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、頭を現実に引き戻して、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、ＯＳⅣ／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。整数型のようで整数型でない、文字型のようで文字型でない、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文には必らずＡＳＳＩＧＮ文がつきものなのである。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は無く、代りにラベル（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：（コロン）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＜ｂ＞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。

＜b＞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。ＯＳⅣ　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。

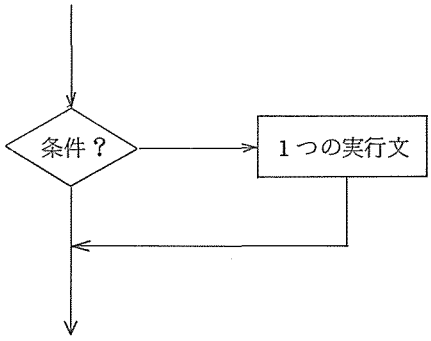

図１３－１

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。

＜b＞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≠B	A．NE．B	A¬＝B
A＜B	A．LT．B	A＜B
A≦B	A．LE．B	A＜＝B
A＞B	A．GT．B	A＞B
A≧B	A．GE．B	A＞＝B
A≮B	.NOT.（A．LT．B）	A¬＜B
A≯B	.NOT.（A．GT．B）	A¬＞B
	（A，Bは算術型）	（A，Bは算術型）
P∧Q	P．AND．Q	P&Q
P∨Q	P．OR．Q	P｜Q
$\overline{P}$	.NOT．P	¬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

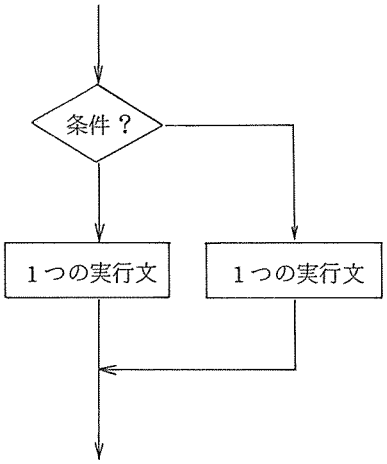

図１３－２

ために、ＦＯＲＴＲＡＮの最新改良版とも言えるＦＯＲＴＲＡＮ７７という新しい言語ではＥＬＳＥ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;
```

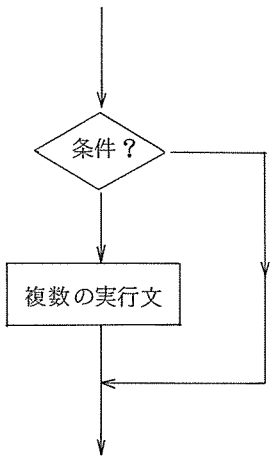

図１３－３

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。

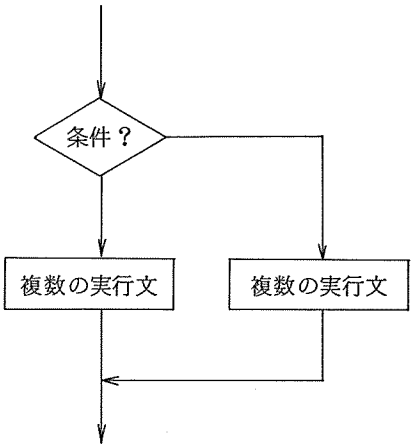

図１３－４

＜ｅ＞ひし形横並び型（ＩＦ文の入れ子）

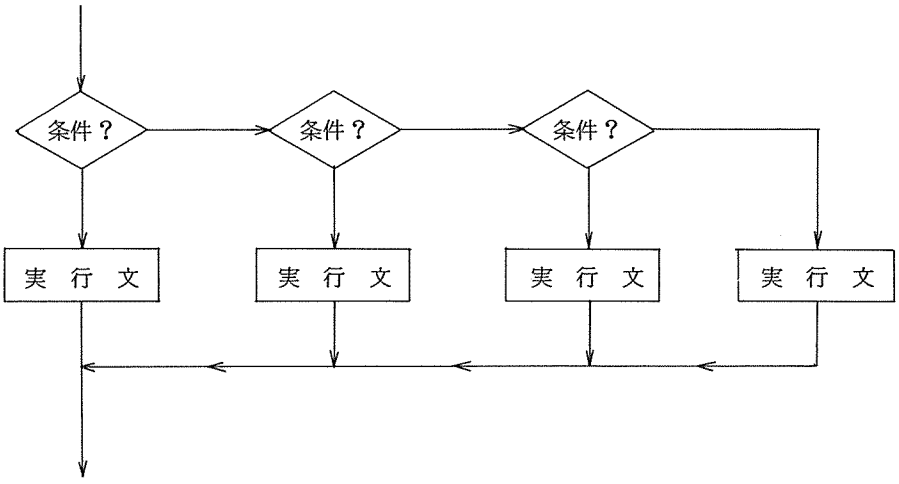

図１３－５

図１３－５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、必らず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。

＜b＞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の後に端末文の文番号は書かず、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文から端末文の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ループ」と同じような意味をもつ）の繰返しの途中でも、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でこれをするとエラーとなるので、

```
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節を伴なうか、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節を伴なう場合は、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，……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節も伴なわず、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、＜b＞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、＜b＞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＜b＞と＜c＞のDO文を組み合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きざみで繰返し実行し、次にM＝50からM＝200まで10きざみで繰返し実行する。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）、パーマネントファイル、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からの読込みは、

```
      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、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，OSⅣ／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ＳＹＳＰＲＩＮＴ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、ＳＹＳＰＲＩＮＴは、ＰＡＧＥＳＩＺＥやＬＩＮＥＳＩＺＥ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。ＰＡＧＥＳＩＺＥとは、１頁に印刷できる行数で、ＳＹＳＰＲＩＮＴについてはＴＹＰＥ（Ａ），（ＡＢ），ＯＳⅣ／Ｆ４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。ＬＩＮＥＳＩＺＥとは、１行当りに印字できる文字数で、ＳＹＳＰＲＩＮＴについてはＴＹＰＥ（Ａ）では１３２文字、ＴＹＰＥ（ＡＢ）とＯＳⅣ／Ｆ４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文は必ら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＞ＰＡＧＥＳＩＺＥやＬＩＮＥＳＩＺＥを指定するとき

前節で述べたようにＴＹＰＥ（Ａ）やＯＳⅣ／Ｆ４ではＳＹＳＰＲＩＮＴ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)  PAGESIZE(66)
        LINESIZE(132);
```

ここで、ファイル名は標準出力ファイルＳＹＳＰＲＩＮＴ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を示しておこう。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とはならずにそれぞれ、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（……）；
```

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

```
PUT LIST（……）；
```

と簡単になる。

＜b＞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。メイントで、

（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