



Title	FORTTRANとの比較によるPL/I 入門 (4)
Author(s)	塩野, 充
Citation	大阪大学大型計算機センターニュース. 1980, 39, p. 41-63
Version Type	VoR
URL	https://hdl.handle.net/11094/65464
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

FORTRANとの比較によるPL/I入門(4)

大阪大学工学部 塩野 充

9月初めの新聞の経済欄に、日本電気がACOSシステム1000を完成したという記事が載っていた。阪大センターのACOSシステム900はもはや上から2番目の機種になり下がってしまったのである。記事によると、ACOS1000は、ACOS900の4倍ないし5倍の能力を持っているそうである。どういう面での比較かは分からないが、単純に考えても現在の阪大センターのACOS900を2台並べたシステムの2倍以上の能力があるということであろう。出荷は来年(昭和56年)の10月ということだから、遅かれ早かれいずれ阪大センターにも出現するであろう。しかし、考えてみるとACOSシリーズは、システム1000で予定終了打ち止めという訳ではないだろうから、システム1100, 1200, ……と続々と、より大きいシステムが日本電気がツブれない限り、開発されてゆくであろう。ACOS2000やACOS3000などというのはどんなコンピュータであろうか。その頃にはFORTRANとかPL/Iとかいうプログラム言語はまだ存在しているだろうか。人間に話しかけるように音声で入出力できたり、知能を持って人間と対抗したり、などとそこまで考えるといささかSF映画の見過ぎという感じもするが、現実的に考えると、少なくとも紙カードや紙テープといった原始的なものは姿を消して、博物館へ行かないと見られなくなっているだろう。磁気テープも今のオープンリールタイプのようなバカでかい重いものではなく、超高密度なマイクロカセット的なものになるか、更に進歩して、テープやディスクといった機械的な回転メカニズムを全く必要とせず、電子的ないし光学的な走査のみによって情報の読み書きができる、いわば半導体カード(?)やホログラムカード(?)のようなものが、今のIDカードのような大きさで、できるのではないかと思う。あと20年後の21世紀まではかからず、20世紀中にできてしまうのではないかと思う。

こんなことを考えていると一体何の原稿を書いているのか忘れてしまいそうなので、頭を現実に戻して、PL/Iの話に入ろう。今回は、GO TO文, IF文, DO文, ファイル入出力文について解説しよう。

第12章 GO TO文

FORTRANのGO TO文には次の3種類がある。

- (i) 無条件GO TO文(単純GO TO文)
- (ii) 計算形GO TO文
- (iii) 割当て形GO TO文

このうち、(iii)の割当て形GO TO文は(ii)の計算形GO TO文で十分代用できるものであり、さして存在価値のあるものとは思われない。多分岐の場合、計算形GO TO文ではどうしても具合が悪く、割当て形GO TO文でなければ使えない、という状況の例は筆者には思い浮かばないのである。ところで、割当て形GO TO文というのをよく知らない人のために少し説明を加えておこう。計算形GO TO文はよく御存知のように、例えば、

```
GO TO (10, 15, 28, 35), K
```

という形で、K=1なら文番号10へ、K=2なら文番号15へ、K=3なら文番号28へ、K=4なら文番号35へ飛べという文である。K=1, 2, 3, 4以外のときの処置はコンパイラによって異なり、ACOS-6ではエラーとなるが、OS IV/F4では、このGO TO文の次の文へ移るようになっている。ここでKは普通の整数型であるが、そんなこと言わなくても当然じゃないかと思われるかも知れないが、実はここが大事なポイントなのである。上例と同じ文を、割当て形GO TO文で書くと、

```
GO TO K, (10, 15, 28, 35)
```

となり、計算形GO TO文のときのKとカッコの順序が逆になる。この文の意味は先程と少し異なり、Kが10なら文番号10へ、Kが15なら文番号15へ、Kが28なら文番号28へ、Kが35なら文番号35へ飛べという命令である。ここで、K=10とか、K=15という書き方をせず、Kが10、Kが15、という書き方をしたのはそれなりの理由がある。ここでのKは見かけは整数型であり、データの型宣言としてはINTEGERなのであるが、実は整数型ではなく、算術演算にも使うことはできないし、代入文で10や15という値を入れることもできない。強いて言うならば"文番号型"なのである。整数型のように整数型でない、文字型のように文字型でない、それが"文番号型"である。ACOS-6のマニュアルでは、これを「スイッチ変数」と呼んでいる。ところで、代入文の使えない"文番号型"に値を代入するにはどうすればよいか。例えば、Kに15を入れるには次のように書く。

```
ASSIGN 15 TO K
```

これは、K=15という代入文とは全く異なる。K=15としたのではKは整数型の15となる。上記のASSIGN文では、Kは"文番号型"の15となる。見かけは同じ15でも整数型の15と、"文番号型"の15とでは計算機の内部表現が全く異なる。文字型でもないから、

```
DATA L/2H15/
```

```
K=L
```

という入れ方でもダメである。従って、割当て形GO TO文には必ずASSIGN文が付きものである。割当て形GO TO文があるのにASSIGN文のないプログラムはありえない。この点からも、計算形GO TO文よりも面倒なのである。従って、ここでは(i)と(ii)につ

いて、PL/Iに置き換えて説明しよう。ところでFORTRANでもPL/Iでも、GO TOを、間に空白を入れずに、GOTOと書いてもよいのだが、ここではやはり英語に忠実にするため空白を入れて書くことにする。

<a>無条件GO TO文（単純GO TO文）

FORTRANでは文を識別するために、文番号というのがあり、カードの第2～6カラムに打った。PL/Iでは文番号というの無く、代わりにラベル（名札）というものを使う。簡単に言えば、文番号は数字であるのに対し、ラベルは英字で始まる英数字を用いているだけのことであり、本質的には変りはない。

FORTRANで例えば、

```
GO TO 10  
.....
```

```
10 A=B
```

をPL/Iで書くと

```
GO TO L10;  
.....
```

```
L10: A=B;
```

となる。ここでL10というのはラベルである。ラベルは変数名と同じく、英字で始まる英数字の列である。その長さはTYPE(A), (AB)では6文字以内、OSⅣ/F4では7文字以内である。しかし、この長さを超えてもエラーにはならず、システム固有の変換規則に従って短縮化が行われ、WARNINGメッセージが出されるだけであるが、そんな長いラベルをつける必要はなく、ラベルは6文字以内と覚えておけばよいだろう。ラベルの付け方は上例のように、付けたい文の前に：（コロ）を介して書けばよい。これは、第10章で述べたPROC文に付けた手続き名と同じ付け方である。ラベルと手続き名をひっくるめて、ラベル型（ラベル定数、ラベル変数、ラベル配列）という。ラベル型は強いて言えば前述したFORTRANの“文番号型”に相当するもので、やはり算術演算には使えない。しかし、“文番号型”とは違って、代入文には使える。これについては次ので述べる。又、PL/Iでは1つの文に2つ以上のラベルを付けることもできる。例えば、

```
TARO: JIRO: A=B+C;
```

などと書いてもよい。しかし、1つの文に2つ以上のラベルを付けなければならない場合というのはあまり考えられない。

計算形GO TO文

FORTRANの計算形GO TO文は前述したように、例えば、

```
GO TO (5, 6, 7, 8, 9), K
.....
5  A=B+C
.....
6  A=D+E
.....
7  A=F+G
.....
8  A=H+P
.....
9  A=Q+R
.....
```

というような形式の多分岐のGO TO文である。これに直接対応するGO TO文は、PL/Iには存在しない。しかし、同じ働きをする文を書くことは可能である。次のように書けばよい。

```
DCL X(5) LABEL INIT(L5, L6, L7, L8, L9);
.....
GO TO X(K);
.....
L5 : A=B+C;
.....
L6 : A=D+E;
.....
L7 : A=F+G;
.....
L8 : A=H+P;
.....
L9 : A=Q+R;
.....
```

このDCL文では、LABEL属性により配列X(5)がラベル型であること(すなわちラベル配列であること)を宣言し、その初期値をL5, L6, L7, L8, L9としている。従って、K

=1ならL5へ、K=2ならL6へ、K=3ならL7へ、K=4ならL8へ、K=5ならL9へ飛ぶ。ここで、ついでながらラベル型についてもう少し補足説明を加えておこう。FORTRANの文番号のうち、FORMAT文の文番号はGO TO文の行先になってはいけない。例えば、

```
GO TO 100
.....
```

```
100 FORMAT (.....)
```

などとするとエラーになる。この意味で、FORMAT文の文番号は普通の実行文などの文番号とは性質が異なるので、区別して"FORMAT文番号"などと書いてある本やマニュアル類もある。このことはPL/Iでも全く同様で、第8章で述べたR書式(リモートフォーマット)のFORMAT文のラベルについて同じことが言える。例えば、

```
GO TO LB;
.....
```

```
LB:FORMAT (.....);
```

などとするとエラーになる。従って、実行文のラベルと、FORMAT文のラベルは明確に区別しなければならない。ACOS-6のTYPE(A),(AB)では、これを区別するため、FORMAT文のラベルを特に"書式データ"と呼んでいる。又、FORMAT文のラベルであることを宣言するためのFORMAT属性というのを設けており、例えば、

```
DCL LB FORMAT;
```

などと書く。OS/IV PL/Iでは特に区別はしておらず、FORMAT属性も設けていない。しかし、LBがGO TO文の行先になればエラーになることに変わりはない。

第13章 IF文

FORTRANのIF文には、

(i) 論理IF文

(ii) 算術IF文

の2つの種類があった。PL/Iにはこのような区別はなく、IF文は(i)の論理IF文に対応するものだけである。しかし、FORTRANでも(ii)の算術IF文は(i)の論理IF文でも代用できるように、PL/IのIF文は両者を十分カバーしているし、それどころか、もっと大きな機能を持っている。以下、様々なタイプのIF文を説明しよう。なお、以下に記した種々の型名は筆者が名付けたものである。

<a>片側単一文実行型

図13-1に示すタイプで、FORTRANで例えば、

```
IF (K. GT. M) A=B+C
```

というような文である。図では、判断を表わすひし形の分岐出口に通常、右にYES、下にNOを書くことが多いが、PL/IではどちらをYESにすることも可能なので、ここでは省略してある。

この文をPL/Iで書くと、

```
IF K>M THEN A=B+C;
```

となる。FORTRANで、.EQ.と書いた

所が=となり、IFの後の条件節を包むカッ

コがなく、条件節と実行文の間に、THENという言葉が入る点が異なっている。ここで、FORTRANの .GT. が>となるように、PL/Iではより通常の数学表現に近くなっている。この .GT. のようなものをFORTRANでは関係演算子 (relational operator) と呼んでいるが、PL/Iでは>のようなものを比較演算子 (comparison operator) と呼んでいる。このように用語は異なるが、その意味は全く同じである。又、FORTRANのIF文で用いる、.AND. (論理積)、.OR. (論理和)、.NOT. (否定) なる3つの演算子は論理演算子 (logical operator) と呼ばれているが、PL/Iでこれに対応する演算子&, |, ~はビット列演算子 (bit stripg operator) と呼ばれ、やはり全く同じ意味である。表13-1にこれらの対応を示す。ここで注意すべきは、<=や、>=を、=<や、=>と順序を間違えて書いてはいけないことである。これはFORTRANでも、.LE.や.GE.を、.EL.や.EG.と書いてはいけないことを想起すれば容易に覚えることができる。同様に、~<=を=~>と書いてはいけないことは、.NE.を.EN.と書いてはいけないことと同じである。~<や、~>はFORTRANでは1つの演算子で直接対応するものはないが、.NOT.を使って書くことができる。

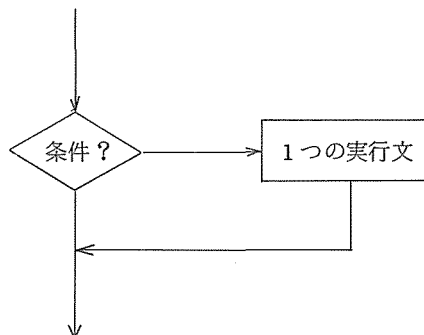


図13-1

両側単一文実行型

図13-2に示すタイプで、FORTRANで例えば、

```
IF (K. GT. M) GO TO 1
```

```
D=E+F
```

```
GO TO 2
```

```
1 A=B+C
```

```
2 .....
```

表 13.1 IF文における演算子

数学的表現	FORTRAN	PL/I
$A=B$	A. EQ. B	A=B
$A \neq B$	A. NE. B	A \neq B
$A < B$	A. LT. B	A<B
$A \leq B$	A. LE. B	A<=B
$A > B$	A. GT. B	A>B
$A \geq B$	A. GE. B	A>=B
$A \nless B$.NOT. (A. LT. B)	A \nless B
$A \ngtr B$.NOT. (A. GT. B)	A \ngtr B
	(A, Bは算術型)	(A, Bは算術型)
$P \wedge Q$	P. AND. Q	P&Q
$P \vee Q$	P. OR. Q	P Q
\overline{P}	.NOT. P	$\neg P$
	(P, Qは論理型)	(P, Qはビット型)

あるいは2つのIF文を使って(この場合、実行速度は遅くなるが)、

IF (K. GT. M) A=B+C

IF (K. LE. M) D=E+F

と書けるようなタイプである。

これをPL/Iで書くと、

IF K>M THEN A=B+C;

ELSE D=E+F;

となる。ここで出てきたELSEというのがFORTRANにはない強みで、FORTRANという言語の最大の欠点の1つは、このELSEがないことである

と言っても過言ではないと思う。ELSEが使えない

ことと、後で述べるようにIFの条件節の後に単一の実行文しか書けないことが、やたらGO TO文を多く発生させ、最近よく言われる構造化プログラム(structured program: 簡単に言えばGO TO文のないプログラム)とは程遠いものになってしまう原因となっている。この

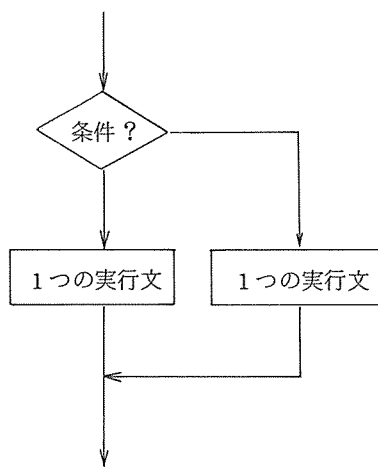


図 13-2

ために、FORTRANの最新改良版とも言えるFORTRAN 77という新しい言語ではELSE Eが使えるようになっている。言うまでもないが、THENは「それなら」、ELSEは「そうでなければ」という英語である。THENとその後の実行文を合せて、THEN節（又はTHEN単位）、ELSEとその後の実行文を合せて、ELSE節（又はELSE単位）と呼ぶ。上例を、フローのひし形の分岐出口のYESとNOを逆にして書くと、

```
IF K > M THEN D = E + F ;
      ELSE A = B + C ;
```

となる。

<c>片側複数文実行型

図13-3に示すタイプで、FORTRANで例えば、

```
IF (K .LE. M) GO TO 1
A = B + C
D = E + F
1 .....
```

というような場合である。これをPL/Iで書くと、

```
IF K > M THEN DO ;
      A = B + C ;
      D = E + F ;
      END ;
      ELSE ;
```

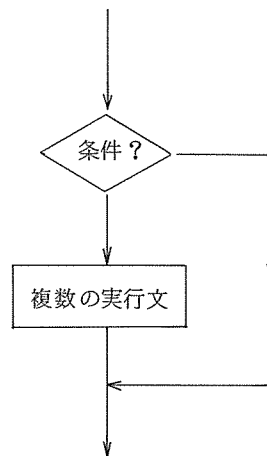


図13-3

と書ける。ELSE ; はELSE節の実行文が空文（すなわち；のみ）であることを示している。すなわちELSE節では何もしないことを意味しており、省略してもかまわない。THEN節（ELSE節でもよい）に複数の実行文を置きたいときはこのように制御変数もなにもないDO文（繰返しのないDO文、又は単純DO文と呼ぶ）を用いてまとめると、単一の実行文として扱われる。勿論、通常の繰返しのあるDO文を書いてもかまわない。FORTRANではこのようなことは許されない。

<d>両側複数文実行型

図13-4に示すタイプで、FORTRANで例えば、

```
IF (K .LE. M) GO TO 1
A = B + C
```

```

D=E+F
GO TO 2
1 U=V+W
X=Y+Z
2 .....

```

というような場合である。これをPL/Iで書くと、

```

IF K>M THEN DO;
    A=B+C;
    D=E+F;
END;
ELSE DO;
    U=V+W;
    X=Y+Z;
END;

```

となる。

<e>ひし形横並び型（IF文の入れ子）

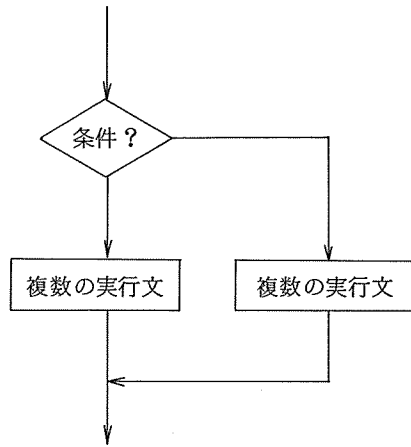


図 1 3-4

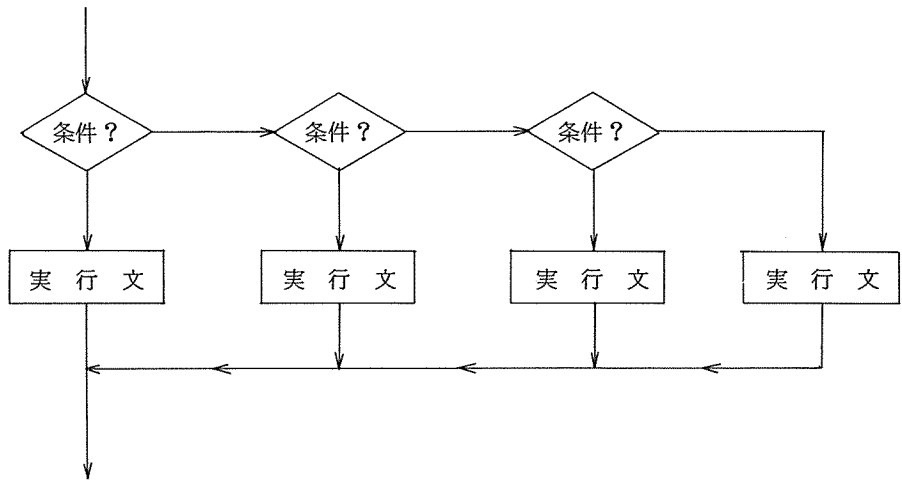


図 1 3-5

図 1 3-5 に示すように、判断のひし形が 2 個以上横に並んでいるタイプである。各々の実行文は単一でも複数でもかまわないが、以下の例では簡単化のため単一にしてある。（複数のときは前

述のDO文を使えばよい)。この一例をFORTRANで書くと、

```
      IF (I. GT. J)  GO TO 1
      A=B+C
      GO TO 4
1     IF (K. GT. L)  GO TO 2
      D=E+F
      GO TO 4
2     IF (M. GT. N)  GO TO 3
      U=V+W
      GO TO 4
3     X=Y+Z
4     .....
```

となる。これが、更にひし形が増えたりすると、まさにGO TOだらけになってしまう。これをPL/Iで書けば、

```
      IF I>J
          THEN IF K>L
                  THEN IF M>N
                          THEN X=Y+Z ;
                          ELSE U=V+W ;
                  ELSE D=E+F ;
          ELSE A=B+C ;
```

となる。このようなものをIF文の入れ子 (nesting) という。FORTRANではDO文の入れ子はあったが、IF文の入れ子はない。入れ子のIF文で注意すべきことは、入れ子になっていない普通のIF文では、前述したようにELSE節が空文のとき (すなわち、ELSE節で何もしないときであり、ELSE ; と書く)、ELSE節そのものを省略してもよいが、入れ子になっているときは、たとえ空文でも省略してはいけないということである。なぜならば、例えば、

```
      IF X>1.5
          THEN IF Y<3.0
                  THEN K=K+1 ;
                  ELSE ;
          ELSE Z=0.0 ;
```

という文で、4行目のELSE節が空文なので省略してしまうと、

```

IF X>1.5
  THEN IF Y<3.0
        THEN K=K+1;
        ELSE Z=0.0;

```

となる。これをよく見ると最後のELSE節は2行目のTHEN節に対応しているとも考えられるし、3行目のTHEN節に対応しているとも考えられ、どちらかはっきりしない。どちらに解釈するかでプログラムは全然異なったものになる。(目で見ると限りは段下げ(カラムの対応)で、2行目のTHEN節に対応しているらしいことは分かるが、PL/Iプログラムは自由形式でカラムの位置は関係ないので、コンパイラはそこまで気を利かせることはしない)。

PL/IのIF文は、ELSE節があることと、THEN節やELSE節の実行文にDO文が使えることの2つの理由により、GO TO文を殆んど必要としないスッキリとした形に書くことができる。

第14章 DO文

DO文については第3章でも触れたが、FORTRANに比べて相当豊富な機能がある。以下、各種のDO文を順に説明していこう。なお、各々の分類名は筆者が名付けた。

<a>繰返しなしのDO文(単純DO文)

これは前章のIF文のTHEN節やELSE節における複数の実行文をまとめて単一の実行文として取り扱うために用いられるもので、繰返しを行わない1回限りのDO文である。例えば、

```

DO;
  A=B*C-D;
  PUT LIST(A);
END;

```

などと書く。このようにDO文の最後(端末文)は、END文となる。FORTRANの場合は、CONTINUE文であった。PL/IでFORTRANのCONTINUE文に相当するのは、;(セミコロン)だけからなる空文であり、何もしないという意味では同じであるが、空文をDO文の端末文に使うことはできず、必ずEND文でなければならない。

上例をもし強いてFORTRANで書こうとすれば、

```

DO 1 I=1, 1
  A=B*C-D
  WRITE(6,*)A

```

1 CONTINUE

となるが、これは全く意味がない。総返しなしのDO文は、THEN節やELSE節の実行文として使えるからこそ意味があるのであって、FORTRANではそれができないのだから、このようなDO文は全く意味がない。

FORTRAN型DO文

FORTRANで例えば、

```
DO 5 K=1, 10, 2
```

```
.....
```

5 CONTINUE

をPL/Iで書くと、

```
DO K=1 TO 10 BY 2 ;
```

```
.....
```

```
END ;
```

となる。すなわち、FORTRANのようにDOの後に文末の文番号は書かず、いきなり制御変数を書く。又、開始値（初期値、出発値）1と終了値（終値）10の間のコンマがTOに、終了値10と増分2の間のコンマがBYに変わっている。増分を省略した場合、すなわち、

```
DO K=1 TO 10 ;
```

```
.....
```

```
END ;
```

とすると、増分は1と見なされることはFORTRANと同じである。FORTRANにはない機能として次の2つがある。

(i) WHILE節による途中終了が指定できる。FORTRANで例えば、

```
DO 1 K=1, 13, 3
```

```
.....
```

```
IF (A. GT. B) GO TO 2
```

```
.....
```

1 CONTINUE

2

というように、DOグループ（PL/IではDO文から文末のEND文までをまとめて「DOグループ」と呼び、FORTRANでいう「DOの範囲」に相当し、通常よく使われる「DOグループ」と同じような意味をもつ）の繰返しの途中で、 $A > B$ となればDOグループを終了して次の

文へ飛びたいとき、PL/Iでは、

```
DO K=1 TO 13 BY 3 WHILE (A<=B) ;  
.....  
END ;
```

というふうに、後にWHILE節を付加しておくだけでよい。WHILE節というのは、WHILE (条件式) という形式であり、条件式が真のあいだだけDOグループを実行し、条件式が偽となればたとえ制御変数が終了値まで来ていなくてもDOグループを終了する。

(ii) 制御変数、開始値、終了値、増分の各々のデータ型は整数でなくてもよいし、整数と実数が混在してもよい。又、それらはマイナスでもよい。従って、開始値は終了値より大きくてもよい。例えば、

```
DO A=-3.5 TO 2 BY 0.5 ;
```

や、

```
DO B=7.1 TO -1.1 BY -1 ;
```

という書き方もできる。

(iii) 開始値や終了値に配列要素名を書いてもよい。例えば配列M(2)があるとき、

```
DO K=M(1) TO M(2) ;
```

のような書き方もできる。FORTRANでこれをするエラーとなるので、

```
M1=M(1)
```

```
M2=M(2)
```

あるいは

```
EQUIVALENCE (M1, M(1)), (M2, M(2))
```

としておいてから、

```
DO 1 K=M1, M2
```

としなければならない。

PL/Iの、FORTRAN型DO文で、FORTRANのDO文と比べて注意すべき点を次に3つ並べる。

(i) GO TO文によってDOグループから外に飛び出すことはかまわないが、外からDOグループの中へ入ってくることは絶対に許されない。FORTRANではDOの拡張範囲といって、DOグループから一旦外へ飛び出して、再び同じDOグループへ戻ってくる場合のみ、これを許していたが、PL/IではこのようなDOの拡張範囲というものは認めていない。但し、PL/Iでも繰返しなしのDO文に関しては外から入ってくることを許している。

(ii) JISを越えたFORTRANでは、開始値が終了値より大きい場合もエラーにはせず、1

回だけDOグループを実行する。例えば、

```
DO 3 I=5, 1, 2
```

という形では、まず制御変数Iが開始値5で、とりあえず1回DOグループを実行し、次に開始値5に増分2を加えて7となった制御変数を終了値1と比較すると、既に終了値を超えているので、DOグループの実行を止める。これに対してPL/Iではこのような場合、1回もDOグループを実行しない。すなわち、

```
DO I=5 TO 1 BY 2;
```

では、まず制御変数Iが開始値5となり、すぐ終了値と比較し、増分がプラスのときはIが終了値を上回っていればDOグループの実行を止める。もし増分がマイナスのときはIが終了値を下回っていればDOグループの実行を止める。従って上例ではDOグループの実行は0回となる。この点はFORTRANとPL/IのDO文の本質的な違いであろう。

(Ⅲ) FORTRANではDO文が入れ子になっているとき、端末文であるCONTINUE文を1つにまとめることができた。例えば、

```
DO 1 I=1, 10, 2
DO 2 J=1, 35, 3
DO 3 K=5, 10
.....
3 CONTINUE
2 CONTINUE
1 CONTINUE
```

のような場合は、

```
DO 1 I=1, 10, 2
DO 1 J=1, 35, 3
DO 1 K=5, 10
.....
1 CONTINUE
```

とまとめることができる。PL/Iの場合は、

```
DO I=1 TO 10 BY 2;
DO J=1 TO 35 BY 3;
DO K=5 TO 10;
.....
END;
```

```
END ;
```

```
END ;
```

における3つのEND文をやはり1つにまとめることができる。そのときは、最も外側のDO文にラベルを付けて、END文にもそのラベルを付ければよい。すなわち、

```
L1 : DO I=1 TO 10 BY 2 ;
```

```
DO J=1 TO 35 BY 3 ;
```

```
DO K=5 TO 10 ;
```

```
.....
```

```
END L1 ;
```

とすればよい。この方法をEND文の重ね閉じ（多重閉鎖）という。

(iv) 終了値のないFORTRAN型DO文は、WHILE節を伴うか、又はIF文によるDOグループの外への飛び出しを含んでいるかのいずれかである。WHILE節を伴う場合は、例えば、

```
DO K=0 BY 3 WHILE (K<L) ;
```

という文では、 $K < L$ が成り立つ限り、 $K = 0, 3, 6, 9, 12, \dots$ といくらでも続く。これは後述する<d>のWHILE型DO文を用いて、

```
K=0 ;
```

```
DO WHILE (K<L) ;
```

```
.....
```

```
K=K+3 ;
```

```
END ;
```

と書いたのと同じである。

IF文による飛び出しを含むのは例えば、

```
DO K=0 BY 3 ;
```

```
IF K>L THEN GO TO L1 ;
```

```
.....
```

```
END ;
```

```
.....
```

```
L1 : .....
```

という形である。終了値のない場合で、WHILE節も伴わず、IF文による飛び出しも含まないときは、無限ループになってしまう。

<c>制御変数値列記型DO文

これは名前の通り、制御変数のとる値を順に並べて書いたDO文である。例えば、

```
DO A=0.5, 1.1, 2.8, 4.5, 7.3, 10.1;
```

という文では、 $A=0.5$ 、 $A=1.1$ 、 $A=2.8$ 、 $A=4.5$ 、 $A=7.3$ 、 $A=10.1$ の6回についてDOグループを順次実行する。並べる制御変数値は大きい順や小さい順になっていなくてもよく、バラバラでよい。このDO文が出たついでに、前述のFORTRAN型DO文に関する注意を補足しておこう。例えばFORTRAN型DO文、

```
DO K=1 TO 6 BY 2;
```

を、FORTRANのときの馴れから、TOとBYをうっかりコンマにして、

```
DO K=1, 6, 2;
```

と書いてしまうと、エラーにはならず、今説明した制御変数値列記型DO文に解釈され、 $K=1$ 、 $K=6$ 、 $K=2$ の3回についてDOグループが実行される。この点、注意が必要である。

なお、厳密にはこの制御変数値列記型DO文というのは、のFORTRAN型DO文で、開始値だけ書いて終了値と増分を省略した型式を多数つないだ形になるのであるが、ここではFORTRANユーザに理解しやすいように別の型として分類した。

<d>WHILE型DO文

このDO文は、のFORTRAN型DO文のWHILE節だけを取り出したようなDO文で、制御変数は存在しない。一般的に、

```
DO WHILE (条件式);
```

の形式である。この文では条件式が真である限りDOグループを繰返し実行し、偽になったら実行を止める。例えば、

```
DO WHILE (A<5.5);
```

のような場合は、 A が5.5未満である限りDOグループの実行を続ける。従ってこの場合、 A は最初5.5より小さく、DOグループの中で増加してゆくことが前提となる。もし、 A が最初から5.5以上の値であればこのDOグループは前述したように1回も実行されない。又、もし A が最初5.5より小さく、かつDOグループの中で減少してゆくのであれば、条件式は永久に真のままとなるので、DOグループは無限ループに陥ってしまう。ところで、次のような書き方は、つい行いたくなりがちであるが誤りである。

```
DO WHILE (0.0<X<1.0);
```

このような場合は、

```
DO WHILE (0.0<X & X<1.0);
```

とすればよい。

<e>やや複雑なDO文

以上述べたと<c>のDO文を組み合わせた形式も許される。例えば

```
DO A=-2.5 TO 3.5 BY 0.5 WHILE (B<12.5), 3.8,  
4.3, 5.9;
```

という文では、 $B < 12.5$ という条件下で、 $A = -2.5$ から $A = 3.5$ まで 0.5 きざみで繰り返し実行し、それが済んだら、 $A = 3.8$ 、 $A = 4.3$ 、 $A = 5.9$ で実行する。 $A = -2.5$ から $A = 3.5$ までの間で、 $B < 12.5$ という条件が成り立たなくなったら、終了値の $A = 3.5$ まで行っていないくても、 $A = 3.8$ へ飛ぶ。従って、最初から $B < 12.5$ という条件が成り立っていないければこのDOグループは $A = 3.8$ から始まる。

```
DO N=1 TO 10 BY 2, 15, 16, 17 TO 30 BY 3;
```

という例では、 $N = 1$ から $N = 10$ まで 2 きざみで繰り返し実行し、次に $N = 15$ 、 $N = 16$ で実行し、その後 $N = 17$ から $N = 30$ まで 3 きざみで実行する。

```
DO M=5 TO 25 BY 5, 50 TO 200 BY 10 WHILE (A  
>B);
```

という例では、 $M = 5$ から $M = 25$ まで 5 きざみで繰り返し実行し、次に $M = 50$ から $M = 200$ まで 10 きざみで繰り返し実行する。但し、 $M = 50$ から $M = 200$ の途中ででも、 $A > B$ という条件が成り立たなくなればDOグループの実行を終了する。

```
DO A=-21.3 TO 12.5 BY 0.7 WHILE (X>Y),  
125.5 TO -58.3 BY -9.5 WHILE (U<V);
```

という例では、 $X > Y$ という条件が成り立つとき、 $A = -21.3$ から $A = 12.5$ まで増分 0.7 で繰り返し実行する。 $A = 12.5$ の実行を終えるか、又は $A = 12.5$ まで行っていないくても条件 $X > Y$ が成り立たなくなるか、のいずれかになると、次は $U < V$ という条件が成り立つとき $A = 125.5$ から $A = -58.3$ まで増分 -9.5 （言い換えれば“減分”（？） 9.5 となる）で繰り返し実行する。 $A = -58.3$ まで行かなくても条件 $U < V$ が成り立たなくなればDOグループの実行を終了する。

第15章 ファイル入出力文

第8章では標準入出力装置 すなわちカードリーダーとラインプリンタを用いた場合の入出力文について解説した。しかし、入出力装置はカードリーダーとラインプリンタと限ったものではなく、磁気テープ（MT）や、カード出力（カードパンチ）、パーマメントファイル、テンポラリファイル等

々がある。本章ではそのような入出力装置（広い意味でのファイルである）のための入出力文について解説しよう。

(15-1) 厳密な入出力文

FORTRANではカードリーダーとラインプリンタの2つの入出力装置は標準入出力装置として設定されており、これらを使う場合は、JCL（ジョブコントロール言語）のファイル定義文が不要である。すなわち、カードリーダーはファイルコード5（ACOS-6 FORTRANでは41でもよい）、ラインプリンタはファイルコード6（同じく42でもよい）を用いる限りJCLは不要である。例えばカードリーダーからの読み込みは、

```
READ (5, 100) A
100 FORMAT (.....)
```

というように、ファイルコード5を用いる。ラインプリンタで印刷するときは、

```
WRITE (6, 200) A
200 FORMAT (.....)
```

というように、ファイルコード6を用いる。ACOS-6 FORTRANではこの他にカード出力も標準出力装置になっており、ファイルコード43で使用できる。例えば、

```
WRITE (43, 300) A
300 FORMAT (.....)
```

などと書く。FORTRANでの、このファイルコード5や6に相当するPL/Iでのファイル名は、SYSINおよびSYSPRINTである。すなわち、数字ではなく英字名となる。SYSINがカードリーダー、SYSPRINTがラインプリンタである。この両者を、標準入出力ファイルという。カード出力は標準入出力ファイルには設定されていない。

ところで、第8章(8-1)で述べたEDIT型出力文の例

```
PUT EDIT (A) (F (7, 3)) ;
```

を、厳密な本来の形式で書けば、

```
PUT FILE (SYSPRINT) EDIT (A) (F (7, 3)) ;
```

となる。又、同様に(8-2)のEDIT型入力文の例、

```
GET EDIT (A) (F (7, 3)) ;
```

を厳密な本来の形式で書けば、

```
GET FILE (SYSIN) EDIT (A) (F (7, 3)) ;
```

となる。すなわち、一般的には、

```
PUT FILE (ファイル名) EDIT (.....) (.....) ;
```

GET FILE (ファイル名) EDIT (……………) (……………) ;
という形式になる。この形式で、ファイル名が標準入出力ファイルSYSIN, SYSPRINT
の2つの場合に限って、

FILE (ファイル名)
という箇所を省略してもよく、そのときの形式が、第8章に述べた入出力文なのである。以上では、
EDIT型入出力文を例にとって述べたが、LIST型やDATA型入出力文についても全く同様
である。

(15-2) ファイル宣言文

FORTRANではファイルコード5, 6等の標準入出力ファイル以外の入出力ファイルを用い
るときは、JCLでファイルコードを定義しておくだけでよく、FORTRANプログラムの中で
は特に宣言文などは必要とせず、例えば、

```
READ (15, 100) X  
100 FORMAT (……………)
```

や、

```
WRITE (22, 110) Y  
110 FORMAT (……………)
```

の15や22のように、そのまま入出力文に書くだけでよい。しかし、PL/Iではファイル宣言
が必要である。今、例えばOSAKAという名前の入力ファイルを宣言するには、

```
DCL OSAKA FILE INPUT;
```

とすればよい。TOKYOという名前の出力ファイルを宣言するには、

```
DCL TOKYO FILE OUTPUT;
```

とすればよい。出力ファイルのうち、特に印刷ファイル(すなわちラインプリンタ)であることを
宣言するには

```
DCL TOKYO FILE PRINT;
```

とすればよい。すなわち、ファイル宣言の一般形は、

```
DCL ファイル名 FILE INPUT (又はOUTPUT, 又はPRINT) ;
```

となる。勿論この他にもSTREAMとRECORDの指定や、ENVIRONMENT属性など、
ファイル宣言文に書けることは山ほどあるが、FORTRANユーザのPL/I入門としては上記
の形で十分であり、ややこしいことを並べたてても頭が混乱するだけなので、それらは全て割愛す
ることにする。ファイル名(上例でのOSAKA, TOKYOなど)の長さは、TYPE (A) ,
(AB) では5文字以内、OSM/F4では7文字以内である。標準出力ファイルSYSPRIN

Tは8文字であるが、これは例外として許されている。

上記の一般形で、最後に書いてある入出力指定 (INPUT, OUTPUT, PRINT) のうち、ラインプリンタを指定する PRINT はあまり使わないだろう。なぜならば、ラインプリンタを使う場合はわざわざファイル名を作って宣言しなくても、標準出力ファイル SYS PRINT を使えばよいからである。FORTRAN でラインプリンタを使う場合、ファイルコード 6 を使わずに、わざわざ JCL でファイル定義の必要な他の番号を使うことに相当する。但し、SYS PRINT は、PAGE SIZE や LINE SIZE が決まっているので、これを変更したいときは、印刷ファイルを宣言しなければならない。PAGE SIZE とは、1 頁に印刷できる行数で、SYS PRINT については TYPE (A), (AB), OSW/F 4 いずれも 60 行である。普通、ラインプリンタはハードウェア的には 66 行印刷可能である。LINE SIZE とは、1 行当りに印字できる文字数で、SYS PRINT については TYPE (A) では 132 文字、TYPE (AB) と OSW/F 4 では 120 文字である。普通、ラインプリンタでハードウェア的に印字可能な文字数は 132 文字である。

1 つのファイルを入力と出力の両方に使いたい場合は、入出力指定を書かないで、例えば、

```
DCL OSAKA FILE;
```

としておいて、後述する OPEN 文によって入出力指定を行えばよい。

(15-3) OPEN 文と CLOSE 文

OPEN 文と CLOSE 文は必ず書かなければならないという文ではなく、省略してもよい。OPEN 文がなくても最初の GET 文や PUT 文によって当該ファイルが OPEN される。又、CLOSE 文がなくてもプログラムの実行が終了すると自動的にファイルが CLOSE される。こう書くと、OPEN 文や CLOSE 文は全くの無用の長物で、存在価値がないように思えるが、実はそんなことはなく、使わなければならないときがある。その 2 つの場合を次に示す。

<a> PAGE SIZE や LINE SIZE を指定するとき

前節で述べたように TYPE (A) や OSW/F 4 では SYS PRINT は 1 頁 60 行、1 行 120 文字なので、1 頁 66 行、1 行 132 文字フルに印刷したいときは次のようなファイル宣言と OPEN 文が必要である。

```
DCL SYSPRINT FILE PRINT;  
OPEN FILE (SYSPRINT) PAGE SIZE (66)  
LINE SIZE (132);
```

ここで、ファイル名は標準出力ファイル SYS PRINT をそのまま使ったが、どうせファイ

(15-4) カード出力の方法

PL/Iでは、カード出力用の標準出力ファイルは設定されていないので、プログラムの中でファイル宣言を行うとともに、JCLでファイル定義をしなければならない。JCLはシステムによって異なり、一般性を持った説明はできないが、ACOS-6の場合について、JCLを示しておく。ファイル名を例えばCARDPとすると、プログラムでは、

```
DCL CARDP FILE OUTPUT ;
OPEN FILE (CARDP) LINESIZE (80) ;
.....
PUT FILE (CARDP) EDIT (.....) (.....) ;
```

となる。OPEN文のLINESIZEが80というのはカードイメージ(80欄カード)だからである。PUT文はEDIT型で書いたが、勿論LIST型やDATA型でもよい。

この場合、JCLは、

```
1カラム      8カラム      16カラム
$           PUNCH      CP, X1S
```

となる。CPはPL/Iにおけるファイルコードであり、FORTRANでは05, 06, 43等の2桁の数字からなるのに対して、2桁の英数字からなっており、1桁目は英字でなければならない。このファイルコードはプログラムで宣言したファイル名の頭とシッポの文字をつなぎ合せた形であり、この例ではファイル名はCARDPゆえ、先頭のCと、最後尾のPをつないで、ファイルコードはCPとなっている。ファイル名がOSAKAであればファイルコードはOA, TOKYOであればTO, MT1であればM1というふうになる。他にOPEN文のTITLEオプションというのを使う方法もあるが、ここでは省略する。ちなみに言っておくと、標準入出力ファイルSYSIN, SYSPRINTのファイルコードだけはSN, STとはならずそれぞれ、I*, P*となるが、これはプログラマーには無関係のことである。上記JCL中のX1Sはロジカルユニットデジグネータであるが、分からない人は上記のまま打っておけばよい。

(15-5) 磁気テープや磁気ディスクファイルの場合

磁気テープの場合も基本的には全く同様で、例えばファイル名をMTAPEとすると、プログラムは、

```
DCL MTAPE FILE ;
OPEN FILE (MTAPE) OUTPUT ;
PUT FILE (MTAPE) LIST (.....) ;
```

などとなる。上例は出力の場合であるが、入力の場合も同様に書けばよい。JCLは、FORTR

ル宣言するのだから他の名前を使ってもよいだろうと考えて、例えばファイル名を JAPAN とすると、PUT 文でファイル名を省略することができないで、いちいち、

```
PUT FILE (JAPAN) LIST (.....) ;
```

などを書かねばならない。これに対してファイル名を SYSPRINT としておく、省略することができて、

```
PUT LIST (.....) ;
```

と簡単になる。

入力と出力を兼ねるとき

入力（読み込み）と出力（書き込み）の両方に使うファイル（磁気テープやパーマネントファイル等）の場合、OPEN 文と CLOSE 文が不可欠となる。

```
DCL MT1 FILE ;
.....

OPEN FILE (MT1) INPUT ;
GET FILE (MT1) LIST (.....) ;
CLOSE FILE (MT1) ;
.....

OPEN FILE (MT1) OUTPUT ;
PUT FILE (MT1) LIST (.....) ;
CLOSE FILE (MT1) ;
END ;
```

この例ではファイル MT1 からデータを読み込んで、何らかの処理をして再び MT1 に書き込む、一種の更新処理を行っている。1 番目の OPEN 文は MT1 を入力ファイルとして開き、2 番目の OPEN 文は同じ MT1 を出力ファイルとして開いている。従って 1 番目の CLOSE 文は必要不可欠であるが、2 番目の CLOSE 文はここでは省略してもよい。

以上が OPEN 文と CLOSE 文の必要な 2 つの場合である。PL/I の OPEN 文は FORTRAN の REWIND 文を含んでいると考えてもよい。又、CLOSE 文は FORTRAN の ENDFILE 文と REWIND 文を含んでいる。FORTRAN の REWIND 文、ENDFILE 文あるいは BACKSPACE 文に単独に相当する PL/I の文は存在しない。従って、磁気テープ上に、情報と EOF マークを交互に書き込んでゆくマルチファイルを作成する場合などは、FORTRAN に比べてやや不便である。

ANで磁気テープを使用する場合と全く同じで、\$TAPE9等を用いればよいが、ただファイルコードが数字ではなく、この場合であればファイル名がMTAPEだから、頭とシッポをつないで、MEとなる。

磁気ディスク上のパーマネントファイルやテンポラリファイルの場合もJCLはFORTRANの場合と同じで、\$PRMFLや\$FILEを使えばよいが、ただファイルコードの部分だけ上記の規則で変えればよい。

今回はGO TO文、IF文、DO文、ファイル入出力文について解説した。次回はON文、構造体、JCLのまとめ等について述べる予定である。 (プログラム相談員)

/* 前回 (第3回) の誤植訂正 */

(1) 86頁, 上から12行目

(誤) できる。次のようにすればよい。メインで、

(正) できる。次のようにすればよい。メインで、

(2) 90頁, 下から9行目

(誤) DCL A INIT (1. 0), K INIT (5), M CHAR (4) INIT

(正) DCL A INIT (1. 0), K INIT (5), M CHAR (4) INIT
(' ABCD ');

以 上