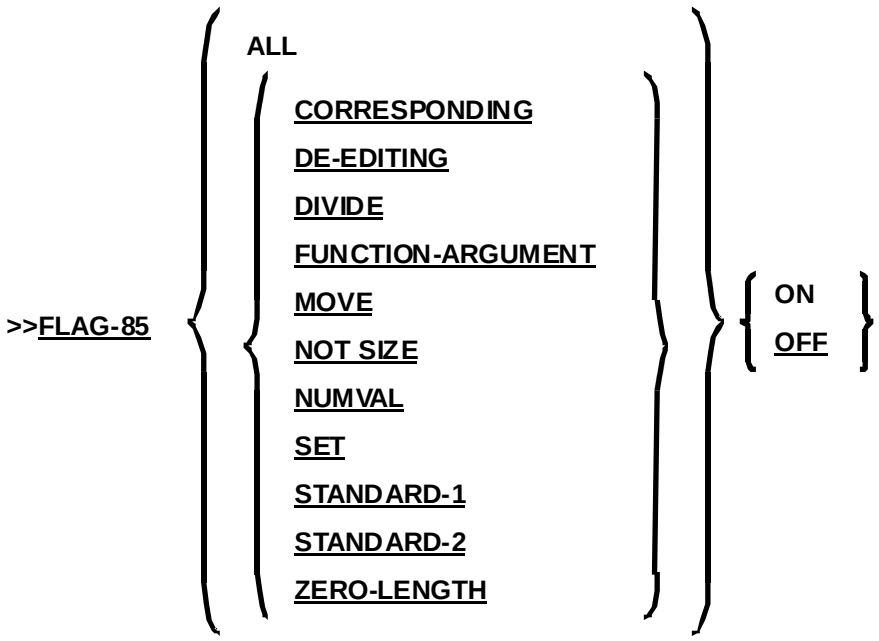
>>IF 定数条件式１ [ ソース行１ ]

>>END-IF

[ >>ELSE [ ソース行２ ] ]

## 1.4 FLAG85 指示

従来の COBOL 規格と COBOL2002 規格の間で、互換性が欠ける構文に関してフラグを立てる項目を指定します。




## 1.5 SOURCE FORMAT 指示

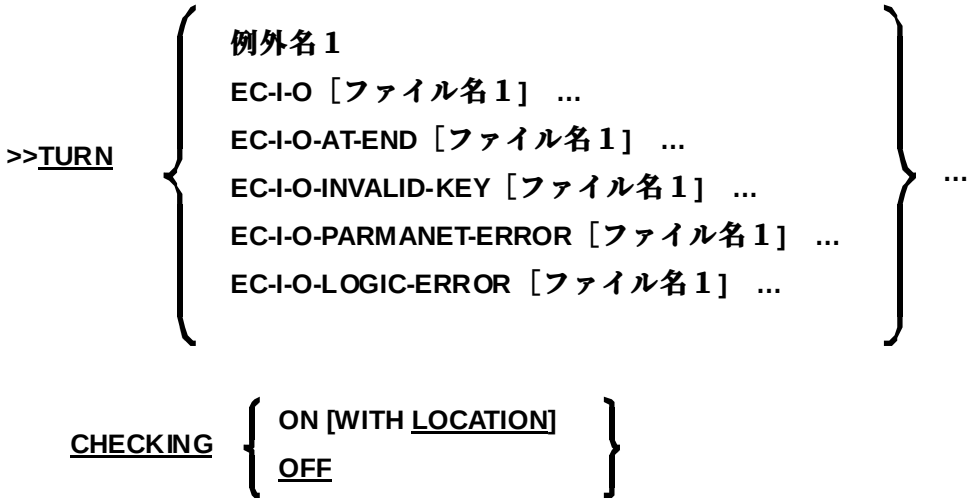
後続のプログラムテキストの書式が、固定形式（FIXED）か自由形式（FREE）かを指定します。


## 1.6 PROPAGATE 指示

後続のプログラムテキストの関数、メソッド、プログラムに対して例外条件の自動的な伝播を有効とします。

# 1.7 TURN 指示

**後続するプログラムテキストに対して、特定の例外条件に対するチェックを行うかどうかを指示します。**

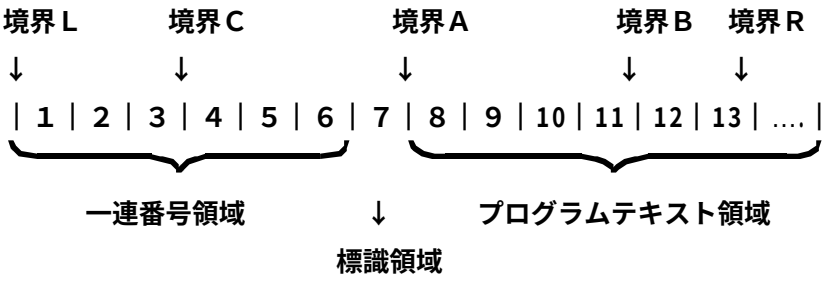


# ２章　自由形式の正書法

　従来の固定形式のソーステキストに加えて自由形式のソーステキストが追加されました。固定形式または自由形式のソーステキストは、コンパイル時指示文（>>SOURCE FORMAT )により変更可能です。また、>>SORUCE　FORMAT 指示により、二つのソーステキスト形式をソーステキストと COPY 原文のなかで混在して記述できます。
さらに、固定形式のソーステキストにも若干の変更が加えられました。

## 2.1 固定形式のソーステキスト

　従来のソーステキストの形式です。COBOL2002 で大きな変更はありませんが、従来あったＡ領域Ｂ領域の識別が削除されました。
これにより、固定形式であってもプログラムテキストの記述はより自由になりました。




## 2.2 自由形式のソーステキスト

　自由形式のソーステキストでは、これまでの一連番号領域や、識別領域等の境界による制約がなくなりソーステキストをどこに書いてもよい形式です。
ただし識別領域がないため　継続、注記等に関しては、特別の規則が定められています。

・継続の規則：定数の継続は、"-　または　'-　によって行います。
　　例　　N"日本語の"-
　　"定数"　.
・注記の規則：コメントは、コメント指示子 *> の後ろにコメントを記述します。
　　例　　MOVE　DATA1　TO　DATA2 . *>コメントデータの転記

# 自由形式のプログラムの例

```
>>SOURCE FORMAT IS FREE
IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-FORM-EXP.
SOURCE-COMPUTER. XYZ WITH DEBUGGING MODE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 長い項目 PIC N(100) VALUE N"長い項目の定数の継続例"-
                        "定数の継続の残り".
01 数字項目 PIC 9(10).
PROCEDURE DIVISION.

paragraph-name.
*>これは、コメントの行です。
           MOVE 0 TO 数字項目.        *>これは、インラインのコメントです。
           MOVE "長い項目の定数の継続の例です。"&
"定数の継続の残り" TO 長い項目.
  .........
```

# 3章 ビット操作機能

この章では、１または０の値を持つブーリアンデータの操作機能について説明します。
COBOL2002 では、１または０の値を持つブーリアンデータ、ブーリアン定数の定義と、論理演算、論理比較機能が追加されました。ブーリアンデータは、その表現形式にＢＩＴ、ＤＩＳＰＬＡＹ、ＮＡＴＩＯＮＡＬが指定でき、用途がＢＩＴである場合にビット操作機能が使用できます。

・ブーリアンデータ

ＰＩＣＴＵＲＥ　１（ｎ）　ＵＳＡＧＥ　ＢＩＴ．
ｎビットのビットデータを定義する。

・ブーリアン定数

Ｂ"｛ブール文字（「０」または「１」)｝…"
例　Ｂ"０１０１００１１１０１００００"
ＢＸ"｛１６進数字｝…"　例　ＢＸ"３Ｆ７５０１２Ａ"

・ブーリアン演算子

Ｂ－ＮＯＴ：論理否定
Ｂ－ＡＮＤ：論理積
Ｂ－ＯＲ：論理和
Ｂ－ＸＯＲ：排他的論理和

## 3.1 ブーリアン式

ブーリアン演算子を使用することで、論理演算が行えます。その使用例を示します。

```
01  BOOLDATA    PIC   1(4)  USAGE  BIT  VALUE B"0000".
01  BOOLDATA2   PIC   1(4)  USAGE  BIT.
...
MOVE  B"0011"  TO  BOOLDATA2                  *> BOOLDATA2 の初期化
...
COMPUTE  BOOLDATA  =  B-NOT  BOOLDATA2
        *> BOOLDATA2 のビット反転を BOOLDATAに設定(B"1100")
...

COMPUTE  BOOLDATA  =  BOOLDATA  B-AND  B"0000"
                    *> BOOLDATAの全てのビットを0に設定(B"0000")
...
COMPUTE  BOOLDATA2  =  BOOLDATA2  B-OR  BX"8"
                    *> BOOLDATA2のビット位置１のビットをON  (B"1011")
```

最後の COMPUTE 命令は、次の MOVE 命令と等価です。

```
MOVE   B"1"   TO    BOOLDATA2(1:1)
```

次に論理比較の場合には、次のように記述します。

```
01  SBIT-DATA    PIC   1   USAGE  BIT.
01  MBIT-DATA    PIC   1(7)  USAGE  BIT.
...
IF  SBIT-DATA  THEN  ...    *>  SBIT-DATA が真（１）なら...
```

複数ビットの場合は、部分参照で次のように記述できます。

```
IF  MBIT-DATA(4:1)  THEN ... *> MBIT-DATA のビット位置 4 のビットが真（１）な
ら...
```

# 3.2 組込み関数

COBOL2002 では、ブーリアンデータと整数データとの相互変換を可能にする次の組込み関数が利用できます。

- **FUNCTION BOOLEAN-OF-INTGER ( 引数 1 , 引数 2 )**

引数 1 の絶対値と等価な二進値をあらわす、用途がビットのブーリアンデータを返します。引数 2 は、返されるブーリアンデータの長さを指定します。

- **FUNCTION INTGER-OF-BOOLEAN ( 引数 1 )**

引数 1 のブーリアン文字列が表現する二進値と等価な整数データ項目を返します。
次にその使用例を示します。

```
01  BIT-DATA     PIC   1(8)  USAGE  BIT.
01  INT-DATA     PIC   9(5)  VALUE  512.
01  INT-DATA2    PIC   9（3）.
…
MOVE  FUNCTION  BOOLEAN-OF-INTGER(  INT-DATA, 6) TO BIT-DATA.
  *>  MOVE 命令実行後、BIT-DATA には、B"00100000" が設定される。

COMPUTE  INT-DATA2 = FUNCTION-OF-BOOLEAN  (BIT-DATA(1:6)).
  *> COMUPTE 命令実行後、INT-DATA2 には、32 が設定される。
```

# 3.3 アライメント

用途がビットであるブーリアンデータのアライメントは、項目定義の始まりがどこであるかに依存します。

- 基本項目である場合

先頭ビットは、バイトアラインされ、続くビットは隣接して配置されます。

例）

```
77  BIT-DATA1  PIC  111  USAGE BIT.
 1  BIT-DATA2  PIC  1(10) USAGE BIT.
```

| ビット位置 | バイト数 |
|---|---|
| 1 2 3 4 5 6 7 8 | |
| 1 1 1 | 1 |
| 1 1 1 1 1 1 1 1 | 1 |
| 1 1 | 2 |

未使用のビット位置は、01 レベルの項目の再定義を除いて使用できません。

- 集団項目に含まれる場合

集団項目内のブーリアンデータの先頭ビットは、バイトアラインされ、それ以後はブーリアンデータ以外のデータが出現するか、もしくは集団項目の終端に到達するまではビットアラインで配置されます。

例）

```
01  GRP1
  02  BIT-DA1  PIC  11  USAGE BIT.
  02  BIT-DA2  PIC  1  USAGE BIT.
  *>  暗黙の無名ビットデータ項目
  02  BIT-DA3  PIC X（3）
  02  BIT-DA4  PIC  1  USAGE  BIT.
  *>  暗黙の無名ビットデータ項目
  02  GRP-2
    03  BIT-DA5  PIC  1  USAGE BIT.
    03  BIT-DA6  PIC  1(4)  USAGE BIT.
  *>  暗黙の無名ビットデータ項目
```

| ビット位置 | バイト数 |
|---|---|
| 1 2 3 4 5 6 7 8 | 1～6 |
| 1 1 | 1 |
| 1 | 1 |
| 1 1 1 1 1 | 1 |
| | 2～4 |
| 1 | 5 |
| 1 1 1 1 1 1 1 | 5 |
| | 6 |
| 1 | 6 |
| 1 1 1 1 | 6 |
| 1 1 1 | 6 |

前記例に対して、GRP-2 に GROUP-USAGE-BIT 句を付加した場合に次のようなアラインメントに変更することが可能です。

| 例） | ビット位置 | バイト数 |
|---|---|---|
| 01 GRP1 | 1 2 3 4 5 6 7 8 | 1～5 |
| … | | |
| 02 BIT-DA4 PIC 1 USAGE BIT. | 1 | 5 |
| 02 GRP-2 GROUP-USAGE-BIT. | | 5 |
| 03 BIT-DA5 PIC 1 USAGE BIT. | 1 | 5 |
| 03 BIT-DA6 PIC 1(4) USAGE BIT. | 1 1 1 1 | 5 |
| *> 暗黙の無名ビットデータ項目 | 1 1 | 5 |

# 4 章 漢字等の多オクテット文字機能

この章では、COBOL における漢字等の多オクテット文字の扱いについて説明します。
国際符号化文字集合(UCS)等の多オクテット文字を標準的にサポートするため、データ定義などに拡張がされています。

## 4.1 文字集合

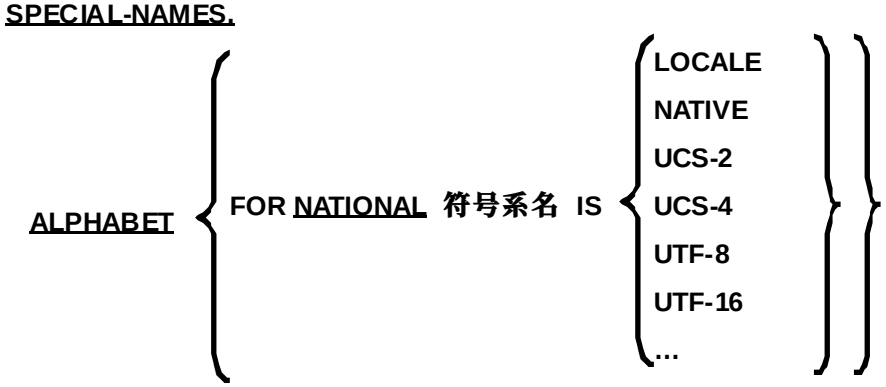
COBOL における文字集合には、COBOL 文字一覧、計算機コード化文字集合、そして符号系の３種類があります。
COBOL 文字一覧は、言語構文を定義するために使用される文字の一組で、従来の COBOL 言語と同様です。
計算機コード化文字集合は、実行時に内部処理用のデータを表現するために使用され、ソースコード中ではコメントや１６進表現でない定数の内容が、計算機コード化文字集合で表現されます。この文字集合は、USAGE DISPLAY と記述された計算機英数字コード化文字集合と、USAGE NATIONAL と記述された計算機国別文字コード化文字集合とがあります。
符号系は、名前のついた、コード化文字集合や照会順序の仕様です。特殊名段落で符号系を命名したり、利用者定義のコード化文字集合や照会順序を指定することができます。

### 4.1.1 特殊名段落（SPECIAL-NAMES）

特殊名段落では符号系名をコード化文字集合や照会順序に関係付けることができます。このとき、国別文字コード化文字集合として、UCS-2、UCS-4、UTF-8、UTF-16 を指定することができます。

#### 4.1.1.1 符号系名句（ALPHABET）

ある名前を、コード化文字集合か照会順序、あるいはその両方に関連付けることができます。符号系名を実行用計算機段落の PROGRAM　COLLATING　SEQUENCE 句において、あるいは SORT 文や MERGE 文の COLLATING　SEQUENCE 指定において参照した場合、符号系名は照会順序を表します。字類条件中や、データ部の CLASS 句、SYMBOLIC CHARACTER 句、あるいはファイル記述句の CODE-SET 句において参照した場合には、コード化文字集合を表します。

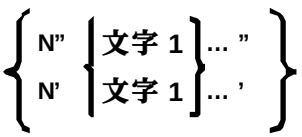



# 4.2 データ部

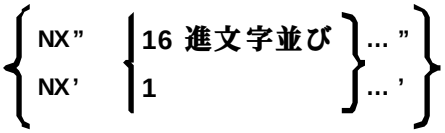
データをできるだけ機種によらないものにするために、データの特性や性質を記述するのに、「図形文字」を使用して表現される標準データ形式を用います。
実行時に計算機上で別なコード化文字集合が有効となる場合、定数は計算機コード化文字集合に変換されます。
ファイル入出力では、論理レコードと物理単位の間で転記が行われる際に、CODE-SET 句が指定されている場合、必要とされる変換、埋め文字の追加・削除が行われます。

## 4.2.1 国別文字定数

書き方1




書き方 2



### 4.2.2 データ記述項

#### 4.2.2.1 PICTURE 句

国別文字項目を定義する場合には PICTURE 文字列として記号「N」だけを使用します。
国別文字編集項目を定義するには、PICTURE 文字列は記号「N」、「B」、「0」、「/」の組み合わせで記述し、「N」が少なくとも 1 つと「B」、「0」、「/」のどれかが共に含まれている必要があります。

#### 4.2.2.2 USAGE 句

計算機の記憶域内でデータ項目を表すために国別文字コード化文字集合を使用することおよび、データ項目を文字境界に桁詰めすることを指定するために、USAGE　NATIONAL を指定します。桁詰めにより、暗黙の FILLER ビット位置が生成されることがあります。
国別文字は、計算機の記憶域中で表現される場合に、計算機の英数字文字集合の文字の大きさ以上の、一定の大きさを持つ文字表現でなければなりません。

## 4.3 組込み関数

国別文字コードを扱うために次の組込み関数が用意されます。

- CHAR-NATIONAL
- DISPLAY-OF
- NATIONAL-OF

また、次の組込み関数で引数として国別文字コードが使用できます。

- BYTE-LENGTH
- LENGTH
- LOCALE-COMPARE
- LOCALE-DATE
- LOCALE-TIME
- LOWER-CASE, UPPER-CASE
- MAX, MIN
- NUMVAL, NUMVAL-C, NUMVAL-F
- ORD, ORD-MAX, ORD-MIN
- REVERSE
- STANDARD-COMPARE
- TEST-NUMVAL, TEST-NUMVAL-C, TEST-NUMVAL-F

### 4.3.1 CHAR-NATIONAL 関数

**FUNCTION CHAR-NATIONAL ( 引数 )**

**国別文字プログラム照会順序中において、引数の値に等しい順序位置にある国別文字の 1 文字を返します。**

### 4.3.2 DISPLAY-OF 関数

**FUNCTION DISPLAY-OF ( 引数 1 [ 引数 2 ] )**

**引数中の国別文字を混在英数字外部データ形式へと変換した値を含んでいる文字列を返します。**

### 4.3.3 NATIONAL-OF 関数

**FUNCTION NATIONAL-OF ( 引数 1 [ 引数 2 ] )**

**引数中の文字を国別文字の内部表現に変換した値を含んでいる文字列を返します。**

# 5 章 浮動小数点データ操作機能

COBOL2002 では、浮動小数点表現を使用した数字データ項目が追加され、従来からの固定小数点データと同じように使用することが出来るようになりました。

## 5.1 浮動小数点数字定数

浮動小数点数字定数は、2 個の固定小数点数字定数を空白なしに英字の「E」で挟んで構成されます。「E」の左側の定数は、有効桁部を表します。この有効桁部は、符号が付いてもよく、小数点を含んでいる必要があります。また、この有効桁部の長さは 1 桁から 31 桁までででなくてはなりません。「E」の右側の定数は、10 を基数にした指数部を表します。この指数部は、符号が付いてもよく、小数点を含んでいてはいけません。また、この指数部の長さは 1 桁から 3 桁までででなくてはなりません。

例：1.2345E-2, -0.3E12

## 5.2 浮動小数点数字項目

浮動小数点数字項目は、2 個の固定小数点数字項目の PICTURE 文字列を空白なしに英字の「E」で挟んで構成されます。「E」の左側の数字項目は、有効桁部を表します。「E」の右側の数字項目は、10 を基数にした指数部を表します。この指数部は、「+9」,「+99」,「+999」,または「+9(n)」（ここで n = 1,2,3）でなくてはなりません。

例：01 FLOAT-01 PICTURE IS +9（1）.9(4)E+9（2）

受取り側データ項目が浮動小数点数字編集項目である場合は、編集される値が 0 でなければ、浮動小数点数字編集項目に転記されるデータは、最左端の桁が 0 にならないように桁寄せされます。

## 5.3 USAGE 句

浮動小数点データに関連して、USAGE 句に次の形式が追加されました。

- FLOAT-SHORT
- FLOAT-LONG
- FLOAT-EXTENDED

これらのフィールドの大きさと許容値の範囲は処理系により定義されますが、FLOAT-SHORT のデータ項目で保持できる数値は、FLOAT-LONG のデータ項目でも表現でき、FLOAT-LONG のデータ項目で保持できる数値は、FLOAT-EXTENDED のデータ項目でも表現できる必要があります。

## 5.4 組込み関数

浮動小数点データに関連して、組込み関数に次の関数が追加されました。

- **FUNCTION NUMVAL-F ( 引数 1 )**

引数 1 で指定された文字列が表現する浮動小数点数値を返します。先行、後続、中間の空白列は無視されます。

- **FUNCTION TEST-NUMVAL-F ( 引数 1 )**

引数 1 の内容が NUMVAL-F 関数の引数 1 に対する仕様に適合することを検証します。適合する場合は、0 を返します。適合しない場合は、誤りのある最初の文字位置を返します。

# 6 章 ポインタ項目とアドレス付け機能

COBOL2002 では、ポインタという新しいデータ・クラスが導入されました。その背景として、従来のビジネス・アプリケーションに加えてシステム・プログラミングを含む様々な用途に COBOL が使用されるようになったこと、また C や C++のようなシステム記述言語との親和性が求められるようになったことが挙げられます。ポインタにはデータアドレスを扱うデータポインタとプログラムアドレスを扱うプログラムポインタの 2 種類があります。
アドレス、ポインタを不用意に使用するとメンテナンス困難なプログラムになる可能性もありますので、使用する際には注意が必要です。

## 6.1 データアドレスとデータポインタ

データアドレスはデータ項目の所在を一意に特定するものであり、次の形式で記述されるデータアドレス一意名によって参照することができます。

ADDRESS OF 一意名 1

データアドレスはデータポインタに保管することができます。データアドレスとデータポインタは別プログラムに渡すことが可能です。またデータポインタを別プログラムから受け取ることが可能です。

- 制限つきデータポインタ
  制限つきデータポインタはある特定されたタイプのデータ項目のアドレスのみを保持することができます。制限つきデータポインタは、データを別の型で扱ってしまうというミスを未然に防ぎ、型の安全性を保証するために有効です。

- **データポインタの例**

  **例１）**

  **次のプロトタイプを持つレコードへのポインタを返すプログラム”Get-next-rec”が存在すると仮定します。**

```
Program-id Get-next-record is prototype.
Data division.
Linkage section.
01 ptr1 usage pointer
Procedure division returning ptr1.
End program Get-next-recotd.
```

  **また、次のような宣言を持つクライアント・プログラムを仮定します。**

```
Repository.
   Program Get-next-record.
…
01 p usage pointer.
01 my-wreck based.
   02 name pic x(30).
   02 addr pic x(30).
```

  **このクライアント・プログラムからプロトタイプに記述された”Get-next-record”を呼ぶには手続き部に次のステートメントを記述します。**

```
Call Get-next-record returning p
```

  **ポインタ p にはレコードの場所が保管されているため、my-wreck を介してデータにアクセスすることができます。**

```
Set address of my-wreck to p
Move “SAM JONES” to name in my-wreck.
```

例２）

ポインタをパラメーターとするインターフェースへのアドレスの渡し方の例を挙げます。

データ部には次のように定義されていると仮定します。

```
01 p2 usage pointer.
01 data-record.
  02 ……
```

プログラム"Process-record"に data-record のアドレスを渡すためには、次のように記述します。

```
Set p2 to address of data-record
Call "Process-record" using p2
```

あるいは、ポインタを使わずに次のように記述することも可能です。

```
Call "Process-record" using address of data-record
```

# 6.2 プログラムアドレスとプログラムポインタ

プログラムアドレスはプログラムの所在を一意に特定するものであり、次の形式で記述されるプログラムアドレス一意名によって参照することができます。

ADDRESS OF PROGRAM { 一意名１ / 定数１ / プログラム原型名 }

プログラムポインタはプログラムアドレスを保管するためのデータ項目です。プログラムアドレスとプログラムポインタはプログラムを呼ぶために使用することができます。プログラムアドレスとプログラムポインタは別プログラムに渡すことが可能です。またプログラムポインタを別プログラムから受け取ることが可能です。

- 制限つきプログラムポインタ

  上記においてプログラム原型名１が指定された場合、プログラムアドレス一意名はプログラム原型名１と同じシグニチャを持つプログラムのアドレスのみを保持する制限つきプログラムポインタとなります。

# 7 章 利用者定義のデータ型機能

COBOL では、レベル番号、項目の名前、PICTURE や USAGE 句の様なデータ属性を指定して、データ項目の構造や形式を表現します。従来の COBOL では、同じ構造のデータ項目を複数定義する場合、レベル番号、項目の名前、各種属性という一連の記述を繰り返し指定する必要がありました。COBOL2002 では、利用者が自由にデータ型(データ構造の雛型)を定義して、そのデータ型を参照することで、同じ構造のデータ項目を簡単に記述することができます。

## 7.1 データ型と型名

データ型とは、あるデータ項目とその従属項目の形式を全て含む、データ構造の雛型です。
型名とは、データ型を識別するために利用者がその雛型に付与する名前です。

利用者定義のデータ型や型名は、データ記述項にTYPEDEF句で宣言します。データ型の記述は、TYPEDEF句を指定すること以外は、データ項目の宣言と同様に、利用者が自由に記述します(データ型の宣言だけでは、記憶域は確保されません)。

「利用者定義のデータ型」をデータ項目の記述に書く場合は、データ記述項にTYPE句(型名を伴う)を指定します。

データ型を定義し、利用する例を次に示します。

```
01  FEATURE  TYPEDEF.      *> この記述で型名  FEATURE を定義する
  02  FEATURE-NAME  PIC  X(15)  OCCURS  10.
01  EQUIPMENT.             *>
  02  EQUIPMENT-ID  OCCURS  100.
    03     EQUIPMENT-LIST  TYPE  FEATURE.
                           *> 型名  FEATURE を利用する
```

前記の例は、次のレコード記述と同じです。

```
01  EQUIPMENT.             *>
  02  EQUIPMENT-ID  OCCURS  100.
    03  QUIPMENT-LIST  PIC  X(15)  OCCURS  10.
```

データ型の形式は、その型を構成する基本データ項目の相対位置と長さ、およびこれらの基本項目のそれぞれに指定された PICUTURE, USAGE, SIGN, SYNCHRONIZED, JUSTIFIED及びBLANK句によって決まります。

型名を使って宣言したデータ項目は、型名宣言のTYPEDEF句にSTRONG指定があるか否かによって、次の二つに分類されます。弱く型付けされた(基本・集団)項目

- 強く型付けされた(集団)項目

強く型付けされた項目とは、集団項目のデータ内容の妥当性を確保するため、COBOL2002で新たに導入された仕組みです。COBOLの集団項目は英数字項目として扱われるため、送り出し項目と受け取り項目のデータ構造が一致しなくても転記できました。これは、柔軟な転記の記述が許される反面、集団項目中の各基本項目に「不正なデータ」を格納する原因にもなっていました。この様な誤りを防ぐため、強く型付けされた項目という仕組みが導入されました。ただし、基本項目は強く型付けすることができません。

## 7.1.1 弱く型付けされた項目

型名を使って宣言したデータ項目のうちで強く型付けされていない項目のことを、弱く型付けされた項目と呼びます。弱く型付けされた項目とは、STRONG 指定のない型宣言を参照するものか、STRONG 指定が有効でない型宣言に従属するものです。弱く型付けされた項目は、基本項目であることも、集団項目のこともあります。

弱く型付けされた項目は、指定された型名からそのデータ構造が決まることを除き、型付けされていない項目と同様に使用できます。ですから、TYPEDEF 句で指定する型宣言は、以降で、一連のデータ記述の組を型名という一単語で記述するための手段といえます。

## 7.1.2 強く型付けされた項目

強く型付けされた項目とは、STRONG 指定付きの型宣言を参照するものか、STRONG 指定付きの型宣言で記述された集団項目だけです。

集団項目中の基本データ項目に格納する内容の整合性を保つため、強く型付けされた(集団項目)は通常の集団項目と比べて許される操作が非常に制限されます。強く型付けされた集団項目中の基本データ項目に格納する内容の整合性を損なう可能性のある操作は全て禁止されます。

＜強く型付けされた項目に関する主な制約＞

- 型の異なる集団項目からの転記は許されない。
- 強く型付けされた項目のデータ記述項に VALUE 句は書けない。
- 強く型付けされた項目やその従属項目を、再定義や再命名してはならない。
- 一部の例外を除いて、部分参照してはならない。
- 強く型付けされた項目を受け取り作用対象として参照できるのは、次の場合のみである。
  - （a） 仮引数や返却項目
  - （b） INITIALIZE文
  - （c） MOVE文
  - （d） READ文のINTO指定
  - （e） RELEASE文のFROM指定
  - （f） RETURN文のINTO指定
  - （g） REWRITE文のFROM指定
  - （h） WRITE文のFROM指定

# 7.2 TYPE 句と TYPEDEF 句

利用者定義のデータ型機能のため、データ部の句として TYPE 句と TYPEDEF 句の二つが導入されました。ここではそれぞれの概要を説明します。

## 7.1.1 TYPE 句

データ記述項中の TYPE 句は、型名で示すデータ型の宣言をこの記述項のデータ記述に使うことを表します。

### 7.1.1.1 書き方

TYPE TO 型名１

### 7.1.1.2 主な規則

- TYPE 句は、この記述項の左辺のデータ記述が、型名１によって指定されることを表す。TYPE 句の効果は、型名１で識別されるデータ記述をこの TYPE 句の代わりに書いたのと同じである。

## 7.1.1 TYPEDEF 句

TYPEDEF 句は、このデータ記述がデータ型の宣言であることを表す。

### 7.1.1.1 書き方

IS TYPEDEF [ STRONG ]

### 7.1.1.2 主な規則

- TYPEDEF 句を指定した場合、このデータ記述項はデータ型の宣言となる。この記述項に指定されたデータ名は型名となる。従属するデータ記述項や、条件名記述項、及び RENAME 句は、このデータ型の型宣言の一部となる。これらの従属項で記述された項目のデータ名は、この型名を使用して定義される集団項目の従属項目としてだけ、参照してもよい。その様な集団項目が複数存在する場合、集団項目名での修飾が必要となる。
- 型宣言には、それに関連する記憶域というものがない。
- 記述項の左辺が基本項目であるとき、STRONG 指定を書いてはならない。

## 7.3 例題

例題として、次の集団項目「人事レコード１」と同じ構造の集団項目「人事レコード２」を記述する場合を考えます。

```
01  人事レコード１.
    05  従業員コード      PIC X(8).
    05  氏名              PIC N(10).
    05  生年月日.
        10  年                  PIC X(4).
        10  月                  PIC X（２）.
        10  日                  PIC X（２）.
```

従来のCOBOLで同じ構造の集団項目「人事レコード２」を記述するには、「人事レコード１」と同様に、レベル番号、項目の名前、各種属性を従属する項目のものを含めて繰り返し記述する必要がありました。

```
01  人事レコード２.
    05  従業員コード      PIC X(8).
    05  氏名              PIC N(10).
    05  生年月日.
        10  年                  PIC X(4).
        10  月                  PIC X（２）.
        10  日                  PIC X（２）.
```

上記の様に何度も同様の記述を書く必要があるため、生産性が悪く、プログラミング時点で誤りを作り込み易いという欠点がありました。又、この様に繰り返し記述したデータ項目の構造を変更する際には、ﾌﾟﾛｸﾞﾗﾑ中で全く別々に宣言された同じ構造のデータ項目を洗い出し、それぞれの宣言に対する修正の要否を検討する必要があり、保守性も悪くなっていました。

COBOL2002では、利用者定義のデータ型をあらかじめ宣言しておいて、同じデータ構造のデータ記述に繰り返し利用することで、上記に示した生産性や保守性の悪さを改善できます。先に述べた「人事レコード１」と「人事レコード２」を、利用者定義のデータ型(人事情報)を使って宣言すると次の様になります。

```
01  人事情報 IS TYPEDEF.
    05  従業員コード        PIC X(8).
    05  氏名                PIC N(10).
    05  生年月日.
        10  年                  PIC X(4).
        10  月                  PIC X（２）.
        10  日                  PIC X（２）.
*> 上記では、「人事情報」という利用者定義のデータ型を宣言しています。
*> 「人事情報」というデータ型の宣言だけで、記憶域は取られていません。
    01  人事レコード１ TYPE  TO 人事情報.
*> 型名「人事情報」を使い、「人事レコード1」を宣言(記憶域の確保)しています。
    01  人事レコード２ TYPE  TO 人事情報.
*> 「人事レコード２」も「人事レコード１」と同様に宣言しています。
```

利用者定義のデータ型を利用する利点を整理すると、次の通りです。

- 同じ構造のデータ項目を簡単に宣言できます。これにより、複雑なデータ構造を何度も記述する必要がないため生産性が向上し、誤りを作りこみにくくなります。
- 同じ構造のデータ項目を同じ型名に統一しておくことで、次の様な副次効果を得ることもできます。
    (ア) 型名を目印に同じ構造のデータ項目を検索できるため、データ型の構造変更による影響範囲の調査が容易になります。
    (イ) データ型の宣言を修正することで、そのデータ型の記述を参照する全てのデータ項目の構造を容易に変更できます。
- 強く型付けされた項目を利用して、集団項目に従属するデータ項目のデータの整合性を確保し易くできます。

# 8 章 利用者定義の関数機能

従来の COBOL でも組込み関数機能が提供されていますが、COBOL2002 では、さらに利用者が関数を定義し、利用することができるようになります。
利用者定義関数を使うには、
　関数本体の定義
　関数リポジトリ宣言
　関数一意名
が必要です。関数を参照するための「関数一意名」から説明していきます。

## 8.1 関数一意名

COBOL では、関数の参照は一意名参照と同じように扱われます。一意名は、項目を一意に参照するための COBOL 構文の要素を示します。関数一意名は、関数で定義された何らかの処理の結果がデータ項目に格納された後に、そのデータ項目を参照することをあらわします。従って厳密には COBOL の関数一意名には、a) 関数定義を呼び出して処理を実行し結果を一時的データ項目に格納する部分と、b) そのデータ項目の値を参照する部分の二つの部分があります。ただ　一般には関数の戻り値を参照していると考えて問題ありません。

関数一意名の構文は利用者定義関数は従来の組込み関数と同じ形式です。

　[ FUNCTION ]　関数名 1 [　( 引数 1　)　]

利用者定義関数の参照で、組み込み関数の参照と異なるのは、次の点です。
（1） 関数名 1 は、利用者が定義したものであること
（2） 引数 1 は、同じく利用者の定義した並びであること
必須語 FUNCTION は、REPOSITORY 段落に宣言した関数名の前では省略できます。なお、上記の 関数名 1 は、正確には、環境部構成節リポジトリ段落で宣言した関数プロトタイプ名を指定する必要があります。

関数の面白みは、それが式の中で使えることです。プログラムの処理の中に、結果として値を返す処理があれば、その処理は関数の候補です。たとえば、4 桁年を引数にとって、その年が閏年か否かを判定するという処理は，典型的な関数です。なぜなら，この処理の

結果で欲しいのは、Yes か No かであり、このような関数参照は、例えば、IF 文の判定に使えます。

4 桁年の変数 今年 を引数にとり、文字列 "Y" か "N" を返却するユーザ定義関数 閏年 を定義したとします。

```
IF 閏年(今年) = "Y" THEN DISPLAY 366.
```

関数 閏年 の返却値を関数一意名で参照し、ことを判定しています。

# 8.2 利用者定義関数とプログラムとの違い

COBOL2002 では、PROGRAM-ID で始まるプログラムにも、RETURNING 指定が書けるようになります。RETURNING 指定は、いわば、関数の戻り値と同じです。プログラムの RETURNING 指定も、利用者定義関数も、C 等の他の言語とのインターフェースが取り易いようにと導入されたものです。どちらも結果を返却するものですので機能が重複しています。

プログラムは CALL 文で呼び出すものですから、関数のように一意名として式や文の一部として使うことはできません。しかし前述のように、関数も戻り値の格納されたデータ項目の値参照になりますので、記述の冗長性さえ気にしなければ、CALL 文と一時変数の組み合わせで、関数と同じ動作を記述できます。

# 8.3 利用者関数の定義

では、関数を定義してみましょう。

```
[ IDENTIFICATION DIVISION. ]
FUNCTION-ID. 利用者関数名 1 [ AS 定数 ].
[ オプション段落 ]
[ 環境部 ]
[ データ部 ]
[ 手続き部 ]
END FUNCTION 利用者関数名 1.
```

ほぼ、プログラムの定義と同じです。異なるのは PROGRAM-ID が FUNCTION-ID に変更され、END FUNCTION が省略できない部分のみです。さらに、手続き部見出しには、RETURNING 指定が必要です。関数は必ず値を返却する必要があります。

PROCEDURE DIVISION [ USING データ名 1 ... ] RETURNING データ名 2

USING で指定するデータ名 1 は仮引数であり、省略しましたが、BY REFERENCE 及び BY VALUE が指定できます。データ名 1、データ名 2 とも、連絡節に記述しておく必要があります。

前述の閏年関数は次のようになります(処理本体も省略しました)。

```
IDENTIFICATION DIVISION.
FUNCTION-ID. 閏年.
     :
DATA DIVISION.
LINKAGE SECTION.
01 西暦年 PIC 9(4).
01 結果 PIC X.
PROCEDURE DIVISION USING 西暦年 RETURNING 結果.
     :
END FUNCTION 閏年.
```

関数の手続き部の処理で、関数から戻る直前にデータ項目 結果 に設定された値が、この関数の戻り値になります。また、データ項目 結果 の型が、この関数の型であり、この関数を参照する一意名の型になります。

# 8.4 利用者定義関数を使うための宣言

利用者定義関数を使うには、環境部の構成節のリポジトリ段落に、あらかじめ関数名を宣言しておかなければなりません。

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
FUNCTION 関数プロトタイプ名 1 [ AS 定数 ]

この環境部を含むプログラムや関数内で、この関数プロトタイプ名で示される関数を利用することができます。関数プロトタイプ名は、同じ翻訳グループに同名の関数定義があれば、それが実態になります。組込み関数と同じように使える関数を、利用者が自由に定義できるようになると理解してください。

# 9 章 ファイルの共用と排他制御の機能

従来の COBOL では、同時に動作する複数のプログラムから、同じのファイルを操作することができませんでした。
COBOL2002 では、ファイル定義単位にファイル共用及び排他制御という新たに導入された機能によって、同時に動作する複数のプログラムから同じファイルを操作することが可能になります。
本機能を利用することで、同時に動作する複数のプログラムから同一ファァイルにアクセスする際、ファイル単位やレコード単位でのアクセスについてデータの保全性を高めることが可能となります。

## 9.1 ファイル共用とレコードロック

ファイル共用とは、複数のプログラムからの同一ファイルへのアクセスを制御する機能です。ファイル共有は、「排他モード」と「共用モード」の２種類に大別されます。
レコードロックとは、共用モードでアクセスされるファイル上のレコードについて、複数のプログラムからレコードの排他制御を行う機能です。
これらの機能を使うことにより、１つのプログラムでのみファイルアクセスを許す（排他モード）業務プログラムの開発や、同時に複数のプログラムで１つのファイルの共用（共用モード）を行う業務プログラムの開発が可能となります。

### 9.1.1 ファイル共用

ファイル共用とは、同時に動作する複数のプログラムから同一ファイルへのアクセスを制御するものです。ファイル共用のモードには、「共用モード」、「読込み専用共用モード」、及び「排他モード」の３種のモードがあります。

「共用モード」とは、同時に動作する複数のプログラムからファイルをアクセスする際に、
各々のプログラムの OPEN モードとして INPUT モード、I-O モード、EXTEND モードの組合わせで共用できるモードです。

「読込み専用共用モード」とは、同時に動作する複数プログラムからファイルをアクセスする際に、OPEN モードとして INPUT モードの場合に限って共用して使用できるモードです。

「排他モード」とは、１つのプログラムでアクセス中は、他のプログラムとの共用を行わ

ないモードです。一旦 OPEN を行うと CLOSE するまでの間、「ファイル共用矛盾」により他のプログラムからのアクセスを拒否されます。

共用ファイルへのアクセスは、共用モードと OPEN モードの組み合わせが正しく、かつ、別プログラムからの同一ファイルに関連するファイル定義との組み合わせにおいても、ファイル共用が許された記述でなければなりません。

### 9.1.2 レコードロック

レコードロック（record locking）とは、同時に動作する複数のプログラムより、同一ファイルの同一レコードへのアクセスを制御する機能です。レコードロック機能は、共用モードでのアクセスが許されたファイルに対してのみ適用可能です。ある入出力文によってあるレコードを参照すると、そのレコードはロックされ、他のプログラムからそのレコードへのアクセスは IGNORING LOCK 指定のある READ 文によるアクセスを除いて、「レコード共用矛盾」が発生してアクセスが拒否されます。
レコードロックを行うには、各々の入出力文でレコードのロック管理を意識することなく行う「自動ロック」、及び各々の入出力文で意識的にレコードのロック管理を行う「手動ロック」の２種類があります。

「自動ロック」は、READ 文が実行された場合、読込んだレコードに対して自動的にレコードロックが行われます。

「手動ロック」は、明示的に LOCK 指定が書かれた入出力文が実行された場合、処理対象となるレコードに対してレコードロックが行われます。

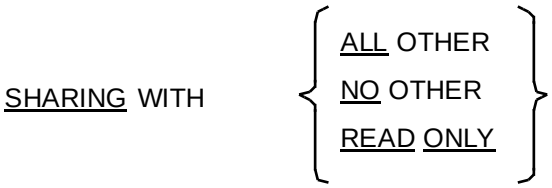
また、同時に保持できる最大レコードロック数の管理方式には、「単一レコードロック」と「複数レコードロック」の２種類があります。

「単一レコードロック」は、１つのファイル定義中でロック可能なレコードは唯一つになります。 例えば、READ 文の実行が成功した場合に読込んだレコードにレコードロックが行われ、以前の READ 文でロックされていたレコードは自動的に解放されます。

「複数レコードロック」は、１つのファイル定義中で複数個のレコードがロック可能です。
例えば、READ 文の実行が成功した場合に読込んだレコードにレコードロックが行われても、以前の READ 文で既にロックされているレコードも解放されずにレコードロックされたままとなります。

# 9.2 SHARING によるファイル共用の指定

ファイル共用はファイル管理記述項の SHARING 句、又は SHARING 指定のある OPEN 文で指定します。両方の指定がある場合は OPEN 文の指定を優先します。指定可能なファイル共用には、次の３種類があります。




（１）NO OTHER は「排他モード」を表し、ファイルへのアクセスが排他的であることを指定します。指定したファイルが、他のプログラムにより開かれている場合、指定したファイルの OPEN は不成功となります。あるプログラムによって OPEN が成功した場合、CLOSE されるまでの間、他のプログラムによって OPEN することができません。NO OTHER では、レコードロックは有効となりません。

（２）READ ONLY は「読取り専用モード」を表し、指定したファイルへのアクセスが他のプログラムとの同時アクセスを入力モードのみに制限します。他のプログラムで入力以外のモードで開かれている場合、指定したファイルの OPEN は不成功となります。指定したファイルの OPEN が成功した場合、他のプログラムからは、入力でないモードでは OPEN することができません。READ ONLY では、レコードロックは有効となります。

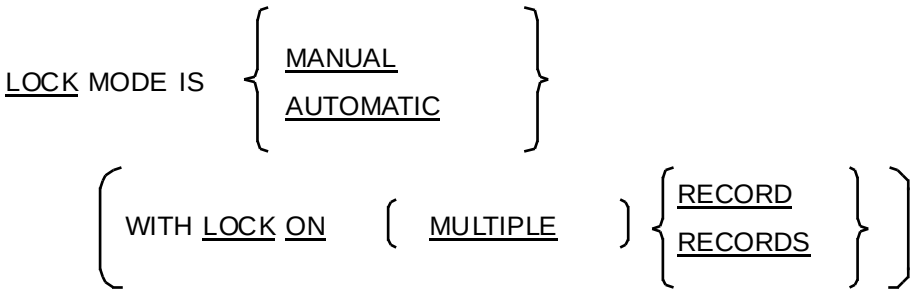
（３）ALL OTHER は「他の全てとの共用モード」を表し、指定したファイルへのアクセスが入力モード、入出力モード、又は拡張モードの場合において、同じ OPEN モードを指定した他のプログラムで、ファイル共用の制限事項に従って同時アクセスを許可するものです。ALL OTHER では、レコードロックは有効となります。

以上のように、ファイル共用は先行の実行単位により既に OPEN されたファイルに対して、後行の実行単位で同一ファイルに OPEN を実行した場合、組み合わせによって OPEN が成功したり不成功となったりします。ファイル共用モードの組み合わせにより、OPEN 文が成功するか否かを次の表に示します。

| 後行 OPEN 要求 / 先行 OPEN 要求 | | NO OTHER | READ ONLY | | ALL OTHER | |
|---|---|---|---|---|---|---|
| | | EXTEND I-O INPUT OUTPUT | EXTEND I-O OUTPUT | INPUT | EXTEND I-O OUTPUT | INPUT |
| NO OTHER | EXTEND I-O INPUT OUTPUT | 不成功 | 不成功 | 不成功 | 不成功 | 不成功 |
| READ ONLY | EXTEND I-O | 不成功 | 不成功 | 不成功 | 不成功 | 正常 OPEN |
| | INPUT | 不成功 | 不成功 | 正常 OPEN | 不成功 | 正常 OPEN |
| | OUTPUT | 不成功 | 不成功 | 不成功 | 不成功 | 不成功 |
| ALL OTHER | EXTEND I-O | 不成功 | 不成功 | 不成功 | 正常 OPEN | 正常 OPEN |
| | INPUT | 不成功 | 正常 OPEN | 正常 OPEN | 正常 OPEN | 正常 OPEN |
| | OUTPUT | 不成功 | 不成功 | 不成功 | 不成功 | 不成功 |

# 9.3 LOCK 句によるレコードロックの指定

レコードロックの方式はファイル管理記述項の LOCK 句で指定します。




1. MANUAL 指定があるとロックモードは手動になり、LOCK 指定が入出力文に明示的に指定されている場合に限り、レコードはロックされます。
2. AUTOMATIC 指定があるとロックモードは自動になり、次の場合にレコードはロックされます。
   （a）IGNORING 指定も NOLOCK 指定もない READ 文が実行された場合
   （b）LOCK 指定のある REWRITE 文は WRITE 文が実行された場合

3. MULTIPLE 指定がない場合、単一レコードロックとなります。単一レコードロックの場合、ある時点に１つのファイル定義でロックできるレコードは唯ひとつに限られます。レコードをロックしたある文の実行が成功すると、そのファイル定義に対してそれ以前にロックされた、そのファイル中のあらゆるレコードロックが解放されます。
4. MULTIPLE 指定がある場合、複数レコードロックとなります。複数レコードロックの場合、１つのファイル定義でロックされたレコードを１つ以上保持することができます。

# 9.4 RETRY 指定 による繰返し排他制御の指定

RETRY 指定は、ファイル又はレコードが既にロックされた状態であっても、排他制御を続けるかを指定します。他のプログラムによって既にロックされたファイル又はレコードにアクセスする場合は「ファイル共用矛盾」又は「レコード共用矛盾」が発生するため、通常プログラムにより繰返しアクセスを行います。RETRY 指定は、共用矛盾が発生したファイル又はレコードに対して繰返しアクセスを自動的に行い、プログラムの処理を軽減します。

自動的な繰返し処理の指定方法には、「再試行回数」の指定、「処理中断時間」の指定、「無条件繰返し」の指定の３種類があります。

```
RETRY { 算術式 1 TIMES
      | FOR 算術式 2 SECONDS
      | FOREVER }
```

1. 算術式 1 は、ロックされている資源のアクセスに失敗してから、入出力操作の要求を完了させるまでの再試行回数を指定します。
2. 算術式 2 は、ロックされている資源のアクセスに失敗してから、処理中断期間の秒数を指定します。処理中断の期間中、ロックされている資源のアクセス権を獲得する試みが繰り返されます。
3. FOREVER 指定は、ロックされている資源のアクセス権を獲得する試みを、入出力操作が完了するまで繰り返します。
4. 入出力操作の最初の試みが不成功となった理由がファイル共用矛盾又はレコード共用矛盾である場合、次を適用します。

   （ａ）RETRY 指定がない場合、あるいは算術式 1 や算術式 2 の評価の結果が有効でない値の場合、この文は不成功に終わり、ファイル共用矛盾やレコード共用矛盾が発生します。

（b）1～3 の規定通り入出力操作を完了させようして成功した場合には入出力操作文は成功に終わりますが、失敗した場合にはファイル共用矛盾又はレコード共用矛盾が発生します。

## 9.5 記述例

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUPUT SECTION.
FILE-CONTROL.
  SELECT USER-FILE ASSIGN TO SYS000
     ORGANIZATION IS SEQUENTIAL
     SHARING WITH ALL OTHER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・①
     LOCK MODE IS AUTOMATIC WITH LOCK ON RECORD. ・・・・・・②
DATA DIVISION.
FILE SECTION.
FD USER-FILE.
01 USER-REC PIC X(10).
PROCEDURE DIVISION.
  OPEN I-O USER-FILE.
  READ USER-FILE RETRY FOREVER ・・・・・・・・・・・・・・・・・・・・・・・・・・・・③
  MOVE ALL 'X' TO USER-REC.
  REWRITE USER-REC. ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・④
  CLOSE USER-FILE.
  STOP RUN.
```

①「他の全てとの共用モード」で OPEN を行います。制限事項に従う限りにおいて、他ファイル結合子との同時アクセスが許可されます。

②単一レコードロックを自動的に行います。つまり、このファイル中のレコードに対する READ 文で自動的にレコードロックを行い、以前の READ 文で獲得したレコードロックを自動的に解放します。

③レコードロックありで１レコードを読み込みます。但し、他のプログラムによってレコードロックされている場合においても、レコードロックが確保できるまで引き続き READ 文は実行されます。この READ 文の実行が成功するとレコードはロックされ、他のプログラムからのアクセスをレコード共用矛盾により拒否します。

④③で読み込んだレコードを更新します。REWRITE 文が終了すると、③で確保したレコードロックは解放されます。

# 10 章 画面処理機能

COBOL における画面処理機能は、規格化される以前からほとんどの COBOL 処理系によって実装されてきましたが、そのサポート範囲や文法はそれぞれの処理系により異なっていました。本章では COBOL 2002 が定める画面処理機能について説明します。
主な追加機能には以下のものがあります。

・データ部の画面節による画面レイアウトの定義
・画面のフィールド属性の定義
・ACCEPT・DISPLAY 文による入出力
・機能キーの検知
・カーソル位置の移動、検知

## 10.1 画面の概念

COBOL が入出力する「画面」とは、「長方形に配列された表示可能な文字位置」であって、「キーボードを介しての入出力機能を提供するもの」とされています。即ち、一般的に「キャラクタ端末」と呼ばれるハードウェアを想定しています。ウィンドウやマウス操作はその概念の中に含まれていません。
COBOL の画面を構成する要素について説明します。

### 10.1.1 フィールド

フィールドとは画面上の連続する文字位置をひとつ以上まとめたものであり、画面に対する入出力操作の基本単位になります。画面中のフィールドは順序付けられており、入力操作の順番を定めることができます。

### 10.1.2 属性

画面上の表示において目に見えるさまざまな属性を持たせることができます。このような属性は画面上の表示位置に関連付けられます。
属性には、背景色、前景色、高輝度、反転、点滅、必須入力、入力非表示などがある。

### 10.1.3 機能キー

一般にファンクションキーと呼ばれるものです。プログラムの実行時にオペレータが機能キーを押すと、そのキーに対応した機能キー番号がプログラムに返されます。

### 10.1.4 CRT 状態

端末入出力操作の状態をあらわす 2 文字の値です。ファイル入出力におけるファイル状態コードと同様に、ACCEPT 文による入力動作の後に値が設定され、動作の成功・失敗、
失敗時の原因、機能キーの値などをプログラムに返します。

### 10.1.5 カーソル

画面上でキーボード操作を可能とする文字位置を示す目に見える記号です。
オペレータがキーボード入力を行うとき、カーソルは常にその入力位置に見えています。
また、プログラムはオペレータに入力位置を指示するためにカーソルを所定のフィールドに移動することができます。
プログラムは、カーソルロケータと呼ばれる 6 文字のデータ項目を経由して、現在のカーソル位置を検知することもできます。

## 10.2 画面処理機能の環境部

環境部では、構成節の特殊名段落に、CURSOR 句と CRT　STATUS 句を記述します。

### 10.2.1 CURSOR 句の書き方

```
CURSOR　IS　データ名 2
```

データ名 2 は以下のどちらかの形式でなければなりません：

```
01　データ名 2　　PIC 9(6).
```

または、

```
01　データ名 2.
    05　X　　PIC 9(3).
    05　Y　　PIC 9(3).
```

ここで、レベル番号は任意です。また、後者の書き方で、従属項目 X、Y の名前も任意です。
データ名 2 のはじめの 3 桁はカーソル位置の画面中の縦座標、後ろの 3 桁は横座標を表します。この値は、ACCEPT 文の実行開始時点でのカーソルの位置を指定します。また ACCEPT 文終了時のカーソルの位置をプログラムに返します。

### 10.2.1 CRT STATUS 句の書き方

CRT STATUS IS データ名 3

データ名 3 は、長さが 4 文字の英数字項目でなければなりません。
この値は、ACCEPT 文の実行後に、以下の値に設定されます：

| | |
|---|---|
| 0000 | 送信キーの押下によって ACCEPT 文が正常に完了した。 |
| 1xxx | 機能キーの押下によって ACCEPT 文が正常に完了した。<br>xxx の数値は機能キーの番号になる。 |
| 2xxx | 文脈依存の機能キーの押下によって ACCEPT 文が正常に完了した。<br>xxx の数値は機能キーの番号になる。 |
| 8000 | 正しい画面位置に入力項目が無いため ACCEPT 文が不成功に終わった。 |
| 9xxx | COBOL 処理系の規定するエラーにより ACCEPT 文が不成功に終わった。<br>xxx の数値は COBOL 処理系によって規定されるエラー番号になる。 |

## 10.3 画面処理機能のデータ部

データ部には、作業場所節などと同様に画面節を記述します。画面節では端末入出力の際に表示される画面のレイアウトを記述します。画面節には、画面レコードと従属画面項目を記述します。

### 10.3.1 画面節の書き方

SCREEN SECTION.

{ 定数名記述項 / 画面記述項 } .

# 10.4 画面記述項

画面中のフィールドのさまざまな属性を記述し、ACCEPT 、DISPLAY 文で参照できるようにします。また、フィールドをデータ項目に関連付け、ACCEPT 文でのキーボード入力と DISPLAY 文での表示をそれぞれ指定されたデータ項目との間で転記されるようにします。

## 10.4.1 画面記述項の書き方

### 【書き方 1 (集団項目)】

```
レベル番号  [ 記述項名 ]
  [IS  GLOBAL ]
  [ LINE NUMBER IS  [ PLUS  ]{ 一意名 1 }  ]
                    [ MINUS ]{ 整数 1  }
  [{ COLUMN }NUMBER IS [ PLUS  ]{一意名 2 }]
   { COL    }          [ MINUS ]{ 整数 2 }
  [BLANK SCREEN ]
  [ 画面属性句群 ]
  [[SIGN IS] { LEADING  } [SEPARATE CHARACTER] ]
             { TRAILING }
  [FULL ]
  [AUTO ]
  [SECURE ]
  [REQUIRED ]
  [OCCURS  整数 5  TIMES ]
  [[USAGE IS] { DISPLAY  } ]
              { NATIONAL }
```

## 【書き方２(基本項目)】

```
レベル番号 [ 記述項名 ]
  [ IS  GLOBAL ]
  [ LINE NUMBER IS [ PLUS  ] { 一意名 1 } ]
                   [ MINUS ] { 整数 1  }
  [ { COLUMN } NUMBER IS [ PLUS  ] { 一意名 2 } ]
    { COL    }           [ MINUS ] { 整数 2  }
  [ BLANK { LINE   } ]
          { SCREEN }
  [ ERASE { END OF LINE    } ]
          { END OF SCREEN  }
          { EOL            }
          { EOS            }
  [ 画面属性句群 ]
  [ { PICTURE } IS 文字列 [ SIZE IS 整数 1  LOCALE  [ IS 呼び名 1 ] ] ]
    { PIC     }
  [ 送り元-宛て先-句 ]
  [ BLANK  WHEN  ZERO ]
  [ { JUSTIFIED } RIGHT ]
    { JUST      }
  [ [SIGN IS] { LEADING  } [ SEPARATE CHARACTER ] ]
              { TRAILING }
  [ FULL ]
  [ AUTO ]
  [ SECURE ]
  [ REQUIRED ]
  [ OCCURS  整数 5  TIMES ]
  [ [ USAGE IS ] { DISPLAY  } ]
                 { NATIONAL }
```

ここで、画面属性句とは：

```
[ BELL ]
[ BLINK ]
[ HIGHLIGHT  ]
[ LOWLIGHT   ]
[ REVERSE-VIDEO]
[ UNDERLINE ]
[ FOREGROUND-COLOR IS { 一意名 3 } ]
                      { 整数 3   }
[ BACKGROUND-COLOR IS { 一意名 4 } ]
                      { 整数 4   }
```

送り元-あて先-句とは：

```
[ USING 一意名 7                          ]
{ [ FROM { 一意名 5 } ]                   }
{        { 定数 1   }                     }
{   TO   一意名 6                         }
[ VALUE IS 定数 2                         ]
```

## 10.4.2 属性句

上記の一般形式に現れる、画面記述項に特有な句は、フィールドの属性や動作を指示するものです。以降にその意味をまとめます。

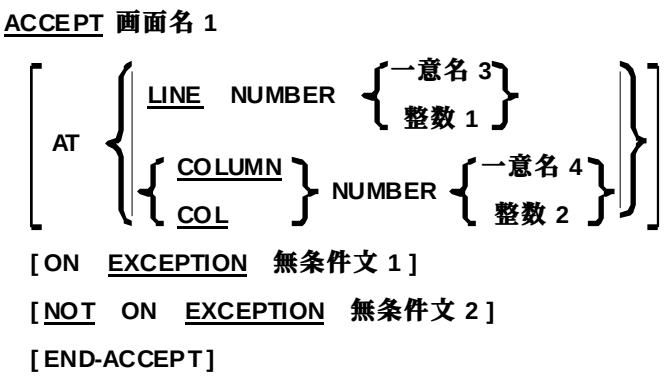
| | |
|---|---|
| AUTO | ACCEPT 文の実行時に次のフィールドへ自動的に移動する。 |
| BACKGROUND-COLOR | 背景色。整数は 0 から 7 までの範囲内で、色番号を指定する。 |
| BELL | DISPLAY 文での表示時に端末の警告音を鳴らす。 |
| BLANK | 項目が表示される前に行や画面全体をクリアする。 |
| BLINK | 文字を点滅させる。 |
| COLUMN | フィールドの横方向の画面座標を指定する。 |
| ERASE | 項目が表示される前に行や画面の一部をクリアする。 |
| FOREGROUND-COLOR | 前景色。整数は 0 から 7 までの範囲内で、色番号を指定する。 |

| | |
|---|---|
| FROM | 表示するデータの送りだし元を指定する。 |
| FULL | フィールドを完全に埋めるか完全に空白のどちらかを強制する |
| HIGHLIGHT | 文字を高輝度で表示する。 |
| LINE | フィールドの縦方向の画面座標を指定する。 |
| LOWLIGHT | 文字を低輝度で表示する。 |
| REQUIRED | フィールドに少なくとも一文字を入力することを強制する。 |
| REVERSE-VIDEO | 文字を反転表示する。 |
| SECURE | キーボードから入力されるデータが画面上に表示されない。 |
| TO | 入力するデータの受けとリ先を指定する。 |
| UNDERLINE | 文字を下線表示する。 |
| USING | FROM と TO の共用のデータ項目を指定する。 |

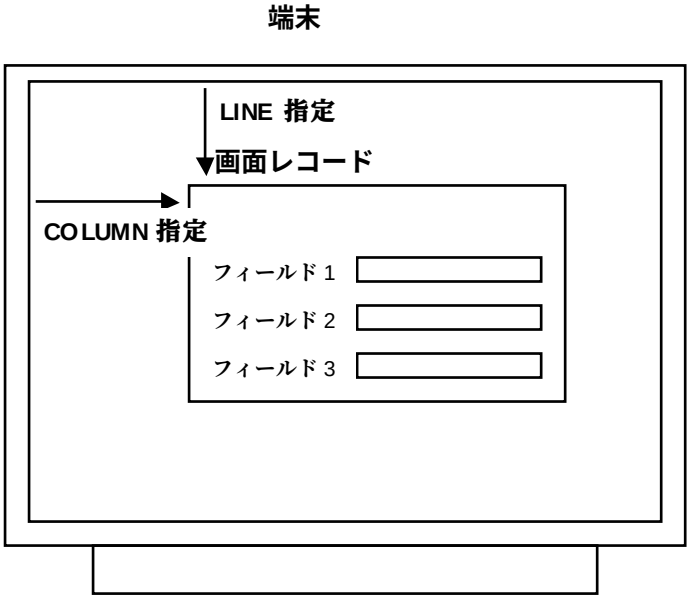
なお、COBOL 2002 の画面処理機能では、罫線に関する属性は仕様化されませんでした。

# 10.5 画面処理機能の手続き部

## 10.5.1 ACCEPT 文の書き方




画面名 1 は、画面節で宣言された基本項目であるか、または集団項目であって TO 句や USING 句を持つ基本項目を含むものでなければなりません。
LINE 指定、COLUMN 指定は、画面名 1 が表す画面レコードの、端末上の相対座標を指定します。画面名 1 が集団項目である場合、従属するフィールドは画面名 1 の開始位置からの相対位置になります。以降にその様子を図示します。




画面からの入力動作が正常に完了すると、もし CRT　STATUS 項目が宣言されていれば、そこに 0000、1xxx、2xxx のいずれかの値が設定され、NOT　ON EXCEPTION 指定が書かれていれば無条件文 2 に制御が移行します
画面からの入力動作が正常に完了しなかった場合、もし CRT　STATUS 項目が宣言されていれば、そこに 8xxx、9xxx のいずれかの値が設定され、ON　EXCEPTION 指定が書かれていれば無条件文 1 に制御が移行します

## 10.5.2 DISPLAY 文の書き方

```
DISPLAY 画面名 1
[ AT { LINE   NUMBER { 一意名 3 }
                     { 整数 1 }
     { { COLUMN } NUMBER { 一意名 4 }
     { { COL    }        { 整数 2 } } } ]
[ ON   EXCEPTION  無条件文 1 ]
[ NOT  ON  EXCEPTION  無条件文 2 ]
[ END-DISPLAY ]
```

画面名 1 は、画面節で宣言された画面項目です。
LINE 指定、COLUMN 指定は、ACCEPT 文と同様です。

# 11 章 例外割り込み処理機能

COBOL 2002 では、他の多くのオブジェクト指向言語がサポートしている例外割り込み処理機能が追加になりました。
伝統的な COBOL プログラムでは、プログラム全体のステップ数に対してエラーチェックの処理が占める割合はかなり多くなります。個々の処理ステップごとに、エラーの種別に応じてログファイルを残したり、コンソールへアラームを表示するプログラミングを記述すると、ステップ数が膨らんで生産性が低化します。さらに、プログラムの本来の処理がエラー処理に埋没してたいへん読みにくいプログラムになってしまいます。
例外割り込み処理機能は、現在の COBOL でもサポートされている、USE 文を持つ DECLARATIVES 手続きと同様に、各種のエラー発生時に宣言部分で記述されたエラー処理手続きが自動的に呼び出される機能です。

## 11.1 例外割り込み処理機能の概要

例外は、COBOL 文の実行時に発生するイベントです。COBOL 実行システムは、このイベントを検知すると、処理系によってまたはプログラマによって定義された所定の手続きに分岐します。これが例外割り込み機能です。
一般には、例外処理はオペレーティングシステムの機能として提供されています。UNIX ではシグナル処理がこれにあたり、汎用機では、十進演算などの COBOL 用の機械語命令が例外を起こすと、COBOL 実行システムがオペレーティングシステムを経由してこれを検知し、エラー報告を行っています。
COBOL 2002 が提供する例外割り込み機能では、このようなハードウェアやオペレーティングシステムが管理する例外を取り扱うことができますが、さらに、COBOL 独自の例外やプログラマが定義する例外も取り扱えるようになっています。
例外が発生すると、宣言部分に書かれた所定の手続きが実行されますが、その後の処理は、
例外の種類によって異なります。「致命的な」例外と呼ばれる一群の例外では、プログラムは実行を停止します。「非致命的な」例外の場合は、引き続き実行を継続します。プログラマが定義する例外はすべて「非致命的な」例外となります。

# 11.2 例外名

例外には、システムが定義するものとプログラマが定義するものとがあります。前者は、さらにCOBOL規格が定義するものとCOBOL処理系作成者が定義するものに分かれます。
COBOL 規格が定義する例外には、表の添え字範囲あふれ、十進データ不正、ゼロによる割り算などのおなじみのものがあります。COBOL 2002 以前は、このようなエラーが発生したときの動作は、ON SIZE ERROR 指定などが書かれていない限り規定されていませんでした。
プログラマが定義する例外とは、例えば、「預金残高不足」とか「在庫数量割れ」といったものになるでしょう。
COBOL 2002 では、これら各種の例外をプログラマが識別するために、例外名を用います。例外名は、以下の 3 階層で分類されています：

レベル 1 例外名：
　　EC-ALL、すべての例外をあらわす。
レベル 2 例外名：
　　EC-I-O など、例外を 19 に類別したそれぞれの代表名。
レベル 3 例外名：
　　レベル 2 例外名に、ハイフンと文字列をつけた名前で、個別の例外をあらわす。

以降に、COBOL 規格が定める例外名の一覧を示します。カテゴリ欄は、例外が致命的であるかどうかを示します。「作成者定義」は、致命的であるかどうかを COBOL 処理系作成者が定めるものを示します：

| 例外名 | カテゴリ | 説明 |
|---|---|---|
| EC-ALL | | すべての例外 |
| EC-ARGUMENT | | 引数エラー |
| EC-ARGUMENT-FUNCTION | 致命的 | 関数引数エラー |
| EC-ARGUMENT-IMP | 作成者定義 | 処理系作成者定義の引数エラー |
| EC-BOUND | | 区域外 |
| EC-BOUND-IMP | 作成者定義 | 処理系作成者定義の区域外 |
| EC-BOUND-ODO | 致命的 | OCCURS DEPENDING データ項目が区域外 |
| EC-BOUND-PTR | 致命的 | ポインタデータ項目の内容が区域外 |
| EC-BOUND-REF-MOD | 致命的 | 区域外の部分参照 |
| EC-BOUND-SUBSCRIPT | 致命的 | 区域外の添え字付け |
| EC-BOUND-TABLE-LIMIT | 致命的 | 動的な表の大きさ違反 |
| EC-DATA | | データ例外 |
| EC-DATA-IMP | 作成者定義 | 処理系作成者定義のデータ例外 |

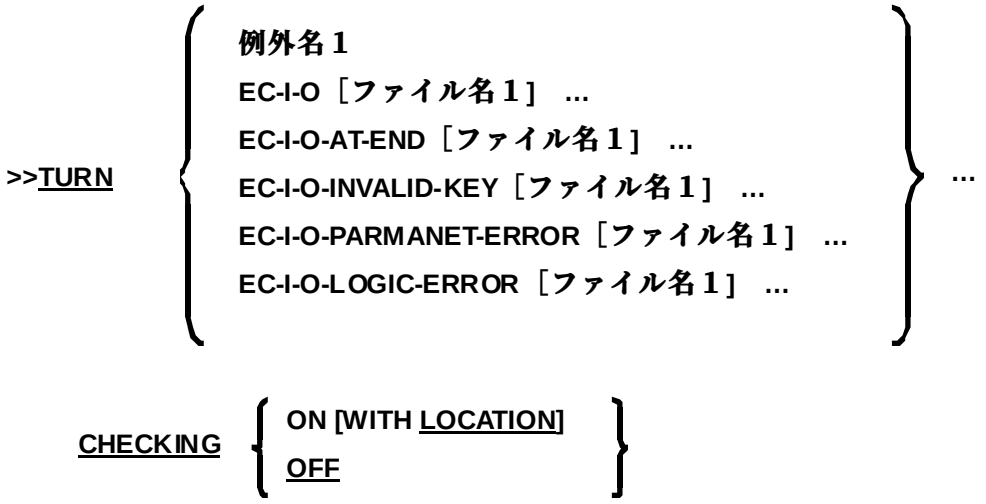
| | | |
|---|---|---|
| EC-DATA-INCOMPATIBLE | 致命的 | 矛盾データ例外 |
| EC-DATA-PTR-NULL | 致命的 | 基底付き項目ポインタの NULL 参照 |
| EC-FLOW | | 実行制御フロー違反 |
| EC-FLOW-GLOBAL-EXIT | 致命的 | 大域的な宣言手続き中での EXIT PROGRAM |
| EC-FLOW-GLOBAL-GOBACK | 致命的 | 大域的な宣言手続き中での GOBACK |
| EC-FLOW-IMP | 作成者定義 | 処理系作成者定義のフロー違反 |
| EC-FLOW-RELEASE | 致命的 | RELEASE が SORT の範囲内にない |
| EC-FLOW-RETURN | 致命的 | RETURN が SORT や MERGE の範囲内にない |
| EC-FLOW-USE | 致命的 | USE 文が別の USE 文を実行させた |
| EC-I-O | | 入出力例外 |
| EC-I-O-AT-END | 非致命的 | 入出力状態「1x」 |
| EC-I-O-FILE-SHARING | 非致命的 | 入出力状態「6x」 |
| EC-I-O-IMP | 作成者定義 | 入出力状態「9x」 |
| EC-I-O-INVALID-KEY | 非致命的 | 入出力状態「2x」 |
| EC-I-O-LINAGE | 致命的 | LINAGE データ項目の値が要求範囲外 |
| EC-I-O-LOGIC-ERROR | 致命的 | 入出力状態「4x」 |
| EC-I-O-PERMANENT-ERROR | 致命的 | 入出力状態「3x」 |
| EC-I-O-RECORD-OPERATION | 非致命的 | 入出力状態「5x」 |
| EC-IMP | | 処理系作成者定義の例外 |
| EC-LOCALE | | 地域固有仕様に関連するすべての例外 |
| EC-LOCALE-IMP | 作成者定義 | 処理系作成者定義の地域固有仕様関連例外 |
| EC-LOCALE-INCOMPATIBLE | 非致命的 | 参照した地域固有仕様に期待の文字指定がない |
| EC-LOCALE-MISSING | 致命的 | 指定した地域固有仕様では使用できない |
| EC-LOCALE-SIZE | 致命的 | 地域固有仕様の編集で数値は切り捨てられた |
| EC-OO | | OO に関連するすべての既定儀例外 |
| EC-OO-CONFORMANCE | 致命的 | オブジェクト修正子に対する不成功 |
| EC-OO-EXCEPTION | 致命的 | 例外オブジェクトが処理されなかった |
| EC-OO-IMP | 作成者定義 | 処理系作成者定義の OO 例外 |
| EC-OO-INTRINSIC | 致命的 | 在来データ項目の型チェックが失敗した |
| EC-OO-METHOD | 致命的 | 要求されるメソッドが使用可能でない |
| EC-OO-NULL | 致命的 | NULL オブジェクト参照によるメソッド起動 |
| EC-OO-RESOURCE | 致命的 | オブジェクトのためのシステム資源不足 |
| EC-OO-UNIVERSAL | 致命的 | 実行時の型チェックが失敗した |
| EC-ORDER | | 順序付け例外 |
| EC-ORDER-IMP | 作成者定義 | 処理系作成者定義の順序付け例外 |
| EC-ORDER-NOT-SUPPORTED | 致命的 | 順序付けが未サポート |
| EC-OVERFLOW | | 桁あふれ条件 |
| EC-OVERFLOW-IMP | 作成者定義 | 処理系作成者定義の桁あふれ条件 |
| EC-OVERFLOW-MEMORY | 致命的 | システム限界のため動的表が拡張不可能 |

| | | |
|---|---|---|
| EC-OVERFLOW-STRING | 非致命的 | STRING 文の桁あふれ条件 |
| EC-OVERFLOW-UNSTRING | 非致命的 | UNSTRING 文の桁あふれ条件 |
| EC-PROGRAM | | プログラム間連絡例外 |
| EC-PROGRAM-ARG-MISMATCH | 致命的 | 引数の不一致 |
| EC-PROGRAM-ARG-OMMITED | 致命的 | 省略された引数への参照 |
| EC-PROGRAM-CANCEL-ACTIVE | 非致命的 | 取り消されるプログラムが活性状態 |
| EC-PROGRAM-IMP | 作成者定義 | 処理系作成者定義のプログラム間連絡例外 |
| EC-PROGRAM-NOT-FOUND | 致命的 | 呼ばれるプログラムが見つからず |
| EC-PROGRAM-PTR-NULL | 致命的 | NULL ポインタによる CALL 文 |
| EC-PROGRAM-RECURSIVE-CALL | 致命的 | 呼ばれるプログラムが活性状態 |
| EC-PROGRAM-RESOURCES | 致命的 | 呼ばれるプログラムで資源が使用不能 |
| EC-RAISING | | EXIT RAISING 例外 |
| EC-RAISING-IMP | 作成者定義 | 処理系作成者定義の EXIT RAISING 例外 |
| EC-RAISING-NOT-SPECIFIED | 致命的 | ユーザ定義例外名が USING に書かれていない |
| EC-RANGE | | 範囲例外 |
| EC-RANGE-IMP | 作成者定義 | 処理系作成者定義の範囲例外 |
| EC-RANGE-INDEX | 致命的 | 指標が負の値か限界を超えている |
| EC-RANGE-INSPECT-SIZE | 致命的 | INSPECT 文での置き換え項目の大きさが異なる |
| EC-RANGE-PERFORM-VARYING | 致命的 | PERFORM VARYING の項目の値が負 |
| EC-RANGE-SEARCH-INDEX | 非致命的 | SEARCH 文の指標の初期値が範囲外 |
| EC-RANGE-SEARCH-NO-MATCH | 非致命的 | SEARCH 文で探索基準に合致する要素なし |
| EC-REPORT | | 報告書作成機能例外 |
| EC-REPORT-ACTIVE | 致命的 | 既に INITIATE された報告書に対する INITIATE |
| EC-REPORT-COLUMN-OVERLAP | 非致命的 | 報告書項目の重なり |
| EC-REPORT-FILE-MODE | 致命的 | 報告書ファイルのオープンモードが不正 |
| EC-REPORT-IMP | 作成者定義 | 処理系作成者定義の報告書作成機能例外 |
| EC-REPORT-INACTIVE | 致命的 | 未 INITIATE の報告書での GENERATE/TERMINATE |
| EC-REPORT-LINE-OVERLAP | 非致命的 | 報告書行の重なり |
| EC-REPORT-LINE-WIDTH | 非致命的 | 行幅を超えた |

| | | |
|---|---|---|
| EC-REPORT-NOT-TERMINATED | 非致命的 | 未 TERMINATE の報告書ファイルの CLOSE |
| EC-REPORT-PAGE-LIMIT | 非致命的 | 縦方向のページの境界を越えた |
| EC-REPORT-SUM-SIZE | 致命的 | 合計カウンタの桁あふれ |
| EC-SCREEN | | 画面操作例外 |
| EC-SCREEN-FIELD-OVERLAP | 非致命的 | 画面フィールドの重なり |
| EC-SCREEN-IMP | 作成者定義 | 処理系作成者定義の画面操作例外 |
| EC-SCREEN-ITEM-TRUNCATED | 非致命的 | 行に対して画面フィールドが長すぎる |
| EC-SCREEN-LINE-NUMBER | 非致命的 | 画面項目の行位置が端末の大きさを超えた |
| EC-SCREEN-STARTING-COLUMN | 非致命的 | 画面項目の開始位置が行の大きさを超えた |
| EC-SIZE | | 桁あふれ例外 |
| EC-SIZE-EXPONENTIATION | 致命的 | べき乗演算規則の違反 |
| EC-SIZE-IMP | 作成者定義 | 処理系作成者定義の桁あふれ例外 |
| EC-SIZE-OVERFLOW | 致命的 | 計算での算術桁あふれ |
| EC-SIZE-TRUNCATION | 致命的 | 格納での有効桁切り捨て |
| EC-SIZE-UNDERFLOW | 致命的 | 浮動小数点の下位桁あふれ |
| EC-SIZE-ZERO-DIVIDE | 致命的 | ゼロによる除算 |
| EC-SORT-MERGE | | SORT/MERGE の例外 |
| EC-SORT-MERGE-ACTIVE | 致命的 | オープン済みファイルへの SORT/MERGE |
| EC-SORT-MERGE-FILE-OPEN | 致命的 | |
| EC-SORT-MERGE-IMP | 作成者定義 | 処理系作成者定義の SORT/MERGE の例外 |
| EC-SORT-MERGE-RELEASE | 致命的 | RELEASE レコードが長すぎまたは短すぎ |
| EC-SORT-MERGE-RETURN | 致命的 | AT END 後に RETURN が実行された |
| EC-SORT-MERGE-SEQUENCE | 致命的 | ファイルに対する順序エラー |
| EC-STORAGE | | 記憶領域割り当て例外 |
| EC-STORAGE-IMP | | 処理系作成者定義の記憶領域割り当て例外 |
| EC-STORAGE-NOT-ALLOC | | 割り当てられていない領域の FREE |
| EC-STORAGE-NOT-AVAIL | | ALLOCATE 文で領域取得不可能 |
| EC-VALIDATE | | VALIDATE 例外 |
| EC-VALIDATE-CONTENT | | VALIDATE の内容エラー |
| EC-VALIDATE-FORMAT | | VALIDATE の形式エラー |
| EC-VALIDATE-IMP | | 処理系作成者定義の VALIDATE 例外 |
| EC-VALIDATE-RELATION | | VALIDATE の比較エラー |
| EC-USER | | 利用者定義の例外 |
| EC-USER-XXX | | レベル３の利用者定義の例外 |

# 11.3 TURN コンパイル指令

プログラムで例外のチェックを行うかどうかは、コンパイル時に TURN 指令を使用して、例外名ごとに指定することができます。
この指令の省略値は何の例外もチェックしません。従って、例外処理機能を使用するには、以下の指令を指定することが必要です。



# 11.4 例外割り込み処理機能の手続き部

## 11.4.1 手続き部の枠組み

例外処理を使用するプログラムの手続き部は、次の形をとります：

```
PROCEDURE DIVISION RAISING 例外名 1. Á 例外名 1 を起こす副プログラムを宣言
DECLARATIVES.
例外処理 SECTION.
    USE AFTER EXCEPTION 例外名 2.      Á 例外名 2 に対する処理手続きの宣言
        …
    RESUME                             Á 例外処理手続きからの戻り
        …
END DECLARATIVES.
手続き処理 SECTION.
        …
    IF エラー
      RAISE EXCEPTION 例外名 2.                Á プログラマ定義例外の発生
        …
    EXIT PROGRAM
      RAISING EXCEPTION 例外名 1.      Á 呼び出し側に例外を発生させる
```

## 11.4.2 手続き部見出し

手続き部見出しの RAISING 指定では、プログラマ定義の例外名 (EC-USER-XXX) を書き、プログラムが EXIT 文で発生させる可能性のある例外を宣言します。

```
PROCEDURE DIVISION
  [USING 指定]
  [RETURNING データ名 2]
[ RAISING { 例外名 1          } ... ]  .
          { クラス名 1        }
          { インタフェース名 1 }
```

例えば、「倉庫から商品を出荷する」というプログラムが、「在庫数量割れ」という例外を発生するように作成したとします。このプログラムを呼び出したプログラムは、CALL 文でその例外が発生し、呼び出し側に宣言された例外処理手続きが実行されることになります。

## 11.4.3 宣言部分

宣言部分の構成は、従来の入出力エラー処理手続きの宣言と変わりありません。変わったのは USE 文の新しい書き方が追加になっただけです。
11.2 で説明したように、例外名には入出力例外も含まれていますので、従来のファイル例外についての宣言手続きの書き方と、新しい例外処理の宣言手続きは、同じ目的のために同居することができます。

```
[ DECLARATIVES.
  { 節名  SECTION .  USE 文 .
    [完結文] ...  [ 段落名.  [ 完結文 ] ... ] ... } ...
  END  DECLARATIVES.  ]
```

## 11.4.4 EXIT PROGRAM 文

手続き部見出しの RAISING 指定で宣言した例外名は、EXIT PROGRAM 文の RAISING 指定に書くことによって例外を発生させます。
RAISING 指定の無い EXIT PROGRAM 文は、例外を発生させること無く呼び出し側プログラムに正常リターンします。呼び出し側プログラムに対して例外の発生を知らせたいときだけ RAISING 指定のある EXIT PROGRAM 文で戻ります。
従来のプログラミングでは、副プログラムを呼び出すと必ずそのあとで副プログラムが返すエラーコードをチェックするコードが書かれていました。この機能を使うと、呼び出し側で宣言手続きを書くだけで、各呼出し毎に毎回チェックする必要がなくなります。

```
EXIT  PROGRAM
[  RAISING { 例外名 1          } ... ]  .
           { クラス名 1        }
           { インタフェース名 1 }
```

## 11.4.5 RESUME 文

致命的でない例外については、宣言手続きの実行後、プログラム実行を再開することができます。通常は、宣言手続きが末尾まで実行されるか、EXIT 文に到達するところで、例外を生じた文の次の実行文から再開することになります。宣言手続き中に RESUME 文を書くと、即座に例外処理を停止して処理を再開することができます。

```
RESUME  AT  { NEXT STATEMENT }
            { 手続き名 1      }
```

## 11.4.6 RAISE 文

プログラマ定義の例外を明示的に発生させるには RAISE 文を使います。

```
RAISE  { EXCEPTION 例外名 1 }
       { 一意名 1           }
```

## 11.4.7 USE 文

例外処理で使用する宣言手続きに対しては、以下の形式の USE 文を使用して、特定の例外や特定のファイルに対する特定の例外についての例外処理を指定します。

```
USE AFTER EXCEPTION  { 例外名 1                    }
                     { 例外名 2 { ファイル名 2 }   }
```

例外名には、レベル 1、レベル 2、レベル 3 のどの例外名を書くこともできます。例外が発生したときに、その例外の種別が宣言部分に書かれた複数の USE 文に合致する場合、レベル 3 の例外名に関する USE 文が書かれていればそれが最優先となり、次いでレベル 2、レベル 3 の順で選択されます。

## 11.4.8 例外処理の組み込み関数

例外処理を使用するために便利ないくつかの組み込み関数が用意されています。

| | |
|---|---|
| EXCEPTION-FILE 関数 | 最近に発生した例外に関連するファイルのステータス値とファイル名を返す。 |
| EXCEPTION-LOCATION 関数 | 最近に発生した例外の発生箇所を返す。プログラム名、段落名、行番号などの処理系作成者定義形式。 |
| EXCEPTION-STATEMENT 関数 | 最近に例外を発生した COBOL 文の名前 (「MOVE」など)を返す。 |
| EXCEPTION-STATUS 関数 | 最近に発生した例外の例外名を返す。 |

# 12 章 データの妥当性検査機能

「オブジェクト指向」ということばが一般的になる 1980 年頃までは、「非手続き的」という言葉がありました。これは、プログラムの動作を手続きロジックで詳細に指示する代わりに、プログラムの主処理を記述する前に宣言的に動作を規定することを意味するものです。例えば、COBOL の報告書作成機能は、報告書のレイアウトや制御切れ、見出し、集計などをデータ部で宣言しておけば、手続き部は GENERATE 文を記述するだけで良いというものです。COBOL の言語仕様にはこのような「非手続き的」処理が少なくありません。
実行文の実際の動作が､そのオペランドの内部状態や型によって規定されると言う意味で、「非手続き的」という概念はそのまま「オブジェクト指向」に引き継がれています。
本章で述べる妥当性検査は、検査条件を事前に宣言しておき、手続き部のロジックでは VALIDATE 文で一括して自動的な検査を行うものです。従って、これも COBOL が新たにサポートした「非手続き的」言語機能ということができます。

COBOL で記述される典型的なデータ処理プログラムでは、しばしば入力データの妥当性のチェックに大きなステップ数が費やされます。外部から取りこんだデータを業務データベースに格納する前にデータがクリーンなものであることを検証することが重要であるからです。COBOL 2002 の妥当性検査機能を使用すると、このようなプログラミングの生産性を大きく向上させることができます。

## 12.1 妥当性検査機能の概要

この機能によってチェックできる妥当性には、次のものがあります。

⑴形式妥当性
基本データ項目の内容が、それに対する PICTURE 句、USAGE 句、SIGN 句の記述に矛盾しないかどうかをチェックします。

⑵内容妥当性
データ項目の内容が、プログラマの定義する特定の値、範囲、字類、英大文字・小文字に合致しているかどうかをチェックします。

⑶関係妥当性
データ項目の内容が、他のデータ項目との関連についての条件を満たすかどうかをチェッ

クします。複数の入力値に対する複合条件のチェックが可能です。

内容妥当性と関係妥当性については、COBOL　2002 でデータ記述項に新たに追加された句によって、プログラマが定義できます。
これらのチェックは、手続き部で VALIDATE 文が明示的に実行されたときになされます。
プログラマは VALIDATE 文を使って次のことができます。

⑴各チェック対象項目を指定された送り先へ転記します。これを入力分配と呼びます。形式妥当性に違反する場合に転記する省略時解釈値を DEFAULT 句で定義しておくこともできます。
⑵ チェックの結果判明したエラーの種別に応じたステータス値を、ERROR 句で定義した項目に設定します。これをプログラムで判定することによって、詳細なエラー後の後始末を記述できます。
⑶ チェックの結果エラーがあれば、その種別に応じた EC-VALIDATE 例外が発生します。
従って、適切な USE 文が書かれた宣言手続きを実行させることができます。

# 12.2 妥当性検査機能のデータ部

内容妥当性と関係妥当性の検査条件を記述するための句が新たに追加されました。

## 12.2.1 ALLOW 句

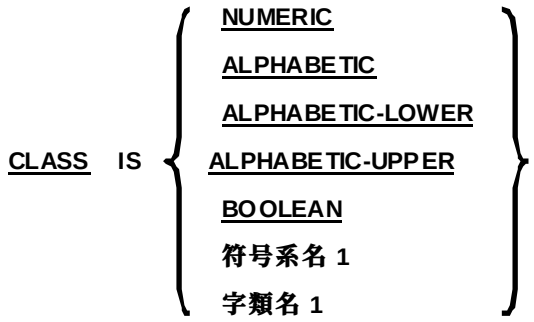
ALLOW 句は、すべての検査に優先して許容される値を指定できます。ここに指定した、ひとつまたは複数の定数値があれば、他の検査の結果にかかわらずエラーとしません。

ALLOW [ONLY] 定数 1 [OR 定数 2 ]... [WHEN 条件 1 ]

条件 1 には、ALLOW 句を適用するための前提条件を記述できます。まずこの条件が検査され、この結果が真であるときにだけ ALLOW 句は適用されます。
ONLY は、ここに指定された値だけが許容されることを示します。即ち、指定された定数値以外の値であれば、関係妥当性チェックのエラーと判断されます。

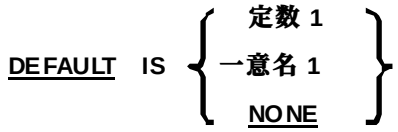
## 12.2.2 CLASS 句

CLASS 句は、字類の内容妥当性検査の仕様を指定します。この句の指定は、条件式の字類検査と同様です。




## 12.2.3 DEFAULT 句

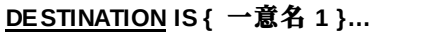
VALIDATE 文の実行によって、データ項目の内容は DESTINATION 句に書かれた送り先へ転記されます。このとき、もしデータ項目の内容が形式妥当性に違反しているか、空白であったら、この送り先へ転記することは無意味になってしまうかもしれません。
DEFAULT 句はこのような場合に代わりに転記する省略時解釈値を指定する。




NONE の指定は、データ項目の内容が形式妥当性に違反しているか、空白であったら転記を行わないことを意味します。

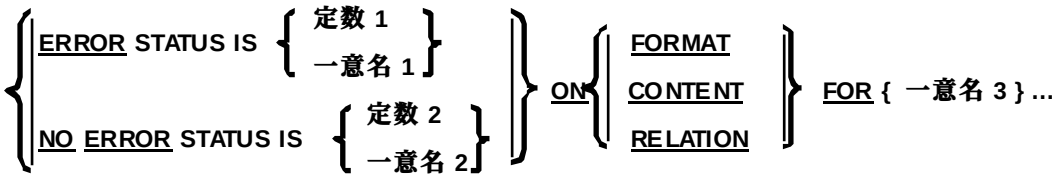
## 12.2.4 DESTINATION 句

DESTINATION 句は、VALIDATE 文実行時のデータの転記先を指定します。




## 12.2.5 ERROR 句

ERROR 句は、妥当性検査の結果を自動的にフラグ項目に設定することを指示します。



例えば､次のプログラムは VALIDATE 文による SHORI-CODE の内容検査に違反すると、「処理コードの内容不正」というメッセージを表示します。

```
01   ERROR-MESSAGE   PIC N(80) VALUE SPACE
         ERROR STATUS IS N"処理コードの内容不正"
         ON CONTENT SHORI-CODE.

     VALIDATE SHORI-CODE.
     DISPLAY ERROR-MESSAGE.
```

## 12.2.6 INVALID 句

INVALID 句は、関係妥当性の検査条件を指定します。

```
INVALID [ WHEN 条件 1 ]
```

この句が書かれたデータ項目に対する VALIDATE 文の実行時に条件 1 が検査され、結果が真であれば関係妥当性違反となります。

## 12.2.7 VALID/INVALID VALUE 句

妥当性検査機能の VALUE 句は、検査対象データ項目の記述項に従属する 88 レベルに書き、内容妥当性の検査仕様を指定します。

```
{ INVALID } { VALUE  } [ IS  ] { 定数 1 [ { THROUGH } 定数 2 ] } IN 符号系名 1
{ VALID   } { VALUES } [ ARE ] {         [ { THRU    }       ] }
  [ WHEN 条件 1 ]
```

WHEN 指定を書くと、内容妥当性検査に先だって条件 1 が検査され、結果が真であるときにだけ内容妥当性検査が行われます。
INVALID VALUE の場合は、定数で指定される値または範囲に適合する場合に内容妥当性違反になり、VALID VALUE の場合は、定数で指定される値または範囲に適合しない場合に内容妥当性違反になります。

# 12.3 VALIDATE 文

VALIDATE 文の形式は、次のように簡潔です。

VALIDATE { 一意名 1 } ...

一意名を複数個書いた場合は、各一意名に対する VALIDATE 文を順番に書いたのと同じ結果になります。
一意名が集団項目である場合、それに従属するすべての項目が妥当性検査の対象になります。検査対象のデータ項目には、内部的に暗黙の標識データ項目が割り当てられ、VALIDATE 文による検査の結果を保持するのに使用されます。これを「内部標識」と呼びます。VALIDATE 文の実行開始時にすべての内部標識はゼロでクリアされます。
VALIDATE 文は以下の 5 段階で実行されます。

⑴ 形式妥当性検査
すべての検査対象項目に対して、その内容が PICTURE 句、SIGN 句、USAGE 句の記述に矛盾しないかがチェックされ、違反している項目の内部標識には「形式が不当」の状態が記録されます。

⑵ 入力分配
検査対象項目に DESTINATION 句が書かれていれば、その受け側への転記が行われます。

⑶ 内容妥当性検査
検査対象項目に CLASS 句が書かれていれば、指定された字類検査が行われ、検査対象項目に INVALID 句 または VALID 句のある 88 項目が従属していれば、指定された値チェックが行われます。ここでの検査の結果違反が検出されると、違反している項目の内部標識には「内容が不当」の状態が記録されます。

⑷ 関係妥当性検査
検査対象項目に INVALID 句か ALLOW ONLY 句が書かれていれば、指定された関係妥当性検査が行われます。ここでの検査の結果違反が検出されると、違反している項目の内部標識には「関係が不当」の状態が記録されます。
ONLY の無い ALLOW 句が書かれていれば、指定された許容値に一致する場合、ここまでの段階で設定された内部標識はゼロにリセットされます。

⑸ 誤り指摘
以上の検査の結果、内部標識がゼロでないものがひとつでもある場合、EC-VALIDATE 例

外が発生します。「形式が不当」の内部標識があれば、EC-VALIDATE-FORMAT 例外が発生し、「内容が不当」の内部標識があれば、EC-VALIDATE-CONTENT 例外が発生し、「関係が不当」の内部標識があれば、EC-VALIDATE-RELATION 例外が発生します。
次に、検査対象項目を FOR 指定に書いた ERROR 句のあるデータ項目への転記が行われます。

# 13 章 オブジェクト指向機能

この章では、COBOL2002 のオブジェクト指向機能について説明します。
COBOL2002 では、COBOL にとってまったく新しい概念であるオブジェクト指向機能を、従来の機能と完全な互換を保ちながら取り込んでいます。そのため、既存資産を活かしたオブジェクト指向開発が可能です。その一方で、オブジェクト指向プログラミングに必要な機能を一通りサポートしており、他のオブジェクト指向言語と比較しても遜色のないものになっています。

## 13.1 クラスとオブジェクト

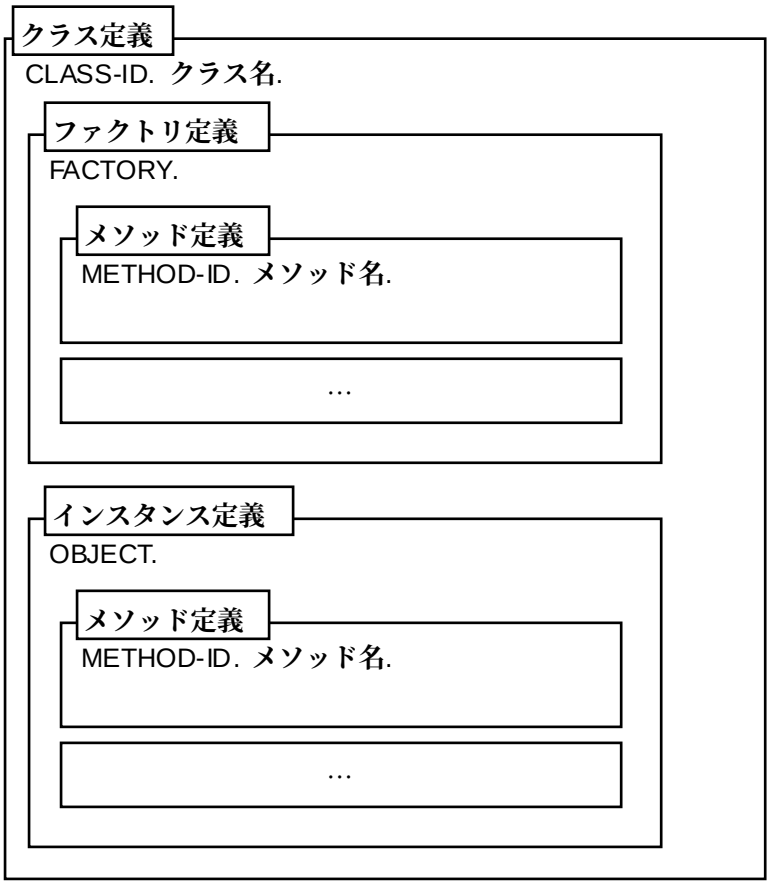
オブジェクト指向プログラミングで扱うオブジェクトは、現実世界の“ もの ”あるいは“ 概念 ” を計算機上でモデル化したものです。オブジェクト指向プログラミングは、システムをオブジェクトの集まりと考え、それらを相互に連携させながらシステムを開発する方法です。
オブジェクトは、それ自身の内部にデータや手続きのコードを持っていますが、それらは外部からは一切見えません。外部から見えるのは、公開された手続き（メソッド）のインタフェースだけです。つまり、オブジェクト内部の実現方法は外部から完全に隠蔽されており、公開されたメソッドを通じてだけアクセスできます。
オブジェクトの共通の性質（データレイアウトと手続き）はクラスにより定義されます。あるクラスに属するオブジェクト、すべて同じデータレイアウトと手続きを持ちます。
たとえば、銀行業務アプリケーションでは、たくさんの預金口座オブジェクトを扱います。
それぞれの預金口座オブジェクトは様々なデータ（口座番号、名義人、残高など）を持ち、外部からアクセスするための手続き（預け入れ、払戻し、残高照会など）を持っています。
すべての預金口座オブジェクトは、同じデータレイアウトと手続きを持っており、それらを定義するのが預金口座クラスです。

## 13.1.1 クラス

クラスは、各々のオブジェクトが持っているデータのレイアウト及び、手続きであるメソッドを定義したものです。同じクラスに所属するオブジェクトは、すべて同じデータレイアウトを持ち、同じメソッドを持っています。
COBOL のクラス定義は、次の二つの部分から構成されています。

・インスタンス定義
・ファクトリ定義

インスタンス定義では、そのクラスに属するインスタンスオブジェクトを定義し、ファクトリ定義は各クラスに一つだけ存在するファクトリオブジェクトを定義します。これらのオブジェクトの違いについては、“13.1.2 インスタンスオブジェクト”と“13.1.3 ファクトリオブジェクト”で説明します。
COBOL のクラス定義は、次のような構造になっています。クラス定義はファクトリ定義とインスタンス定義を含み、それぞれの定義はメソッド定義を含んでいます。



## 13.1.2 インスタンスオブジェクト

クラスに属するオブジェクトをインスタンスオブジェクトまたは単にインスタンスと呼びます。
同じクラスに属するインスタンスはインスタンス定義で定義されたデータ定義を共有します。しかし、それぞれのインスタンスはぞれぞれ固有のデータ値を持っています。また、同じクラスに属するインスタンスはメソッド定義も共有します。
インスタンスオブジェクトは、そのクラスのファクトリオブジェクトによって生成されます。インスタンス生成時には、インスタンス定義中のデータ記述項にしたがってオブジェクトの領域を割りつけます。インスタンスのデータ領域は、そのクラスのインスタンスメソッドからしかアクセスできません。そのため、インスタンスデータを外部から完全に隠すことができます。
たとえば、銀行業務アプリケーションでは、それぞれの預金口座の情報は預金口座クラスのインスタンスである預金口座インスタンスが持っています。また、それぞれの預金口座インスタンスは、名義人、残高などの口座に関する情報をもっています。これらの領域は、預金口座クラスのインスタンス定義で定義されています。
次の例では、預金口座インスタンスは、残高と名義人の二つのインスタンスデータを持ち、預け入れ、引出し、残高照会の三つのインスタンスメソッドを持っています。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
…
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 残高          PIC S9(16)V9（２）.
01 名義人        PIC N(40).
PROCEDURE DIVISION.
METHOD-ID. 預け入れ.
…
END METHOD 預け入れ.
METHOD-ID. 引出し.
…
END METHOD 引出し.
METHOD-ID. 残高照会.
…
END METHOD 残高照会.
END OBJECT.
END CLASS 預金口座.
```

## 13.1.3 ファクトリオブジェクト

各クラスは、ファクトリオブジェクトと呼ばれる特殊なオブジェクトを一つだけ持っています。ファクトリオブジェクトは、オブジェクトインスタンスを生成したり、クラス共通の情報を管理したりするために使います。
ファクトリオブジェクトは、インスタンスオブジェクトと同様に、データ（ファクトリデータ）とメソッド(ファクトリメソッド）を持ちます。ファクトリデータは、クラス内で共通なデータを格納するのに使い、ファクトリメソッドはオブジェクトインスタンスの生成などに使います。
たとえば、預金口座クラスは、預金口座ファクトリオブジェクトと呼ばれるファクトリオブジェクトを持っています。新しい預金口座インスタンスを作るには、預金口座ファクトリオブジェクトのファクトリメソッドを使います。
次の例では、預金口座ファクトリオブジェクトは、新しい預金口座インスタンスを生成するための新規口座メソッドを持っています。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
PROCEDURE DIVISION.
METHOD-ID. 新規口座.
DATA DIVISION.
LINKAGE SECTION.
01 作成口座      USAGE OBJECT REFERENCE ACTIVE-CLASS.
PROCEDURE DIVISION   RETURNING 作成口座.
…
END METHOD 新規口座.
END FACTORY.
OBJECT.
…
END OBJECT.
END CLASS 預金口座.
```

## 13.1.4 定義済みオブジェクト

COBOL では、次のオブジェクトが予め用意されており、オブジェクトを使用できる場所で使うことができます。

- EXCEPTION-OBJECT： 例外オブジェクト
- NULL： NULL オブジェクト（何も指していない状態を表す）
- SELF： メソッドを実行中のオブジェクト
- SUPER：メソッドの探索をスーパークラスから始める（メソッド呼出し時に使用）

## 13.2 継承と多態

オブジェクト指向の特徴の一つとして、アプリケーションの一部を簡単に再利用できることが挙げられます。継承は、アプリケーションの再利用を行うための機構です。
多態とは、“一つのものが複数の形態をとれる”ことです。オブジェクト指向プログラミングでは、多態を継承の機構を使って実現しています。

### 13.2.1 適合

二つのクラス A, B があり、クラス A のオブジェクトがクラス B のオブジェクトとして振る舞える場合を、“A は B に適合する”と言います。
クラス A のオブジェクトがクラス B のオブジェクトとして振る舞えるとは、クラス A のオブジェクトが、クラス B のオブジェクトが受け入れることができる要求をすべて受け入れられるということです。具体的には、クラス A のオブジェクト上で、クラス B のオブジェクト上で呼び出せるメソッドと同名かつ同じシグニチャのメソッドをすべて呼び出せるということです。
A が B に適合していれば、実際のオブジェクトがクラス A のものであっても、クラス B のオブジェクトだと思って使うことができます。クラス A のオブジェクトに対してクラス B のインタフェースでメソッドを呼び出しても、A のオブジェクトは必ず同名・同シグニチャのメソッドを実行できるからです。

### 13.2.2 継承

継承の機構により、既存のクラスをベースに、新しいデータを追加し、新しいメソッドを追加したり、既存のメソッドを置き換えたりすることにより、新しいクラスを定義することができます。つまり、既存のクラスとの差分をコーディングするだけで、新しいクラスを定義することができます。このとき、既存のクラスをスーパークラス、新しいクラスをサブクラスと呼びます。
スーパークラスで定義されたデータは、サブクラスにも引き継がれます。ただし、スーパークラスで定義したデータには、サブクラスのメソッドからはアクセスできません。また、スーパークラスのすべてのメソッドは、サブクラスのメソッドとしても使うことができます。
クラスを継承した場合、サブクラスは、スーパークラスに必ず適合しています。したがって、サブクラスのオブジェクトは、必ずスーパークラスのオブジェクトとして振舞うことができます。適合の要件を満たすために、継承したクラスはスーパークラスで定義されたメソッドのインタフェースを削除・変更することはできません。

継承により、クラスを階層化することができます。あるクラスのオブジェクトを参照するために宣言されたデータ項目は、そのクラスを継承したすべてのクラスのオブジェクトを参照できます。

継承の例として、銀行業務アプリケーションについて考えてみます。銀行業務で扱う預金口座は一種類だけではありません。例えば、普通預金、定期預金、当座預金などがあります。しかしそれらはすべて“預金口座”であることに変わりはありません。たとえば、当座預金と普通預金を比べる、それらは多くの共通点（たとえば、名義人と口座残高の情報を持つ）といくつかの相違点（たとえば、当座預金は小切手を発行できるが利息はつかない）があります。
このような場合、ベースとなる預金口座クラスを作り、当座預金と普通預金を定義するために継承を使います。預金口座クラスはすべての預金口座に共通な部分を定義し、当座預金クラスには当座預金に固有の部分を、普通預金クラスには普通預金に固有の部分を定義します。
このとき、当座預金クラスのコードは次のようになります。当座預金クラスは、預金口座クラスを継承しています。そのため、当座預金クラスのオブジェクトは、預金口座クラスで定義されたインスタンスデータ（残高、名義人）とインスタンスメソッド（預け入れ、引出し、残高照会）も持っています。また、それ以外に当座預金クラスで新しく定義されたインスタンスメソッド“小切手発行”を持っています。

```
CLASS-ID. 当座預金 INHERITS 預金口座.
…
FACTORY.

END FACTORY.
OBJECT.
DATA DIVISION.
…
PROCEDURE DIVISION.
METHOD-ID. 小切手発行.
…
END METHOD 小切手発行.
END OBJECT.
END CLASS 預金口座.
```

### 13.2.3 多態

多態とは、“一つのものが複数の形態をとれる”ことです。
銀行業務アプリケーションの例では、普通預金クラスも当座預金クラスもどちらも預金口座クラスのサブクラスです。そのため、どちらも預金口座クラスの性質を引き継いでおり、預金口座オブジェクトとしても振舞うことができます。逆に言えば、プログラム上は預金口座オブジェクトと書かれていても、実際のオブジェクトは普通預金オブジェクトであったり、当座預金オブジェクトであったりするわけです。
たとえば、預金口座クラスが解約処理メソッドを持っている場合、それを継承した普通預金クラスも当座預金クラスも、同じインタフェースの解約処理メソッドを持っています。このような場合、プログラム上で“預金口座オブジェクト対し、解約処理を呼び出す”とさえ記述しておけば、実際のオブジェクトが普通預金オブジェクトであれば普通預金用の解約処理を、当座預金オブジェクトであれば当座預金用の解約処理を呼び出すことができます。

### 13.2.4 インタフェース

それぞれのクラスは二つのインタフェースを持っています。一つはインスタンスオブジェクトのインタフェースであり、すべてのインスタンスメソッド（継承されたメソッドも含む）から構成されます。もう一つはファクトリオブジェクトのインタフェースであり、すべてのファクトリメソッド（継承されたメソッドも含む）から構成されます。
オブジェクトは、それぞれのクラスで定義したインタフェースを持っています。インスタンスオブジェクトはインスタンス定義で定義したインタフェース持ち、ファクトリオブジェクトはファクトリ定義で定義したインタフェースを持ちます。
COBOL では、クラスとは独立にインタフェースだけを定義することができます。 オブジェクトを参照するデータ項目を定義する際にインタフェースを指定すると、そのデータ項目からはそのインタフェースを実装するオブジェクトなら、クラス階層に関係なく参照することができます。このようなデータ項目では、代入の可否やメソッド呼出し時のパラメタの妥当性はすべてインタフェースを実装しているか否かによりチェックされます。
たとえば、銀行業務アプリケーションは、 オブジェクトの情報を印刷するクラスがたくさんあります。たとえば、預金口座オブジェクトは名義人と残高を印刷し、顧客オブジェクトは顧客の名前と住所を印刷します。両者に共通の印刷ルーチンを作成する場合、印刷関連のメソッドを含むインタフェースを定義し、そのインタフェースを通して両者の印刷関連メソッドを呼び出します。両者の間に継承関係はありません、印刷のための同じインタフェースを実装しています。これにより、このインタフェースを実装するオブジェクトは、すべてこのルーチンで印刷可能になります。

```
*> 印刷処理インタフェースの定義
INTERFACE-ID. 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END INTERFACE 印刷処理.

*> 預金口座クラスの定義
*> 預金口座インスタンスは、印刷処理インタフェースを実装する。
CLASS-ID. 預金口座 INHERITS BASE.
…
OBJECT IMPLEMENTS 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END OBJECT.
END CLASS 預金口座.

*> 顧客クラスの定義
*> 顧客インスタンスも、印刷処理インタフェースを実装する。
CLASS-ID. 顧客 INHERITS BASE.
…
OBJECT IMPLEMENTS 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END OBJECT.
END CLASS 顧客.

*> 上記のクラス/インタフェースを使うプログラム
PROGRAM-ID. 印刷ルーチン.
…
DATA DIVISION.
WORKING-STORAGE SECTION.
01 口座          USAGE OBJECT REFERENCE 預金口座.
01 名義人        USAGE OBJECT REFERENCE 顧客.
01 印刷オブジェクト      USAGE OBJECT REFERENCE 印刷処理.
…
PROCEDURE DIVISION.
```

```
...
    INVOKE 預金口座 "新規口座" RETURNING 口座.
    INVOKE 顧客 "New" RETURNING 顧客情報.
    ...
    SET 印刷オブジェクト TO 口座.
    INVOKE 印刷オブジェクト "印刷".
    ...
    SET 印刷オブジェクト TO 顧客情報.
    INVOKE 顧客情報 "印刷".
    ...
END PROGRAM 印刷ルーチン.
```

# 13.3 オブジェクト参照

それぞれのオブジェクトには、実行時システムが割り当てた参照値を持っています。参照値はオブジェクトへのポインタと考えることができ、オブジェクトを特定する場合には必ずこの値を使います。オブジェクトの参照値は、オブジェクト参照と呼ばれるデータ項目に格納します。

## 13.3.1 オブジェクト参照の定義

オブジェクト参照は、USAGE 句に OBJECT　REFERENCE を指定して定義します。

**預金口座クラスまたはそのサブクラスのオブジェクトを参照できるデータ項目**

```
01　オブジェクト　　USAGE OBJECT REFERENCE 預金口座.
```

**預金口座クラスのオブジェクトしか参照できないデータ項目**

```
01　オブジェクト　　USAGE OBJECT REFERENCE 預金口座 ONLY.
```

**印刷処理インタフェースを実装するオブジェクトを参照できるデータ項目**

```
01　オブジェクト　　USAGE OBJECT REFERENCE 印刷処理.
```

**メソッドを実行しているオブジェクトのクラスのインスタンスオブジェクトを参照するデータ項目**

```
01　オブジェクト　　USAGE OBJECT REFERENCE ACTIVE-CLASS
```

また、上記のほかに、参照できるオブジェクトに一切制約を設けない普遍オブジェクト参照を使用することができます。USAGE 句の OBJECT　REFERENCE 指定で何も指定しなければ、普遍オブジェクト参照になります。

**どのようなオブジェクトでも参照できるデータ項目**

```
01　オブジェクト　　USAGE OBJECT REFERENCE.
```

## 13.3.2 適合チェック

オブジェクト参照は、データ記述項での指定を満たすオブジェクトしか参照できません。この規則は代入とパラメタの受渡しに適用されます。
たとえば、銀行業務アプリケーションでは、次の二つのオブジェクト参照を定義します。

```
01 口座－１　　USAGE OBJECT REFERENCE 預金口座.
01 口座－２　　USAGE OBJECT REFERENCE 普通預金.
```

口座－１は、預金口座クラスとそのサブクラスのインスタンスを参照できます。たとえば、

普通預金クラスは預金口座クラスのサブクラスなので、口座－１は普通預金インスタンスを参照できる。逆に、口座－２は普通預金インスタンスは参照できますが、その他のクラスのインスタンスは参照できません。口座－２に預金口座インスタンスを代入しようとすると、コンパイラがエラーを検出します。

### 13.3.3 手続き部での扱い

オブジェクト参照の代入には、MOVE 文ではなく SET 文を使います。このとき、送り側項目のクラスは受取り側項目のクラスと同じかまたはサブクラスでなければなりません。また、比較は = と NOT = しか使用できません。比較は、オブジェクト参照の値しか比較しません。したがって、同じオブジェクトを指していれば等しいとみなされます。

### 13.3.4 オブジェクトビュー

オブジェクト参照がスーパークラスのインスタンスを参照するように定義されていても、実行時にサブクラスのインスタンスを参照することがあります。たとえば、オブジェクト参照が預金口座インスタンスを参照するように宣言されていても、実行に普通預金インスタンスへの参照が格納されることがあります。このような場合には、普通預金インスタンスを参照するオブジェクト参照に代入できる必要があります。
このような場合、COBOL ではオブジェクトビューを使います。オブジェクトビューを使うことにより、預金口座インスタンスを参照するオブジェクト参照を、普通預金インスタンスを参照するオブジェクト参照とみなすことができます。
たとえば、

```
01  口座－１      USAGE IS OBJECT REFERENCE 預金口座.
02  口座－２      USAGE OBJECT REFERENCE 普通預金.
```

と宣言した場合、

```
SET 口座－１ TO 口座－２
```

は書けますが、

```
SET 口座－２ TO 口座－１
```

はエラーになります。普通預金クラスは預金口座クラスのサブクラスですが、預金口座クラスは普通預金クラスのサブクラスではないからです。

しかし、口座－１に普通預金インスタンスへの参照が入っていることもあり、口座－２への代入が必要になることがあります。このような場合、オブジェクトビューを使って、次のように書きます。

```
SET 口座－２ TO 口座－１ AS 普通預金
```

このように書くと、口座－１が参照するオブジェクトが普通預金クラスのオブジェクトとみなされるので、代入が可能になります。ただし、口座－１が普通預金クラスおよびそのサブクラスの以外のオブジェクトを参照していた場合は実行時エラーになります。

# 13.4 メソッド

メソッドは、そのクラスのオブジェクト上で実行される手続きコードです。オブジェクトの内部には、メソッドを呼び出すことによってアクセスできます。
たとえば、預金口座クラスは、預け入れ、引出し、残高照会などの情報をもっています。アプリケーションが預金口座インスタンスの残高を求めたい場合には、残高照会メソッドを呼び出します。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
…
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  残高          PIC S9(18).
…
PROCEDURE DIVISION.
METHOD-ID. 預け入れ.
…
END METHOD 預け入れ.
METHOD-ID. 引出し.
…
END METHOD 引出し.
METHOD-ID. 残高照会.
DATA DIVISION.
LINKAGE SECTION.
01  現在残高      PIC S9(18).
PROCEDURE DIVISION   RETURNING 現在残高.
    MOVE 残高 to 現在残高.
    EXIT METHOD.
END METHOD 残高照会
…
END OBJECT.
END CLASS 預金口座.

*> 残高照会メソッドを呼び出すプログラム
PROGRAM-ID. 残高チェック.
…
DATA DIVISION.
WORKING-STORAGE SECTION.
…
01  口座          USAGE OBJECT REFERENCE 預金口座.
```

```
01  現在残高       PIC S9(18).
…
PROCEDURE DIVISION.
    …
    INVOKE 口座 "残高照会"      RETURNING 現在残高.
    …
END PROGRAM 残高チェック.
```

## 13.4.1 メソッドの一意性

メソッドは、名前だけで区別されます。そのため、同じクラスには同じ名前のメソッドを複数定義することはできません。継承したクラスが継承されたクラスの中のメソッドと同じ名前のメソッドを定義した場合、継承したクラスのメソッドで上書きされます。

## 13.4.2 メソッド呼出し

メソッドは、オブジェクトとメソッド名を指定することにより呼び出します。メソッドはそのオブジェクトのクラスで定義されているかも知れないし、直接又は間接的に継承したクスラで定義されているかも知れません。
メソッド呼出しには、行内呼出しと、 INVOKE 文を使う方法があります。行内呼出しは一意名の書けるところに指定することができ、実行時にメソッドの復帰値に置き換えられます。
たとえば、預金口座からある金額をひき出す処理は、次のように書きます。

```
INVOKE 口座 "引出し" USING 金額
```

また、残高紹介は次のように書きます。

```
INVOKE 口座 "残高照会" RETURNING 金額
```

さらに、行内呼出しでは次のようになります。

```
MOVE 口座:: "残高照会"  TO 金額
```

また、パラメタを伴う行内呼出しは、次のように書きます。

```
MOVE 口座:: "残高照会－２" (パラメタ)  TO 金額
```

# 13.5 オブジェクトプロパティ

COBOL では、オブジェクトのデータは外部から完全に隠蔽されています。オブジェクトデータにアクセスするためには、そのためのメソッドを呼び出さなければなりません。
しかし、オブジェクトのデータを参照するのに一々メソッド呼出しの文を書いたりメソッドを定義したりするのは面倒です。そのため、特殊な構文のメソッド（プロパティメソッド）を定義することにより、通常の OF による修飾と同じ構文でメソッドの呼出しを可能にしています。また、データ記述項に PROPERTY 句を書くことにより、プロパティメソッドを自動的に生成することもできます。

## 13.5.1 オブジェクトプロパティの設定/参照

一意名の書ける所に "プロパティ名 OF 一意名" と書くと、次のメソッド呼出しと同じになります。

[受取側の場合]

```
送り側項目 を temp-data へ代入
INVOKE 一意名 "プロパティ設定メソッド" USING temp-data.
```

[送り側項目の場合]

```
INVOKE 一意名 "プロパティ参照メソッド" RETURNING temp-data
temp-data を 受取側項目 へ代入
```

## 13.5.2 PROPERTY 句

ファクトリ定義およびインスタンス定義の作業場所節のデータ記述項に PROPERTY 句を指定することにより、そのデータ項目に値を設定/参照するプロパティメソッドを自動的に生成することができます。
たとえば、

```
01  残高 PIC S9(18) PROPERTY.
```

と書くと、次の二つのプロパティメソッドが生成されます。

```
METHOD-ID. GET PROPERTY 残高.
LINKAGE SECTION.
01  現在残高       PIC S9(18).
PROCEDURE DIVISION RETURNING 現在残高.
    MOVE 残高 TO 現在残高
    EXIT METHOD.
END METHOD.
```

```
METHOD-ID. SET PROPERTY 残高.
LINKAGE SECTION.
01  現在残高       PIC S9(18).
PROCEDURE DIVISION USING 現在残高.
    MOVE 現在残高 TO 残高
    EXIT METHOD.
END METHOD.
```

## 13.6 オブジェクトの寿命

**インスタンスオブジェクトは、実行時に割り当てられ、オブジェクト参照を通して参照されます。インスタンスオブジェクトは、ファクトリオブジェクトのメソッドを呼び出す事によって生成されます。インスタンスオブジェクトは、それ以上アクセスされなくなると、必要に応じて自動的に削除されます。**

## 13.7 リポジトリ

クラスを使用するプログラムを翻訳する場合、適合チェックなどのために、プログラムで使用しているクラスの情報が必要になります。COBOL では、これらの情報をやり取りするために、外部リポジトリを使用します。

### 13.7.1 クラス/インタフェースの情報

COBOL コンパイラは、クラスのソースを翻訳する際に、クラス参照時に必要となる情報を外部リポジトリに格納します。そして、クラスを参照しているソースを翻訳する際に、リポジトリからクラス情報を取り出し、使用します。
さらに、COBOL では、クラスと同じようにインタフェースも独立して定義することができます。そのため、外部リポジトリにはクラスの情報だけでなく、インタフェースの情報も格納されます。

### 13.7.2 REPOSITORY 段落

翻訳時に外部リポジトリからクラス・インタフェース情報を取り出すために、環境部の構成節の REPOSITORY 段落に、ソース中で参照するクラス・インタフェースをすべて宣言します。
たとえば、BASE クラス、預金口座クラスと印刷処理インタフェースを参照している場合、次のように宣言します。

```
REPOSITORY.
    CLASS BASE
    CLASS 預金口座
    INTERFACE 印刷処理.
```

# 13.8 その他のオブジェクト指向機能

## 13.8.1 パラメタ化クラス

パラメタ化クラスは、あらかじめクラスの骨組みを用意しておき、利用時にパラメタを渡すことによって目的のクラスを作成するための機能です。オブジェクトの集合を扱うコレクションクラスを定義する際に､格納するオブジェクトのクラスをパラメタにしておけば、様々なクラスのコレクションを作成することができます。

## 13.8.2 例外オブジェクト

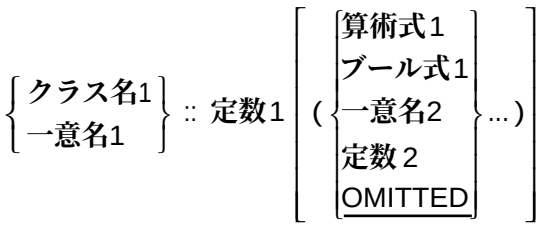
COBOL2002 では例外名による例外処理が追加されますが、オブジェクト指向機能を使った場合は、例外名の代わりに例外オブジェクトを使用できます。例外オブジェクトは、オブジェクト内に例外に関する詳細な情報を持たせることができるので、きめ細かい例外処理が可能です。

## 13.8.3 クラスライブラリ

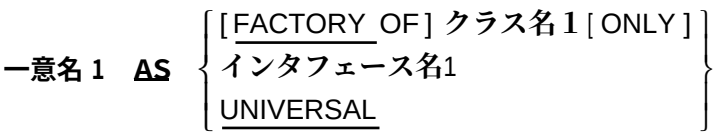
オブジェクト作成、初期化および他の便利な基本的な機能を含む BASE クラスを提供します。インスタンスオブジェクトを作成する NEW メソッドも、BASE クラスのメソッドです。

# 13.9 オブジェクト指向関連の構文

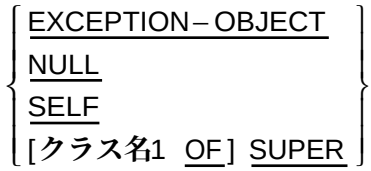
**【行内メソッド呼出し】**



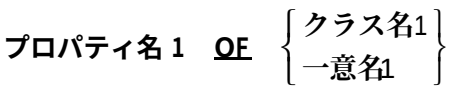

**【オブジェクトビュー】**




**【定義済みオブジェクト参照】**




**【オブジェクトプロパティ】**



**【クラス定義】**

[ IDENTIFICATION DIVISION. ]

CLASS-ID. クラス名 1 [ AS 定数 1 ] [ IS FINAL ]

[ INHERITS FROM { クラス名 2} ... ]

[ USING { パラメタ名 1 } ... ] .

[ オプション段落 ]

[ 環境部 ]

[ ファクトリ定義 ]

[ インスタンス定義 ]

END CLASS クラス名 1.

**【インタフェース定義】**

[ IDENTIFICATION DIVISION. ]

INTERFACE-ID. インタフェース名 1 [ AS 定数 1 ]

[ INHERITS FROM { クラス名 2} ... ]

[ USING { パラメタ名 1 } ... ] .

[ オプション段落 ]

[ 環境部 ]

[ 手続き部 ]

END INTERFACE インタフェース名 1.

**【ファクトリ定義】**

[ IDENTIFICATION DIVISION. ]

FACTORY. [ IMPLEMENTS { インタフェース名 1 } ... . ]

[ オプション段落 ]

[ 環境部 ]

[ データ部 ]

PROCEDURE DIVISION.

[ { メソッド定義 } ... ]

END FACTORY.

**【インスタンス定義】**

[ IDENTIFICATION DIVISION. ]

OBJECT. [ IMPLEMENTS { インタフェース名 2 } ... . ]

[ オプション段落 ]

[ 環境部 ]

[ データ部 ]

PROCEDURE DIVISION.

[ { メソッド定義 } ... ]

END OBJECT.

**【メソッド定義】**

[ IDENTIFICATION DIVISION. ]

METHOD-ID. { メソッド名1 [ AS 定数1] / { GET / SET } PROPERTY プロパティ名1 }

[ OVERRIDE ] [ IS FINAL ] .

[ オプション段落 ]

[ 環境部 ]

[ データ部 ]

[ 手続き部 ]

END METHOD [ メソッド名 1 ] .

**【リポジトリ段落】**

REPOSITORY.

[ { クラス指定子 / インタフェース指定子 / 関数指定子 / プログラム指定子 / プロパティ指定子 }... ]

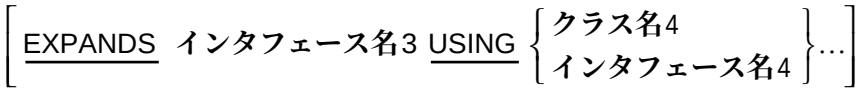
**【クラス指定子】**

CLASS クラス名 1 [ AS 定数 1 ]

[ EXPANDS クラス名2 USING { クラス名3 / インタフェース名1 }... ]

**【インタフェース指定子】**

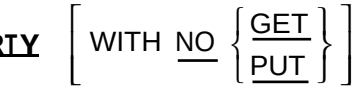
INTERFACE インタフェース名 2 [ AS 定数 2 ]




**【プロパティ指定子】**

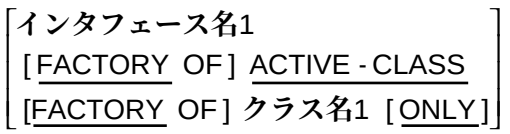
**PROPERTY** プロパティ名1 [ **AS** 定数5 ]

**【PROPERTY句】**



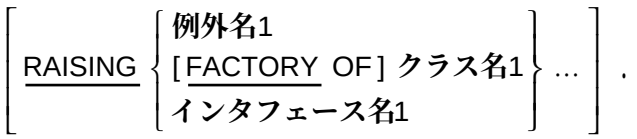

**【USAGE句】**

[ **USAGE** IS ] **OBJECT REFERENCE**

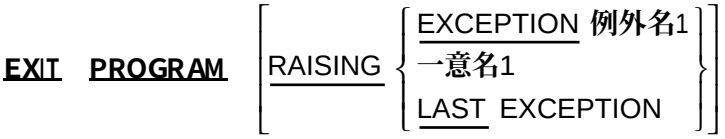


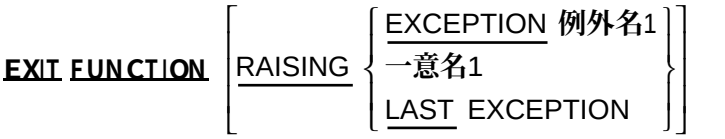

**【手続き部の見出し】**

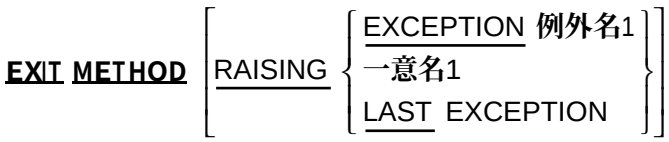
**PROCEDURE DIVISION** [ **USING** 指定 ] [ **RETURNING** データ名2 ]




**【EXIT文】**




EXIT METHOD
RAISING
EXCEPTION 例外名1
一意名1
LAST EXCEPTION

**【GOBACK 文】**

```
GOBACK [ RAISING { EXCEPTION 例外名1 }
                 { 一意名1          }
                 { LAST EXCEPTION   } ]
```

**【INVOKE 文】**

```
INVOK { クラス名1 } { 一意名2 }
      { 一意名1   } { 定数1   }

[ USING { [BY REFERENCE ] { 一意名3 }                    }
        { OMITTED         }                              }
        { [BY CONTENT ] { 算術式1 }                      }
        {               { ブール式1 }                    }
        {               { 一意名5 }                      } ...
        {               { 定数2 }                        }
        { [BY VALUE ] { 算術式1 }                        }
        {             { 一意名5 }                        }
        {             { 定数2 }                          } ]

[RETURNING 一意名 4]
```

**【RAISE 文】**

```
RAISE { EXCEPTION 例外名1 }
      { 一意名1           }
```

**【USE 文】**

```
USE AFTER { EXCEPTION OBJECT } { クラス名1          }
          { EO               } { インタフェース名1 }
```

**【SET 文】**

```
SET { 一意名 3 } ... TO { クラス名1 }
                        { 一意名4   }
```

# 14 章 言語間連絡の拡張

この章では、言語間連絡を主題として、連絡機能の拡張について説明します。

## 14.1 連絡機能の拡張

次に COBOL2002 での連絡機能の拡張を、項目毎に見て行きます。

（１）PROCEDURE DIVISION や CALL 命令で RETURNING の指定が可能になりました。前者は組み込み関数の様な値を返却するプログラム、関数を COBOL で記述出来る様になったという事です。また後者もその様な関数タイプの手続きを CALL 命令で呼び出す事が出来ます。

（２）PROCEDURE DIVISION や CALL 命令のパラメータ受け渡し方式として、BY VALUE の指定が可能になりました。従来の規格では BY REFERENCE と BY CONTENT だけでしたが、この２方式は両方、アドレス渡し方式です。従来規格ではポインタデータも無いし、アドレスを渡している意識は特に無かったかも知れません。REFERENCE と CONTENT の差は前者が呼び出し側のデータそのものを渡すのに対し、後者はデータのコピーを渡します。
これらを纏めると次の表になります。

| | REFERENCE | CONTENT | VALUE |
|---|---|---|---|
| 渡し方 | アドレス | アドレス | データ値 |
| 渡す物 | オリジナル | コピー | コピー |

表 14.1　パラメータの受け渡し方式

（３）PROCEDURE DIVISION や CALL 命令で各パラメータの前に OPTIONAL 指定が出来、パラメータが省略可能である事を表現出来ます。また引数が省略されているかどうかを下記形式の「引数省略条件」で判定出来ます。

**データ名　IS　[NOT]　OMITTED**

例えば次の様に、パラメータを最大３つ渡せるが、必須なのは先頭パラメータだけである様なプログラムを作る事が出来ます。

```
PROCEDURE   DIVISION   USING
                SYORI-SYUBETU
                OPTIONAL   PARAM-2
                OPTIONAL   PARAM-3.
    IF   PARAM-2   IS   OMITTED
        第２パラメータ省略時の処理
    END-IF   …
```

（４）コンパイル指示子 CALL-CONVENTION が指定出来ます。これはプログラムやメソッドの相互作用の詳細を決定するものです。つまりプログラム呼び出し規約の種類を指定出来ると言う事です。詳細については処理系作成者定義になります。

（５）GOBACK 命令が新設されました。これは EXIT PROGRAM と STOP RUN の機能を併せ持つ働きをします。つまり呼ばれたプログラム内では EXIT PROGRAM と同様の動作を行い、メインプログラムでは STOP RUN と同じになります。

（６）プログラム・プロトタイピング（原型）機能が利用出来ます。環境部・構成節の「REPOSITORY」段落でプロトタイプ宣言を行う事によって、下記の様なプログラムの性質をチェック可能になります。

**パラメータの渡し方(BY VALUE 他)**
**パラメータの省略可否**
**パラメータの型**
**呼び出し規約**

＜使用例＞

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. proto-program-name-1  IS  PROTOTYPE.
DATA  DIVISION.
LINKAGE  SECTION.
01  param-1   PIC  ...
01  param-2   PIC  ...
01  return-val   PIC  ...
PROCEDURE  DIVISION   USING  BY  VALUE  param-1,  OPTIONAL   param-2
                              RETURNING  return-val.
END PROGRAM  proto-program-name-1.

IDENTIFICATION  DIVISION.
PROGRAM-ID. main-program-name-1.
ENVIRONMENT  DIVISION.
CONFIGURATION  SECTION.
REPOSITORY.
      PROGRAM  proto-program-name-1.
PROCEDURE  DIVISION.
      CALL  "proto-program-name-1"  USING  BY VALUE argument-1,  OMITTED
                                    RETURNING return-code.
```

＜説明＞

**上記の翻訳グループは２つの翻訳単位で構成されています。第１の翻訳単位はプロプラム原型定義であり、第２は従来通りのプログラム定義(メイン・プログラム)です。メイン・プログラムの REPOSITORY 段落では、何処かで定義されたプロプラム原型情報を参照する事を宣言し、このプログラム内で記述されている CALL 命令でのプログラム性質のチェックを可能にしています。**

（７）PROGRAM-POINTER 型データを定義出来ます。このデータ項目は CALL,  INVOKE 等で使用でき、主にオブジェクト指向機能と結びついています。
OBJECT REFERENCE 型データも同じ目的のものです。

（８）連絡関係の例外
EC-PROGRAM... EC-ARGUMENT... EC-FLOW など。

## 14.2 言語間連絡の拡張について

ここで説明している連絡機能の殆どは、COBOL 言語内だけの機能にとどまりません。COBOL 間での連絡機能も勿論であり、その機能範囲が広がった事も重要な点ですが、他言語との連携の可能性を広げたというポイントも見逃せません。特に BY　VALUE 方式によるパラメータ渡しや、呼び出し規約の指定は現在幾つかの他言語で利用されている手法であり、COBOL にこの様な機能が取り込まれた結果として、呼び出される側、または呼び出す側としても COBOL 以外の言語で作ったモジュールを想定する事が容易になります。また既に一定のレベルで実装し始めている処理系も有ります。

またオブジェクト指向機能の詳細についてはここでは説明しませんが、この機能も他言語連絡も見据えて来ています。当然仕様策定の上では既存のオブジェクト指向言語(C++,Java 等)の仕様も踏まえ、同等の機能や考え方が多く取り入れられている事は言うまでも有りません。これは仕様的な整合性を取るという意味ばかりでなく、実装に当たって相互の連携利用の道を開くものと言えます。つまり C++から COBOL で作ったオブジェクトを呼び出したり、COBOL プログラムから Java　Beans の機能を利用したりする可能性が予測されます。

特に CORBA(Common Object Request Broker Architecture)や「.net」等最近の業界動向の中でフレームワーク（枠組み、構成）の統合、連携の方向性が強く出てきており、これら枠組みの中に COBOL という言語開発手法も浸透して行く可能性を予感させるものであると言う事も出来ます。

# 15 章 標準算術演算と 31 桁への拡張

この章では算術演算の３１桁拡張と、標準算術演算について説明します。

## 15.1 算術演算の３１桁拡張について

COBOL 言語は従来でも 18 桁の誤差無し整数演算を定義していました。これは他言語と比較しても精度の高いものであり、COBOL 言語の特徴の１つです。18 桁でも相当大きい数字だと思いますが、これは兆の上の京という単位で 2 桁を表現出来る事になります。COBOL2002 ではこれが 31 桁に拡張されます。これは京の上の、上の、もう１つ上の「壌（じょう）」を３桁表現出来ます。こんな数字を何に使うのでしょうか？しかし科学技術的な範囲も含めて十分余裕のある桁数を扱える事は良い事ですね。やはり時代の流れが、より高度な情報処理技術を求めている、と言う事が出来るでしょう。

31 桁算術機能は、下記の様な場所で利用出来ます。

（１）数字定数
（２）数字項目、数字編集項目の定義
（３）算術演算およびその中間結果(注)
（４）MOVE 命令他、転記の規則が適用されるところ(注)
（５）浮動小数点データを表現する基数、べき数など

(注)演算の中間結果等については、次項の標準算術演算が関係します。

## 15.2 標準算術演算とは

標準算術演算とはどの様なものでしょうか。算術演算の性質として、どの処理系でも同じ様に動作すべき範囲と、場合によっては異なる処理系毎に異なった結果を生むかも知れない範囲が有ります。算術演算が異なった結果を生じる、と言うと奇異に聞こえるかも知れませんが、例えば演算途中の中間結果の精度等がこれに当たります。

過去の規格に於いては殆ど共通的な範囲のみを定義し、処理系毎に異なるかも知れない処理範囲については明確な規定は有りませんでした。この為逆に演算結果が曖昧になったり、どの様な結果をもたらすかの定め自体がなかったりと、算術演算についての曖昧性が無視出来ない問題になっていたと言えます。

COBOL2002 では算術演算に関する規定を、次の３種類に分類しました。
（１）共通算術演算（注）
（２）処理系固有算術演算
（３）標準算術演算

つまりどのコンパイラに於いても、この様に動作しなければならない、と定めた範囲が（１）、処理系によって異なるかも知れない扱いの部分が（２）です。ですが（２）の部分についても何も指標が無いと前の規格と大差なくなってしまうので、標準的にはこうある事が望ましい、或いは１つの処理案としてこの様な事が考えられる、という事を纏めたのが（３）の標準算術演算であると言えます。

（注）「共通算術演算」という呼び方はこの章の解説の為に、便宜的にこう呼んだもので有って、規格の中には有りません。

## 15.3　演算種類毎の規定概要

次に各規定について概観します。

（１）共通算術演算
① 算術演算の中で括弧を、算術演算の優先順を明示・変更する為に使用出来る。括弧内の演算は括弧外の演算に先立って行われる。括弧が入れ子になっている場合、最も内側の組から評価される。
②同一レベルの演算では種類によって、次の優先順位が適用される。

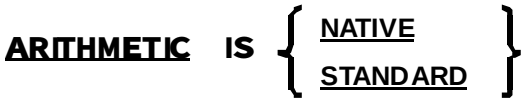
１番目：単項の + と -
２番目：べき乗 **
３番目：乗算 * と除算 /
４番目：加算 + と減算 -

③括弧無しに同じ順位の演算が連続する場合は、左から右へ向かう順序で演算が行われる。括弧を使用するのは、同じ順位の演算が連続している場合の論理的な曖昧さを排除するため、通常の実行順序を変更した順序を指定するため、あるいは通常の順序を強調して明白にする為である。
④算術式中で可能な記号、演算子、作用対象の組合せ（注）
⑤ べき乗の評価規則（注）
（注）詳細は省略

（2）処理系固有算術演算

標準算術演算規則を採用するのか、処理系固有算術演算とするのかの範囲のうち、その処理系で固有の処理に関して実現者は、何が固有で有ってその範囲ではどの様に動作するのかを明示しなければなりません。

見出し部、OPTIONS 段落で次の標記が出来ます。




省略時は NATIVE(処理系固有)です。次の様な演算では上記指定に関わらず常に、処理系固有と見なされます。

①usage of binary-char, binary-short, binary-long, binary-double,float-short, float-long, あるいは float-extended データ項目を含む演算
②べき乗や組み込み関数で近似が使用される場合
③翻訳指令の FLAG-NATIVE-ARITHMETIC が有効な場合

（3）標準算術演算

標準算術演算が有効な場合、下記の演算動作が定義されます。標準算術演算は最も共通的な演算方式で、予測可能、合理的また可搬性のある結果をもたらす事が目的です。ここで可搬性のあるとは、異なる処理系間でも同じ結果を生じる事です。

①標準中間データ項目

標準中間データ項目というのは算術演算の中間結果を格納する為の、数字データ項目であってその内部表現は、処理系作成者が規定する。

②標準中間データ項目が取りうる値は一意なゼロか、抽象的な、符号付き

正規化 10 進浮動小数点形式で、絶対値が 10E-100 以上 10E99 - 10E67 で精度が 10 進 32 桁の範囲である。標準中間データ項目は明示的に指定され無い限り、切り捨てや四捨五入によって 32 桁未満になる事はない。

③ゼロでない正規化された標準中間データ項目は、小数点の左に数字を持たず、小数点の右の桁がゼロ以外の数字である。

④標準中間データ項目は次の場合に 31 桁に四捨五入される。

比較のとき、関数の引数になるとき、及び ROUNDED 指定のない結果データに転記されるとき。

この規則は必要以上に四捨五入が行われる事を排除する。

⑤加算では作用対象の厳密な和を 32 桁へ四捨五入し、正規化し、標準中間データ項目へ格納する。

⑥減算は次の通りとする。

（ 作用対象１ +（- 作用対象２ ））

⑦乗算、除算

加算と同様である。

⑧べき乗

省略。

## 15.4　標準算術演算の性質

それでは標準算術演算は規格の中で提案されているので、全てこれに従った実現方式が理想という事でしょうか。これはそうでは有りません。標準算術演算はあくまで指針であって、必ず従わなければならない物でも無くまた、従う事が理想という物でも有りません。
例えばある処理系作成者が標準算術演算と異なった、中間データの精度３５桁をサポートしよう、と考えたとします。その精度が必要と思われる根拠があるので有れば、それは立派な態度であり、その点での機能は標準算術演算に従った場合よりも優れているという見方も出来ます。また別の処理系作成者が作成者側の都合から中間データの精度は 31 桁しか確保出来ないとしましょう。その場合でもその様な仕様である事を明示すれば仕様準拠という観点では問題ないのです。

ただどちらの場合でも絶対必要なのは、その様な処理系固有算術演算を採用している事を処理系の仕様書等で明示する事であり、これが守られている限りどちらの処理系も正しく COBOL2002 に対応した物であると言う事が出来ます。

但し標準算術演算の中でも「合理的である」とか「予測可能である」等は重要な標準であるので、守るべきである、とするのが処理系提供者の正しい姿勢と言えるでしょう。

# 16 章 その他

## 16.1 POSIX(注)のロケールに対応した地域・文化固有機能

本解説の第４章で「漢字等の多オクテット文字処理機能」の説明が有ります。その部分を読んだ読者の多くは、日本語を扱う上での機能を想定して読んだかと思いますが、この機能は日本だけの為では有りません。日本は勿論ですがアジアを中心とした２バイト文字文化圏を対象としていると供にそれだけではなく、文字コードは１バイト内に収まるが、これまでの標準化ＣＯＢＯＬの中では既定の無かった非英語圏のヨーロッパ初め諸国での利用も踏まえた、国際化規格になっています。この節では新規格の、国際化対応、地域・文化固有機能について解説します。

（注）POSIX とは...(Portable Operating System Interface for UNIX)
IEEE(アイトリプルイー、米国電気電子学会)によって定められたものです。後にこの規格は ISO(国際標準化機構)によって国際規格として定められています。UNIX 標準化に端を発した、システム機能の標準化規格であり、この仕様に準拠する事でポータビリティ確保に効果があると考えられます。米国連邦政府が調達規格に採用した事も有って、各方面で準拠すべき規格と意識、対応されています。

## 16.2 国際化、他国語対応とは何か

元々国際化、他国語対応とはどの様な仕組みの事でしょうか。
（１）注釈、定数で各国語が扱える
　→従来も殆ど問題は無かった。
（２）プログラミング言語内で使用可能な文字の拡張
　→開発の生産性・信頼性・保守性の向上
（３）国際化プログラム開発が容易化
（４）符号系変換機能があり、ファイル変換等が省力化できる

上記は扱える文字コードセットや、符号系に関わる機能が殆どですが、国際化とは下記の様な機能も含みます。
（５）文字の照合順序
（６）通貨記号、金額の単位

（7）数字の表現
（8）日付・時刻の表現など

この様に文化的（国・地域・言語・習慣による違い）な相違も標準規格において考慮することが重要との認識から、マルチオクテット文字処理機能をより一般化し、"国際化機能"と呼んでいます。

ちなみに新規格の言葉では有りませんが、国際化の事を「i18n」と呼ぶそうです。これは国際化を表す internationalization の先頭「i」と最後の「n」、そしてその間に 18 文字のアルファベットが有る事からです。また関連用語として、多言語化(multilingualization) ： m17n 、地域化(localization) : l10n と言うのも有ります。これらの言葉は COBOL 言語以外の国際化の話題中でも比較的広く使われている様です。

# 16.3 ロケール

地域、国際化の指定の為に、言語規格として次の様な機能を持たせています。この指定機能の事を地域固有仕様（ロケール）と呼びます。

（1）ロケールの分類
ロケールは次の様な分類として定義されています。

| 分類 | 意味 |
|---|---|
| LC_COLLATE | 比較順序 |
| LC_CTYPE | 文字分類と大・小文字変換規則 |
| LC_MESSAGES | 情報、対話メッセージと応答の形式 |
| LC_MONETARY | 通貨表現 |
| LC_NUMERIC | 数値表現、形式情報(小数点文字など) |
| LC_TIME | 日付、時刻表現形式 |
| LC_ALL | 上記ロケールの全てを含むカテゴリ |

（2） 特殊名段落
特殊名段落で次の指定が出来ます。

**LOCALE** { 分類名 / 定数 } **IS 地域固有仕様名**

（3） SET 命令で地域固有仕様の値を設定又は参照できます。
（4）EC-LOCALE... ロケール関連の例外です。

（５）新設の組み込み関数
BYTE-LENGTH, CHAR-NATIONAL, LOCALE-COMPARE, LOCALE-DATE,
LOCALE-TIME,NATIONAL-OF
（６）機能拡張された組み込み関数
LOWER-CASE, UPPER-CASE

# 16.4 国際化の具体例

具体的な例として国際化機能を持ったCOBOLコンパイラとはどの様なものでしょうか。

例えば貴方は有るユーザ向けに開発済みのシステムを持っていて、これは英語のみを使っていたとします。このシステムを日本語化(例えば SJIS)する場合、どの様な修正が必要になるでしょうか。

まず第１に画面表示メッセージや、印刷用定数等は書き換える必要が有ります。修正量が多ければその部分はプログラムの外に切り出して、メッセージファイル化しておく事は、後々にも役立ちます。
英語機能の部分も残して置きたければ「コンパイラ指示機能」による、コンパイル時選択行の指定機能が有用です。また幾つかの場所では日本語（２バイト）データとして扱いたい場合が有るかも知れません。その場合は、データ定義を日本語項目(PIC N)とします。手続き部はどうでしょうか。
実は手続きは基本的に何ら改造を必要としないのです。扱うデータが英語なのか、日本語なのかによって取るべき動作が変わる命令は数多く有ります。例えば次の STRING 命令を例に、説明します。

```
STRING data-1 DELIMITED BY SPACE
       data-2 DELIMITED BY SIZE
  INTO data-3
  WITH POINTER counter-1
```

この時に data-1,2, および 3 が１バイト英数字項目(PIC X)の定義で有れば、この命令動作は１バイト文字単位での動作になります(POINTER 句で指定されたカウンタ・データは、１バイト文字としての文字数で扱われる)。
そして data-1,2, および 3 が日本語項目(PIC N)の場合は、２バイト文字単位での動作、すなわちカウンタ・データは、２バイト文字としての文字数で扱われる訳です。これにより日本語を扱うプログラムで有っても、１バイト文字のみ扱っていたものから、手続き部

に関する改造は要らない事になります。

そしてこのソース・プログラムをコンパイルする時の操作として、英語部分ではなく日本語部分を有効にする、日本語メッセージファイルを利用する、また日本語項目の中身はＳＪＩＳとして扱う、等の指示情報をコンパイラに与える必要が有ります。その為にコンパイラに次の指定を入力します。

LC_ALL　　Japanese_Japan.932

これは現在 Windows で用いられている方式をベースに想定した、１つの例です。下線の前は言語、下線とピリオドの間が国、最後がコードページと言う符号化文字集合・方式を表しています。

それでは次の段階として、上記プログラムを別国語対応させる事を想定します。例えばこのシステムを利用しているユーザー企業が、中国・香港支店を開く事になったとします。
同じアプリケーションを、日本だけでなく中国の人も使いたい訳です。勿論画面表示のための定数や印刷用ヘッダ等は中国語の物に差し替える必要が有ります。上で説明したとおり、メッセージファイル等に分離して有れば、プログラムの修正は殆ど無いでしょう。また PIC　N 等の定義は修正不要です。
そしてプログラムをコンパイルする時に、次のロケールをコンパイラに指示します。

LC_ALL　　Chinese_China.950

最後の「950」は代表的な中国語文字セットの１つである「BIG5」を示しています。同時に中国での代表的な通貨記号や日付、時刻表現等も上記指定だけで、自動的に行われると思って良いでしょう。つまり本当に僅かな改修のみで他国語対応のシステム構築が出来る事になります。もちろん具体的な実装においてはもう少し複雑な部分、考慮を要すべき事柄等が有るかも知れません。ですが基本的な事は、ここで述べた部分で表現可能です。
ますますグローバル化が進んでいる社会、そしてその中で求められる情報技術（ＩＴ）の上で、今後国際化の重要性が増加していく事は間違いないと考えられます。その様な要求に応える、規格になっていると言えるのではないでしょうか。

# 16.4 国際化のまとめ

国際化の説明の最後に、規格化に携わった方々の論文から引用する形で、国際化規格制定の上でどの様な点を考慮し、苦心したのかを国際化の流れ、意義等を汲みながら見ておきたいと思います。COBOL2002 国際規格の中の国際化機能は特に、日本から標準化に参加した方々の努力によって実った成果です。標準化に労された方々に心から感謝致します。

「COBOL 日本語機能の一般化による、国際規格化提案と国際化機能の開発」

＜標準化前の状況＞

**日本語化機能は実装が先行していた。各社が未統一のまま実装した為に非関税障壁と海外から批判される危険性が有り日本のみでの標準化は避けるべき状況であった。**

このため国内だけの JIS にはせずに、日本から国際規格として提案することにした。

＜主要課題＞

**世界各国の賛同が必要である。**

日本固有の地域性を排除した公平な汎用仕様にしなければならない。

**既存の日本語機能を使用した膨大な資産からの移行性を確保する必要がある。**

＜仕様作成方針＞

（1） 使用可能な文字を世界各国・地域の文字に拡張する。

（2） 世界の多くの符号系に対応可能な枠組を設定する。

（3） 既存資産との共存を重視し、符号系切替のシフトコードの存在を意識しない方式とする。

（4）文字データ型として各国文字項目を追加し、その機能（定義と操作）は既存の英数字項目と同等とする。

（5）各国・地域に適用可能な国際化プログラムの記述が可能な仕様とする。

（6）既存の日本語仕様機能の中で使用頻度の高いものは国際規格仕様の核となる構文として採用する。

（7）日本特有の機能や名称、時代制約的な機能、ハードウェア依存性の高い仕様を排除する。

1 つは、世界各国の賛同を獲得するために日本という地域性を排除することである。このため、中国や韓国などのマルチオクテット文字を使っている国・地方はもちろんのこと、アルファベット 26 文字以外のヨーロッパ各国文字（ドイツ語のウムラルトやロシア文字など）を含む多種多様な文字とそれらが規定される多くの符号系を対象とする"普遍的な枠組み"を定めることにした。

2 つ目の課題は、利用者のために日本語機能を用いた既存資産の移行性を確保することである。

＜結果的に提案した国際化規格案＞

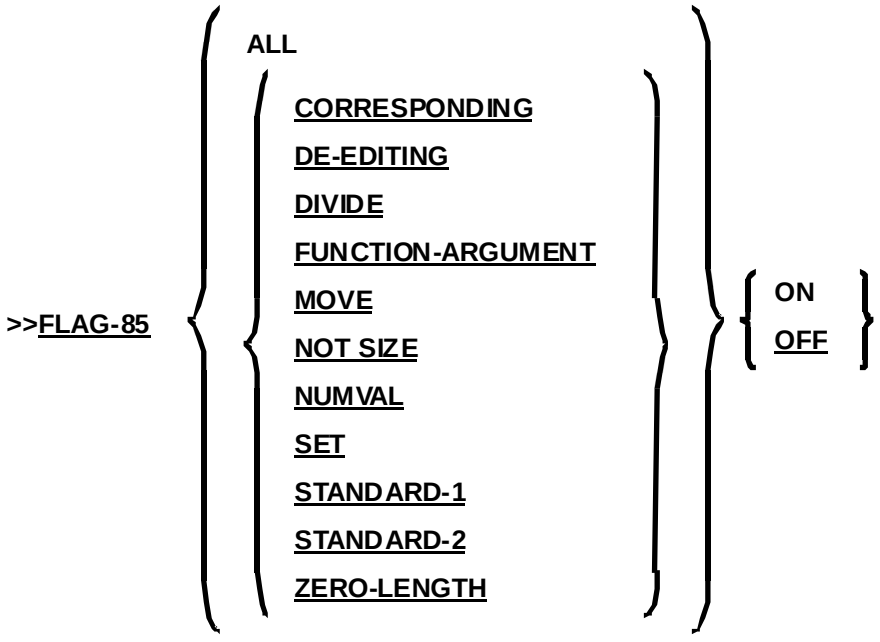
（1） 対象とする文字を、中国、韓国、タイ、ロシア、ドイツ、北欧、東欧等アルファベット 26 文字以外の世界各国の文字に拡張するとともに、多くの符号系に対応可能な普遍的な枠組を設定する。

（2） 既存の文字型（英数字項目）に加え、漢字などのマルチオクテット文字を扱う国別文字（national character）項目を新設し、機能は英数字項目と同等とする。特に同じ業務プログラムを各国ごとに適用するときには、データ定義だけを修正しロジックの変更が不要で済むように国際化仕様を設定する。

（3） 国内で使用頻度の高い日本語処理機能を選定し、それを国際化機能の核となる仕様として採用することにより、既存資産から国際規格仕様への移行性を実現する。

# 16.5 既存プログラムとの互換

COBOL2002 では「FLAG-85」コンパイル時指示機能によって、前規格との互換性に問題が出るかも知れない機能について、警告メッセージを表示する方法が提供されます。



（1）CORRESPONDING

CORRESPONDING 句指定の ADD, MOVE, または SUBTRACT 命令に於いて、定数以外の添え字付けが有る時、警告する。

（2）DE-EDITING

数字編集項目の逆編集が必要になる MOVE 文を、警告する。

（3）DIVIDE

REMAINDER 付きの DIVIDE 文で、商のデータ項目が符号無しであり、除数または被除数が符号付きの場合警告する。

（4）FUNCTION-ARGUMENT

関数 RANDOM の引数に算術式を指定し、その結果を組み込み関数の引数にしようとする時、関数 RANDOM に警告を出力する。

（5）MOVE

英数字の定数又はデータが数字項目に転記され、その桁数が 31 桁を越えるとき警告を出力する。

（6）NOT SIZE

算術命令で NOT SIZE ERROR が指定され、SIZE ERROR 句の指定が無い場合警告する。

（7）NUMVAL

組み込み関数 NUMVAL と NUMVAL-C は警告する。

（8）SET

条件設定の SET 文が可変長集団項目を対象にしている場合、警告する。

（9）STANDARD-1

特殊名段落の ALPHABET 句に STANDARD-1 の指定がある場合、警告する。

（10）STANDARD-2

特殊名段落の ALPHABET 句に STANDARD-2 の指定がある場合、警告する。

（11）ZERO-LENGTH

長さゼロになりうるデータ定義を警告する。

（12）ALL

上記（1）～（11）を全て含む。

# 保護用紙