第2章　データの構造

指導目標

本章では、ＣＯＢＯＬ言語におけるデータ記述の概念を理解させる。

オープンシステム化がソフトウェア、ハードウェア分野で叫ばれているが、これを実現するためには異機種のコンピュータを接続する必要がある。接続手段にはいろいろあるがデータ形式から観た互換性も重要である。
コンピュータのデータの内部表現形式は機種により差異があるが、これらの差異をプログラミング上でなくすために言語規格でどのように定義されているかを指導する。

ＣＯＢＯＬ言語の「標準データ形式」は、コンピュータの基数系によらず数値は１０進数を用いて表現する。また、文字データ項目はＣＯＢＯＬ文字集合の全てを用いることでデータの物理的特性を切り離している。
この切り離されている部分の知識を指導することで、ハードウェアとソフトウェアの理解をより深めることがプログラミング技法の向上につながることと、互換性のあるプログラムとは何かを理解させる。

内容のあらまし

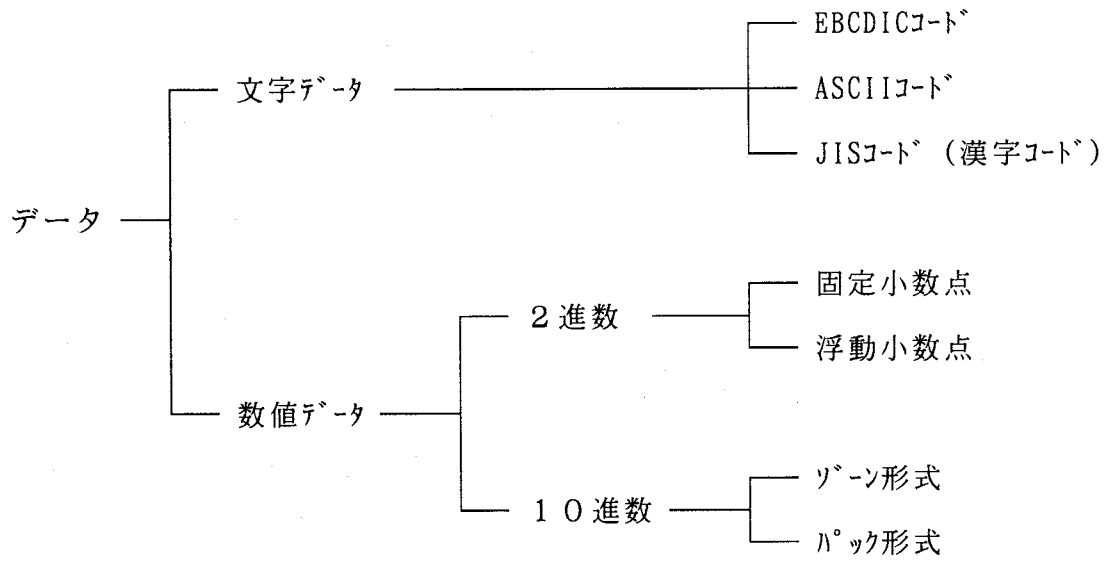
内　　容	説　明	議　論	机上実習	計算機実習
データの型	ｺﾝﾋﾟｭｰﾀの内部データ形式とCOBOLのデータの字類の概念を説明する。	一般的なﾃﾞｰﾀの型と使用法について議論する。		ﾃﾞｰﾀの型とﾊｰﾄﾞｳｪｱの知識が整理できるように実習する。
データの発生源	COBOLでのﾃﾞｰﾀ発生源と各部の役割を説明する		ﾃﾞｰﾀの発生源を詳細に列挙させ整理する	
データの定義	DATA DIVISIONの各SECTIONのﾃﾞｰﾀ定義の役割と機能の概要を説明する。			
データ記述項	ﾃﾞｰﾀの論理的実体の定義についてﾃﾞｰﾀ記述項を説明する。			
基本項目と集団項目	ﾚﾍﾞﾙ番号の種類とﾚﾍﾞﾙの概念でﾚｺｰﾄﾞが構成されることを説明する。		ﾃﾞｰﾀ構造の例をあげ集団項目の記述方法を実習する。	
ＰＩＣＴＵＲＥ句	ﾃﾞｰﾀ項目の性質と編集の形式の意味を定義するPICTURE句について説明する。			編集項目を重点に記述と機能を実習する
ＵＳＡＧＥ句	ﾃﾞｰﾀの内部表現をﾃﾞｰﾀの型について説明を行いﾌﾟﾛｸﾞﾗﾑでの使用法を理解させる。	処理に応じた効率的なﾌﾟﾛｸﾞﾗﾑとは何かを議論する。	数値をあげて2進数、16進数表現に変換させる。	
ＶＡＬＵＥ句	ﾃﾞｰﾀ項目の初期値の与え方とその利用法について説明する。			VALUE句の記述規則をｺﾝﾊﾟｲﾙを通して実習する。

内　　容	説　明	議　論	机上実習	計算機実習
REDEFINES句	ﾃﾞｰﾀ項目の再定義と利用法について説明する。	再定義の必要性について議論する。	例をあげて利用法を	
配列	処理手続きでの配列（表）の有効性について解説を行いCOBOLでの表の定義と参照方法について説明する。	配列を使用しない場合の違いを議論する。	図の例をあげ配列が使いこなせるようにする。	添字と指標の使い分けを実習する。

第２章　データの構造

２．１　データの型

コンピュータで使用されるデータの概要を分類すると、次のようになる。



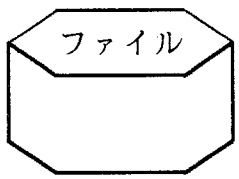
ＣＯＢＯＬでは、データ項目を五つの項類（category）に分類する。五つの項類は、「英字」、「数字」、「英数字」の三つの字類（class）に分けられる。

項目のレベル	字類（class）	項類（category）	PICTURE文字列の例
基本項目	英字	英字	A(4)
	数字	数字	9(3)　S9(4)　99V9
	英数字	数字編集 英数字編集 英数字	ZZ9　---,--9　+9(3) AABAA　XX/XX/XX X(10)　9X9X
集団項目	英数字	英字 数字 数字編集 英数字編集 英数字	A(4)AA 99　PP9(3)　9(3)PP **,**9　+++,+++.++ XXOXX　X/B/0 AX9　9(5)X(5)

注：項類「英字編集」はない。PICTURE　AABAABAAは「英数字編集」である。

ＣＯＢＯＬでのデータ定義と内部表現形式の例は、次のとおり。

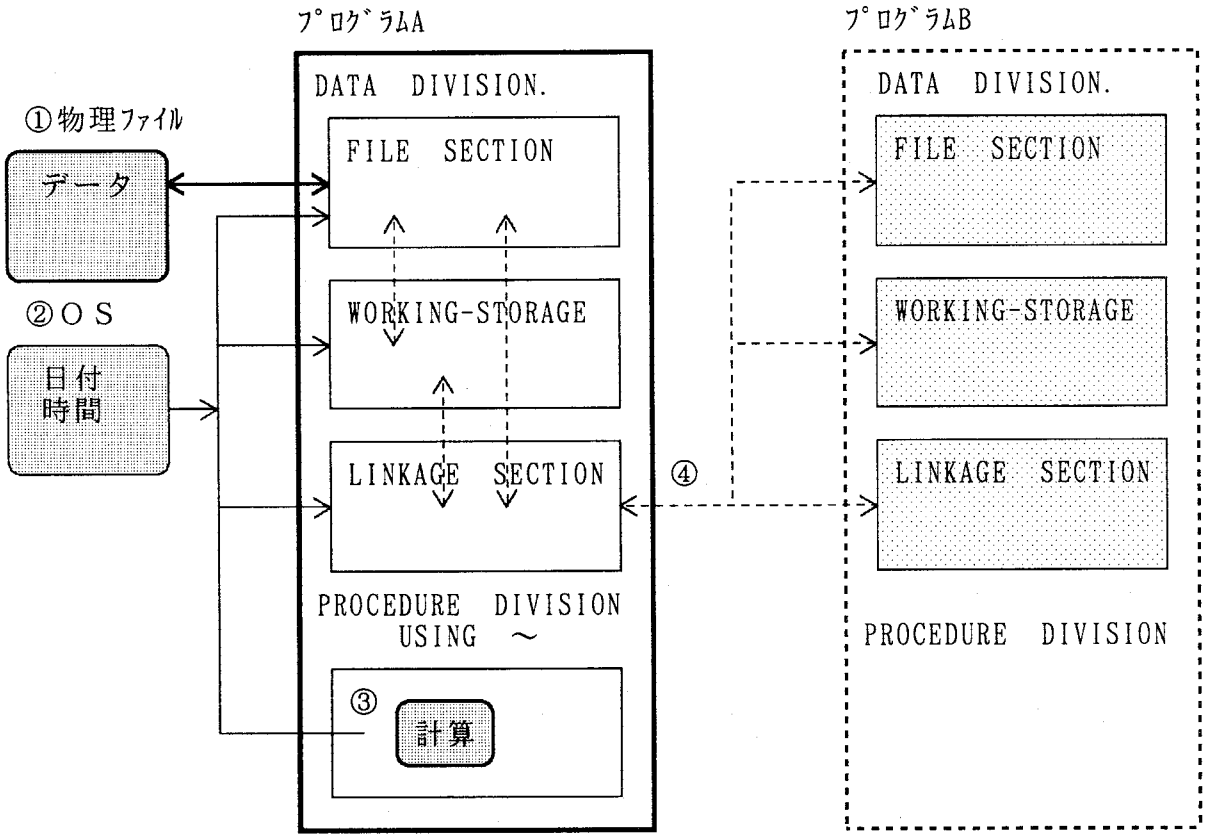
```
IDENTIFICATION DIVISION.
PROGRAM-ID.   DUMP.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
    SELECT DMPFILE ASSIGN TO "DUMPDATA".
DATA DIVISION.
FILE SECTION.
FD  DMPFILE.
01  DMPREC.
    03  AN     PICTURE  X(4).
    03  ZD     PICTURE  9(4).
    03  ZDS    PICTURE S9(4).
    03  PD     PICTURE  9(7)  USAGE PACKED-DECIMAL.
    03  PDS    PICTURE S9(7)  USAGE PACKED-DECIMAL.
    03  BN     PICTURE  9(8)  USAGE BINARY.
    03  BNS    PICTURE S9(8)  USAGE BINARY.
PROCEDURE DIVISION.
開始.
    OPEN OUTPUT DMPFILE
    MOVE "ABCDEF" TO AN
    MOVE  12      TO ZD, ZDS
    MOVE  12      TO PD, PDS
    MOVE  12      TO BN, BNS
    WRITE DMPREC
    CLOSE DMPFILE.
終了.
    STOP RUN.
```



ファイルの内容（ＪＩＳコード、１６進数形式）

```
                                                    (文字形式)
41 42 43 44 30 30 31 32 30 30 31 32 00 00 01 2F    ABCD00120012.../
   ＡＮ       ＺＤ       ＺＤＳ       ＰＤ
00 00 01 2C 00 00 00 0C 00 00 00 0C                ....,.........
   ＰＤＳ      ＢＮ       ＢＮＳ
```

２．１．２　ＣＯＢＯＬに於けるデータの発生源



注：　通信機能、特殊ﾚｼﾞｽﾀ（LINAGE-COUNTER）を除く。

①物理ファイルからの入力データ

```
READ　ﾌｧｲﾙ名　INTO　ﾚｺｰﾄﾞ作業域　AT　END　GO　TO　終了.
```

②オペレーティング・システムからのデータ

```
ACCEPT　日付　FROM　DATE
ACCEPT　時間　FROM　TIME
```

③手続き部での計算

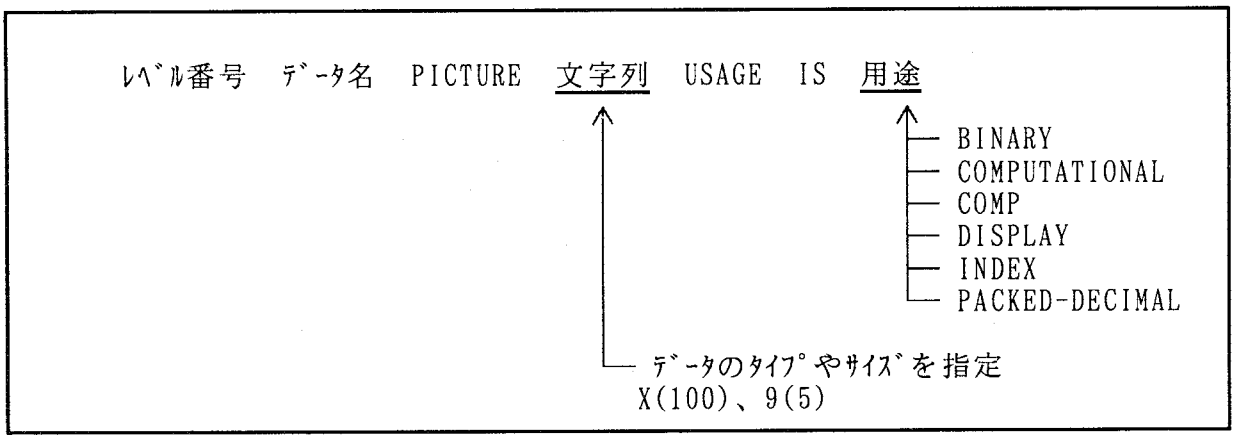
```
COMPUTE　税金　=　価格　*　0.03.
```

④他のプログラムからのデータ

```
PROCEDURE　DIVISION　USING　ﾃﾞｰﾀ名1　ﾃﾞｰﾀ名2.
```

２．１．２　定義の概要

ＣＯＢＯＬでは、データ項目を次のように定義する。



浮動小数点ﾃﾞｰﾀ、漢字ﾃﾞｰﾀの定義については、各々ＣＯＢＯＬコンパイラにより異なる。

言語別のデータ形式の定義は次のとおり。

	ＣＯＢＯＬ	Ｃ	ＦＯＲＴＲＡＮ
文字ﾃﾞｰﾀ	ﾃﾞｰﾀ名 PIC X(ｻｲｽﾞ) DISPLAY	char　変数名[ｻｲｽﾞ]	CHARACTER変数名*ｻｲｽﾞ CHARACTER*ｻｲｽﾞ変数名
数値ﾃﾞｰﾀ 2進数 （整数）	ﾃﾞｰﾀ名 PIC S9(桁数) BINARY	short　変数名 int　変数名	INTEGER*ｻｲｽﾞ　変数名
（実数）		double　変数名	REAL*ｻｲｽﾞ　変数名
10進数 （ﾊﾟｯｸ）	ﾃﾞｰﾀ名 PIC　S9(桁数) PACKED-DECIMAL		
（ｿﾞｰﾝ）	ﾃﾞｰﾀ名 PIC　S9(桁数) DISPLAY		
ﾎﾟｲﾝﾀ ﾃﾞｰﾀ		char　*変数名 int　*変数名	
論理ﾃﾞｰﾀ			LOGICAL　変数名
複素数 ﾃﾞｰﾀ			COMPLEX　変数名

注：*ｻｲｽﾞはＪＩＳにはない。

２．２　データの定義

ＣＯＢＯＬでは手続き部で使用するデータ項目を「DATA　DIVISION」で記述する。

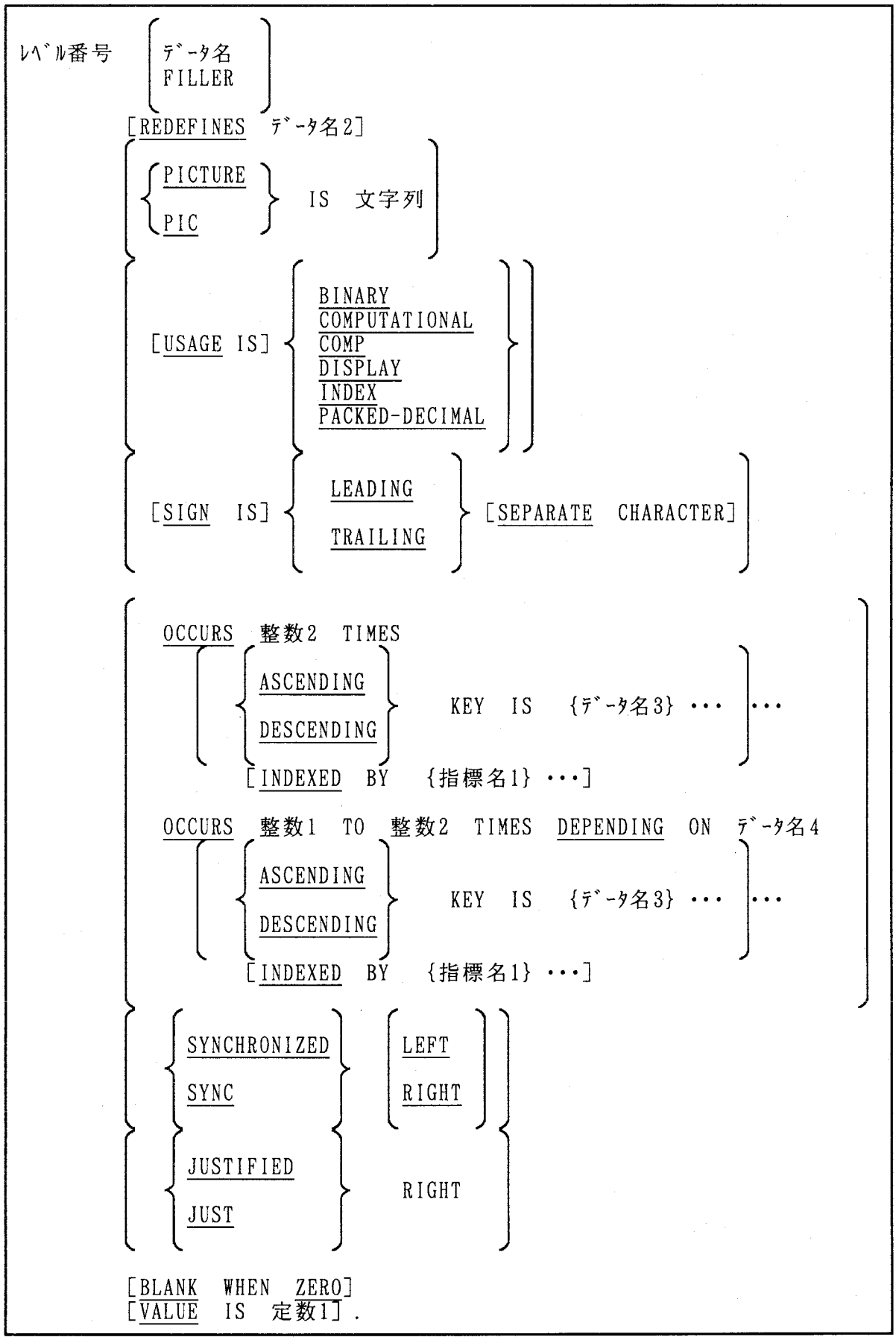
DATA　DIVISIONの構成は、次のとおり。

```
DATA  DIVISION.
  FILE  SECTION.
    ファイル記述項 －　ラベル、ブロック情報を記述
      FD  ﾌｧｲﾙ名  LABEL RECORD IS  STANDARD
                  BLOCK  CONTAINS  100  RECORDS.
    レコード記述項 －　レコードの構造とデータ種類を記述
      01  学生.
          02  氏名          PIC  X(20).
          02  生年月日      PIC  9999/99/99.
          02  住所          PIC  X(80).
          02  教科          PIC  99  OCCURS  10  TIMES.
  WORKING-STORAGE  SECTION.
    内部データ記述　－　テーブル、中間データの作業領域を記述
      01  年月日.
          02  年            PIC  9999  VALUE  1994.
          02  月            PIC  99    VALUE  12.
          02  日            PIC  99    VALUE  1.
      01  テーブル          PIC  X(5)  OCCURS  200  TIMES  INDEXED  BY  X.
  LINKAGE  SECTION.
    外部データ記述　－　別プログラムで定義されているデータの参照
      01  処理指示          PIC  X.
      01  結果              PIC  9(3).
  COMMUNICATION  SECTION.
    通信記述項　－　　ＣＯＢＯＬ通信機能のデータについて記述
  REPORT  SECTION.
    報告書記述項　－　報告書作成機能のデータと制御情報を記述
```

注：　ファイル記述項については、「第５章ファイル入出力」を参照のこと。

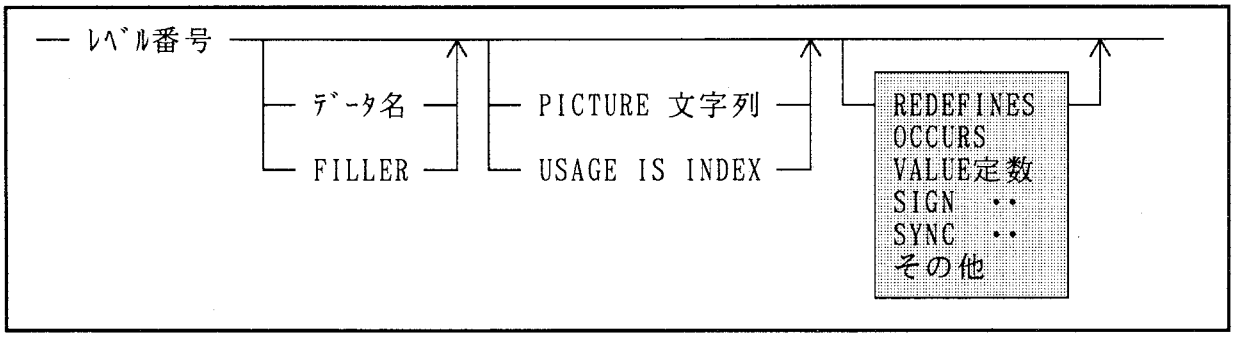
２．３　データ記述項

データ記述項は、DATA　DIVISIONでデータの名前やデータの属性を定義する。



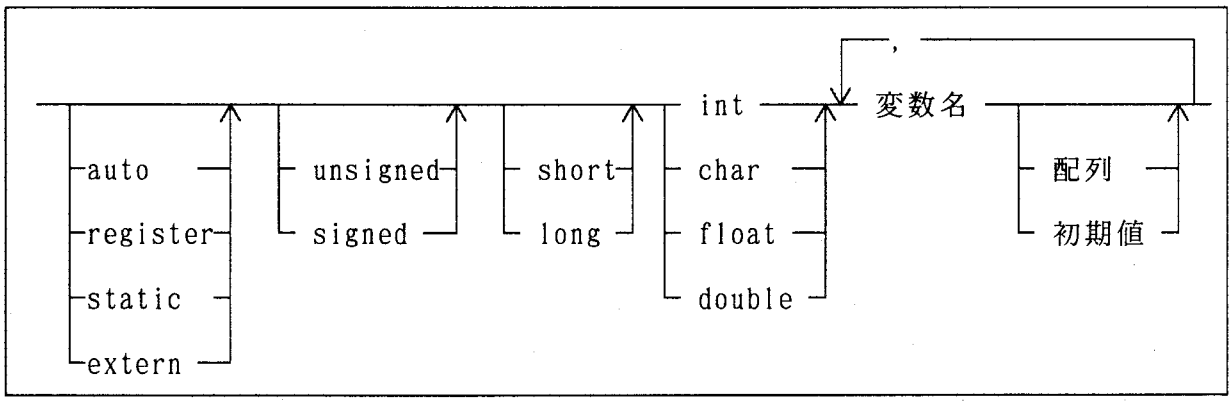
データの定義は、「レベル番号」、「データ名／FILLER」の順に記述し、残りの句の順序は任意であるが、基本項目には「PICTURE」句または「USAGE　IS　INDEX」を含むこと。
また、集団項目は「レベル番号」が必須であり、「PICTURE」句や「USAGE　IS　INDEX」は指定できない。

ＣＯＢＯＬデータ記述の基本形は次のとおり。



レベル番号は、レコードの階層構造中のデータ項目の位置を表す。
レベル番号は、０１～４９、６６、７７、８８が指定できる。
レベル番号の０１～０９は一桁（１～９）で記述できる。
レベル番号０１は、最初の記述項であり、４９が最後の記述項である。

Ｃ言語のデータ定義の概要



手続き部で参照するデータ項目には、データ名として名前を付ける。
ＦＩＬＬＥＲ（無名項目）は手続き部で明示的に参照しないが、データ構造を定義する場合に用いられる。

```
DATA  DIVISION.
    :
01  ﾚｺｰﾄﾞ名.                              ←参照可能
    02  FILLER    PICTURE  X(5).          ←参照不可
    02  ﾃﾞｰﾀ名1  PICTURE  X(5).          ←参照可能
    02  FILLER.                           ←参照不可
        03  ﾃﾞｰﾀ名2  PICTURE  9(3).       ←参照可能
        03  FILLER   PICTURE  9(3).       ←参照不可

PROCEDURE  DIVISION.

    MOVE  SPACE  TO  ﾚｺｰﾄﾞ名
    MOVE  123    TO  ﾃﾞｰﾀ名2.

    MOVE  "ABCD" TO  FILLER      ←誤り
```

データ名またはＦＩＬＬＥＲを省略すると、そのデータ項目はＦＩＬＬＥＲが指定されたものとして扱われる。

```
01  ﾚｺｰﾄﾞ名.                              ←参照可能
    02           PICTURE  X(5).            ←参照不可
    02  ﾃﾞｰﾀ名1  PICTURE  X(5).          ←参照可能
    02           .                         ←参照不可
        03  ﾃﾞｰﾀ名2  PICTURE  9(3).       ←参照可能
        03           PICTURE  9(3).       ←参照不可
```

Ｃ言語の構造体の定義例は、次のとおり。

```
  struct  構造体ﾀｸﾞ  {
                     int      ﾃﾞｰﾀ名1;
                     char     ﾃﾞｰﾀ名2;
                     char     ﾃﾞｰﾀ名3;
                     struct   構造体ﾀｸﾞx   構造体ﾃﾞｰﾀ名x;
                     int      ﾃﾞｰﾀ名4;
                     };

  struct  構造体ﾀｸﾞ   構造体ﾃﾞｰﾀ名;
```

２．４　基本項目と集団項目

レベル番号（01、02～49、66､77､88）でデータ構造を定義する。

ﾚﾍﾞﾙ番号	意　　味
０１	最上位のﾚﾍﾞﾙ番号であり、Ａ領域に記述する。 指定できる項目は、基本項目と集団項目である。 01　基本項目1　　PICTURE　文字列. 01　集団項目1. 　02　基本項目2　PICTURE　文字列. 　02　基本項目3　PICTURE　文字列. 01　基本項目4　　PICTURE　文字列.
０２～４９	このﾚﾍﾞﾙ番号はＢ領域に記述する。 指定できる項目は、基本項目、集団項目である。 01　集団項目1. 　10　集団項目2. 　　20　集団項目3. 　　30　集団項目4. 　　40　集団項目5. 　　49　基本項目　　PICTURE　9(5).
６６	既に定義された項目を再定義する。 「RENAMES」を含んでいなければならない。 01　個人情報. 　02　姓　　PICTURE　X(20). 　02　名　　PICTURE　X(20). 　02　住所　PICTURE　X(40). 　02　収入　PICTURE　9(8). 　66　氏名　RENAMES　姓　THRU　名.
７７	基本項目のみの定義（独立項目） 使用する場合、01ﾚﾍﾞﾙの項目より前に定義する。 77　独立項目1　PICTURE　文字列. 77　独立項目2　PICTURE　文字列. 01　集団項目1. 　02　基本項目1　PIC　文字列..
８８	条件名の記述項目 「VALUE」を含んでいなければならない。 88　条件名1　VALUE　定数1. 88　条件名2　VALUE　定数1　THRU　定数2,　定数3.

（１）　レベル番号とデータ階層

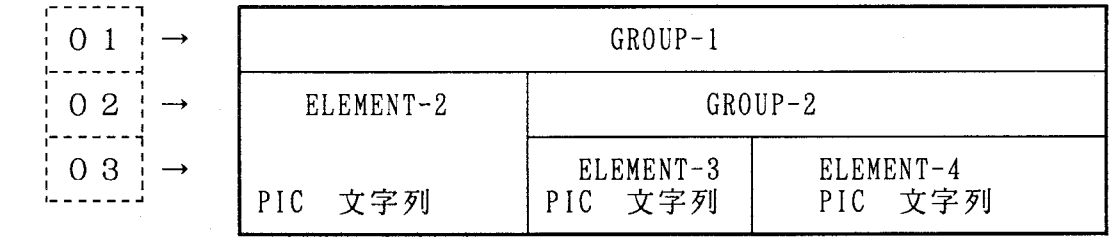
基本項目（elementary　item）とは、論理的にそれ以上は細分されないデータ項目である。

```
01  ELEMENT-1              PICTURE 文字列.
```
←基本項目

集団項目（group　item）とは、従属するデータ項目で構成されるデータ項目である。

```
01  GROUP-1.
    02  ELEMENT-2          PICTURE 文字列.
    02  GROUP-2.
        03  ELEMENT-3      PICTURE 文字列.
        03  ELEMENT-4      PICTURE 文字列.
```
←集団項目
←基本項目
←集団項目
←基本項目
←基本項目

レベル番号



２．５　ＰＩＣＴＵＲＥ句

ＰＩＣＴＵＲＥ句は、データのサイズ（桁数）を定義すると同時に、指定する文字列の組み合わせでデータ項目の種類（英字、数字等）も定義する。
ＰＩＣＴＵＲＥ句で決まるデータの項類は、「英字」、「数字」、「英数字」、「英数字編集」、「数字編集」の５種類である。「英字編集」はない。

```
{ PICTURE }
{ PIC     } IS  文字列
```

最大３０文字で記述する。

英字	A(4)
数字	9(3)、S9(4)、9V9
数字編集	ZZ9、--,--9、+9(3)
英数字編集	AABAA、XX/XX/XX
英数字	X(10)、9X9X

ＰＩＣＴＵＲＥ句は基本項目に書くことができ、記述できる文字列の長さは最大３０文字である。
文字列で使用できる文字記号は、次のとおり。

・文字列で２個以上指定できる記号
　Ａ　Ｂ　Ｐ　Ｘ　Ｚ　９　０　／　，　＋　－　＊　￥（通貨編集用文字）

・文字列で１個しか指定できない記号
　Ｓ　Ｖ　．　ＣＲ　ＤＢ

文字Ａ、Ｂ、Ｐ、Ｘ、Ｚ，ＣＲ、ＤＢの大文字と小文字は等価とする。

文字列中の括弧の中の数字は、括弧の前の文字の繰り返し（反復回数）を表す。
反復回数が書ける文字は、Ａ　Ｘ　９　Ｐ　Ｚ　＊　Ｂ　／　０　＋　－　￥　，

```
PICTURE    --,---,--9.
PICTURE    -(2),-(3),-(2)9.
```

通貨編集用文字（１文字）は環境部の「CURRENCY　SIGN」で変更できるが使用できる文字には制限がある。

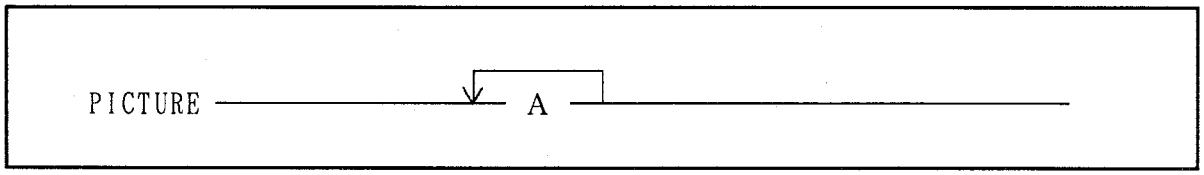
```
ENVIRONMENT  DIVISION.
      :
SPECIAL-NAMES.
        :
       CURRENCY  SIGN  IS  @

DATA  DIVISION.
      :
01  ﾃﾞｰﾀ名     PICTURE    @@@,@@@,@@@.99.

                          @1,234.50  <-----  実行結果

PROCEDURE  DIVISION.
      :
    MOVE  1234.5  TO  ﾃﾞｰﾀ名    ----------
```

ＰＩＣＴＵＲＥ文字列とデータ項目の関係は次の通り。

項　　類	使用文字	意　　味
数字項目	９ Ｐ Ｓ Ｖ	数字１桁の記憶域を表す 仮想の数字１桁を表す、記憶域は取られない データの符号の有無を表す 文字列の左端の文字、記憶域は取られない 仮想小数点の位置を表す 記憶域は取られない （例） ９９９　　Ｓ９（４）　９９Ｖ９９ ＳＰＰ９９９　　９９９ＰＰＰ
英字項目	Ａ	英字１桁の記憶域を表す （例） ＡＡＡ　　Ａ（１０）
英数字項目	Ｘ Ａ ９	英数字１桁の記憶域を表す 英字１桁の記憶域を表す（全てＡは不可） 数字１桁の記憶域を表す（全て９は不可） （例） ＸＸＸ　Ｘ（１００）　ＡＡＸＸ Ｘ９Ａ
英数字編集項目	Ａ Ｘ ９ Ｂ ０ ／	英字１桁の記憶域を表す 英数字１桁の記憶域を表す 数字１桁の記憶域を表す 空白を挿入する記憶域を表す 数字のゼロを挿入する記憶域を表す 文字「／」を挿入する記憶域を表す （例） ＸＸＢＸ　００Ｘ　ＸＸ／ＸＸ ＡＡ００　ＡＡ／ＡＡ／ＡＡ　ＡＡＢＡ
数字編集項目	Ｂ Ｐ Ｖ Ｚ ９ ０ ／ ， ． ＋ － ＣＲ ＤＢ ＊ ￥	空白を挿入する記憶域を表す 仮想の数字１桁を表す、記憶域は取られない 仮想小数点の位置を表す 記憶域は取られない 先行ゼロ列のゼロを空白に置き換える 記憶域は取られる 数字１桁の記憶域を表す 数字のゼロを挿入する記憶域を表す 文字「／」を挿入する記憶域を表す 文字「，」を挿入する記憶域を表す 小数点を表す編集用文字　記憶域は取られる 符号編集用文字　記憶域は取られる 符号編集用文字　記憶域は取られる 符号編集用文字　記憶域は取られる 符号編集用文字　記憶域は取られる 先行ゼロ列のゼロを星印に置き換える 記憶域は取られる 通貨編集用文字　記憶域は取られる （例） 9,999,999　-99,999　--,--9 ZZ,ZZZ,ZZ9+　¥¥¥,¥¥¥CR

（１）　英字項目（alphabetic）

文字列は、文字「Ａ」だけからなる。

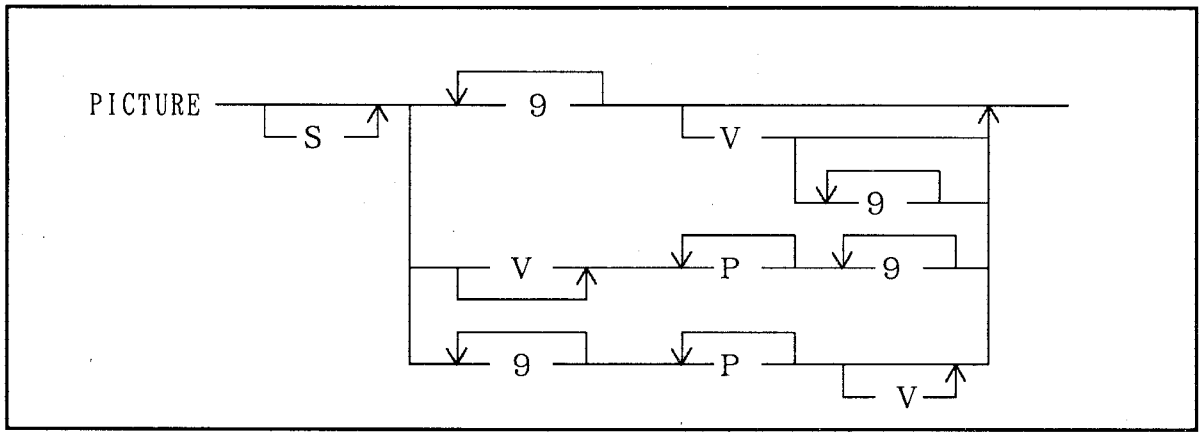


項目の内容は、一つ以上の英字（英文字、空白）でなければならない。

```
01  英字項目1  PICTURE  A(14)  VALUE  "COBOL LANGUAGE".
01  英字項目2  PICTURE  A(14)  VALUE  SPACE.
```

（２）　数字項目（numeric）

文字列は、文字「９」、「Ｐ」、「Ｓ」、「Ｖ」だけからなる。
数字（文字９）の桁数は１～１８桁とする。

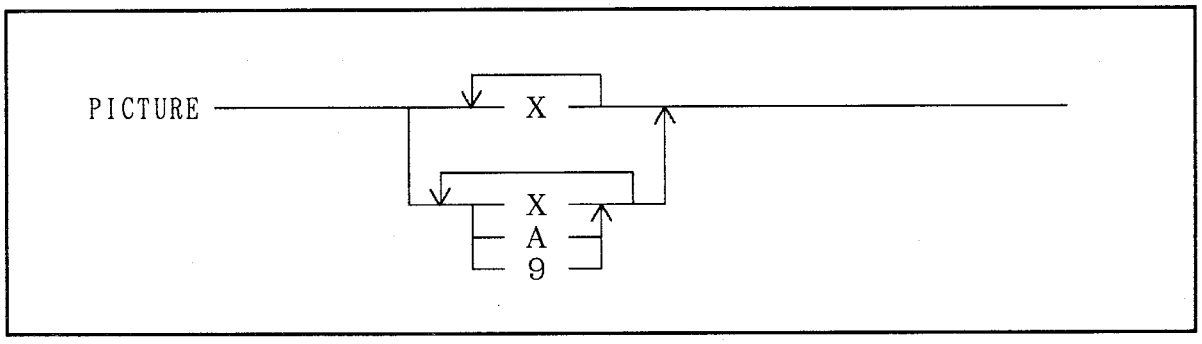


項目の内容は、一つ以上の数字（０～９）でなければならない。
項目が符号付きならば、「＋」、「－」または作成者の定めた演算符号を含むことができる。

```
01  数字項目1  PICTURE  S99V99  VALUE  -1.2.
01  数字項目2  PICTURE  S99V99  VALUE  -1.2
                SIGN  IS  LEADING  SEPARATE  CHARACTER.
01  数字項目3  PICTURE  S99V99  VALUE  ZERO.
```

（３）　英数字項目（alphanumeric）

文字列は、文字「Ｘ」、「Ａ」、「９」の組合せからなり、「Ａ」だけ「９」だけからなる文字列は英数字項目ではない。



項目の内容は、計算機文字集合の一つ以上の文字を含まなければならない。
文字位置の内容と対応するPICTURE文字の矛盾についてはチェックされない。

```
01  英数字項目1  PICTURE  X(14)  VALUE  "JIS-COBOLｹﾞﾝｺﾞ".
01  英数字項目2  PICTURE  X(14)  VALUE  LOW-VALUE.
01  英数字項目3  PICTURE  X(14)  VALUE  ZERO.
01  英数字項目4  PICTURE  XA9    VALUE  "123".
```

（４）　英数字編集項目（alphanumeric　edited）

文字列は、文字「Ａ」、「Ｘ」、「９」、「Ｂ」、「０」、「／」の組合せからなり「Ａ」か「Ｘ」を含み、かつ「Ｂ」か「０」か「／」を少なくとも一文字含まなければならない。

項目の内容は、計算機文字集合の二つ以上の文字を含まなければならない。
文字位置の内容と対応するPICTURE文字の矛盾についてはチェックされない。

```
01  英数字編集項目1  PICTURE  XX/XX/XX.
01  英数字編集項目2  PICTURE  9999XXX.
01  英数字編集項目3  PICTURE  AX9B0.
01  英数字編集項目4  PICTURE  /X/.
```

（５）　数字編集項目（numeric　edited）

文字列は、文字「Ｂ」、「／」、「Ｐ」、「Ｖ」、「Ｚ」、「０」、「９」、「，」、「．」、「＊」、「＋」、「－」、「ＣＲ」、「ＤＢ」、「￥（通貨編集用文字）」の組合せからなり、数字の桁数は１～１８桁とする。
「．」、「Ｖ」、「ＣＲ」、「ＤＢ」は文字列中に一つしか書けない。

```
01  数字編集項目1  PICTURE  99.9.
01  数字編集項目2  PICTURE  9999/99/99.
01  数字編集項目3  PICTURE  ZZ,ZZZ,ZZ9+.
01  数字編集項目4  PICTURE  ***,***V***.
```

数字編集項目にVALUE句で初期値を設定する場合、文字位置の内容と対応するPICTURE文字と矛盾しないものでなけらばならない。

ＰＩＣＴＵＲＥ句による編集

PICTURE句で指定できる編集の方法は、「挿入」と「ゼロ抑制」の２種類がある。
挿入編集は４種類、ゼロ抑制編集は２種類ある。

編集	挿入	単純挿入	挿入文字は「,　Ｂ　０　／」 文字列の例　XXBXX　99/99/99
		特殊挿入	挿入文字は「. 」 文字列の例　999.999
		固定挿入	挿入文字は「＋　－　CR　DB　＄」 文字列の例　+999　999-　$99.99DB
		浮動挿入	挿入文字は「¥　＋　－」 文字列の例　¥¥,¥¥¥　+++,+++　--9
	ゼロ抑制	空白による ゼロ抑制	挿入文字は「Ｚ」 文字列の例　ZZZ.99
		星印による ゼロ抑制	挿入文字は「＊」 文字列の例　***,***.**

編集できる項類は、次のとおり。

項類（category）	編集の種類
英字項目	なし
数字項目	なし
英数字項目	なし
英数字編集項目	「０　Ｂ　／」の単純挿入
数字編集項目	浮動挿入とゼロ抑制の組み合わせ以外の編集

ＰＩＣＴＵＲＥ句の文字列に於ける順序規則

先行文字 ＼ 後続文字		浮動挿入以外の挿入文字									浮動挿入文字						その他の文字					
		B	0	／	,	.	＋－	＋－	CR DB	通貨	Z＊	Z＊	＋－	＋－	通貨	通貨	9	AX	S	V	前P	後P
浮動挿入以外の挿入文字	B	○	○	○	○	○	○			○	○	○	○	○	○	○	○	○		○		○
	0	○	○	○	○	○	○			○	○	○	○	○	○	○	○	○		○		○
	／	○	○	○	○	○	○			○	○	○	○	○	○	○	○	○		○		○
	,	○	○	○	○	○	○			○	○	○	○	○	○	○	○			○		○
	.	○	○	○	○		○			○	○		○		○		○					
	＋－																					
	＋－	○	○	○	○	○				○	○	○			○	○	○			○	○	○
	CR DB	○	○	○	○	○				○	○	○			○	○	○			○	○	○
	通貨							○														
浮動挿入文字	Z＊	○	○	○	○		○			○	○											
	Z＊	○	○	○	○	○	○			○	○	○								○		○
	＋－	○	○	○	○					○			○									
	＋－	○	○	○	○	○				○			○	○						○		○
	通貨	○	○	○	○		○								○							
	通貨	○	○	○	○	○	○								○	○				○		○
その他の文字	9	○	○	○	○	○	○			○	○		○		○		○	○	○	○		○
	AX	○	○	○													○	○				
	S																					
	V	○	○	○	○		○			○	○		○		○		○		○		○	
	前P	○	○	○	○		○			○	○		○		○		○		○		○	
	後P						○			○									○	○		○

２．６　ＵＳＡＧＥ句

ＵＳＡＧＥ句は、コンピュータでデータを表現するのに幾つもの形式がある場合の表現形式を指定する。

```
[USAGE  IS] { BINARY
              COMPUTATIONAL
              COMP
              DISPLAY
              INDEX
              PACKED-DECIMAL }
```

USAGE句を省略すると、「USAGE　IS　DISPLAY」とみなされる。
「DISPLAY」は、コンピュータで標準のデータ形式が用いられ、英字、英数字、英数字編集、数字、数字編集のデータ項目に指定できる。「INDEX」は、指標データ項目に指定する。「BINARY」、「COMPUTATIONAL」、「PACKED-DECIMAL」は、数字項目に指定できる。
USAGE句は、集団データ項目に指定できるが、集団に従属する基本データ項目と異なるUSAGE句は指定できない。

	項目レベル		項　　類				
ＵＳＡＧＥ	集団	基本	英字	英数字	英数字編集	数字	数字編集
BINARY	○	○	×	×	×	○	×
COMPUTATIONAL	○	○	×	×	×	○	×
DISPLAY	○	○	○	○	○	○	○
INDEX	○	○	−	−	−	−	−
PACKED-DECIMAL	○	○	×	×	×	○	×

（１）　ＢＩＮＡＲＹ

「BINARY」は、数字項目を表すために基数として「２進数」が用いられる。
データ領域の大きさは、PICTURE文字列で表現される範囲の最大値を含むことができる領域を割り付ける。

```
01  ﾃﾞｰﾀ名  PICTURE  9(4)  USAGE  BINARY  VALUE  123.
                          ↑── ４桁での値の範囲は１～９９９９なので
                              データ領域は２バイト必要である。
```

００	７Ｂ

（内容は１６進数表現）

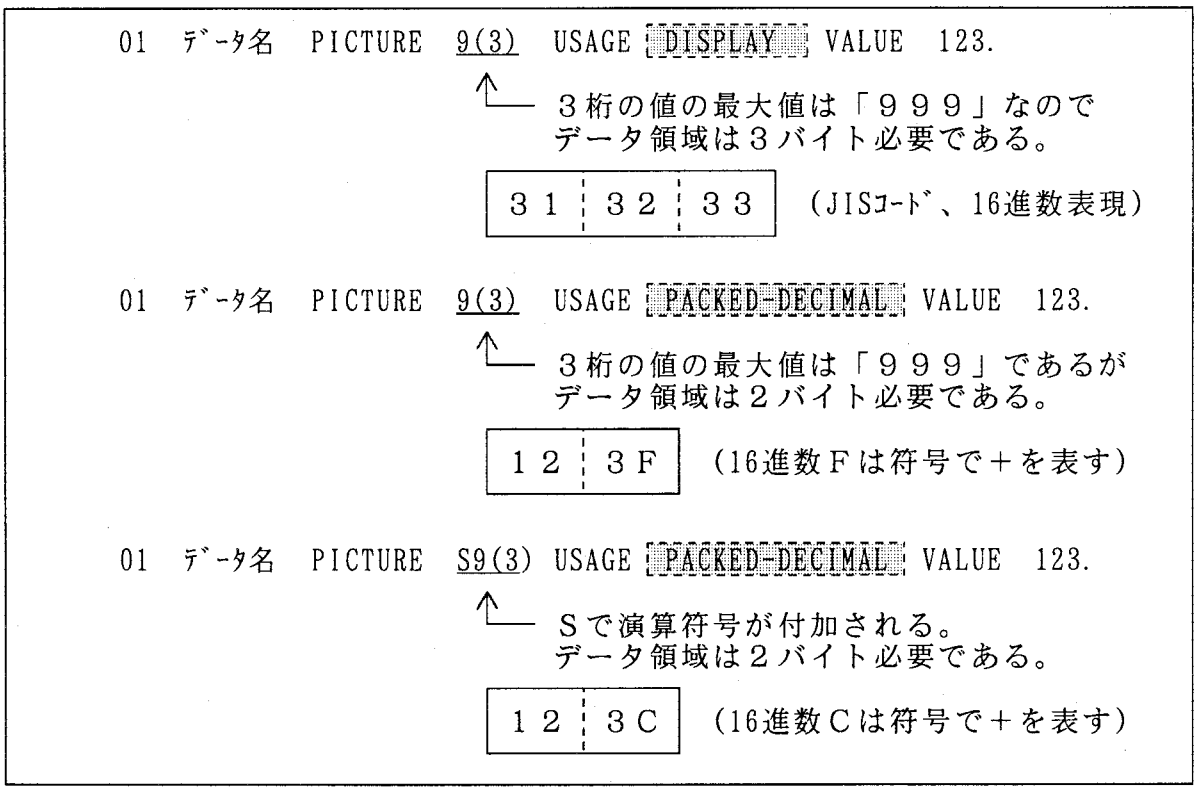
```
01  ﾃﾞｰﾀ名  PICTURE  9(8)  USAGE  BINARY  VALUE  123.
                          ↑── ８桁での値の範囲は１～99999999なので
                              データ領域は４バイト必要である。
```

００	００	００	７Ｂ

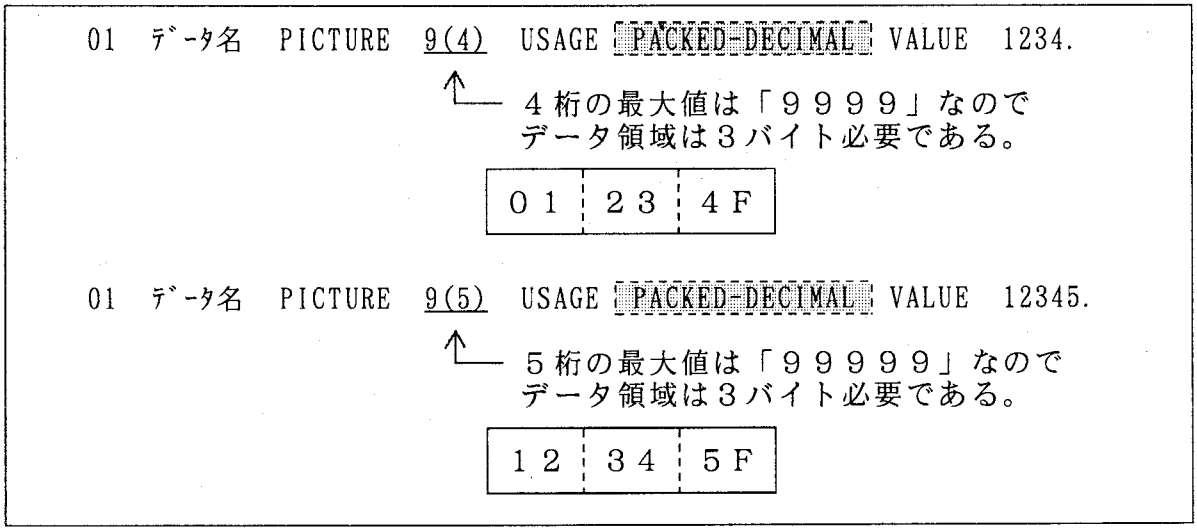
負の値は、２の補数や１の補数で表現する場合が多い。

（２）　ＰＡＣＫＥＤ－ＤＥＣＩＭＡＬ

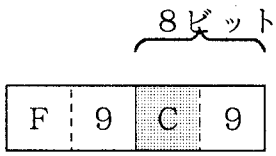
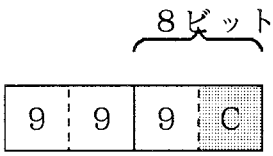
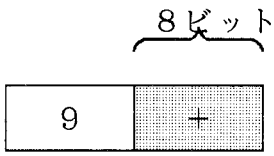
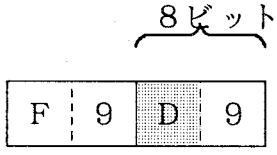
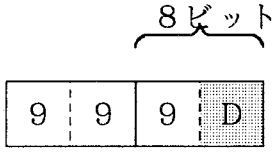
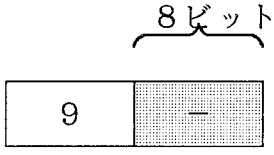
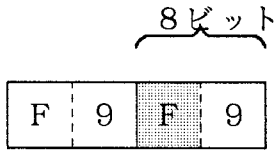
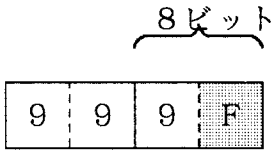
「PACKED-DECIMAL」は、数字項目を表すために基数として「１０進数」が用いられる。通常、「DISPLAY」は、ゾーン１０進数（ZONED DECIMAL）形式が用いられ、「PACKED-DECIMAL」にはパック１０進数形式が用いられる。



「PACKED-DECIMAL」では、桁数が異なってもデータ領域の大きさは同じ場合がある。



数字項目の符号は、次のとおり。

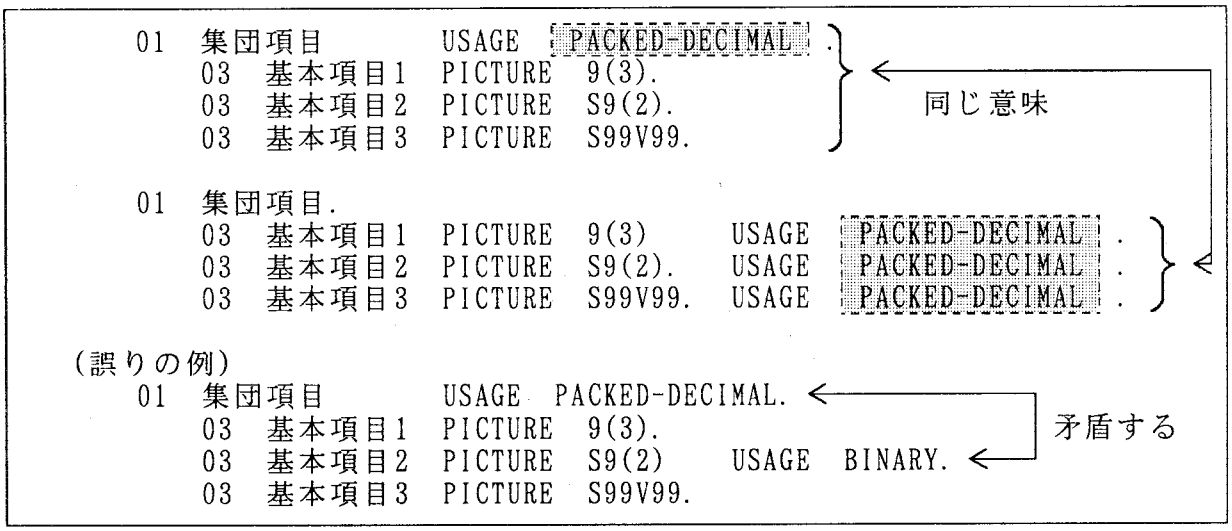
	DISPLAY	PACKED-DECIMAL	SIGN TRAILING SEPARATE CHARACTER
符号付き（＋）	 （ASCIIでは３）	 （ASCIIでは３）	
符号付き（－）	 （ASCIIでは７）	 （ASCIIでは７）	
符号なし	 （ASCIIでは３）	 （ASCIIでは３）	

内容は１６進数表現でＥＢＣＤＩＣコード体系の場合。

「COMPUTATIONAL」は、数字項目を表すための基数と形式がコンパイラ作成者によって決められるためコンパイラにより異なる。
次の表は、あるコンパイラの「USAGE　COMPUTATIONAL」の拡張割付の例である。

ＵＳＡＧＥ	データ表現形式
COMPUTATIONAL-1 COMP-1	浮動小数点（単精度）
COMPUTATIONAL-2 COMP-2	浮動小数点（倍精度）
COMPUTATIONAL-3 COM-3	パック１０進数 (PACKED-DECIMAL)
COMPUTATIONAL-4 COMP-4	２進数 (BINARY)
COMPUTATIONAL-5 COMP-5	２進数 (BINARY)

USAGE句は、集団データ項目にも指定できる。
集団データ項目に「BINARY」、[COMPUTATIONAL」、「PACKED-DECIMAL」を指定すると従属する基本データ項目には異なるUSAGE句は指定できない。



指標データ項目の「USAGE INDEX」には、「PICTURE」句を指定できない。

```
01  ﾃﾞｰﾀ名        PICTURE  9(5)  USAGE  BINARY.

01  指標ﾃﾞｰﾀ名    USAGE  INDEX.
```

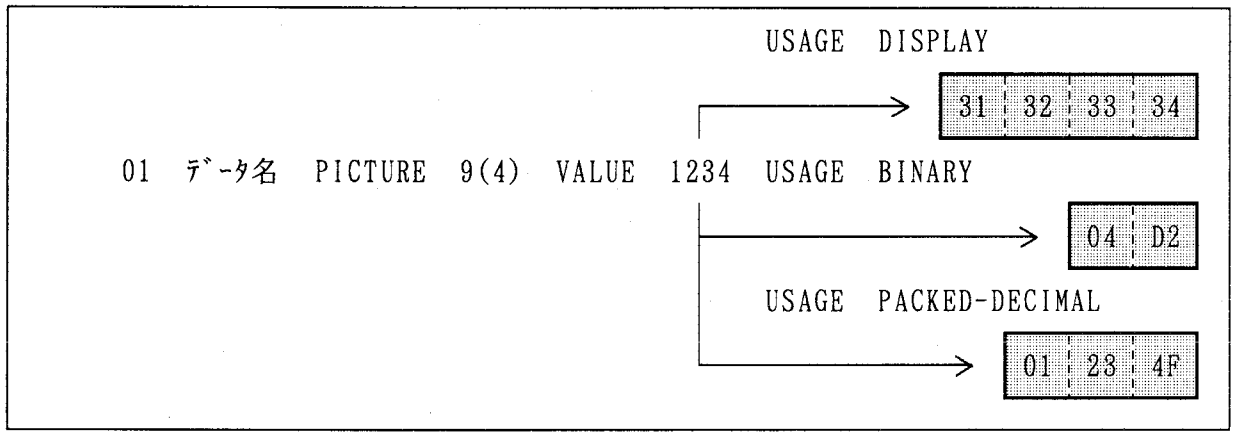
指標データ項目は、表要素（配列）の出現番号に対応する値を含むため、その領域の大きさはコンパイラーにより定義される。
また、指標データ項目は、ＳＥＡＲＣＨ文、ＳＥＴ文、比較条件、ＣＡＬＬのＵＳＩＮＧで参照できるがＭＯＶＥ文などで直接参照できない。
集団データ項目に指標データ項目が含まれる場合は、ＭＯＶＥ文で参照できるが内容の変換等は行われない。

```
01  集団項目名1.
    03  指標ﾃﾞｰﾀ名1     USAGE  INDEX.

01  集団項目名2.
    03  指標ﾃﾞｰﾀ名2     USAGE  INDEX.


MOVE  指標ﾃﾞｰﾀ名1  TO  指標ﾃﾞｰﾀ名2  ←×  誤り
MOVE  集団項目名1  TO  集団項目名2  ←○  誤りではない

SET   指標ﾃﾞｰﾀ名2  TO  指標ﾃﾞｰﾀ名1  ←○  通常の使い方
```

ＵＳＡＧＥ句と記憶域の割り当ての例

ＣＯＢＯＬでは、計算機でデータを表現するのに幾つもの形式がある場合には、データ記述で特に指定しない限り標準データ形式が用いられる。基本データ項目や集団項目の大きさは、その項目を標準データ形式で表現したときの文字数である。
けたづめ（SYNC）と用途（USAGE）により、この大きさと内部表現に必要な大きさとが異なる場合があってもよい。
次の表は、PICTUREの桁数と記憶域のバイト数の関係を示す。

ＰＩＣＴＵＲＥ 桁数	記憶域のバイト数			
	USAGE IS DISPLAY	USAGE IS BINARY バｲﾄ割付	USAGE IS BINARY ﾜｰﾄﾞ割付	USAGE IS PACKED-DECIMAL
９（１）	1	1	2	1
９（２）	2	1	2	2
９（３）	3	2	2	2
９（４）	4	2	2	3
９（５）	5	3	4	3
９（６）	6	3	4	4
９（７）	7	3	4	4
９（８）	8	4	4	5
９（９）	9	4	4	5
９（１０）	１０	5	8	6
９（１１）	１１	5	8	6
９（１２）	１２	5	8	7
９（１３）	１３	6	8	7
９（１４）	１４	6	8	8
９（１５）	１５	7	8	8
９（１６）	１６	7	8	9
９（１７）	１７	8	8	9
９（１８）	１８	8	8	１０



内容はＪＩＳコード、16進数表現である。

２．７　ＶＡＬＵＥ句

ＶＡＬＵＥ（値）句は、作業場所節（WORKING-STORAGE　SECTION）及び通信節でデータ項目に初期値を定義する。データ項目にＶＡＬＵＥ句を書かない場合の初期値の規定はない。ファイル節（FILE　SECTION）のデータ項目には指定できない。
また、ＶＡＬＵＥ句はレベル番号８８と組み合わせて条件名に関連する値を定義することができる。

書き方１

```
        VALUE  IS  定数1
```

書き方２

```
        { VALUE  IS  }  { 定数2 [ { THRU      } 定数3 ] } ...
        { VALUES ARE }            { THROUGH }
```

書き方２は、条件名を宣言する場合に使用する。ＴＨＲＵで範囲を指定する場合の値の順序は、次のとおり。

```
        定数2  <  定数3
```

データ部（DATA　DIVISION）の各SECTIONでのＶＡＬＵＥ句の記述規則は次のとおり。

ＳＥＣＴＩＯＮ名	データ項目の初期値	条件名に関連する値
FILE	×	○
WORKING-STORAGE	○	○
LINKAGE	×	○
COMMUNICATION	○	○
REPORT	○	○

REDEFINES句を含むデータ項目にデータ項目初期値（書き方１）のVALUE句は書けない。

```
01  ﾃﾞｰﾀ名1  PICTURE  X(10)
01  ﾃﾞｰﾀ名2  REDEFINES  ﾃﾞｰﾀ名1.
    03  ﾃﾞｰﾀ名3  PIC  X(3)  VALUE  "678".          ←  ×
    03  ﾃﾞｰﾀ名4  PIC  X(7)  VALUE  "ABCDEFG".      ←  ×

01  ﾃﾞｰﾀ名1  PICTURE  X(10)  VALUE  "678ABCDEFG".  ←  ○
01  ﾃﾞｰﾀ名2  REDEFINES  ﾃﾞｰﾀ名1.
    03  ﾃﾞｰﾀ名3  PIC  X(3).
    03  ﾃﾞｰﾀ名4  PIC  X(7).

01  ﾃﾞｰﾀ名1  PICTURE  X(10)  VALUE  SPACE.         ←  ○
01  ﾃﾞｰﾀ名2  REDEFINES  ﾃﾞｰﾀ名1.
    03  ﾃﾞｰﾀ名3  PIC  X(3).
        88  条件名1    VALUE  "DOG".               ←  ○
        88  条件名2    VALUE  "CAT".               ←  ○
    03  ﾃﾞｰﾀ名4  PIC  X(7).
```

JIS-COBOLでは、OCCURS句を含むデータ項目にデータ項目初期値（書き方１）のVALUE句は指定可能と規定されているが、OCCURS句にVALUE句を指定できないコンパイラもある。

データの項類（category）とVALUE句の関係は次のとおり。

データ項目	一般規則
数字	・定数は全て数字定数（表意定数ZEROも含む）であること。 01 ﾃﾞｰﾀ名 PICTURE 9(3) VALUE IS ZERO. 01 ﾃﾞｰﾀ名 PICTURE 9(3) VALUE IS 123. 01 ﾃﾞｰﾀ名 PICTURE 9(4) VALUE IS 123 USAGE COMP. ・定数は小数点位置規則に従ってデータ項目に収められる。そして、ゼロ以外の数字が切り捨てられないこと。 01 ﾃﾞｰﾀ名 PICTURE 99V99 VALUE 1.1. 01 ﾃﾞｰﾀ名 PICTURE PP9 VALUE 0.09. 01 ﾃﾞｰﾀ名 PICTURE 9PP VALUE 100. ・定数に符号（+、-）を含む場合は、PICTURE句の文字列に「S」が指定されていること。. 01 ﾃﾞｰﾀ名 PICTURE S9(2) VALUE -50. 01 ﾃﾞｰﾀ名 PICTURE S9V99 VALUE +1.03. 01 ﾃﾞｰﾀ名 PICTURE 9(2) VALUE +50. ←誤りである
英字 英数字 英数字編集 数字編集	・定数は全て文字定数と表意定数であること。 01 ﾃﾞｰﾀ名 PICTURE X(4) VALUE "1234". 01 ﾃﾞｰﾀ名 PICTURE X(1000) VALUE ALL SPACE. ・ＰＩＣＴＵＲＥ句で定義したサイズを超えないこと。 01 ﾃﾞｰﾀ名 PICTURE X(2) VALUE "AB". 01 ﾃﾞｰﾀ名 PICTURE X(2) VALUE "ABC". ←誤りである ・編集項目にVALUE句で初期値を与えてもＣＯＢＯＬの編集規則は適用されない。 01 ﾃﾞｰﾀ名 PICTURE XX/XX/XX VALUE "AABBCC". 01 ﾃﾞｰﾀ名 PICTURE XX/XX/XX VALUE "AA/BB/CC". 01 ﾃﾞｰﾀ名 PICTURE ZZZ,ZZ9 VALUE "1234". 01 ﾃﾞｰﾀ名 PICTURE ZZZ,ZZ9 VALUE " 1,234". 01 ﾃﾞｰﾀ名 PICTURE +99 VALUE "-56".

集団項目にVALUE句を指定する場合は、文字定数または表意定数である。VALUE句を持つ集団項目に従属するデータ項目には、「書き方１」のVALUE句は指定できないが、レベル番号８８の条件名には「書き方２」のVALUE句が指定できる。

```
01  集団項目                          VALUE  SPACE.  ──┐
    03  ﾃﾞｰﾀ名1  PICTURE  X(5).          ←──┤
    03  ﾃﾞｰﾀ名2  PICTURE  X(3).          ←──┤
    03  ﾃﾞｰﾀ名3  PICTURE  X(10).         ←──┘
        88  条件名1    VALUE  "TOKYO".
        88  条件名2    VALUE  "NEWYORK".

01  集団項目                          VALUE  ZERO.  ──┐  ←誤り
    03  ﾃﾞｰﾀ名1  PIC  9(5)  USAGE  BINARY.          ←──┤  USAGE別に
    03  ﾃﾞｰﾀ名2  PIC  9(3).                         ←──┤  定数は変換
    03  ﾃﾞｰﾀ名3  PIC  9(10) USAGE  PACKED-DECIMAL.  ←──┘  されない。
```

２．８　ＲＥＤＥＦＩＮＥＳ句

REDEFINES（再定義）句は、同じ記憶領域を別のデータ項目として記述する。

```
ﾚﾍﾞﾙ番号  { データ名1 }  REDEFINES  データ名2
          { FILLER    }
```

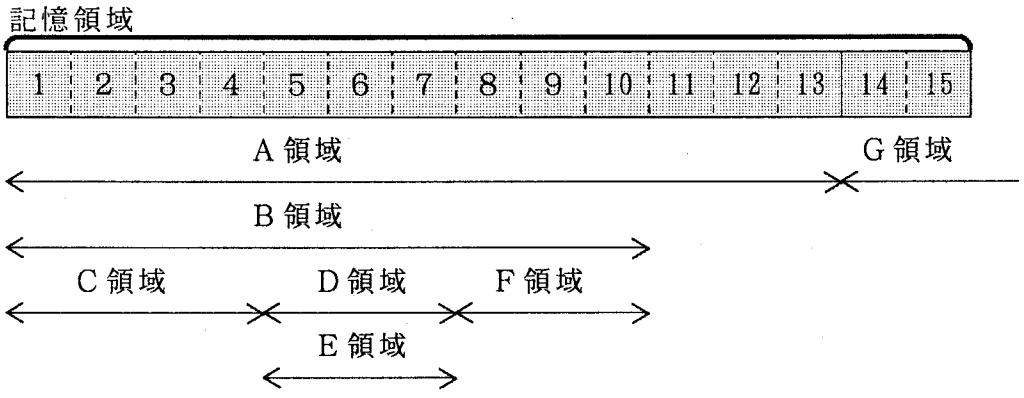
REDEFINES句は、データ名1またはFILLERの直後の句として記述する。
データ名1とデータ名2のレベル番号は同じでなければならいない。ただし、レベル番号が「６６」や「８８」であってはならない。

再定義データ項目の終わりは、データ名2のレベル番号と同じレベル番号が現れるか、それより数値の小さいレベル番号が現れたときに再定義は終了する。

```
01  A       PICTURE  X(13).
01  B   REDEFINES  A.
    03  C     PICTURE  9999.
    03  D     PICTURE  XXX.
    03  E   REDEFINES  D  PICTURE  999.
    03  F     PICTURE  99V9.
01  G         PICTURE  9(8).
```

AとBは同じ領域
但しｻｲｽﾞは異なる

DとEは同じ領域



同じ記憶領域を何回も再定義する場合は、データ名2はいずれもその記憶領域を最初に定義したデータ名を使用する。

```
03  BASE                    PICTURE  999.
03  R1  REDEFINES  BASE     PICTURE  99V9.
03  R2  REDEFINES  BASE     PICTURE  999PP.
03  R3  REDEFINES  BASE     PICTURE  PP999.
```

データ部の「FILE　SECTION」の０１レベル記述項には「REDEFINES」を書けない。
これは、コンパイラが自動的に再定義するからである。

```
FILE  SECTION.
FD  ﾌｧｲﾙ名
    DATA  RECORDS  ARE  ﾚｺｰﾄﾞ名1  ﾚｺｰﾄﾞ名2  ﾚｺｰﾄﾞ名3.
01  ﾚｺｰﾄﾞ名1.  <-+
    03  ～       |
01  ﾚｺｰﾄﾞ名2.  <-+---- コンパイラが同じ領域に割り付ける
    03  ～       |
01  ﾚｺｰﾄﾞ名3.  <-+
    03  ～

WORKING-STORAGE  SECTION.
01  ﾃﾞｰﾀ名1.                          <-+
    03  ～                               |
01  ﾃﾞｰﾀ名2  REDEFINES  ﾃﾞｰﾀ名1.      <---+---- 同じ領域にする場合は
    03  ～                               |     REDEFINES句を指定する
01  ﾃﾞｰﾀ名3  REDEFINES  ﾃﾞｰﾀ名1.      <-+
    03  ～
```

再定義される項目（データ名2）のデータ記述項には、ＯＣＣＵＲＳ句を含んでいてはいけない。ただし、ＯＣＣＵＲＳ句を含むデータ記述項に従属していてもよい。

```
01  AA.
    05  BB         PICTURE  X(100).
    05  CC     REDEFINES  BB  PICTURE  X(10)  OCCURS  10 TIMES.

                    ↓  正しくは次のように記述する。
01  AA.
    05  BB         PICTURE  X(100).
    05  FILLER     REDEFINES  BB.
        10  CC     PICTURE  X(10)  OCCURS  10  TIMES.
```

再定義するデータ項目の用途（USAGE）は、異なってもよい。

```
01  ALL-TYPES.
    03  AN                  PICTURE  X(4).
    03  ZD  REDEFINES  AN   PICTURE  9(4)  USAGE  DISPLAY.
    03  PD  REDEFINES  AN   PICTURE  9(7)  USAGE  PACKED-DECIMAL.
    03  BN  REDEFINES  AN   PICTURE  9(8)  USAGE  BINARY.
```

０１以外のレベル番号で再定義する場合は、領域の大きさは再定義する領域（ﾃﾞｰﾀ名２）以内であること。
０１のレベル番号の領域を再定義する場合は、任意である。

```
01  BASE-SIZE                          PICTURE  X(100).
01  FREE-SIZE1  REDEFINES  BASE-SIZE   PICTURE  X(050).  ←  ○
01  FREE-SIZE2  REDEFINES  BASE-SIZE   PICTURE  X(150).  ←  ○

01  AA.
    03  BB                  PICTURE  X(50).
    03  CC  REDEFINES  BB   PICTURE  X(49).  ←○
    03  DD  REDEFINES  BB   PICTURE  X(50).  ←○
    03  EE  REDEFINES  BB   PICTURE  X(51).  ←×  BBのｻｲｽﾞを超える
```

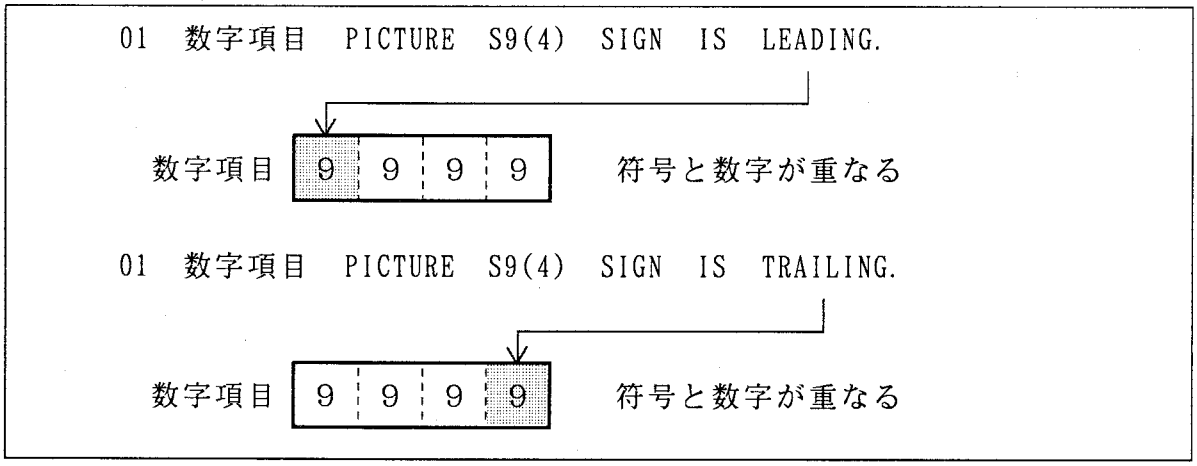
２．９　ＳＩＧＮ句

ＳＩＧＮ（符号）句は、数字項目の符号（＋、－）の位置と表現形式を指定する。
ＳＩＧＮ句を指定する場合は、PICTURE文字列に「Ｓ」が含まれていること。また、その用途（USAGE）は「DISPLAY」であること。

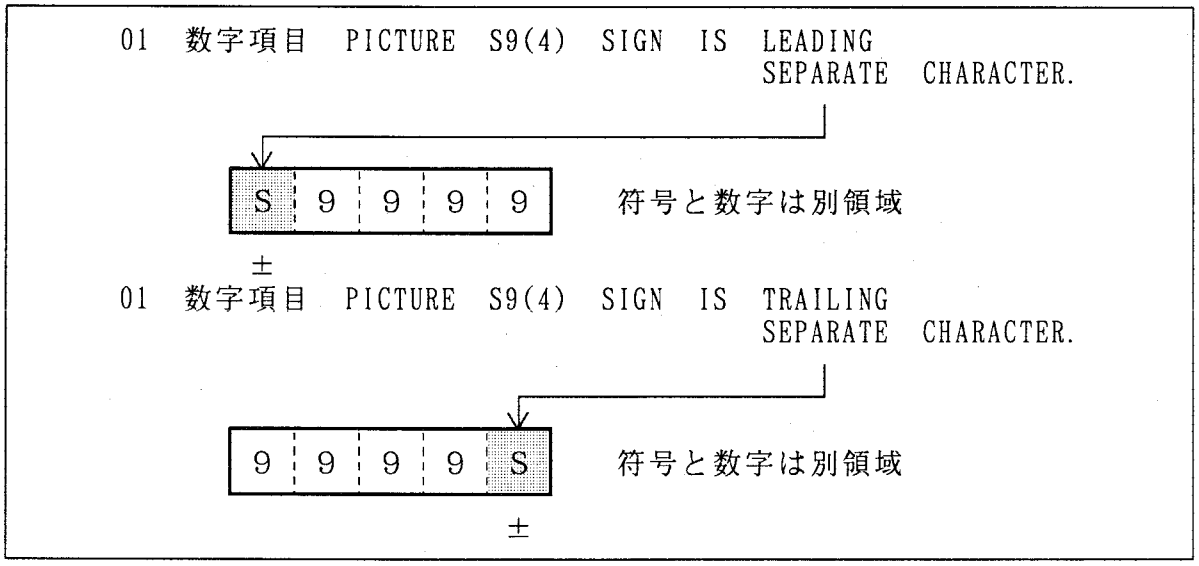
```
[SIGN  IS] { LEADING  }  [SEPARATE  CHARACTER]
           { TRAILING }
```

PICTURE句の文字列中の「S」は、演算符号の存在を示すだけで、その位置や表現形式を示すものではない。

演算符号は、LEADINGの場合は左端に、TRAILINGの場合は右端に付く。



ＳＩＧＮ句で「SEPARATE　CHARACTER」を指定した場合は、PICTURE句の文字列中の「S」はデータ項目の大きさ（サイズ）に含められる。
そして、演算符号文字の正は「＋」であり、負の文字は「－」である。



ＳＩＧＮ句が省略された場合の演算符号の位置と表現形式は、コンパイラ作成者が規定する。プログラムの実行性能を向上させる場合は、ＳＩＧＮ句を省略するとよい。

２．１０　ＢＬＡＮＫ　ＷＨＥＮ　ＺＥＲＯ句

BLANK　WHEN　ZERO句は、データ項目の値がゼロのとき、その項目を空白だけにする。
指定できる項目は、数字項目または数字編集項目の基本項目である。
そして、用途（USAGE）は「DISPLAY」であること。

```
BLANK　WHEN　ZERO
```

PICTURE句の文字列では数字項目のデータ項目に、BLANK　WHEN　ZERO句を指定すると、その項目は数字編集項目とみなされる。

```
DATA DIVISION.
    01　ﾃﾞｰﾀ名　PICTURE　9(5)　BLANK　WHEN　ZERO.　←┐
PROCEDURE　DIVISION.                                  │
      COMPUTE　ﾃﾞｰﾀ名　=　算術式                      │
                  └──────────────────────────────────┘
                  算術式の結果が「０」の場合、ﾃﾞｰﾀ名の内容は空白になる。
```

２．１１　ＪＵＳＴＩＦＩＥＤ句

JUSTIFIED句は、英字項目または英数字項目が受取り側の場合の標準けたよせ規則を無効にする。

```
{JUSTIFIED}
{JUST     } RIGHT
```

送出し側データ項目の大きさが受取り側データ項目より大きければ、左端の文字を切り捨て、小さければ左端の余りに空白を補う。

```
DATA DIVISION.
                                                    実行結果
    01  ﾃﾞｰﾀ名1  PICTURE  X(8).                  ←   ABC
    01  ﾃﾞｰﾀ名2  PICTURE  X(8)  JUSTIFIED  RIGHT. ←      ABC
PROCEDURE  DIVISION.
      MOVE  "ABC"  TO  ﾃﾞｰﾀ名1
      MOVE  "ABC"  TO  ﾃﾞｰﾀ名2
                  |_____________________________|
      MOVE  ﾃﾞｰﾀ名1  TO  ﾃﾞｰﾀ名2      ←項目の大きさが同じであるので
                                      単にデータの移動が行われる。
```

標準けたよせ規則とは、次のとおり。

・受取り側データ項目が数字項目の場合
データは、小数点位置に合わせて格納する。必要に応じてゼロを補ったり、端を切り捨てたりする。
小数点が明示されていない場合には、データ項目の右端にあるものみなす。

```
MOVE  123.45  →  PIC  9V9          34
              →  PIC  9(5)V9(3)    00123450
```

・受取り側データ項目が数字編集項目の場合
データは、小数点の位置に合わせて編集転記する。必要に応じてゼロを補ったり、端を切り捨てたりする。ただし、編集の指定によって先行ゼロ列は空白などで置き換えられる。

```
MOVE  123.45  →  PIC  +9(5)        +00123
              →  PIC  ZZZ,ZZ9.9        123.4
```

・受取り側データ項目が英字、英数字、英数字編集の場合
送出し側データは、受取り側文字位置に左端をそろえて転記される。必要に応じて右端に空白を補ったり、右端を切り捨てたりする。

```
MOVE "COBOL"  →  PIC  XX           CO
              →  PIC  X(8)         COBOL
```

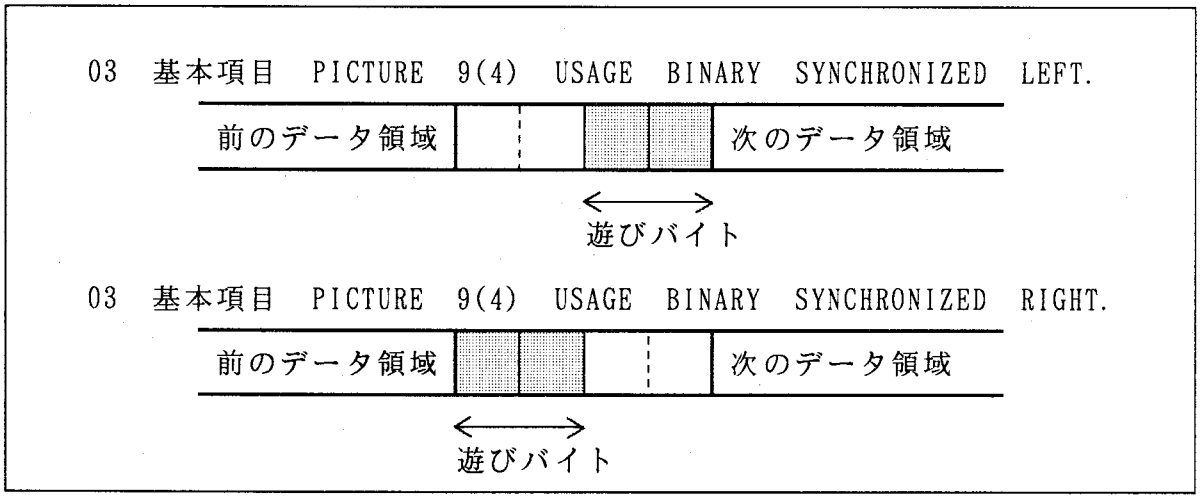
２．１２　ＳＹＮＣＨＲＯＮＩＺＥＤ句

SYNCHRONIZED句は、基本データ項目をコンピュータの記憶装置の固有の境界に従った配置（割り付け）にする。

```
{SYNCHRONIZED}  [LEFT ]
{SYNC        }  [RIGHT]
```

「LEFT」は、基本項目を固有境界の左端の文字位置から割り付ける。
「RIGHT」は、基本項目を固有境界の右端の文字位置で終わるように割り付ける。
「LEFT」、「RIGHT」を省略した場合は、コンパイラー作成者がその割り付けを規定する。

次の例は、コンピュータの記憶装置が４バイト単位の境界に割り付けることを基本とする場合に、２バイトの「BINARY」基本項目の配置例である。



この句は、ハードウェアに依存するので有効に使用するとプログラムの実行性能を向上させることができる。

SYNCHRONIZED句により使用されない領域（遊びバイト）が、集団項目に従属する場は、その遊びバイトは集団項目に含まれる。

```
01  集団項目Ａ.                              ←基本項目Ａは４バイト
    03  基本項目1  PIC  9(4)  BINARY.
    03  基本項目2  PIC  9(4)  BINARY.

01  集団項目Ｂ.                              ←基本項目Ｂは８バイト
    03  基本項目1  PIC  9(4)  BINARY  SYNCHRONIZED.
    03  基本項目2  PIC  9(4)  BINARY  SYNCHRONIZED.
```

ハードウェアの異なるコンピュータで作成したファイルのレコードの中に「SYNCHRONIZED」句や「USAGE BINARY」句がある場合の入力データについては注意する必要がある。

２．１３　配列

ＣＯＢＯＬでは、配列を表（table）という。表は論理的に連続したデータ項目の組であって、データ部でＯＣＣＵＲＳ句によって定義する。

```
DATA  DIVISION.
   :
01   表.
     02   表要素    OCCURS  100  TIMES.
          03  ﾃﾞｰﾀ項目1  PICTURE  X(10).
          03  ﾃﾞｰﾀ項目2  PICTURE  9(5).
```

表中の反復する項目の組に属する１個のデータ項目を表要素（table　element）という。

どの表要素も同じデータ名になるので、特定の要素を参照するためには、その要素が何番目かを示す番号（出現番号）が必要になる。この番号を「添字」と呼ぶ。

表要素の最初の添字は「１」から始まり最後の要素の番号はOCCURS句の繰り返し回数である。

```
表要素データ名　（添字）
                 ↑―― 最初の要素は１である。
                      最後の要素はOCCURS句の繰り返し回数である。
```

言語別の配列の定義の例

ＣＯＢＯＬ	定義 01　table1. 02　一次元　OCCURS　50　TIMES. 03　ﾃﾞｰﾀ名1　PICTURE　X (2) 01　table2. 02　一次元　OCCURS　50　TIMES. 03　２次元　OCCURS　40　TIMES. 04　ﾃﾞｰﾀ名2　PICTURE　X (2) . 参照 ﾃﾞｰﾀ名1　（添字） ﾃﾞｰﾀ名2　（添字　添字）
ＦＯＲＴＲＡＮ	定義 CHARACTER*ｻｲｽﾞ　配列名 (50) CHARACTER*ｻｲｽﾞ　配列名 (50,40) 注：反復回数に下限と上限が指定できる。 配列名（下限：上限）　　配列名 (-10:20) 参照 配列名（添字） 配列名（添字，添字）
Ｃ	定義 int　配列名［50］ int　配列名［50］［40］ 参照 配列名［添字］ 配列名［添字］［添字］ 注：最初の要素の添字は「０」ある。

２．１３．１　配列の定義

配列を定義するために「OCCURS」を使用して繰り返し回数を指定する。
ここで重要なことは、「OCCURS」句はレベル番号０１、６６、７７、８８、に指定できないことである。

```
書き方１
    OCCURS  整数2  TIMES
            [{ASCENDING  }  KEY  IS  {ﾃﾞｰﾀ名2} ･･･] ･･･
            [{DESCENDING }
            [INDEXED  BY  {指標名1} ･･･]

書き方２
    OCCURS  整数1  TO  整数2  TIMES  DEPENDING  ON  ﾃﾞｰﾀ名1
            [{ASCENDING  }  KEY  IS  {ﾃﾞｰﾀ名2} ･･･] ･･･
            [{DESCENDING }
            [INDEXED  BY  {指標名1} ･･･]
```

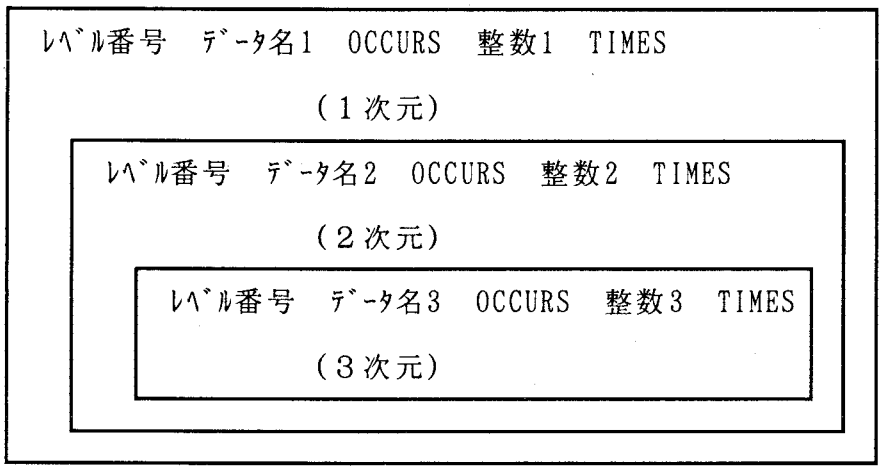
基本的な配列の定義の方法

```
01  ﾃｰﾌﾞﾙ名.
    03  ﾃｰﾌﾞﾙ要素  OCCURS  繰り返し回数  TIMES.
        05  ﾃﾞｰﾀ名1  PICTURE  文字列.
        05  ﾃﾞｰﾀ名2  PICTURE  文字列.
```

例）
```
01  一週間.
    03  日     OCCURS  7  TIMES.
        05  商品名     PICTURE  X(20).
        05  売上額     PICTURE  9(8).
```

一週間						
日	日	日	日	日	日	日
商品名 売上額	商品名 売上額	商品名 売上額	商品名 売上額	商品名 売上額	商品名 売上額	商品名 売上額

２次元以上の配列の定義は、OCCURS句を入れ子形式で定義する。



JIS-COBOLでは、７次元まで定義できる。

２．１３．２　配列の参照

表要素の出現番号を指定する添字には、次のものが使える。

添　字　付　け	整数	データ名（ 12 ）
	整数基本項目（数字項目）	データ名（ データ名 ）
	指標名（INDEXED　BY）	データ名（ 指標名 ）

添字は括弧で囲み、表要素のデータ名の直後に指定する。
２次元以上の添字は、次元の低い添字から順に指定する。

```
表要素データ名　（１次元の添字，　２次元の添字，　・・・）
```

（１）　数字定数（整数）またはデータ名の添字付け

添字とは、配列の要素を指定する整数値（１以上）である。
添字は数字定数と数字項目のデータ名が使用できる。

```
ﾃﾞｰﾀ名　（添字　［添字　［添字］］）
```

TABLE1(2)　　－　定数は１以上の整数である。
TABLE2(X,2)　－　添字の区切りは「,」または空白である。。
TABLE3(X Y Z)　－　添字がデータ名の場合は数字基本項目である。

データ名の添字に、ﾃｰﾌﾞﾙ要素番号を設定する場合は、「ＭＯＶＥ」を使用する。

```
01  X  PICTURE  9(4).
       :
    MOVE  ﾃｰﾌﾞﾙ要素番号  TO  X.
    COMPUTE  TABLE1(10)  =  TABLE1(X)  +  KINGAKU.
```

添字に定数ではなくデータ名を使用する場合、実行効率を向上させるために「USAGE IS　BINARY」の宣言を行うと良い。

```
01  X  PIC  S9(4).

MOVE  ZERO  TO  TABLE1(X)
```

実行効率が悪い

```
01  X  PIC  S9(4)
           USAGE  BINARY.

MOVE  ZERO  TO  TABLE1(X)
```

実行効率が良い

（２）　指標名による添字付け（指標付け）

指標付けを使用する場合は、「OCCURS」に「INDEXED　BY」で指標名を指定する。「INDEXED　BY」で指定した指標名の領域は、ＣＯＢＯＬコンパイラーが自動的に確保する。

```
ﾚﾍﾞﾙ番号　ﾃﾞｰﾀ名　OCCURS　整数　TIMES　INDEXED　BY　指標名
```

指標付けは、ハードウェア特性を有効に利用して実行効率を向上させるために、コンパイラーが、テーブルのアドレッシング情報としてアドレス、ポインター、オフセットなどの値を使用する。

```
(定義)
 01  ﾃｰﾌﾞﾙ名.
     03  ﾃｰﾌﾞﾙ要素　OCCURS　整数　TIMES　　INDEXED BY　指標名
                 PICTURE　文字列.
(参照)
        ﾃｰﾌﾞﾙ要素 ( 指標名 )
```

指標名にﾃｰﾌﾞﾙ要素番号の値は、「ＭＯＶＥ」ではなく「ＳＥＴ」を使用する。

```
SET　指標名　TO　ﾃｰﾌﾞﾙ要素番号.
COMPUTE　TABLE1(指標名)　=　TABLE1(指標名)　+　TANKA.
```

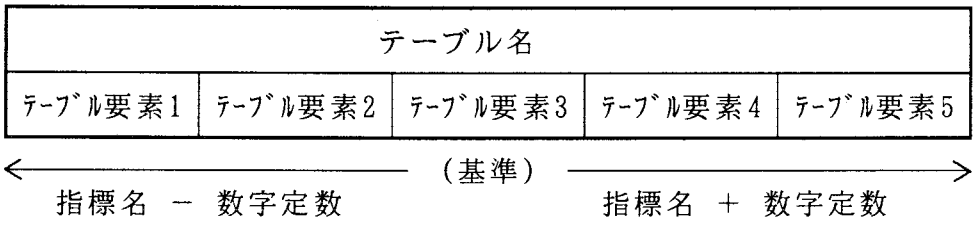
また、指標名は「USAGE　IS　INDEX」の指標ﾃﾞｰﾀ項目や数字項目に保管できる。

```
01　指標ﾃﾞｰﾀ名　　USAGE　IS　INDEX.
01　数字ﾃﾞｰﾀ名　　PICTURE　9(5).　　←整数であること。

SET　指標ﾃﾞｰﾀ名　TO　指標名.
SET　数字ﾃﾞｰﾀ名　TO　指標名.
```

相対指標付け

相対指標付けの記述は、指標名の後に空白を、そして「＋」、または「－」を、そして符号なし数字定数を指定する。

```
ﾃﾞｰﾀ名　(指標名 [± 数字定数]
            [指標名 [± 数字定数]　[ [指標名 [±数字定数] ] )
```



２．１３．３　ＳＥＴ文

指標名に表要素の出現番号を指定する場合に「ＳＥＴ」文を使用する。

書き方１

```
SET { 指標名1 } ･･･ TO { 指標名2 }
    { 一意名1 }        { 一意名2 }
                       { 整数1   }
```

書き方２

```
SET {指標名3} ･･･ { UP BY   } { 一意名3 }
                  { DOWN BY } { 整数2   }
```

書き方３

```
SET { {呼び名1} ･･･ TO { ON  } } ･･･
                      { OFF }
```

書き方４

```
SET {条件名1} ･･･ TO TRUE
```

指標名は、OCCURS句に「INDEXED　BY」指定を書くことで、そのテーブルと関連付けられる。

指標データ項目（USAGE　INDEX）は、指標名の保管領域として使用する。

添字付け

```
01  TABLE1.
    03 SSSS OCCURS 10 TIMES
            PIC  9(4).

01  X       PIC  9(2).

    MOVE 1 TO X.
LOOP.
    MOVE ZERO TO SSSS(X).
    COMPUTE  X  =  X  +  1.
    IF   X < 11 GO TO LOOP.

または
    PERFORM  VARYING  X  FROM  1
               BY 1 UNTIL X > 10
        MOVE ZERO  TO  SSSS(X)
    END-PERFORM
```

指標付け

```
01  TABLE1.
    03 SSSS OCCURS 10 TIMES
         INDEXED BY X   PIC 9(4)

    SET X TO 1.
LOOP.
    MOVE ZERO TO SSSS(X).
    SET X UP BY 1.
    IF  X < 11 GO TO LOOP.

または
    PERFORM  VARYING  X  FROM  1
               BY 1 UNTIL X > 10
        MOVE ZERO  TO  SSSS(X)
    END-PERFORM
```

書き方１のＳＥＴ文における組合せは、次のとおり。

送出し側	受取り項目		
	整数ﾃﾞｰﾀ項目	指標名	指標ﾃﾞｰﾀ項目
整数	×	○	×
整数ﾃﾞｰﾀ項目	×	○	×
指標名	○	○	○無変換
指標ﾃﾞｰﾀ項目	×	○無変換	○無変換

指標の値の設定は、ＳＥＴ文の他に「ＰＥＲＦＯＲＭ」文と「ＳＥＡＲＣＨ」がある。

```
SET    指標名1  TO  整数1

PERFORM  手続き名1  VARYING 指標名1 FROM 1 BY 1  UNTIL  指標名1 > 100
                    AFTER 指標名2 FROM 1 BY 1  UNTIL  指標名2 >  50


SEARCH  ALL  一意名1  WHEN  ﾃﾞｰﾀ名  =  一意名3  ～
(上記で使用される指標名はINDEXED  BYに指定した最初の指標名が使われる)
```

２．１３．４　配列の修飾

データ名を「ＩＮ」または「ＯＦ」で修飾してから配列の要素を指定する。
必要ならば、配列の要素の後に部分文字列を指定する。

配列の修飾は次の通り。

```
ﾃﾞｰﾀ名1  [ { IN } ﾃﾞｰﾀ名2] ･･･ [ { IN } { ファイル名1 } ]
           { OF }               { 通信記述名1 }
                                { 報告書名1   }
        [ ( {添字} ･･･) ] [ (最左端文字位置 : [長さ] ) ]
```

```
データ名１  OF  修飾名  (データ名２  OF  修飾名)
                  ↑              ↑
                  |              └ 指標名は修飾できない。
                  └ データ名１を含む集団項目名
```

```
01  INPUT-DATA.
    03  FILLER  OCCURS  10  TIMES.
      05  TANKA      PIC  999V9.
      05  SUURYOU    PIC  9(5).
      05  KINGAKU    PIC  9(8).

    COMPUTE  KINGAKU (1)  =  TANKA (2)  *  SUURYOU (3)

01  OUTPUT-DATA.
    03  FILLER  OCCURS  10  TIMES.
      05  TANKA      PIC  999V9.
      05  SUURYOU    PIC  9(5).
      05  KINGAKU    PIC  9(8).

  COMPUTE KINGAKU OF OUTPUT-DATA (1) =
          TANKA OF INPUT-DATA (2)  *  SUURYOU OF INPUT-DATA (3).
```

２．１３．５　配列の初期値

（１）　ＶＡＬＵＥ句の使用

表要素の個々のデータ項目に同じ初期値を設定する場合は、次のとおり。

```
01  TABLE1.
    03  ELEMENT  OCCURS  100  TIMES.

        05  数字  PICTURE  S9(5)  VALUE  ZERO.
        05  英字  PICTURE  X(10)  VALUE  SPACE.
```

配列全体に一回の記述で初期値を設定する場合は、表意定数（ZERO、SPACE等）やALL"定数"を使用する。
表要素が「USAGE IS　DISPLAY」の場合に使用する。

```
01  TABLE1    VALUE  ZERO.

    03  ELEMENT    OCCURS  100  TIMES.

        05  TANKA      PIC  999V9.
        05  SUURYOU    PIC  9(5).
        05  KINGAKU    PIC  9(5)V9.
        05  GOUKEI     PIC  S9(7).
```

用途は数字項目
（USAGE DISPLAY）

```
01  TABLE2    VALUE  SPACE.

    03  ELEMENT    OCCURS  100  TIMES.

        05  SIMEI      PIC  X(20).
        05  JYUUSYO    PIC  X(80).
        05  TEL-NO     PIC  X(10).
```

用途は英数字項目

```
01  TABLE3    VALUE  ALL  "　　000".

    03  ELEMENT    OCCURS  100  TIMES.

        05  CODE1      PIC  X(4).
        05  ATAI       PIC  9(3).
```

←空白4個とゼロ3個

SPACEとZEROの定数

```
01  TABLE4    VALUE  "ABCDEFGHIJKLMNOPQRSTUVWXYZ".

    03  ALPHA    OCCURS  26  TIMES  PICTURE X.
```

次の場合の配列にはVALUE句で初期値を設定できない。USAGEがDISPLAYでない。

```
01  TABLE5    VALUE  ZERO.

    03  ELEMENT    OCCURS  100  TIMES.

        05  SAIDAI     PIC  9(4)  USAGE  BINARY.
        05  SAITEI     PIC  S9(8)  USAGE  PAKED-DECIMAL.
```

不可

（２）　ＲＥＤＥＦＩＮＥＳ句の使用

表要素に別々の初期値を設定する場合は、REDEFINES句を使用する。
手順は、先にVALUE句で初期値を設定する。そして、REDEFINES句で初期値を設定した領域を再定義して、OCCURS句で配列を定義する。

```
01  VALUE-TABLE.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  01.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  02.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  03.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  04.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  05.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  06.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  07.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  08.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  09.
    03  FILLER    PIC 9(2)  USAGE BINARY VALUE  10.

01  OCCURS-TABLE    REDEFINES    VALUE-TABLE.

    03  OCCURS-XX  OCCURS  10  TIMES  PIC  9(2) USAGE BINARY.
```

個々の表要素に
初期値を設定する。

２次元の配列の初期値は、次のように設定する。

```
01  VALUE-TABLE.

    03  FILLER.
        05  FILLER  PIC  9(2)  VALUE   11.    ← ( 1 ,  1 )
        05  FILLER  PIC  9(2)  VALUE   12.    ← ( 1 ,  2 )
        05  FILLER  PIC  9(2)  VALUE   13.    ← ( 1 ,  3 )
        05  FILLER  PIC  9(2)  VALUE   14.    ← ( 1 ,  4 )

    03  FILLER.
        05  FILLER  PIC  9(2)  VALUE   21.    ← ( 2 ,  1 )
        05  FILLER  PIC  9(2)  VALUE   22.    ← ( 2 ,  2 )
        05  FILLER  PIC  9(2)  VALUE   23.    ← ( 2 ,  3 )
        05  FILLER  PIC  9(2)  VALUE   24.    ← ( 2 ,  4 )

    03  FILLER.
        05  FILLER  PIC  9(2)  VALUE   31.    ← ( 3 ,  1 )
        05  FILLER  PIC  9(2)  VALUE   32.    ← ( 3 ,  2 )
        05  FILLER  PIC  9(2)  VALUE   33.    ← ( 3 ,  3 )
        05  FILLER  PIC  9(2)  VALUE   34.    ← ( 3 ,  4 )

01  OCCURS-TABLE    REDEFINES  VALUE-TABLE.
    03  OCCURS-1  OCCURS  3  TIMES.
        05  OCCURS-2  OCCURS  4  TIMES    PIC  9(2).
```

（３）　ＰＥＲＦＯＲＭ　ＶＡＲＹＩＮＧ文の使用

```
  PERFORM  VARYING  X  FROM  1  BY  1  UNTIL  X  >  100

        MOVE  ZERO   TO  ELEMENE-A(X)
        MOVE  SPACE  TO  ELEMENT-B(X)

  END-PERFORM
```

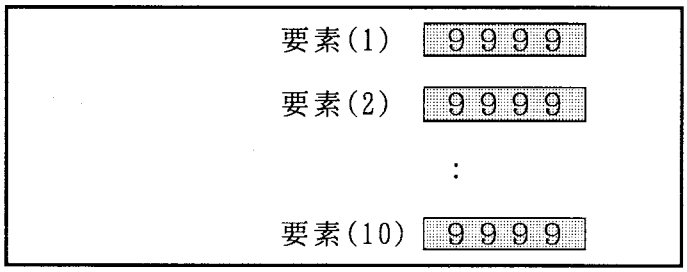
２．１３．６　配列の定義と構造

（１）　１次元

１次元の配列は、次のように定義できる。

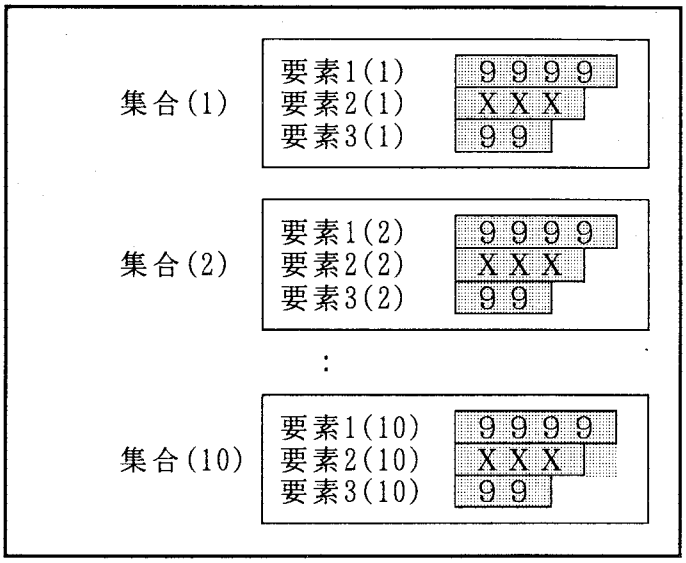
①要素が１個の場合

```
01  table1.
    03  要素  OCCURS  10  TIMES  PICTURE  S9(4).
```



②要素が複数個の場合

```
01  table1.
    03  集合  OCCURS  10  TIMES.
        05  要素1    PICTURE  S9(4).
        05  要素2    PICTURE  X(3).
            :
        05  要素n    PICTURE  9(2).
```



１次元の配列の例は、次のとおり。

```
01  一日.
    03  時間  OCCURS  24  TIMES.
```

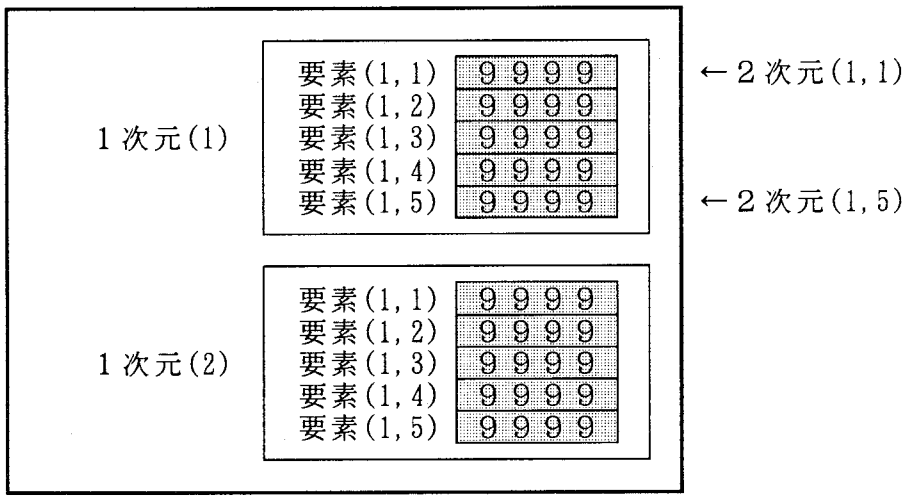
一日					
00時	01時	02時	～	22時	23時

（２）　２次元

２次元の配列は、次のように定義できる。

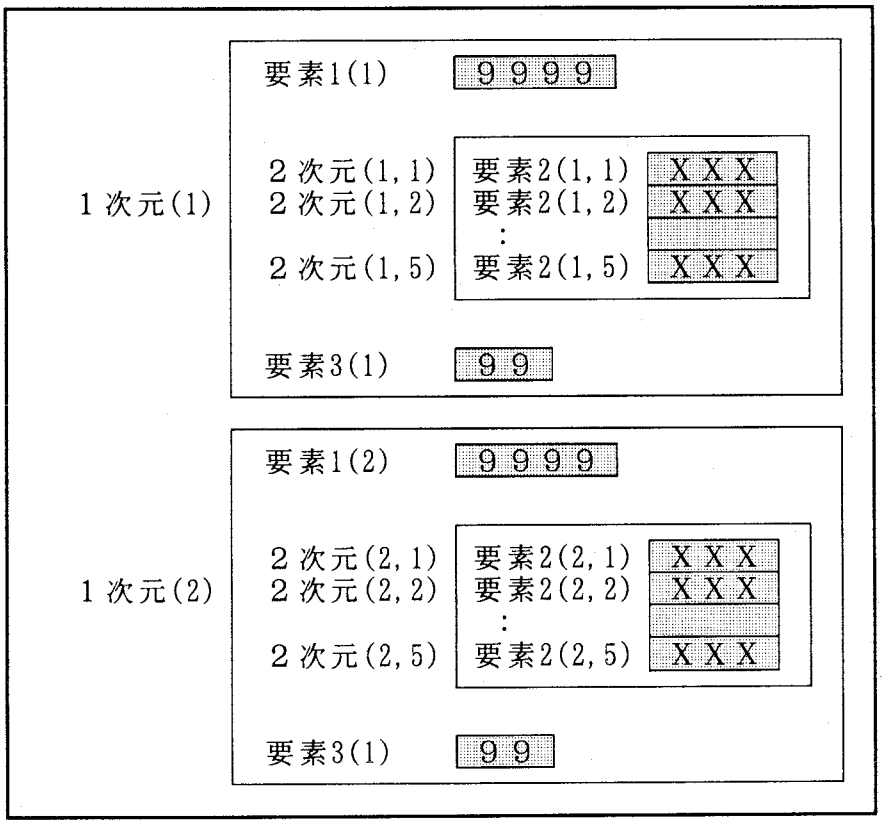
①単純な２次元の場合

```
01  table1.
    03  １次元  OCCURS  2  TIMES.
        05  ２次元  OCCURS  5  TIMES.
              07  要素  PICTURE  S9(4).
```



②複雑な２次元の場合（１次元との組み合わせ）

```
01  table1.
    03  １次元  OCCURS  2  TIMES.
        05  要素1          PICTURE  S9(4).
        05  ２次元  OCCURS  5  TIMES.
              07  要素2  PICTURE  X(3).
        05  要素3          PICTURE  S9(2).
```

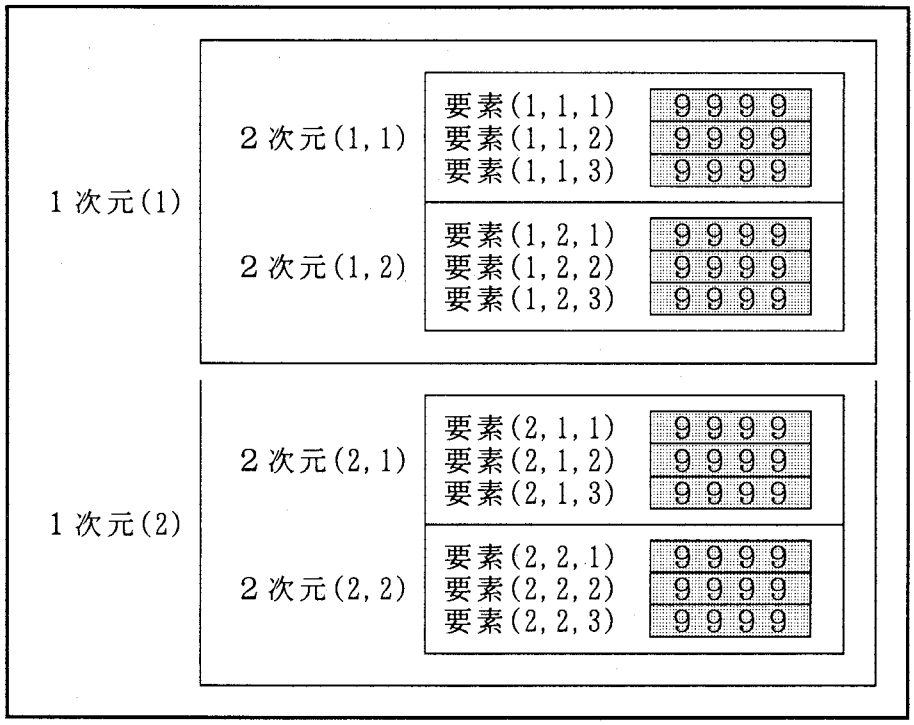


（３）　３次元

３次元の配列は、次のように定義できる。

①単純な３次元の場合

```
01  table1.
    03  １次元  OCCURS  2  TIMES.
        05  ２次元  OCCURS  2  TIMES.
            07  ３次元  OCCURS  3  TIMES.
                09  要素  PICTURE  S9(4).
```



②複雑な３次元の場合（１次元、２次元との組み合わせ）

```
01  table1.
    03  １次元  OCCURS  2  TIMES.
        05  要素1          PICTURE  S9(4).
        05  ２次元  OCCURS  2  TIMES.
              07  要素2  PICTURE  X(3).
              07  ３次元  OCCURS  3  TIMES.
                  09  要素3  PICTURE  S9(2).
        05  要素4          PICTURE  S9(2).
    03  要素5              PICTURE  X(5).
```

なお、ＪＩＳ－ＣＯＢＯＬでは、最大７次元まで使える。

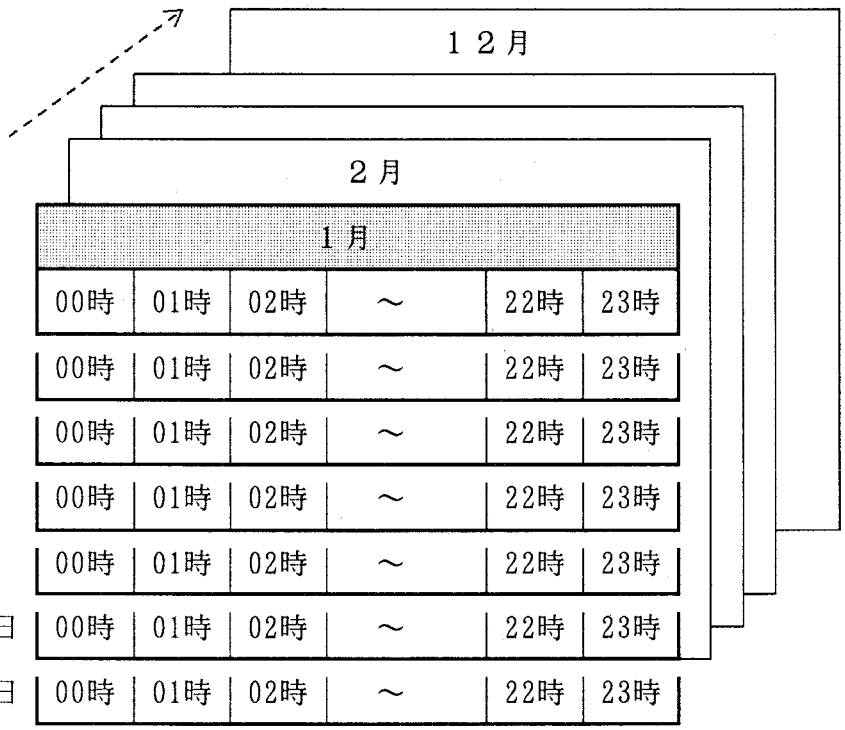
２次元の配列の例は、次のとおり。

```
01  月.
    03  日     OCCURS  31  TIMES.
        05  時間  OCCURS  24  TIMES.
```

	月					
1日	00時	01時	02時	～	22時	23時
2日	00時	01時	02時	～	22時	23時
3日	00時	01時	02時	～	22時	23時
	00時	01時	02時	～	22時	23時
：						
	00時	01時	02時	～	22時	23時
30日	00時	01時	02時	～	22時	23時
31日	00時	01時	02時	～	22時	23時

３次元の配列の例は、次のとおり。

```
01  年.
    03  月     OCCURS  12  TIMES.
        05  日     OCCURS  31  TIMES.
            07  時間  OCCURS  24  TIMES.
```



２．１３．７　配列の操作

配列の要素を順次参照する場合には、ＰＥＲＦＯＲＭ文のＶＡＲＹＩＮＧ指定を使う。
そして、２次元、３次元にはＡＦＴＥＲ指定を追加して使う。

（１）　１次元の操作
例では、添字１が初期値１から終了条件１になるまで増分１が加算される。

```
PERFORM  VARYING  添字１  FROM  初期値１  BY  増分１  UNTIL  終了条件１
        配列要素の参照は「ﾃﾞｰﾀ(添字１)」で行える
END-PERFORM
```

（２）　２次元の操作
例では、最初に添字２が初期値２から終了条件２になるまで増分２が加算される。
そして、終了条件２になると添字１に増分１が加算されて終了条件１が成立しないならば、再度「AFTER」指定を最初から実行する。

```
    PERFORM  手続き名１  [THRU  手続き名２]
        VARYING  添字１  FROM  初期値１  BY  増分１  UNTIL  終了条件１
          AFTER  添字２  FROM  初期値２  BY  増分２  UNTIL  終了条件２

手続き名１.
        配列要素の参照は「ﾃﾞｰﾀ(添字１,添字２)」で行える
```

（３）　３次元の操作

```
    PERFORM  手続き名１  [THRU  手続き名２]
        VARYING  添字１  FROM  初期値１  BY  増分１  UNTIL  終了条件１
          AFTER  添字２  FROM  初期値２  BY  増分２  UNTIL  終了条件２
          AFTER  添字３  FROM  初期値３  BY  増分３  UNTIL  終了条件３

手続き名１.
        配列要素の参照は「ﾃﾞｰﾀ(添字１,添字２,添字３)」で行える
```

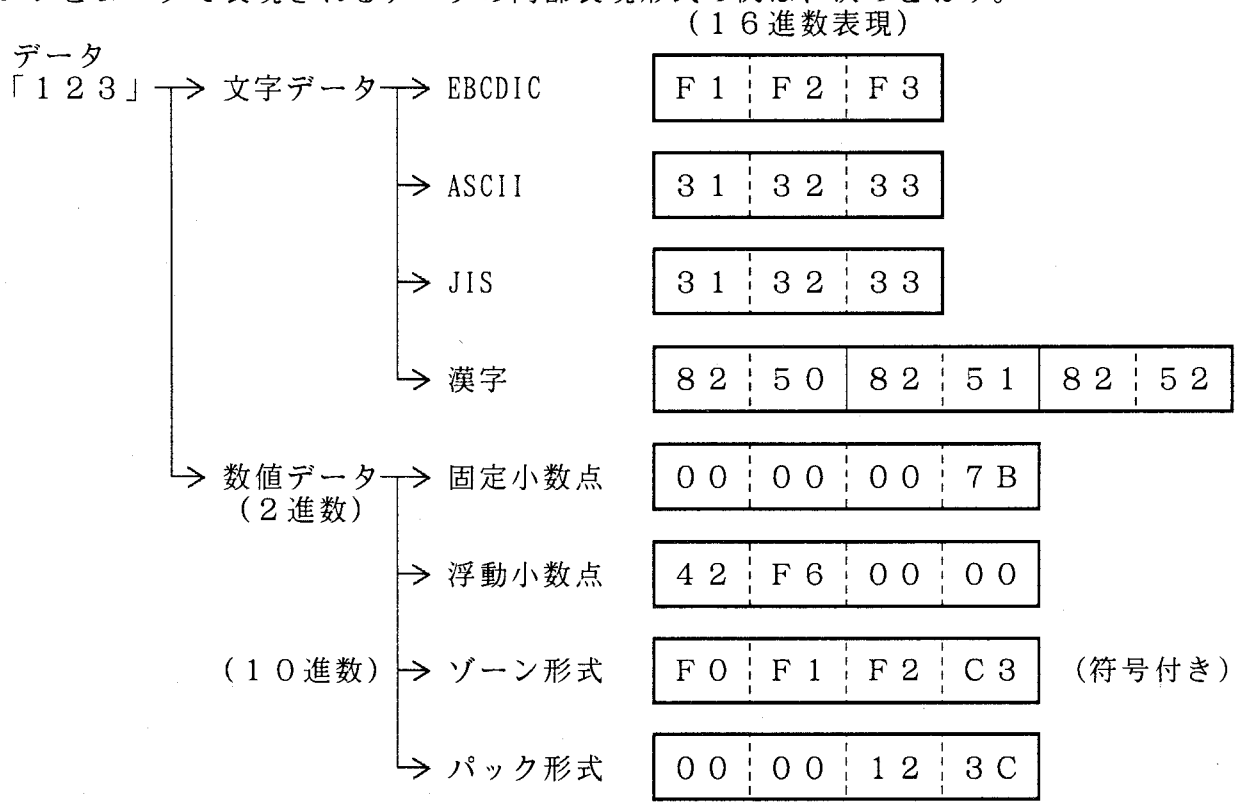
ＳＥＡＲＣＨ　ＡＬＬ文を使用すると自動的に配列の最初に定義した指標名が変化する。

```
01  TABLE1.
    03  表１  OCCURS  100  TIMES  ASCENDING  KEY  IS  要素１
                                   INDEXED  BY  指標名１  指標名２.
        05  要素１  PICTURE  S9(4).
        05  要素２  PICTURE  X(10).

    SEARCH  ALL  表１  AT  END CONTINUE
        WHEN  要素１(指標名１)  =  2001
              :
    END-SEARCH
```

表１の最初の指標名が使われる

指導上の留意点

◎データの表現形式
コンピュータで表現されるデータの内部表現形式の例は、次のとおり。



◎ＣＯＢＯＬの字類とデータの内容は次のとおり。

・英字　　英文字（英大文字、英小文字）または空白
・英数字　計算機文字集合の任意の文字
・数字　　０～９の文字
　　符号付きの場合、「＋」、「－」またはその他の符号を表す文字

◎データの定義
ＣＯＢＯＬではデータ部で定義するが、他の言語では、次のように定義する。

ＦＯＲＴＲＡＮ　－　実行文（命令文）より前に定義する。

PROGRAM／SUBROUTINE／FUNCTION／BLOCKDATA
宣言文 { 整数型　INTEGER　変数名 実数型　REAL　　変数名 複素数型　COMPLEX　変数名 論理型　LOGICAL　変数名 文字型　CHARACTER　変数名
実行文 宣言しない変数も使用できる。
END

ＰＬ／Ｉ　－　データ定義の順序は任意（実行文の後に宣言してもよい）

```
名前:  PROCEDURE  ;
------------------------------------------------------------
                          ｺｰﾄﾞ化算術変数
                            REAL | COMPLEX
                            FLOAT | FIXED
                            BINARY | DECIMAL
                          ｽﾄﾘﾝｸﾞ変数
                            BIT | CHARACTER | GRAPHICS
   DECLARE  変数名  {     ﾋﾟｸﾁｬｰ変数
                            PICTURE
                          ﾗﾍﾞﾙ変数
                            LABEL
                          ﾌｧｲﾙ変数（ﾌｧｲﾙ定数もある）
                            FILE  VARIABLE
                          その他

  実行文
            宣言しない変数もｺｰﾄﾞ化算術変数
            として使用できる。
------------------------------------------------------------
  END  名前;
```

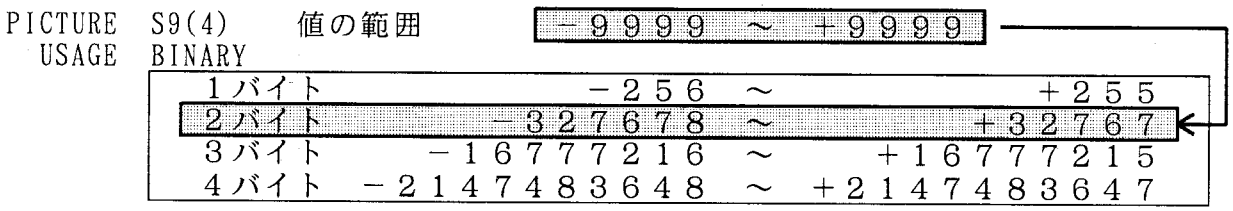
◎データの精度属性
ＣＯＢＯＬでは、PICTURE文字列でデータの表現可能範囲を定義する。
他の言語は、次のとおり。

ＣＯＢＯＬ	PICTURE　Ｓ９（ｎ）Ｖ９（ｎ） 符号の有無／整数部／小数点位置／小数部
ＦＯＲＴＲＡＮ	INTEGER＊長さ　変数名 REAL＊長さ　　変数名 データの型／領域の長さをバイト数で表現
ＰＬ／Ｉ	変数名　FIXED　DECIMAL(桁数，位取り因数) 変数名　FIXED　BINARY（ﾋﾞｯﾄ数，位取り因数） 領域全体の桁数、ﾋﾞｯﾄ数／小数部の桁数、ﾋﾞｯﾄ数
Ｃ	unsigned／signed　short／long　int　変数名 符号の有無／領域の大きさを表す／データの型

注：「＊長さ」はＪＩＳにはない。

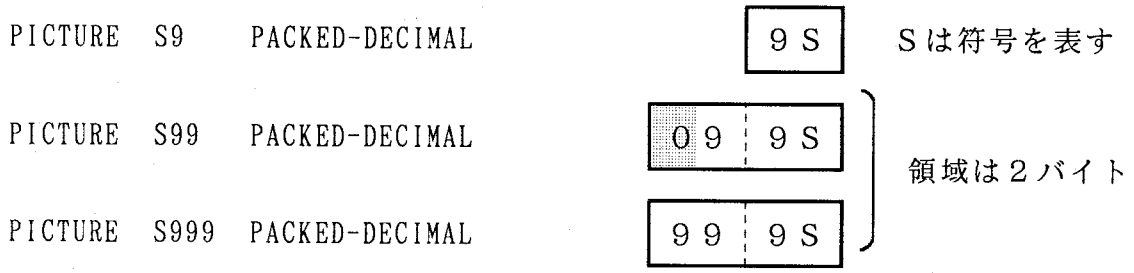
◎２進数の領域

ＣＯＢＯＬではPICTURE文字列で数字の桁数を定義するが２進数（USAGE　BINARY）では、桁数と領域サイズが一致しないのでデータ値の限界に注意する必要がある。



◎１０進数の領域　－　PACKED-DECIMAL

「USAGE　PACKED-DECIMAL」を使う場合には、数字の桁数は奇数桁（１、３、～１７）を定義する。偶数桁（２、４、～１８）で定義すると１桁余分に使えることになるが、「ON　SIZE　ERROR」の指定があれば定義した桁数が厳密にチェックされる。



◎レベル番号

レベル番号０１～４９でデータの階層を定義するとき、プログラムの変更に対処するため０１、０２、０３と連続番号で定義せず、０１、０３、０５などと間をあけるようにするとデータのグループ化などの変更が容易になる。

```
01  ﾃﾞｰﾀ名1.
    02  ﾃﾞｰﾀ名2
          :
        03  ﾃﾞｰﾀ名3
    02  ﾃﾞｰﾀ名4
```

```
01  ﾃﾞｰﾀ名1.
    03  ﾃﾞｰﾀ名2
          :
        05  ﾃﾞｰﾀ名3
    03  ﾃﾞｰﾀ名4
```

◎数字項目の符号

数字項目の定義では、できるだけ符号（Ｓ文字）を付けるようにすると実行性能がよくなる。

汎用コンピュータの１０進演算命令では、常に符号付きデータとして扱われる。

このため符号なし１０進数は、ＣＯＢＯＬコンパイラがデータ領域に格納するときに符号情報を変更する命令を追加することになる。

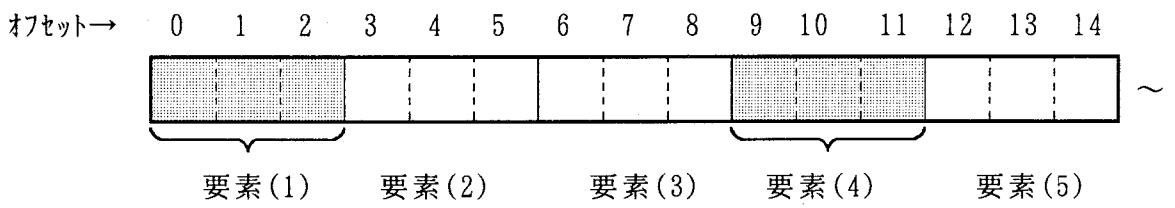
汎用コンピュータの例は、次のとおり。

```
USAGE  DISPLAYの場合  →    UNPK  ﾃﾞｰﾀ領域(5),temp(3)
                           OI    ﾃﾞｰﾀ領域+4,X'F0'      ←符号処理の命令

USAGE  PACKED-DECIMAL →    ZAP   ﾃﾞｰﾀ領域(3),temp(3)
                           OI    ﾃﾞｰﾀ領域+2,X'0F'      ←符号処理の命令
```

◎指標（データ）項目

「USAGE　INDEX」または「INDEXED　BY」の指標（データ）項目の内容はコンピュータのアドレッシングモードにもよるが、オフセット・アドレスを使用している場合が多い。

```
01  指標ﾃﾞｰﾀ  USAGE  INDEX.
01  TAB.
    03  要素  PICTURE  X(3)  OCCURS  10  TIMES  INDEXED  指標名.
```



要素(1)　要素(2)　要素(3)　要素(4)　要素(5)

SET　指標名　TO　1　→　指標名　0

SET　指標名　TO　4　→　指標名　9

指標名の値の計算例は、次のとおり。

指標の値　＝　要素のサイズ　＊　（要素番号　－　1）

このように設定しておくことにより「ＳＥＴ　数字項目　ＴＯ　指標名」で指標名から要素番号へ変換できる。

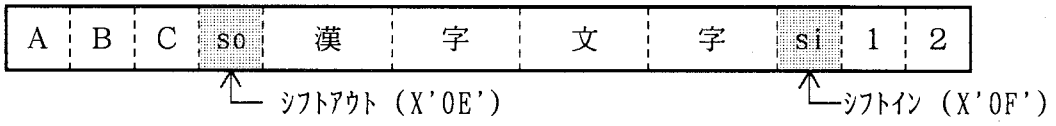
要素番号　＝　指標の値　／　要素のサイズ　＋　1

指標名には属性として要素のサイズがある。指標ﾃﾞｰﾀ名には要素の属性がない。

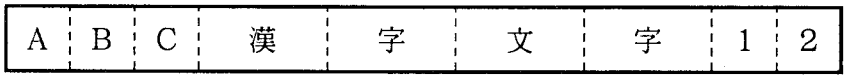
```
ＳＥＴ　指標ﾃﾞｰﾀ名　ＴＯ　指標名
        ↑―――――――――――┘
        変換は行われない。
```

```
ＳＥＴ　指標名１　　ＴＯ　指標名２
        ↑―――――――――――┘
        変換が行われる。
          ①ｔｅｍｐ　＝　指標名２　／　要素２のｻｲｽﾞ　＋　1
          ②指標名１　＝　要素１のｻｲｽﾞ　＊　（ｔｅｍｐ　－　1）
```

◎漢字コード（日本語処理）
汎用コンピュータで多く採用されているＥＢＣＤＩＣコードでは、英数字と漢字コードを区別するために「シフトアウト」、「シフトイン」の制御文字を使用している。



パソコンで多く採用されているシフトＪＩＳコードでは、英数字と漢字コードの混在が可能である。漢字コードは２バイトで構成されるがコードは英数字と重複しないように割り付けられている。



例として、データ部で漢字データ項目の定義は、PICTURE文字列に「Ｎ」を使用する場合が多い。「Ｎ」は漢字ｺｰﾄﾞの２バイトを意味する。
このデータ項目はフィールド全体が漢字（全角）データであり、英数字（半角）データとの混在はできない。全角と半角の混在はPICTURE文字列として「Ｘ」を使う。

```
01  半角データ  PICTURE  X(8)  VALUE  "ABCD1234".
01  混在データ  PICTURE  X(8)  VALUE  "漢字ｺｰﾄﾞ".
01  全角データ  PICTURE  N(4)  VALUE  "漢字項目".
```

編集項目のPICTURE文字列についても、「Ｎ」と組み合わせることができる文字は「Ｂ」や「／」、「０」等に制限される。

```
01  全角編集    PICTURE  NNBNNBNN.
```

表意定数のＳＰＡＣＥは、コード体系には半角と全角のコードが存在するが処理の都合上半角の空白コードを使用する場合が多い。

字類条件などについては、拡張されてデータ内容を検査できるようになっている。

```
IF  一意名  IS  KANJI
IF  一意名  IS  JAPANESE
```

漢字データ項目については、比較条件として「＝」のみで「＜」や［＞」が使えない場合もある。理由としては、漢字ｺｰﾄﾞの大小が論理的意味を持たないためである。

```
IF  漢字ﾃﾞｰﾀ項目1  =  漢字ﾃﾞｰﾀ項目2
```

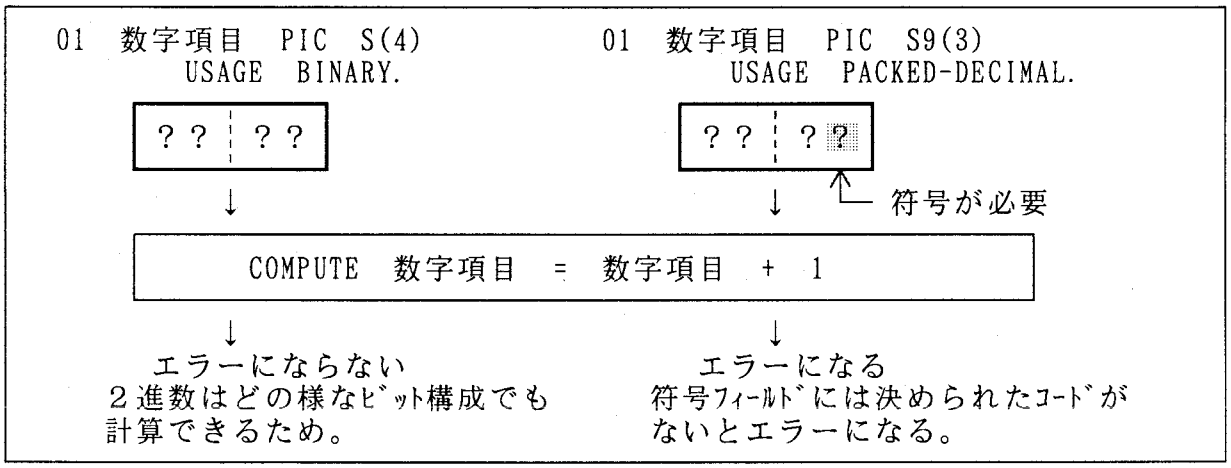
◎領域の再定義
領域の再定義は、記憶装置上の同じ領域に異なる属性のデータを格納できる。

言語名	再定義文	使用例
COBOL	REDEFINES	01 A PICTURE X(4). 01 B REDEFINES A PICTURE S9(4) USAGE BINARY. MOVE LOW-VALUE TO A COMPUTE B = B + 1
FORTRAN	EQUIVALENCE	INTEGER A REAL B EQUIVALENCE(A, B) A = 10 B = 1.23
C	union	union { int A; float B; } 共用体; 共用体.A = 1995; 共用体.B = 1.5;

◎初期値の設定
データ構造により初期値の与え方は異なる。言語によっては繰り返し回数を定数の前に付けることで記述を容易にしている。
データ項目の初期値の設定方法は、次のとおり。

COBOL	ＶＡＬＵＥ　定数 --- 01　ﾃﾞｰﾀ名　PIC　文字列　VALUE　定数. 01　ﾃﾞｰﾀ名　PIC　文字列　VALUE　ALL　定数. 01　集団項目　　VALUE　表意定数. 03　基本項目　PIC　文字列. 03　基本項目　PIC　文字列.
FORTRAN	ＤＡＴＡ　名前ﾘｽﾄ／定数ﾘｽﾄ／ --- DATA　変数　／1／ DATA　配列　／1,2,3,4,5,6,7,8,9,10／ DATA　配列　／繰返し数 *定数／
C	型指定　　宣言子　＝　式 --- int　変数　＝　1; int　配列[10]　＝　{1,2,3,4,5,6,7,8,9,10}; struct　{ int　　変数1; char　変数2[3]; }　変数名　＝　{1,　'A','B','C'};

初期値を設定せず算術演算を実行しても、実行時エラーにならない場合もある。



初期値は変数の記憶属性により、実行開始（ロード）時に１回だけ、または実行時に毎回初期設定される場合がある。

```
関数名()
{
    static  静的変数　＝　定数;
    auto    自動変数　＝　式;　←関数が呼ばれる度に式の値で初期化される。
              ++静的変数;
              ++自動変数;
}
```

◎データ編集

COBOLでは、データ項目を定義するPICTURE文字列でデータの編集形式を指定できる。
Ｃ言語では、printf関数を使用して編集処理を行うことが多い。COBOLのように多彩な編集はできず入出力文と組み合わせて内部データを文字列に変換する程度である。
また、変数名と編集形式はそれぞれ独立している。

```
MOVE  ﾃﾞｰﾀ名  TO    編集項目
                    (数字編集項目)
                    (英数字編集項目)
```

Ｃ言語の編集の例は、次のとおり。

```
printf(編集形式の文字列,  変数名,･･･);

%  ──────────────────────→タイプ ―
   └→フラグ   ↑  └→ 出力幅  ↑  └→・精度  ↑
   (右寄左寄制御)

                    タイプ
                      d   整数形式（符号付き）
                      u   整数形式（符号なし）
                      o   ８進数形式
                      X   １６進数形式（Ａ～Ｆの大文字）
                      f   浮動小数点形式（[-]dddd.dddd）
                      e   浮動小数点形式（[-]d.ddddｅ[±]ddd）
                      c   １文字
                      s   文字列
```

ＦＯＲＴＲＡＮでは、「書式」といい入出力文と組み合わせて編集を指示する。
書式はＦＯＲＭＡＴ文で定義する。

```
        WRITE(装置番号,文番号)  変数名,･･･
文番号  FORMAT(書式仕様)
              Ｉｗ          整数型データの編集
              Ｉｗ．ｄ      同上
              Ｆｗ．ｄ      実数型、倍精度実数型、複素数型データの編集
              Ｅｗ．ｄ      同上
              Ｄｗ．ｄ      同上
              Ｇｗ．ｄ      同上
              Ｌｗ          論理型データの編集
              Ａ            文字型データの編集
              Ａｗ          文字型データの編集（ｗで欄の幅を指定）
             その他、編集制御としてｎＨ、Ｔｃ、ｎＸ、／、ＳＰなどがる。
```

◎データの表現形式

「USAGE　BINARY」のデータ項目は、ハードウェアの特性や言語の特性などにより格納表現がことなる場合がある。
次の例は、同一ハードウエア上でＣ言語とＣＯＢＯＬでデータの受け渡しを行った場合英数字データ（文字列）には問題ないが、BINARYデータに問題が発生してしまう例である。
ただし、これらの問題を解決するために「USAGE　BINARY」を「USAGE　COMP-5」に書き換えれば正しく処理できるようにＣＯＢＯＬの規格にはない機能が拡張されている。

```
#include <stdio.h>
main()
{
FILE *fp;
struct {
      char data1[4];
      int  data2   ;
} rec;
  if ((fp=fopen("XXX.DAT","wb"))==NULL) {
        printf("**open error !!¥n");
  }
  strncpy(rec.data1, "ABCD" ,4);
  rec.data2 =  10  ;
  fwrite((char*)&rec, sizeof(rec), 1, fp);
  fclose(fp);
}
```

ファイルの内容（JISコード　16進数形式）

41	42	43	44	0A	00

DATA1フィールド　DATA2フィールド

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   DAT2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
    SELECT IN-FILE ASSIGN TO "XXX.DAT".
DATA DIVISION.
FILE SECTION.
FD  IN-FILE.
01  IN-REC.
  03  DATA1  PICTURE X(4).
  03  DATA2  PICTURE S9(4) USAGE BINARY.
PROCEDURE DIVISION.
開始.
    OPEN INPUT IN-FILE.
    READ IN-FILE AT END
                DISPLAY "DATA ERROR !"
    END-READ.
    DISPLAY "DATA1=" DATA1
    DISPLAY "DATA2=" DATA2
    CLOSE IN-FILE.
終了.
    STOP RUN.
```

表示結果

```
DATA1=ABCD
DATA2=+2560
```

+2560の16進数表現は「0A00」である。

上記例は、BINARYデータについて異言語間の説明であるが、企業間でのデータの受け渡しではハードウェアが異なる場合があり、データの表現形式やコード体系の知識が必要である。