第4章　制御の流れ

指導目標

ソフトウェア（プログラム）の作成では、生産性と保守性が重要視されている。

本章では、生産性と保守性を実現するために、プログラミング上でどのような工夫があるのかを「制御の流れ」から解説を行って理解させる。

構造化プログラミングの要素である順番、反復、選択の制御構造に焦点をあてＣＯＢＯＬ言語での処理手続きの表現と使い方を指導する。また、ＣＯＢＯＬの条件の表現形式は多様であるので簡潔で適切な表現ができるように指導する。

生産性や保守性を具体的にあげると次のとおり。

○自分がみても他人がみても内容が理解し易い。
○プログラム論理がよく分かる。
○誤りがあっても、比較的はやく気がつく。
○機能拡張など手入れが容易に行える。
○他人にも保守が容易になる。
○独立性の高い適切なモジュール分割により、プログラムの再利用も向上する。
○欠陥を早く修復できる。
○誤りの少ないプログラムを作ることができる。

（「ソフトウェアの基礎編」より）

本章で解説する命令文は、次のとおり。

・ＩＦ文
・ＥＶＡＬＵＡＴＥ文
・ＰＥＲＦＯＲＭ文

内容のあらまし

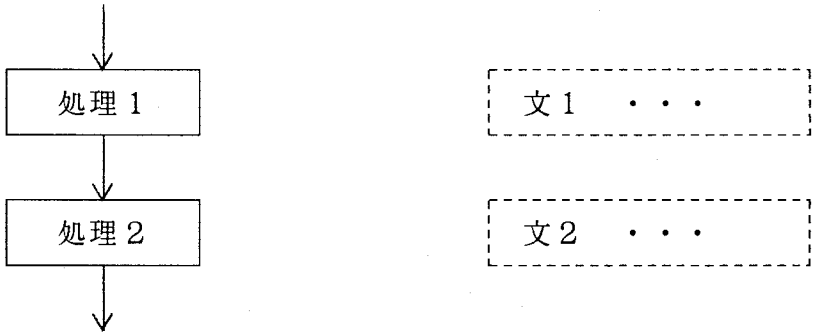
内　　容	説　明	議　論	机上実習	計算機実習
制御構造	順次構造、選択構造、繰り返し構造にの組み合わせでアルゴリズムが記述できることを説明する。		事例をもとに制御構造が表現できるようにﾌﾛｰﾁｬｰﾄを書いて実習する。	SIZE ERRORやAT ENDなどの選択構造になる命令文の書き方をｺﾝﾊﾟｲﾙを通して実習する。
COBOLの条件	制御の流れが分岐する条件について種類と使い方を説明する。	真理値の実現方式について議論する。	比較演算子の優先順位や結合に関わることについて事例をあげ実習させる。	
ＩＦ文	制御構造の選択構造をＩＦ文で説明する。		IFの入れ子の適切な表現を実習する。	
ＥＶＡＬＵＡＴＥ文	選択構造の拡張である多岐選択構造をEVALUATE文で使い方を中心に説明する。	Ｃ言語のswitchとの違いを議論する。		表現が多様にできるのでｺﾝﾊﾟｲﾙを通して習得する。
ＰＥＲＦＯＲＭ文	繰り返し構造を説明するとともにCOBOL独特の実行範囲指定について理解を深める。	関数、ｻﾌﾞﾙｰﾁﾝの考え方との相違点と利用法について議論する	内PERFORMや外PERFORM、UNTIL、VARYING指定が使いこなせるようにする。	VARYING、AFTERの動作について具体的に実習して使いこなせるようにする。

第４章　制御の流れ

４．１　制御構造

（１）　順次構造（Sequence）

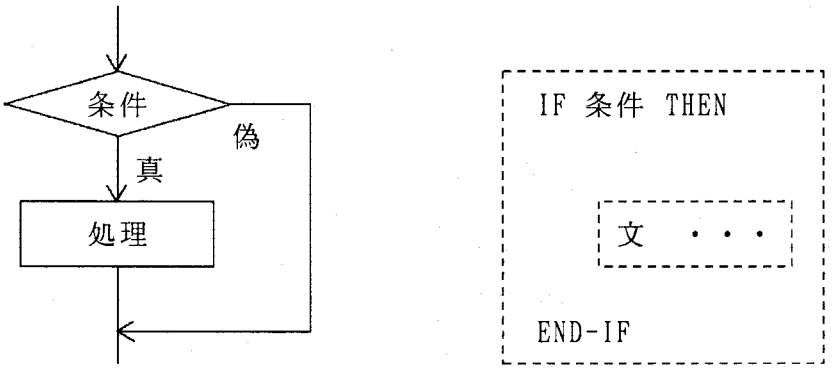
処理の順序が分かれることなく、記述されている命令（文）を逐次実行する。
処理の単位（単一命令、複数命令）には特に制限はないが、モジュール化については考慮する必要がある。
ＣＯＢＯＬの命令文としては、ＭＯＶＥ文などがある。



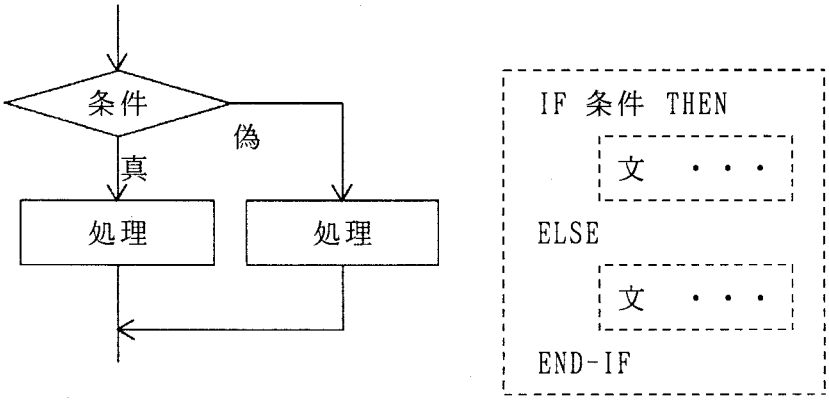
（２）　選択構造(Selection)

選択構造は、条件の真理値が真（true）か偽（false）により実行が選択される。
ＣＯＢＯＬでは、条件の種類として「単純条件」と「複合条件」がある。選択構造の命令文としては、ＩＦ文やＥＶＡＬＵＡＴＥ文などがある。
原始プログラムの記述では、構造範囲を明確にするために字下げ（indentation）して記述する。

①IF-THEN型

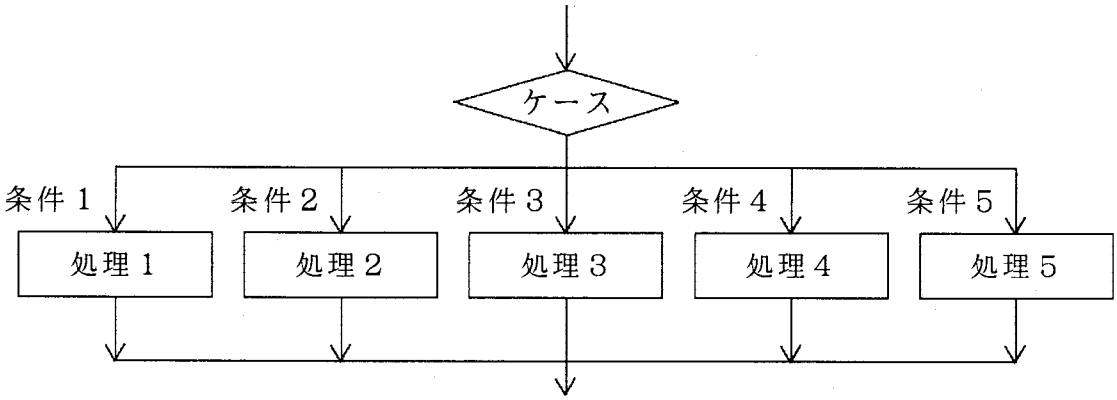


②IF-THEN-ELSE型



③CASE型（多岐選択構造）

構造をわかりやすくするために、１つの入り口と１つの出口にする。
ＣＯＢＯＬの命令文としては、ＥＶＡＬＵＡＴＥ文がある。

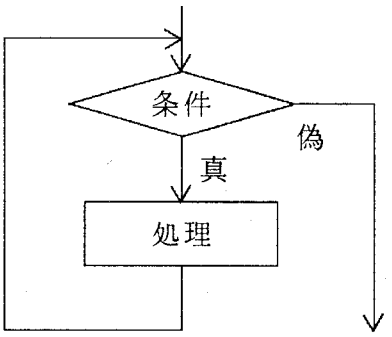



```
EVALUATE 一意名
  WHEN 条件１
     文１  ・・・
  WHEN 条件２
     文２  ・・・
  WHEN 条件３
     文３  ・・・
  WHEN 条件４
     文４  ・・・
  WHEN 条件５
     文５  ・・・
END-EVALUATE
```

（３）　繰り返し構造（Repetition）

ＤＯ－ＷＨＩＬＥ型は、条件が成立する間だけ処理を繰り返す。
ＤＯ－ＵＮＴＩＬ型は、条件が成立するまで処理を繰り返す。
ＣＯＢＯＬでは、繰り返し構造をＰＥＲＦＯＲＭ文で表現できる。どちらの型にするかは「ＴＥＳＴ　ＢＥＦＯＲＥ」指定か「ＴＥＳＴ　ＡＦＴＥＲ」指定で行う。
原始プログラムの記述では、繰り返し範囲を明確にするために字下げして記述する。

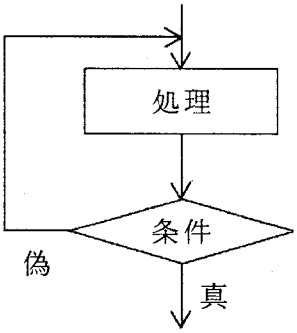
①DO-WHILE型　（処理の実行が０回もありうる）



COBOLのUNTIL指定の条件の書き方に注意がいる

```
PERFORM WITH TEST BEFORE UNTIL 条件

        文 ・・・

END-PERFORM
```

②DO-UNTIL型　（必ず１回は実行される）




```
PERFORM WITH TEST AFTER  UNTIL 条件

        文 ・・・

   END-PERFORM
```

４．１．１　条件文の定義

ＣＯＢＯＬの文（命令）は、無条件文、条件文、翻訳指示文、範囲明示文の４種類がある。
条件文とは、条件の真理値を調べて次にとる動作が真理値に左右される文である。
条件文には、次のものがある。

① EVALUATE文、IF文、SEARCH文、RETURN文
② READ文
　(AT END、NOT AT END指定、INVALID KEY、NOT INVALID　KEY指定)
③ WRITE文
　(INVALID KEY、NOT INVALID KEY指定、END-OF-PAGE、NOT-END-OF-PAGE指定)
④ DELETE文、REWRITE文、START文
　(INVALID KEY、NOT INVALID KEY指定)
⑤ ADD文、COMPUTE文、DIVIDE文、MULTIPLY文、SUBTRACT文)
　(ON SIZE ERROR、NOT ON SIZE ERROR指定)
⑥ RECEIVE文
　(NO DATA、WITH DATA指定)
⑦ STRING文、UNSTRING文
　(ON OVERFLOW、NOT ON OVERFLOW指定)
⑧ CALL文
　(ON OVERFLOW、ON EXCEPTION、NOT ON EXCEPTION指定)

４．１．２　無条件文の定義

無条件文は、無条件動詞で始まり特定の動作を行うのもと、明示範囲符（END-IF等）で区切られた条件文（範囲明示文）がある。
ＣＯＢＯＬの「一般形式」のなかで「無条件文」と指定してある場合、終止符またはいくつかの連続した無条件文のことである。

無条件動詞は、次のとおり。

ＡＣＣＥＰＴ	ＧＥＮＥＲＡＴＥ	ＲＥＬＥＡＳＥ
ＡＤＤ(*)	ＧＯ　ＴＯ	ＲＥＷＲＩＴＥ
ＡＬＴＥＲ	ＩＮＩＴＩＡＬＩＺＥ	ＳＥＮＤ
ＣＡＬＬ(*)	ＩＮＩＴＩＡＴＥ	ＳＥＴ
ＣＡＮＣＥＬ	ＩＮＳＰＥＣＴ	ＳＯＲＴ
ＣＬＯＳＥ	ＭＥＲＧＥ	ＳＴＡＲＴ
ＣＯＭＰＵＴＥ	ＭＯＶＥ	ＳＴＯＰ
ＣＯＮＴＩＮＵＥ	ＭＵＬＴＩＰＬＹ	ＳＴＲＩＮＧ(*)
ＤＥＬＥＴＥ	ＯＰＥＮ	ＳＵＢＴＲＡＣＴ
ＤＩＳＡＢＬＥ	ＰＥＲＦＯＲＭ	ＳＵＰＰＲＥＳＳ
ＤＩＳＰＬＡＹ	ＰＵＲＧＥ	ＴＥＲＭＩＮＡＴＥ
ＤＩＶＩＤＥ(*)	ＲＥＡＤ(*)	ＵＮＳＴＲＩＮＧ(*)
ＥＮＡＢＬＥ(*)	ＲＥＣＥＩＶＥ(*)	ＷＲＩＴＥ(*)
ＥＸＩＴ		

注：(*)付きのものは、次の指定がない場合に無条件動詞となる。
「ON　SIZE　ERROR」や「NOT　ON　SIZE　ERROR」
「INVALID　KEY」や「NOT　INVALID　KEY」
「AT　END」や「NOT　AT　END」
「END-OF-PAGE」や「NOT　END-OF-PAGE」
「ON　OVERFLOW」や「ON　EXCEPTION」、「NOT　ON　EXCEPTION」
「NO　DATA」や「WITH　DATA」

４．１．３　範囲明示文

文の範囲は、明示範囲符と暗黙範囲符で決まる。
明示範囲符を含む文を「範囲明示文」という。明示範囲符は次のとおり。

END-ADD	END-IF	END-REWRITE
END-CALL	END-MULTIPLY	END-SEARCH
END-COMPUTE	END-PERFORM	END-START
END-DELETE	END-READ	END-STRING
END-DIVIDE	END-RECEIVE	END-SUBTRACT
END-EVALUATE	END-RETURN	END-UNSTRING
		END-WRITE

範囲明示文は、条件文を無条件文にする働きもある。

```
IF  条件  THEN  {文１} ･･･  ELSE  {文２} ･･･  [END-IF]
```

文とあるので無条件文、条件文が使え、IFの入れ子が書ける。

```
READ  ﾌｧｲﾙ名  [NEXT]  RECORD  [INTO  一意名]
      [AT  END  無条件文１]
      [NOT  AT  END  無条件文２]
      [END-READ]
```

無条件文とあるので、IFなどの条件文は書けない。
ただし、範囲明示文を使えば条件命令も書ける

暗黙範囲符とは、
(1)完結文の終わりにあって、先行するすべての文を終了させる分離符の終止符（．）。
(2)別の文を含んでいる文中で、含まれる文の範囲を終了させる指定。
　例としては、ＥＬＳＥ指定、ＷＨＥＮ指定、NOT　AT　END指定など。

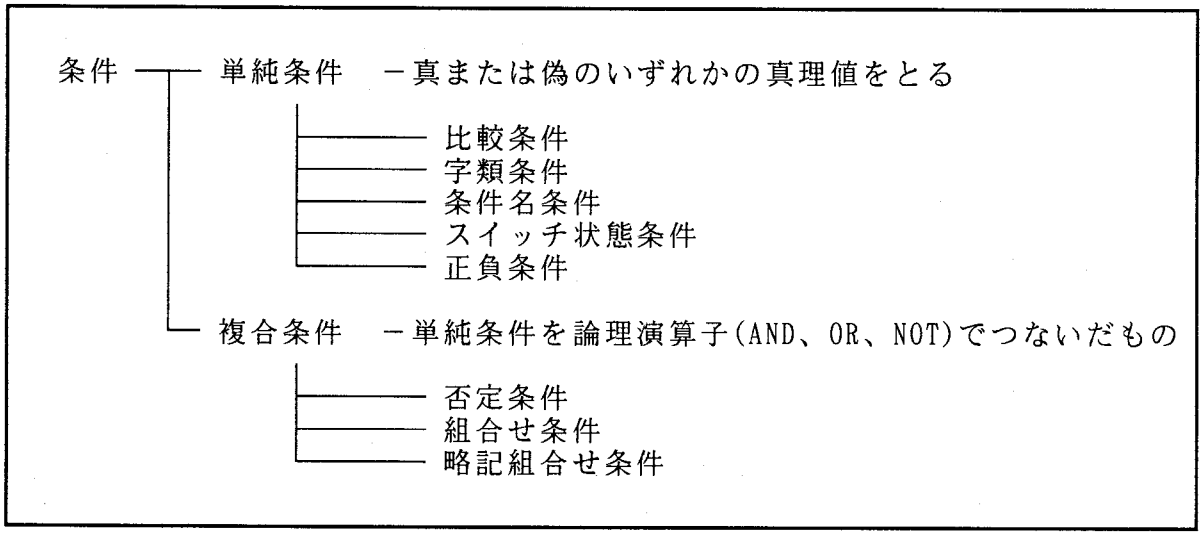
```
READ  ﾌｧｲﾙ名  AT  END
        文１
        文２ .

文３ .

IF  条件１  THEN
        文１
        文２
        :
ELSE
        文３
        文４
END-IF
```

４．２　ＣＯＢＯＬの条件

条件は、その真理値を検査することで手続きの実行を選択するのに使う。
条件式は、ＩＦ、ＥＶＡＬＵＡＴＥ、ＰＥＲＦＯＲＭ、ＳＥＡＲＣＨ文で使う。

```
IF   条件1   THEN  文･･･  ELSE  文･･･  END-IF

EVALUATE  一意名
  WHEN  条件1    無条件文･･･
  WHEN  条件2    無条件文･･･
  WHEN  OTHER    無条件文･･･
END-EVALUATE

PERFORM  手続き名  UNTIL  条件1   無条件文･･･END-PERFORM

SEARCH  一意名  AT  END  無条件文
  WHEN  条件1    無条件文･･･
  WHEN  条件2    無条件文･･･
END-SEARCH
```

ＣＯＢＯＬにおける条件は、次のとおり。



４．２．１　比較条件

データ項目、定数等の二つの比較を行うのに使う。
条件が成立した場合を「真」、成立しない場合を「偽」とする。

```
{一意名1 }   {IS  [NOT] GREATER  THAN      }   {一意名2 }
{定数1   }   {IS  [NOT] >                  }   {定数2   }
{        }   {IS  [NOT] LESS   THAN        }   {        }
{算術式1 }   {IS  [NOT] <                  }   {算術式2 }
{指標名1 }   {IS  [NOT] EQUAL  TO          }   {指標名2 }
             {IS  [NOT]=                   }
             {IS  GREATER THAN OR EQUAL TO }
             {IS  >=                       }
             {IS  LESS THAN OR EQUAL TO    }
             {IS  <=                       }
```

比較対象が両方とも数字の場合は、それぞれのデータ内部表現（USAGE句）が何であっても比較できる。これ以外はUSAGEが同じでなければならない。

```
01  価格    PICTURE  S9(5)  USAGE  PACKED-DECIMAL.
01  税金    PICTURE  S9(4).
01  合計    PICTURE  S9(8)  USAGE  BINARY.

IF  合計  =  価格  +  税金  THEN  ～
```

比較対象のどちらかが集団項目の場合は、文字比較の規則が適用される。

項目のﾚﾍﾞﾙ	字類	項類
基本項目	英字	英字
	数字	数字
	英数字	数字編集 英数字編集 英数字
集団項目	英数字	英字 数字 数字編集 英数字編集 英数字

言語における比較条件の表現形式の概要は、次のとおり。

演算子	COBOL	FORTRAN	C
＝	＝	.EQ.	＝＝
≠	NOT ＝	.NE.	！＝
＜	＜	.LT.	＜
＞	＞	.GT.	＞
≦	＜＝、NOT ＞	.LE.	＜＝
≧	＞＝、NOT ＜	.GE.	＞＝
論理積	AND	.AND.	＆＆
論理和	OR	.OR.	｜｜
否定	NOT	.NOT.	！

４．２．２　字類条件

データが数字、英字、英小文字、英大文字かどうか調べる。

```
一意名　ＩＳ　［NOT］{ NUMERIC
                      ALPHABETIC
                      ALPHABETIC-LOWER
                      ALPHABETIC-UPPER
                      字類名 }
```

データ項目は「USAGE　DISPLAY」であること。

字類条件の意味は、次のとおり。

条　件	字類・項目	比　較　内　容
NUMERIC	数字	０、１、２～９、または、符号
ALPHABETIC	数字以外	英大文字(A～Z)、英小文字(a～z)、空白
ALPHABETIC-LOWER	数字以外	英小文字（ａ～ｚ）、空白
ALPHABETIC-UPPER	数字以外	英大文字（Ａ～Ｚ）、空白
字類名	数字以外	SPECIAL-NAMESで定義した文字の種類

```
01  入力ﾃﾞｰﾀ1    PICTURE  9(3).
01  入力ﾃﾞｰﾀ2    PICTURE  X(10).

IF  入力ﾃﾞｰﾀ1 IS NUMERIC
    DISPLAY  "データは数字です"
ELSE
    DISPLAY  "データは数字でない"

IF  入力ﾃﾞｰﾀ2 IS ALPHABETIC-UPPER THEN   DISPLAY  "英大文字です".

EVALUATE  TRUE
  WHEN  入力ﾃﾞｰﾀ2  IS  NUMERIC          ～
  WHEN  入力ﾃﾞｰﾀ2  IS  ALPHABETIC-LOWER  ～
  WHEN  入力ﾃﾞｰﾀ2  IS  ALPHABETIC-UPPER  ～
  WHEN  OTHER                           ～
END-EVALUATE
```

４．２．３　条件名条件

条件データ項目の値が条件名に割り当られている値に等しいかどうか比較する。

```
条件名
```

使用例は、次のとおり。

```
01  数字ﾃﾞｰﾀ        PICTURE  9(1).

    88   零           VALUE  0.
    88   奇数         VALUE  1  3  5  7  9.
    88   偶数         VALUE  2  4  6  8.
    88   四捨         VALUE  1  THRU  4.
    88   五入         VALUE  5  THRU  9.
          ↑----- 条件名

IF    奇数    THEN  DISPLAY  "値は奇数です。".   ←条件名条件の例

IF  数字ﾃﾞｰﾀ  =  1  OR  数字ﾃﾞｰﾀ  =  3  OR      ←条件名条件を
    数字ﾃﾞｰﾀ  =  5  OR  数字ﾃﾞｰﾀ  =  7  OR        使用しない場合
    数字ﾃﾞｰﾀ  =  9  THEN                          の記述例
                    DISPLAY  "値は奇数です。".

IF    四捨    THEN  DISPLAY  "値を切り捨てます。".

IF  数字ﾃﾞｰﾀ  >=  1  AND  数字ﾃﾞｰﾀ  <=  4  THEN
                    DISPLAY  "値を切り捨てます。".
```

ＳＥＴ文で条件データ項目の値を「真」に設定できる。

```
SET   { {条件名１} ･･･  TO  TRUE } ･･･
```

条件名１のVALUE句に複数の定数を指定した場合には、最初に指定した定数の値が条件データ項目に設定される。

```
01  数字ﾃﾞｰﾀ        PICTURE  9(1).

    88   零           VALUE  0.
    88   奇数         VALUE  1  3  5  7  9.
    88   偶数         VALUE  2  4  6  8.
    88   四捨         VALUE  1  THRU  4.
    88   五入         VALUE  5  THRU  9.
          ↑----- 条件名
    SET   奇数   TO  TRUE      ←数字ﾃﾞｰﾀの値は「1」になる。

    SET   四捨   TO  TRUE      ←数字ﾃﾞｰﾀの値は「1」になる。
```

４．２．４　スイッチ状態条件

作成者の定義したスイッチのオン、オフの状態を調べる。
環境部の特殊段落（SPECIAL-NAMES）でスイッチのオン、オフに条件名を付けて使用する。

```
条件名
```

スイッチのオン、オフはプログラム実行時に外部から与えられる。

```
ENVIRONMENT  DIVISION.

SPECIAL-NAMES.
      実行ｽｲｯﾁ名    ON  STATUS  IS  続行.
                    OFF STATUS  IS  中断.
        ---- 作成者語                  ---ｽｲｯﾁ状態条件
             (呼び名)                        ON/OFFがある

PROCEDURE  DIVISION.

    IF  中断  THEN  DISPLAY  "処理を中断します。"
                    STOP RUN.
```

ＳＥＴ文でスイッチ状態条件の値を変更できる。

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

```
ENVIRONMENT  DIVISION.

SPECIAL-NAMES.
      実行ｽｲｯﾁ名    ON  STATUS  IS  続行.
                    OFF STATUS  IS  中断.
        ---- 作成者語                  ---ｽｲｯﾁ状態条件
                                        ON/OFFがある

PROCEDURE  DIVISION.

    SET  実行ｽｲｯﾁ名  TO  ON

    SET  実行ｽｲｯﾁ名  TO  OFF
```

４．２．５　正負条件

データ項目や算術式の値がゼロより、大きい（正:POSITIVE）、小さい（負:NEGATIVE）、等しい（零:ZERO）かを調べる。

```
算術式　ＩＳ　[NOT] { POSITIVE }
                     { NEGATIVE }
                     { ZERO     }
```

正負条件と比較条件の書き方の違いは、次のとおり。

```
01  数値データ     PICTURE  S9(18).

    IF  数値データ  POSITIVE  THEN  ～     ←正負条件
    IF  数値データ  >  ZERO    THEN  ～     ←比較条件

    IF  数値データ  NEGATIVE  THEN  ～     ←正負条件
    IF  数値データ  <  ZERO    THEN  ～     ←比較条件

    IF  数値データ  ZERO       THEN  ～     ←正負条件
    IF  数値データ  =  ZERO    THEN  ～     ←比較条件
```

４．２．６　否定条件

否定条件は、条件の前にＮＯＴを用いて真理値を逆転（真→偽、偽→真）させる。

```
        NOT  単純条件
```

否定条件に括弧を付けても、その真理値は変わらない。

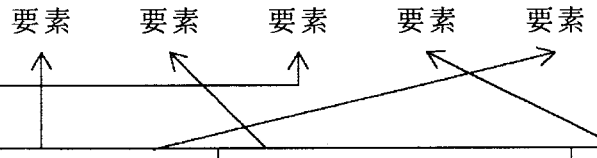
```
NOT { 単純条件　（真または偽のいずれかの真理値をとる）
      ・比較条件
      ・字類条件
      ・条件名条件
      ・スイッチ状態条件
      ・正負条件 }

IF  NOT  ﾃﾞｰﾀ名1  NOT  =  ﾃﾞｰﾀ名2
     ↑                   ↑
     └── 否定条件        └── 比較条件
```

４．２．７　組合せ条件

複数の条件を論理演算子ＡＮＤ、ＯＲを組み合わせて一つの条件を作る。

条件1 { { AND / OR } 条件2 } …

条件と論理演算子、括弧の組み合わせの規則は次の通り。



条件の要素	条件式の位置 先頭	末尾	直前にきてよい要素（先頭を除く）	直後にきてよい要素（末尾を除く）
単純条件	○	○	OR、NOT、AND、(○　条件　OR　条件 ×　)　条件	OR、AND、) ○　条件　AND　条件 ×　条件　NOT　条件
ＯＲ，ＡＮＤ	×	×	単純条件、) ○　条件　AND　条件 ×　AND　OR	単純条件、NOT、(○　OR　NOT　条件 ×　OR　AND　条件
ＮＯＴ	○	×	OR、AND、(○　AND　NOT　条件 ×　条件　NOT 条件	単純条件、(○　NOT　(条件) ×　NOT　NOT
(	○	×	OR、NOT、AND、(○　OR　(条件) ×　条件　(条件)	単純条件、NOT、(○　(NOT　条件) ×　(AND　条件)
)	×	○	単純条件、) ○　(（条件）) ×　(条件 AND)　条件	OR、AND、) ○　(条件)　AND 条件 ×　(条件)　NOT 条件

○は許される組み合わせ
×は許されない組み合わせ

４．２．８　略記組合せ比較条件

比較条件（＞、＝、＜等）を論理演算子（AND、OR）で組み合わせて記述する場合に、比較条件の左辺、または、左辺と比較演算子を省略できる。

```
比較条件 { { AND } [NOT]  [比較演算子]  右辺 }…
         { { OR  }
```

左辺の省略例は、次のとおり。

```
IF 金額 = 100 OR 金額 = 1000 THEN ～
IF 金額 = 100 OR      = 1000 THEN ～
```

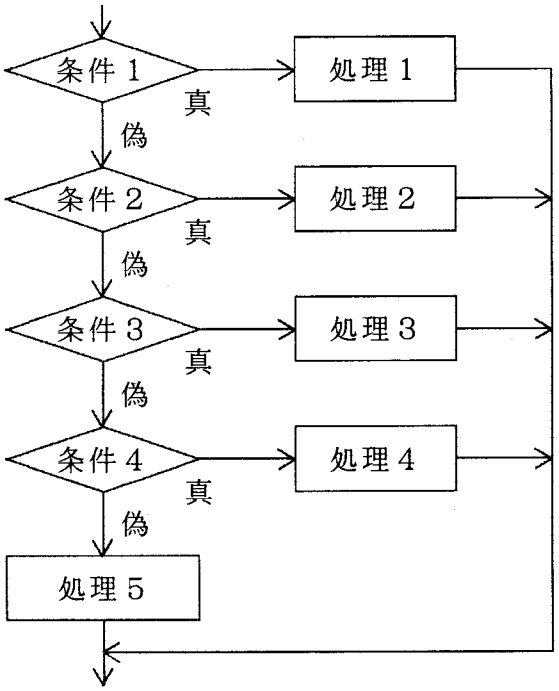
左辺と比較演算子の省略例は、次のとおり。

```
IF 金額 = 100 OR 金額 = 1000 THEN ～
IF 金額 = 100 OR      1000 THEN ～
```

４．３　ＩＦ

ＩＦ文は、条件を評価して、その真理値（真、偽）で次にとる動作が決まる。

```
IF  条件1  THEN { {文1} ･･･
                { NEXT SENTENCE }

{ ELSE     {文2} ･･･ [END-IF] }
{ ELSE   NEXT SENTENCE        }
{ END-IF                      }
```



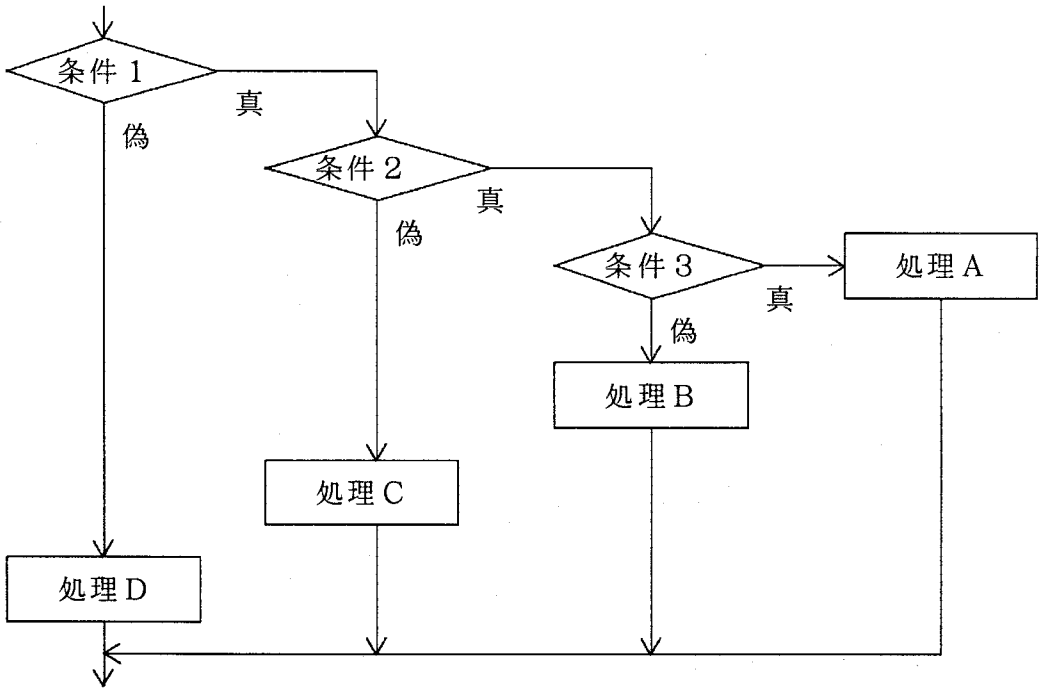

```
IF  条件１  THEN
              処理１

ELSE  IF  条件２  THEN
              処理２

ELSE  IF  条件３  THEN
              処理３

ELSE  IF  条件４  THEN
              処理４

ELSE
              処理５

END-IF
```

上記のＩＦ文をＥＶＡＬＵＡＴＥ文で記述すると次のようになる。

```
EVALUATE  TRUE
   WHEN  条件１     処理１
   WHEN  条件２     処理２
   WHEN  条件３     処理３
   WHEN  条件４     処理４
   WHEN  OTHER      処理５
END-EVALUATE
```

（１）　ＩＦの入れ子

ＩＦ文のＴＨＥＮの後やＥＬＳＥの後には、命令文としてＩＦ文が書ける。




```
ＩＦ　条件１　ＴＨＥＮ
　　ＩＦ　条件２　ＴＨＥＮ
　　　　ＩＦ　条件３　ＴＨＥＮ
　　　　　処理Ａ
　　　　ＥＬＳＥ
　　　　　処理Ｂ
　　　　ＥＮＤ－ＩＦ
　　ＥＬＳＥ
　　　処理Ｃ
　　ＥＮＤ－ＩＦ
ＥＬＳＥ
　処理Ｄ
ＥＮＤ－ＩＦ
```

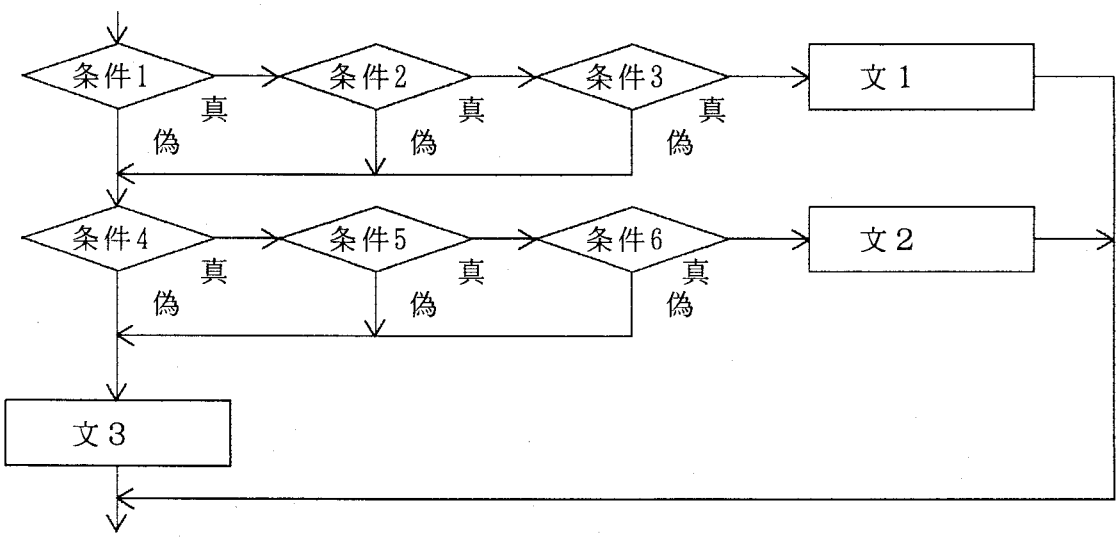
４．４　ＥＶＡＬＵＡＴＥ

ＥＶＡＬＵＡＴＥ（評価）文は、多岐分岐、多岐結合の構造を簡単に記述できる。

```
EVALUATE { 一意名1 | 定数1 | 式1 | TRUE | FALSE }  [ALSO { 一意名2 | 定数2 | 式2 | TRUE | FALSE } ] ･･･

{ {WHEN { ANY | 条件1 | TRUE | FALSE | [NOT] { { 一意名3 | 定数3 | 算術式1 } [THRU { 一意名4 | 定数4 | 算術式2 }] } }

[ALSO { ANY | 条件2 | TRUE | FALSE | [NOT] { { 一意名5 | 定数5 | 算術式3 } [THRU { 一意名6 | 定数6 | 算術式4 }] } }
  ] ･･･} ･･･

   無条件文1} ･･･

[WHEN  OTHER  無条件文2]
[END-EVALUATE]                                   注：THROUGH／THRUは省略
```

ネスト条件の簡略化

```
EVALUATE  一意名1    一意名2    一意名3
    WHEN  条件1      条件2      条件3      文1
    WHEN  条件4      条件5      条件6      文2
    WHEN  OTHER                            文3
END-EVALUATE
```



４．５　ＰＥＲＦＯＲＭ

ＰＥＲＦＯＲＭ文は、明示的（手続き名1）に幾つかの手続きに制御を移し、指定した範囲の手続きの実行が終わると、暗黙的に制御を戻す。

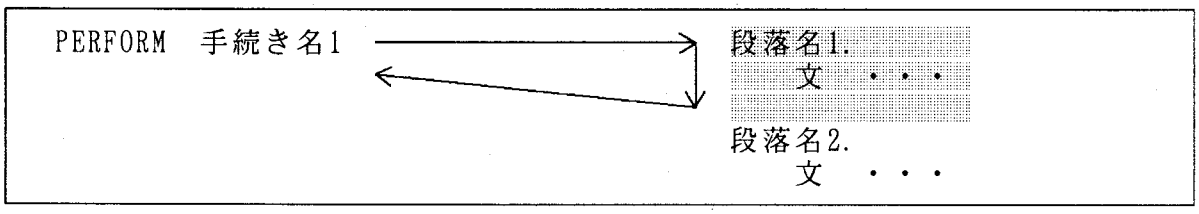
```
PERFORM　手続き名1　［THRU　手続き名2］

PERFORM　［文］・・・　END-PERFORM
```

手続き名を指定した場合（外PERFORM）は、END-PERFORMを書いてはならない。
手続き名を省略した場合（内PERFORM）は、END-PERFORMを書かなければならない。
ＰＥＲＦＯＲＭ文には、次の形式がある。

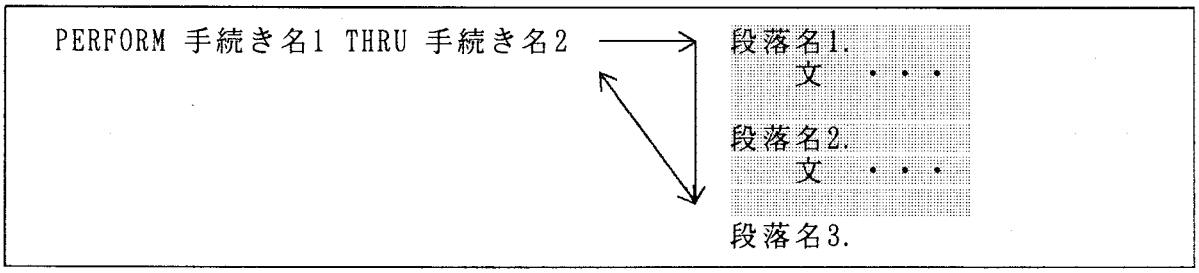
- 基本　ＰＥＲＦＯＲＭ
- ＴＩＭＥＳ指定　ＰＥＲＦＯＲＭ
- ＵＮＴＩＬ指定　ＰＥＲＦＯＲＭ
- ＶＡＲＹＩＮＧ指定　ＰＥＲＦＯＲＭ

「手続き名1」は制御が最初に移るところである。制御の戻り（範囲の終了）は手続き名の種類（段落名、節名）により異なる。

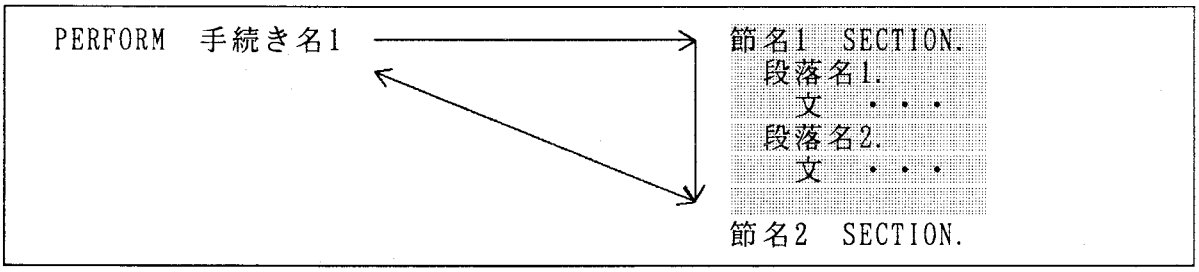
①手続き名が段落名の場合、その段落の最後の文の後で戻る。



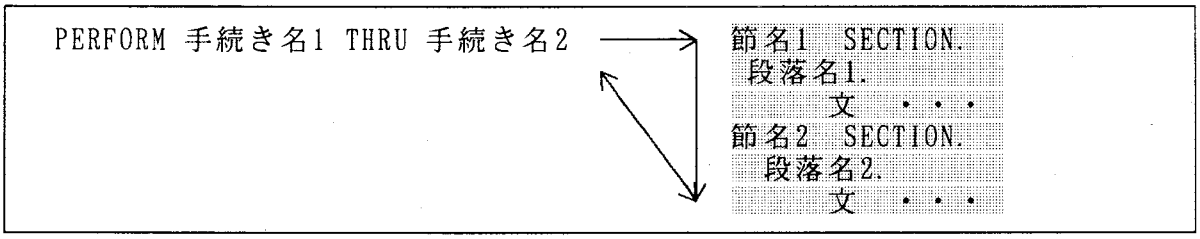
②手続き名1と手続き名2が指定され、それが段落名の場合、手続き名2の最後の文の後で戻る。



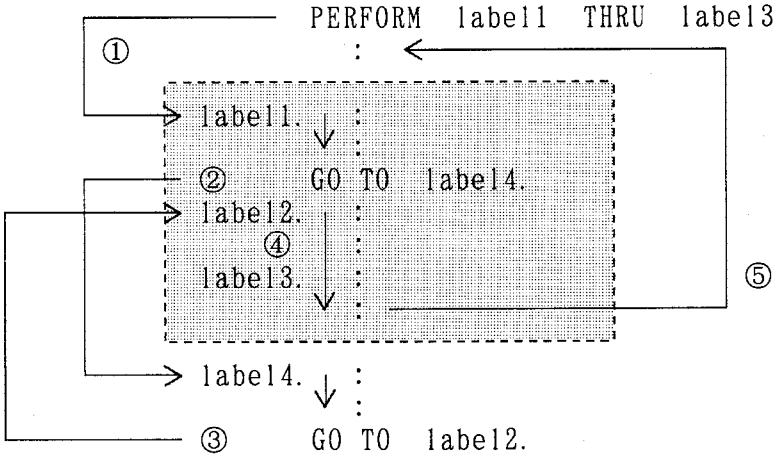
③手続き名が節名の場合、その節名の最後の文の後で戻る。



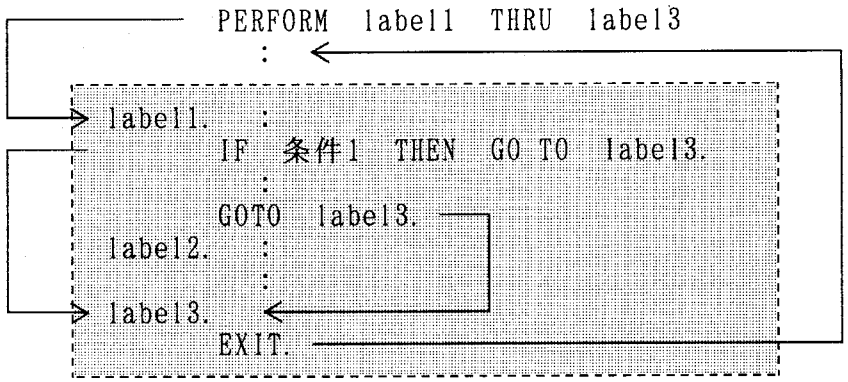
④手続き名1と手続き名2が指定され、それが節名の場合、手続き名2の節の最後の文の後で戻る。



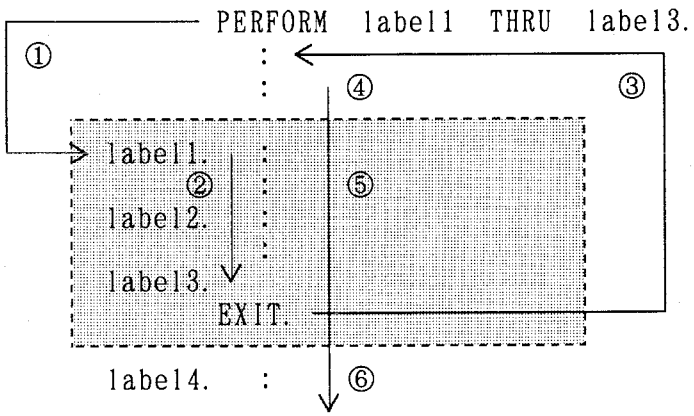
ＰＥＲＦＯＲＭ文の手続き名1と手続き名2の関係は、制御の実行が手続き名1から始まって手続き名2で終了するだけでよい。
手続き名1から手続き名2の間に、ＧＯ　ＴＯやＰＥＲＦＯＲＭで範囲外で分岐してもその後範囲内に戻ればよい。



手続き名1と手続き名2の範囲に複数の経路ある場合、制御を戻すには手続き名2はＥＸＩＴ文だけの段落にする。



手続き名1と手続き名2にＰＥＲＦＯＲＭ文以外で制御が移った場合、制御は範囲の最後の文の次に移る。

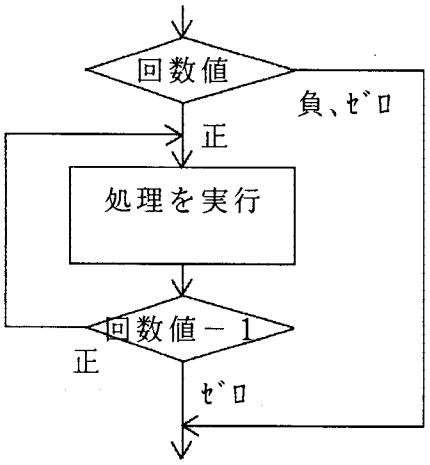


４．５．１　ＴＩＭＥＳ指定

ＴＩＭＥＳ指定は、一意名1または定数1で指定された回数だけ実行する。
指定する回数は整数値である。

```
PERFORM　手続き名1　［THRU　手続き名2］　一意名1／定数1　TIMES

PERFORM　一意名1／定数1　TIMES
　　［文］・・・
END-PERFORM
```

回数がゼロまたは負の場合は、実行されず次の文を実行する。



ＰＥＲＦＯＲＭを実行中に、一意名の値を変更しても、最初に指定した回数は変わらない。

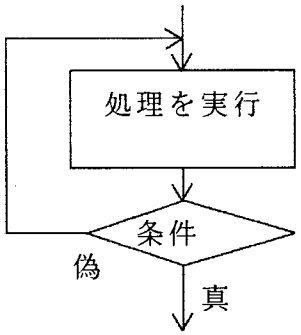
```
01　回数　　PICTURE　S9(8)　　USAGE　BINARY.
　　　　：
　　MOVE　10　TO　回数
　　　　：
　　PERFORM　回数　TIMES
　　　　：
　　　MOVE　ZERO　TO　回数　　←　実行回数は変わらない。
　　　　：　　　　　　　　　　　PERFORMを抜け出る場合は、GO TOを
　　　　　　　　　　　　　　　　使う。
　　END-PERFORM
```

４．５．２　ＵＮＴＩＬ指定

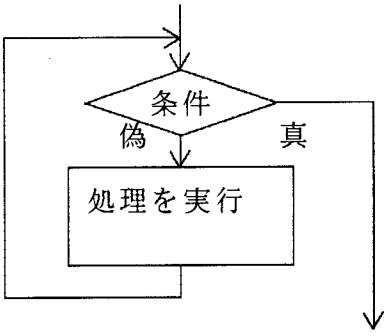
ＵＮＴＩＬ指定で書かれた条件が真になるまで実行を繰り返す。
「TEST　BEFORE」またはＴＥＳＴを省略した場合は、条件の検査を実行前に行う。
「TEST　AFTER」は条件の検査を実行後に行う。

```
PERFORM　手続き名1　[THRU　手続き名2]
          [WITH　TEST　BEFORE／AFTER]　　UNTIL　条件

PERFORM　　[WITH　TEST　BEFORE／AFTER]　　UNTIL　条件
      [文] ・・・
END-PERFORM
```

WITH　TEST　AFTER



WITH　TEST　BEFORE



条件のデータ項目は、条件が検査されるたびにデータ項目の内容や添字等が評価される。

```
01　回数　　PICTURE　S9(8)　　USAGE　BINARY.
        :
    MOVE　ZERO　TO　回数
        :
    PERFORM　UNTIL　回数　>　10
        :
      MOVE　11　TO　回数　　←　PERFORMを抜けでる場合
        :
    END-PERFORM
```

４．５．３　ＶＡＲＹＩＮＧ指定

ＶＡＲＹＩＮＧ指定は、ＰＥＲＦＯＲＭ実行中に幾つかの数値データ項目や指標名を規則的に変化させる場合に使う。

```
PERFORM  手続き名1  [THRU  手続き名2]
     [WITH  TEST  BEFORE／AFTER]
      VARYING  一意名1  FROM  一意名2  BY  一意名3  UNTIL  条件1
     [AFTER    一意名4  FROM  一意名5  BY  一意名6  UNTIL  条件2]
     [AFTER    一意名7  FROM  一意名8  BY  一意名9  UNTIL  条件3] ･･･

PERFORM    [WITH  TEST  BEFORE／AFTER]
       VARYING  一意名1  FROM  一意名2  BY  一意名3  UNTIL  条件1

        [文] ･･･    （内PERFORMの場合にはAFTER指定は使えない）

END-PERFORM
```

VARYINGやAFTERの一意名で指標名を指定する場合は、FROMとBYのデータ項目は整数項目か整数の定数でなければならない。

FROMで指標名を指定する場合は、VARYINGやAFTERの一意名は整数項目でなければならない。

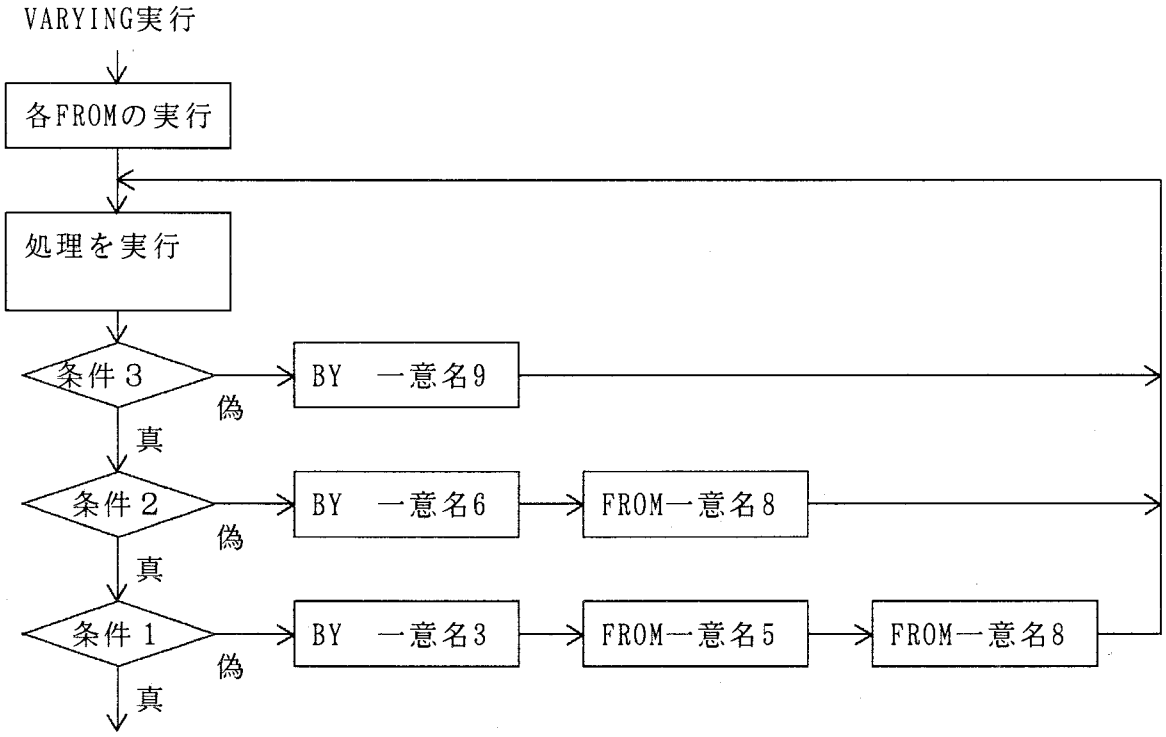
AFTER指定の最大繰り返しは、６回までとする。これは配列の最大次元が７レベルまで許しているためである。

```
     PERFORM  手続き名
              VARYING   １次元添字  FROM  開始番号  BY  増分  UNTIL  ～
                AFTER   ２次元添字  FROM  開始番号  BY  増分  UNTIL  ～
                AFTER   ３次元添字  FROM  開始番号  BY  増分  UNTIL  ～
       ６回     AFTER   ４次元添字  FROM  開始番号  BY  増分  UNTIL  ～
                AFTER   ５次元添字  FROM  開始番号  BY  増分  UNTIL  ～
                AFTER   ６次元添字  FROM  開始番号  BY  増分  UNTIL  ～
                AFTER   ７次元添字  FROM  開始番号  BY  増分  UNTIL  ～

 手続き名.
          文・・・
```

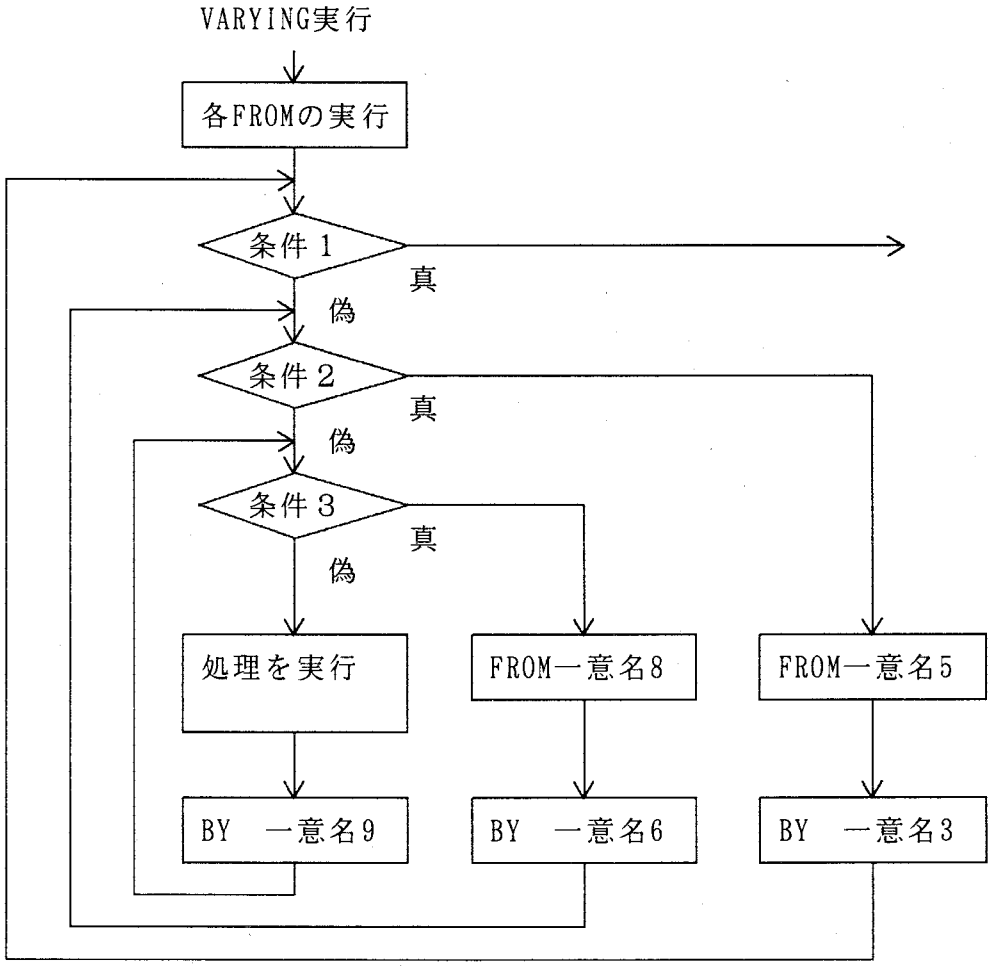
AFTER指定は、内PERFORMには使えないのでVARYING指定を入れ子にするよい。

```
  PERFORM  手続き名
           VARYING  一意名1  FROM  一意名2  BY  一意名3  UNTIL  条件1
             AFTER  一意名4  FROM  一意名5  BY  一意名6  UNTIL  条件2
     :
手続き名.
     文･･･
                                ↓
  PERFORM  VARYING  一意名1  FROM  一意名2  BY  一意名3  UNTIL  条件1
    PERFORM VARYING  一意名4  FROM  一意名5  BY  一意名6  UNTIL  条件2
              文･･･
    END-PERFORM
  END-PERFORM
```

４．５．４　ＴＥＳＴ　ＡＦＴＥＲ指定



４．５．５　ＴＥＳＴ　ＢＥＦＯＲＥ指定　（省略時の解釈はＢＥＦＯＲＥ指定である）



指導上の留意点

◎制御構造に関連する命令文の比較
構造化プログラミングの「順次」、「選択」、「繰り返し」を構成する命令文を各言語で比較すると、選択のＩＦ文には言語間の差異はほとんどない。
選択のＣＡＳＥ型は、言語により呼び方や機能が少し異なる。

```
switch(式)  {
  case  定数式1:
          文1
  case  定数式2:
          文2
  default:
          文3
}
```

```
EVALUATE  式
   WHEN    定数1
           無条件文1
   WHEN    定数2
           無条件文2
   WHEN  OTHER
           無条件文3
END-EVALUATE
```

ﾃﾞｰﾀ名（変数）が使える　→
ﾃﾞｰﾀ名を使用した式が使える →

```
EVALUATE  式
   WHEN    一意名
           無条件文
   WHEN    算術式
           無条件文
   WHEN  OTHER
           無条件文
END-EVALUATE
```

```
switch(式)  {
  case  定数式:
          文1
          break;
  case  定数式:
          文2
          break;
}
```

←breakがないと文2が実行される。
EVALUATE文はbreakの機能が組み込まれている

繰り返しは、Ｃ言語の「式」とCOBOLの「条件」では真偽が逆になることに注意する。

```
while(式)  {
      文
}
```

```
PERFORM  TEST  BEFORE  UNTIL  条件
   文
END-PERFORM
```

```
do  {
      文
}  while  (式)
```

```
PERFORM  TEST  AFTER   UNTIL  条件
   文
END-PERFORM
```

ＣＯＢＯＬのGOTO DEPENDING指定はＣ言語のswitch文に似ている。

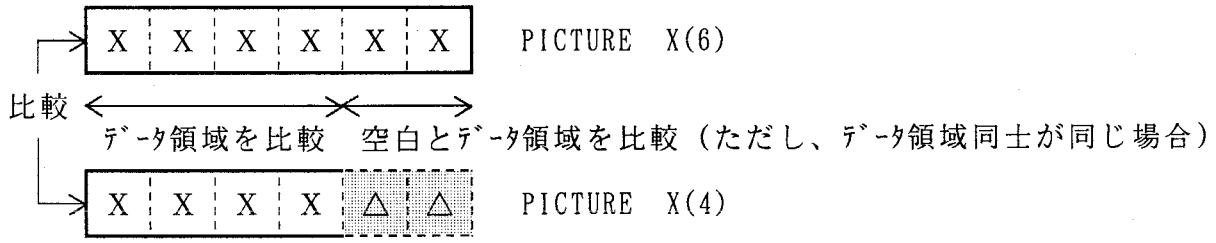
```
switch(変数)  {
  case  1:
          文1
  case  2:
          文2
}
```

```
GO TO 手続き1 手続き2 DEPENDINGﾃﾞｰﾀ名
手続き1.
        文1.
手続き2.
        文2.
範囲外の値の場合はGOTOの次の文が実行さ
れてしまうがswitchにはdefault指定がある
```

◎Ｃ言語を元にＣＯＢＯＬの命令文と比較すると、次のとおり。

命令文	比較説明
break	Ｃではdo while、for、switchから抜けでることができる COBOLにbreakはないが、EVALUATEにはbreakの機能が含まれている。
continue	Ｃでは、ループの繰り返しを止め、ループの条件判定を行 COBOLのCONTINUEは無操作文である。
do ～ while(式)	COBOLのPERFORM TEST AFTERと同じ
for(式1;式2;式3)	COBOLのPERFORM VARYINGで同じような記述はできるが、 Ｃ言語の式1と式2と式3には「,」を使って複雑な記述ができる。 PERFORMのAFTERをfor文で行うにはforを入れ子にする。
goto ﾗﾍﾞﾙ	COBOLのGO TO手続き名と同じ。 GO TOのDEPENDINGはＣ言語にはない。
return 式	COBOLのEXIT PROGRAMには式がない。
switch(式)	COBOLのEVALUATEでは複雑な条件式が使える。
while(式)	COBOLのPERFORM TEST BEFOREと同じ
代入（割当）	MOVE 0 TO A B C D ←COBOL D＝C＝B＝A＝0； ←C

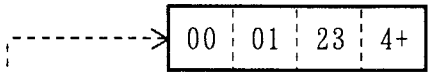
◎比較対象について － 英数字項目
英字、英数字項目の比較はできるだけ同一サイズで行うようにするとよい。
サイズが異なると不足分は空白（SPACE）を比較する機械命令が生成される。



次のように、空白と比較する場合には、表意定数を使用する方が効率がよい。

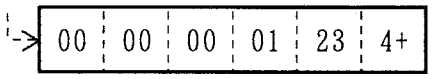
```
IF ﾃﾞｰﾀ名 = "△" ～
           ↓
IF ﾃﾞｰﾀ名 = SPACE ～
```

◎比較対象について　－　数字項目
数字項目の比較はできるだけ用途（USAGE）が同じもの同士を比較する。
用途が異なると、どちらかのタイプに変換してから比較することになる。



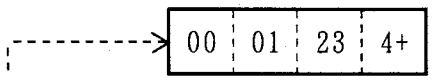
PICTURE　9(7)　PAKED-DECIMAL

比較←→ 桁数が異なっても一回で比較できる。（汎用ｺﾝﾋﾟｭｰﾀ）



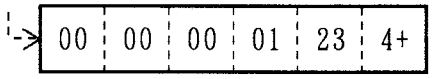
PICTURE　9(11)　PACKED-DECIMAL

・小数点の位置が異なると、小数点の桁合わせが行われた後で比較される。



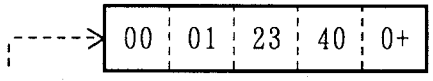
PICTURE　9(6)V9(1)　PAKED-DECIMAL

比較できない



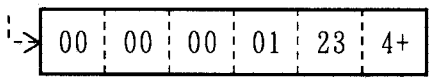
PICTURE　9(8)V9(3)　PACKED-DECIMAL

↓　小数点以下の桁数の多い方に合わせてから比較する。



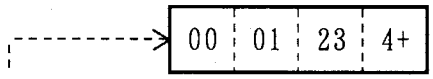
PICTURE　9(6)V9(3)　PAKED-DECIMAL

比較できる



PICTURE　9(8)V9(3)　PACKED-DECIMAL

・汎用ｺﾝﾋﾟｭｰﾀでは、用途（USAGE）にBINARYやPACKED-DECIMALを使うと効率がよい。
USAGE　DISPLAYの数字項目は変換しないと比較できない。

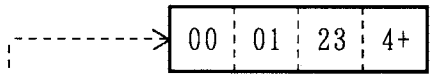


PICTURE　9(6)V9(1)　PACKED-DECIMAL

比較できない

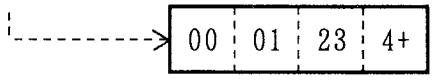
PICTURE　9(5)V9(1)　DISPLAY

↓　DISPLAY形式をPAKED-DECIMALに変換して比較する。



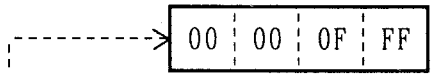
PICTURE　9(6)V9(1)　PACKED-DECIMAL

比較できる



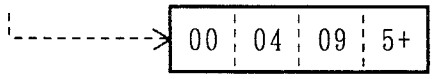
PICTURE　9(6)V9(1)　PACKED-DECIMAL

・用途がBINARYとPACKED-DECIMALも一旦どちらかに変換してから比較する。



PICTURE　9(7)　BINARY

比較できない



PICTURE　9(7)　PACKED-DECIMAL

◎PERFORMの指定
PERFORMで実行先を指定するが、できるだけ手続き名は段落名ではなくセクション名を使用するとプログラムの修正が容易になる。

```
    PERFORM   段落名1
     :
段落名1.          ┐
    文            │
    文            ├ 段落名が追加されるような命令文を書いた場合には、PERFORM文も
     :            │ 修正する必要がある。
段落名2.          ┘
```

```
    PERFORM   節名1.
     :
節名1  SECTION.
段落A.            ┐
    文            │
     :            │
    文            ├ 段落名が追加されるような命令文を書いた場合でも、PERFORM文の
     :            │ 修正は必要ない。
段落B.            │
    文            ┘
```