The University of Osaka
Institutional Knowledge Archive

Title	FORTRANとの比較によるPL/Ⅰ入門 (2)
Author(s)	塩野, 充
Citation	大阪大学大型計算機センターニュース. 1980, 37, p. 57-74
Version Type	VoR
URL	https://hdl.handle.net/11094/65446
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

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

The University of Osaka

資　料

FORTRANとの比較によるPL／I入門(2)

大阪大学工学部　塩　野　　充

前回はPL／Iのさわりだけを紹介したにすぎなかったが、今回から内容に入り、データの型、型宣言文、入出力文について説明しよう。その前に、前回PL／I言語の使用比率の例として東大大型計算機センターの76年のデータを表1として掲げたが、少し古すぎるので、もう少し新しいデータはないかと捜した結果、京大大型計算機センターの78年の例が公表されているのでそれを表2に示す。

表2.　京大大型計算機センターにおける言語使用比率
（77年12月～78年4月）

言　語	ステップ数	比　率
FORTRAN GE	93903	58.6%
FORTRAN HE	63002	39.3%
PL／I	2913	1.8%
COBOL	434	0.3%
ALGOL	8	0.0%

（ステップとはACOSでいうアクティビティに相当する。%は筆者の計算による。）
飯田記子："バッチ・ジョブにおけるプログラムの使用頻度について"京都大学大型計算機センター広報，Vol。11, No.3，1978（8月）より抜萃。

前回の表1ではPL／Iは1.1%だったが表2では1.8%となっており、大して差がないと言ってしまえばそれ迄だが、少しは大きいと言えよう。阪大大型計算機センターにおける言語使用比率にも大変興味があるが、まだ公表されていないようである。筆者の推測では阪大でのPL／I使用比率は上記2例よりもはるかに低いのではないかと思う。当誌の「センターだより」の欄には毎月の処理状況が詳細に載っているが、この欄に言語使用比率も是非掲載してもらいたいものだと思う。PL／Iのみならず、APL，PASCAL，LISPといった新しい言語がどの程度使われているか、筆者以外にも興味を持っている人は多いであろう。

前回、第５章ではＰＬ／Ｉのコーディング様式について説明したが、今回の第６章から内容の本格的な説明に入る。ところで、前回にも少し書いたが本稿はＦＯＲＴＲＡＮを一通りマスターした人がＰＬ／Ｉを容易に理解できることを目的としているので、通常のＰＬ／Ｉ入門書とは少し異なる。従って各種のステートメントを厳密な一般形で表わすようなことはせず、ＦＯＲＴＲＡＮの例文をそのままＰＬ／Ｉに変換して示すことにより、賢明なる読者の頭の中にステートメントの一般形が組み立てられるものと思う。クドクドしい説明はなるべく少なくし、“答一発”ではないが、“例文一発”で即理解して頂けるよう配慮してあるつもりである。人間はコンピュータと違って、高度なパターン認識能力と学習能力を持っているので、このようなことは朝めし前の筈（？）である。とは言ってもこちらの書き方がまずいために朝めし前でない向きもあるかも知れないが、だからと言ってあなたのパターン認識能力、学習能力にケチをつけるつもりは毛頭ありません。

第６章　データの型

ＦＯＲＴＲＡＮでは数値型変数名の場合、Ｉ，Ｊ，Ｋ，Ｌ，Ｍ，Ｎで始まるものは整数型、それ以外、すなわちＡ～Ｈ，Ｏ～Ｚで始まるものは実数型という、暗黙の型宣言があった。ＰＬ／Ｉでも全く同様で、Ｉ，Ｊ，Ｋ，Ｌ，Ｍ，Ｎで始まる変数名は整数型、それ以外の場合は実数型と見なされる。これをＩ－Ｎルールと呼ぶ。ＦＯＲＴＲＡＮにおける暗黙の型宣言のように、プログラマがいちいち指定しなくてもコンパイラが「気を利かせて」解釈する機能をＰＬ／Ｉでは省略時解釈機能（或いは標準ルール、英語ではｄｅｆａｕｌｔ　ｒｕｌｅ，又はｄｅｆａｕｌｔ　ｆｅａｔｕｒｅ という。ｄｅｆａｕｌｔ とは不履行，怠慢，欠席，欠乏等の意味である）と呼ぶ。ＦＯＲＴＲＡＮにも上述の暗黙の型宣言以外に、例えば最初のＲＥＡＤ文やＷＲＩＴＥ文でファイルのオープンを自動的に行うなど、この種の機能を備えているが、ＰＬ／Ｉの方がより豊富である。尤もＦＯＲＴＲＡＮにはもともとファイルをオープンする文はないが……。

ＦＯＲＴＲＡＮでは文字型はＪＩＳでは規定されておらず、通常は整数型として扱われる。但し、ＡＣＯＳ－６　ＦＯＲＴＲＡＮでは文字型を設けている。ＰＬ／Ｉでは文字型が許される。この他にＦＯＲＴＲＡＮの論理型に相当するＢＩＴ（ビット）型がある。複素数型はＦＯＲＴＲＡＮのように実数型、整数型と区別されて独立した型としてあるのではなく、実数型、整数型いずれにもオプション的に指定できる。従ってＦＯＲＴＲＡＮのように複素数型といえば実部も虚部も実数型とは限らず、実部と虚部が整数型の複素数型もありうる。ＦＯＲＴＲＡＮでは複素定数、例えば 2.5 ＋ 4.5 ｉを表わすには（2.5，4.5）と書いたが、ＰＬ／Ｉでは 2.5 ＋ 4.5 Ｉと、虚数部にＩを付けて和（差）の形で書く。単に 4.5 Ｉとすれば純虚数 4.5 ｉを表わす。

ＰＬ／Ｉにおけるデータの型にはこの他にラベル型等、プログラム制御用データの型があるがここでは省略する。

ＦＯＲＴＲＡＮでは変数名は６文字以内と決められていたが、ＰＬ／Ｉでは普通３１文字迄許している。（但し、ＡＣＯＳ－６ではＴＹＰＥ（Ａ），ＴＹＰＥ（ＡＢ）とも２５６文字迄許している。しかし、そんな長い名前を付けると落語のジュゲムジュゲムのように混乱のもととなるだろう）。変数名として使える文字はＦＯＲＴＲＡＮでは英字と数字だけだったがＰＬ／Ｉではこれに加えて、ＴＹＰＥ（ＡＢ）では、@（アットマーク，単価記号），＃（ナンバーマーク，井桁印），$（又は¥），＿（マイナスではなくて下線，アンダーライン）の４文字が使える。ＴＹＰＥ（Ａ）では$と＿は使えるが、@と＃は使えない。これら４文字のうち＿以外の@，＃，$は英字として扱われるので変数名の頭に使ってもよい。なお、前回でも触れたが＿はマイナスとまぎらわしくなる恐れがあるのでなるべく使わない方が無難であろう。

ＰＬ／Ｉでは変数名，外部手続き名（サブルーチン名等），ファイル名等をまとめて名標と呼んでいる。

第７章　型宣言文

数値型のデータ（算術データ）は暗黙の型宣言（Ｉ－Ｎルール）に従う場合は無宣言で使ってよいが、そうでない場合は明確に宣言する必要がある。

なお、ＡＣＯＳ－６ではＴＹＰＥ（ＡＢ）にＩ－Ｎルールがあるが、ＴＹＰＥ（Ａ）にはＩ－Ｎルールがないので要注意である。これは不自然かつ不便なことだと筆者は思う。ＴＹＰＥ（Ａ）でＩ－Ｎルールを付加する方法はあるが章末で述べる。又、文字型やビット型では明確に宣言しなければならない。ＰＬ／Ｉでは宣言文は全て、ＤＣＬという言葉から始まる。ＤＣＬはDECLAREの省略型である。省略型というのは、本来、ＤＥＣＬＡＲＥと書くべき所をＤＣＬと書いてもよいということである。省略型は殆んどのＰＬ／Ｉで互換性があり、パンチミスも少なくできるので、以降は省略型を主に使っていくことにする。

(1)　整数型

ＦＯＲＴＲＡＮで、

```
INTEGER A
```

なる整数型宣言は、ＰＬ／Ｉでは、

```
DCL A FIXED BIN;
```

となる。ＢＩＮはＢＩＮＡＲＹ（２進数）の省略型である。２進数というのは計算機内部での話であり、気にする必要はない。Ａは普通の１０進数の整数と考えてよい。変数名が２つ以上のとき、すなわち、

```
INTEGER A, B, C
```

のような場合は、

DCL (A, B, C) FIXED BIN;

とカッコでくくればよい。FIXEDとBINの順序は逆でもよい。

(2) 実数型

FORTRANで、

REAL K

なる実数型宣言は、PL/Iでは、

DCL K FLOAT DEC;

とすればよい。DECはDECIMAL（10進数）の省略型である。同様に、

REAL K, L, M

はPL/Iでは、

DCL (K, L, M) FLOAT DEC;

とカッコでくくればよい。FLOATとDECの順序も逆でもよい。

(3) 複素数型

前述の整数型又は実数型宣言にCPLX（COMPLEXの省略型）を付け加えればよい。すなわち、実部と虚部が整数型からなる複素数Zの宣言は、

DCL Z CPLX FIXED BIN;

とすればよい。又、実部と虚部が実数型からなる複素数Zの宣言は、

DCL Z CPLX FLOAT DEC;

とすればよい。変数名が2つ以上あるときは前と同様にカッコでくくればよい。変数名が暗黙の型宣言に従うときはCPLXの後は省略してよい。例えば、実部と虚部が整数型からなる複素数Kの宣言は、

DCL K CPLX;

でよいし、実部と虚部が実数型からなる複素数Aの宣言は、

DCL A CPLX;

とすればよい。

(4) 倍精度型

倍精度実数型の宣言は、FLOAT DECの後に（p）を付け加えればよい。カッコ内の数字pは精度を表わすもので、コンパイラによって異なる。ACOS－6の場合、TYPE（A）ではp＝59，TYPE（AB）ではp＝31となる。ちなみに京大センターのFACOM M200用のOSⅣ/F4 PL/Iコンパイラではp＝33となる。これらのpの値を書いた場合、前述の普通の実数型宣言のときの5倍程度の精度となる。FORTRANの、

DOUBLE PRECISION X

は、PL／IのTYPE（AB）では、

DCL X FLOAT DEC（31）；

とすればよい。同様に倍精度複素数、

DOUBLE PRECISION COMPLEX Z

は、TYPE（AB）では、

DCL Z CPLX FLOAT DEC（31）；

とすればよい。TYPE（A）やOSⅣ／F4では（31）をそれぞれ（59），（33）とすればよい。変数名が2つ以上のときはカッコでくくる。

ところで、もしベテランの方が読まれていて苦情が出るといけないので書いておくが、(1)，(2)，(3)の宣言文にも実際には精度指定を書くべきかも知れないが、本稿はFORTRANユーザが、なるべく容易にPL／Iを理解できることが目的であるため、簡略化できる所は極力簡略化して書いている。又、実数型はFORTRANと厳密に対応させる場合は、FLOAT BINであり、更にPL／IプログラムをFORTRANプログラムと結合させる場合には精度指定もきっちり行わなければならない。しかし、ここでの話はまだその段階ではないので、実数型は暗黙の型宣言に合せてFLOAT DECとしてある。整数型についても同様である。

(5) 文字型

前述のように文字型はJIS－FORTRANでは規定されていないが、PL／Iでは次の例のように宣言する。

DCL A CHAR（10）；

ここで、CHARはCHARACTERの省略型である。（10）というのは文字列の長さが10文字であることを示し、もっと長くてもよい。FORTRANのように1つの変数名には4文字ないし6文字しか入らないということはない。許される長さの最大値はTYPE（A）では254，TYPE（AB）では512となっている。カッコを書かず単にCHARとすれば長さは1と見なされる。変数名が2つ以上のときはカッコでくくる。

文字型のデータはFORTRANではnH○○……○で表わしたがPL／Iでは両側を引用符ではさんで表わす。例えば、5HJAPANは、'JAPAN'となる。これはJISを越えたFORTRANでも使われている。

(6) ビット型

ビット型はFORTRANにおける論理型を拡張したものと考えてよい。FORTRANの論理型宣言は例えば、

LOGICAL A

と書くが、これに対応するPL／Iのビット型宣言は、

DCL A BIT(1);

となる。カッコ内の1はビット列の長さであり、もっと長くてもよい。ここがFORTRANと違うところで、FORTRANの場合、論理型はその値として.TRUE.か、.FALSE.すなわち、真（ビットで言えば1）か、偽（ビットで言えば0）のいずれかしかとれない。従ってFORTRANの論理型はPL／Iのビット型の長さ1の場合に相当する。ビット型で許される長さの最大値はTYPE（A）では253，TYPE（AB）では511となっている。長さを省略すると1と見なされる。変数名が2つ以上のときはやはりカッコでくくる。ビット型のデータは引用符ではさんで後にBを付ける。例えば、'1011010'Bというように表わす。

(7) 配列の宣言

配列の宣言は簡単で、今まで述べた(1)〜(6)の各変数名のところを配列名に変えればよい。例えば、暗黙の型宣言に従うサイズ5×10の2次元配列A（5，10）は、

DCL A(5, 10);

とすればよい。暗黙の型宣言に従わないときは、例えば整数型として、

DCL A(5, 10) FIXED BIN;

というようにすればよい。配列の添字はFORTRANでは正整数と決まっていたが、PL／Iでは負整数でもよい。例えば、

DCL A(−2:2, −6:3);

は配列のサイズは前の例と同じ5×10であるが、配列の第1添字は−2，−1，0，1，2，第2添字は−6，−5，−4，−3，−2，−1，0，1，2，3の各値をとりうる。このように：（コロン）で添字の上限と下限を表わす。従って、

DCL A(5, 10);

は、

DCL A(1:5, 1:10);

と等価である。

この機能はFORTRANの最新版とも言えるFORTRAN77にも取り入れられているが、整数座標を配列に置き換えて考えるときなど非常に便利である。配列の次元数はFORTRANでは最大7次元までであるが、PL／IではTYPE（A），（AB）共に31次元、OSⅣ／F4では15次元まで使用できる。配列で気を付けなければならないことは、2次元以上の配列の要素の並び方である。2次元以上の配列の要素は実際の主記憶上では1次元的に直列に並んでいる。その並び方がFORTRANとPL／Iでは異なる。例えば2次元配列A（2，3）は、FORTRANでは、

A（1，1）
A（2，1）
A（1，2）
A（2，2）
A（1，3）
A（2，3）

の順に並んでいるが、ＰＬ／Ｉでは、

A（1，1）
A（1，2）
A（1，3）
A（2，1）
A（2，2）
A（2，3）

という順に並んでいる。つまり、ＦＯＲＴＲＡＮでは左側にある添字ほど忙しく変化していくが、ＰＬ／Ｉでは右側にある添字ほど忙しく変化していく。２次元の場合は画面と考えればＦＯＲＴＲＡＮでは縦走査、ＰＬ／Ｉではテレビと同じ横走査と言えよう。

２つ以上の配列や、スカラと配列をまとめて宣言することもできる。例えば、

```
DCL （A（5，10），B（8），C，D） FIXED BIN；
```

などとできる。ＣとＤはスカラである。

２つ以上の宣言文を１つにまとめるにはカンマでつなげばよい。例えば、

```
DCL X（5）；
DCL Y FIXED BIN；
DCL S CHAR（30）；
DCL T BIT（20）；
```

は次のように書いてもよい。

```
DCL X（5），Y FIXED BIN，S CHAR（30），T BIT（20）；
```

しかし、先のように分けて書いた方が見易いかも知れない。ところで、ＦＯＲＴＲＡＮでは宣言文は必らず実行文より前になければならなかったが、ＰＬ／Ｉではどこにあっても前にあるのと同等に見なされる。

以上、各種の型および配列の宣言文について説明した。ＰＬ／Ｉにはこの他に構造体という、ツリー状のデータを格納するのに便利なものがあるが、ＦＯＲＴＲＡＮには存在しないのでもっと後章で改めて説明することにする。なお、最後に付け加えておくがＰＬ／Ｉでは暗黙の型宣言のように省略時解釈を適用した名標は全てＷＡＲＮＩＮＧとして出力される。この場合のＷＡＲＮＩＮＧ

はＦＯＲＴＲＡＮの場合の「警告」というような強い意味ではなく、「確認」「念押し」程度の意味である。

又、前述したようにＡＣＯＳ－６のＴＹＰＥ（Ａ）にはＩ－Ｎルールがないが、プログラムで付け加えることができる。それにはＦＯＲＴＲＡＮのＩＭＰＬＩＣＩＴ文に対応するＰＬ／ＩのＤＥＦＡＵＬＴ文を使えばよい。すなわち、ＦＯＲＴＲＡＮにおける、

```
IMPLICIT REAL (A-H)
IMPLICIT INTEGER (I-N)
IMPLICIT REAL (O-Z)
```

をＰＬ／Ｉで書いて、

```
DFT (RANGE (A:H)) FLOAT DEC;
DFT (RANGE (I:N)) FIXED BIN;
DFT (RANGE (O:Z)) FLOAT DEC;
```

とすればよい。ＤＦＴはＤＥＦＡＵＬＴの省略型である。

第８章　入出力文

ＰＬ／Ｉの入出力文は大別して２種類あり、１つはＧＥＴ文（入力文）とＰＵＴ文（出力文）からなるストリーム型入出力文であり、もう１つはＲＥＡＤ文（入力文）とＷＲＩＴＥ文（出力文）からなるレコード型入出力文である。このうち、後者のレコード型入出力文はＣＯＢＯＬの入出力文に対応するので、ＦＯＲＴＲＡＮユーザにはあまり関係がない。ＦＯＲＴＲＡＮのＲＥＡＤ，ＷＲＩＴＥ文に対応するのはＧＥＴ，ＰＵＴ文，すなわちストリーム型入出力文である。

ストリーム型入出力文も更に次の３種類に分けられる。

ストリーム型入出力文 ｛ ＥＤＩＴ型入出力文
　　ＬＩＳＴ型入出力文
　　ＤＡＴＡ型入出力文

このうち、ＦＯＲＴＲＡＮの書式つき入出力文に対応するのはＥＤＩＴ型入出力文である。ＬＩＳＴ型入出力文はＦＯＲＴＲＡＮのリスト指示入出力文（暗黙書式つき入出力文）に対応する。ＤＡＴＡ型入出力文はＦＯＲＴＲＡＮのネームリスト型入出力文をずっと使い易くしたものと考えてよいだろう。

以下に出力文と入力文について解説するが、ここでは簡単化のため、出力装置はラインプリンタ、入力装置はカードリーダとして考える。

（８－１）出力文（ラインプリンタ）

ここではまず出力文について解説する。入力文より出力文を先に説明するのは奇異に思えるかも

知れないが、入力文のないプログラムはあっても出力文のないプログラムはまずないと言ってもよい程、出力文は不可欠なものなので、まず出力文から説明することにする。

＜a＞　ＥＤＩＴ型出力文

ＦＯＲＴＲＡＮの書式つき出力文、例えば、

```
        WRITE（6, 100）A
100  FORMAT（F7.3）
```

をＰＬ／Ｉで書くと、

```
PUT  EDIT（A）（F（7, 3））;
```

となる。すなわち、一般的には、

```
PUT  EDIT（変数名）（書式）;
```

となる。ＦＯＲＴＲＡＮにおけるＦ書式（固定小数点書式）のＦ７．３はＰＬ／Ｉでは、Ｆ（7, 3）と数字部分をカッコで包み、ドットはカンマにする。同様にＥ書式（浮動小数点書式）、例えば、Ｅ１３．５はＰＬ／ＩではＥ（１３，５）となる。又、出力様式も多少異なり、－１２３４をＦＯＲＴＲＡＮのＥ１１.4で出力すると、－0.１２３４Ｅ＋０４となるがＰＬ／ＩのＥ（１１,４）で出力すると、－1.２３４０Ｅ＋０３となり、１の位に有効数字が入る。Ｉ書式（整数書式）はＰＬ／Ｉにはなく、Ｆ書式を用いる。すなわち、例えばＦＯＲＴＲＡＮでのＩ８は、ＰＬ／ＩではＦ（８）とすれば同じ機能となる。ＦＯＲＴＲＡＮではＦ８という書き方は許されない。Ｆ書式では出力すべきデータの値が下位桁で書式よりはみ出していれば丸め（絶対値の四捨五入）が行われる。例えば、2.5や－5.5をＦ（２）で書かせるとそれぞれ、３，－６になる。

Ａ書式（文字列書式）は、例えばＡ４はＰＬ／ＩではＡ（４）となる。ここでカッコと数字を省略して単にＡとしてもよい。この場合、長さは可変長となり、出力すべきデータの長さに合わされるので非常に便利である。

空白を作るためのＸ書式（空白書式）は、例えば１０Ｘは、ＰＬ／Ｉでは、Ｘ（１０）となり、数字部分がカッコに包まれて後にくる点が異なる。とにかく、Ｆ，Ｅ，Ａ，Ｘ各書式いずれにしてもＰＬ／ＩではＦＯＲＴＲＡＮと違って、数字部分をカッコで包まなくてはならないこと、及びドットはカンマにするということを覚えておくとよい。

ＰＬ／Ｉではこの他に次のような書式がある。ビット型データのためのＢ書式（ビット列書式）は、Ａ書式と同様で、長さが５ならＢ（５）と書く。これも（５）を省略して単にＢとしてもよい。

複素数のためのＣ書式（複素数書式）は、Ｆ又はＥ書式が２つペアになった形で、

```
C（F（5, 1）, F（7, 3））
```

などと書く。

C（F（5，1））

と1つだけ書けば、

C（F（5，1），F（5，1））

と同じ書式を2つ書いたものと見なされる。C書式の場合、虚数単位を示す文字Iは出力されない。

この他にP書式（ピクチュア書式）があるが、COBOL的なものなので省略する。

同じ書式が続くときは1つにまとめることができる。FORTRANでは、

```
      WRITE（6，100）X，Y，Z
100   FORMAT（3F10．3）
```

と書くような場合，PL／Iでは、

```
PUT EDIT（X，Y，Z）（3 F（10，3））；
```

と書くか、又は、

```
PUT EDIT（X，Y，Z）（（3）F（10，3））；
```

と書く。両者の違いは繰返しの回数を示す3の部分にあり、前者は3とFの間に空白を入れ（入れなくてもよいコンパイラもある）、後者は3をカッコで包んである。どちらを使ってもよいが、後者のカッコで包む方を覚えておいた方が間違いも少なく、あとあと都合がよい。

EDIT型入出力文はFORTRANの書式つき入出力文に対応すると書いたが、厳密に対応するのはEDIT型入出力文のうちのR書式を使った場合である。R書式はリモートフォーマット（遠隔書式）と呼ばれ、FORTRANのFORMAT文を使う場合にぴったり対応する。すなわち、FORTRANで、

```
      WRITE（6，100）A，K
100   FORMAT（F7．3，I6）
```

をR書式を用いたPL／Iで書くと、

```
      PUT EDIT（A，K）（R（L1））；
L1：FORMAT（F（7，3），F（6））；
```

となる。FORTRANの場合のFORMAT文番号100が、PL／IのFORMAT文のラベルL1に対応する。すなわち、R書式のR（ ）のカッコ内にあるのがFORMAT文のラベルを表わす。ラベルというのは、FORTRANの場合の文番号に相当する。PL／Iでは文番号というのは付けないで、その代りに必要ならば文に名前を付ける。その名前のことをラベルというのである。上例ではL1がFORMAT文のラベルである。ラベルと文の間には必らず：（コロン）をはさむ。R書式ではリモートフォーマットの名の通り、FORMAT文は入出力文から遠く離れた所にあってもよく、この点FORTRANのFORMAT文と全く同じである。R書式は同じFORMATを使う入出力文が沢山あるときや、場合によってFORMATを種々切り換えて使いたい

ときに便利である。これは先の例で言えば、例えばR（LX）としておき、場合に応じて、LX＝L1，L2，L3，……と種々変化させて所望のFORMAT文を引っ張ってくればよい。しかし通常は普通のPUT　EDIT文で十分であろう。

次にFORTRANにおけるnH○○……○のように文字列を出力させる方法を述べよう。

FORTRANで例えば、

```
        WRITE（6, 100）X
  100   FORMAT（F7. 2, 9HKILOMETER）
```

はPL／Iでは、

```
  PUT EDIT（X, 'KILOMETER'）（F（7, 2）, A（9））;
```

となり、KILOMETERという文字列が、FORTRANでは書式の所に置かれるのに対し、PL／Iでは変数名（データ）の所に並べられる点が異なる。A（9）はKILOMETERと書くための書式であり、前述したように単にAと書いてもよい。

次に改頁や改行等の書式の制御について述べよう。書式の制御には制御書式を用いる方法と、書式制御用オプションを用いる方法の２通りある。まず制御書式について述べる。制御書式には次のようなものがある。

PAGE書式

SKIP書式

LINE書式

COLUMN書式

X書式も制御書式の１つであるが既に述べたので省略する。

PAGE書式は改頁を行い、FORTRANの1H1に相当する。例えば、

```
        WRITE（6, 100）K
  100   FORMAT（1H1, I5）
```

をPL／Iで書くと、

```
  PUT EDIT（K）（PAGE, F（5））;
```

となる。

SKIP書式は改行を行い、FORTRANの1H＋，1H0，1H　等に相当する。通常SKIP（n）の形式で用い、nはn行改行することを表わす。n＝0なら改行しないで重ね打ちすることを表わし、FORTRANの1H＋に相当する。n＝1なら1行改行で1H　に相当する。n＝2なら2行改行で1H0に相当する。nは3以上でもよい。

次のようなFORTRANの例、

```
        WRITE（6, 100）A, K
```

```
100  FORMAT(1H1, F10.3/////1H , I3)
```

は、PL／Iでは、

```
PUT EDIT(A, K) (PAGE, F(10, 3), SKIP(5), F(3));
```

となる。（n）を省略して単にSKIPと書くとSKIP（1）と見なされる。

LINE書式はその頁の上から何行目に印刷せよという意味で、FORTRANにはない機能である。LINE（n）の形で用いられ、上からn行目に印刷することを意味する。

```
PUT EDIT(X) (LINE(10), F(10, 2));
```

とすれば、その頁の上から10行目にXの値を印刷する。LINE書式の（n）は省略できない。又、逆に前述のPAGE書式には（n）は付けられない。2回改頁しようと思って、PAGE(2)などと書くと誤りで、そんなときは、PAGE，PAGEと2つ並べればよい。

COLUMN書式はその行の左端から何カラム目に印刷するかを表わし、JISを越えたFORTRANにある位置欄記述子Tに相当する。FORTRANで例えば、

```
      WRITE(6, 100)A
100  FORMAT(1H , T21, F7.3)
```

をPL／Iで書くと、

```
PUT EDIT(A) (SKIP, COLUMN(20), F(7, 3));
```

となる。FORTRANの場合は1カラム目は制御文字として使われるので、T21と書いても実際には20カラム目から印刷される。COLUMN書式も（n）は省略できない。

以上が制御書式であるが、前述したように書式制御用オプションを用いる方法もある。このオプションには次の3つがある。

PAGEオプション

SKIPオプション

LINEオプション

すなわち、COLUMN書式とX書式に相当するものがないだけで、あとは使い方も意味も全く同じだが、書く場所が異なる。オプションは、PUTとEDITの間か、又は一番後へ書く。例えば、

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

あるいは、

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

などと書く。両者は全く同じで、前者はAを印刷する前にスキップするが、後者は印刷してからスキップするという意味ではなく、どちらも印刷する前にスキップする。すなわち、SKIP書式で、

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

と書いたのと同じである。なお、ついでに述べると、Aを印刷してからスキップさせようと思って、

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

としてもSKIP(3)は無視される(書いていないのと同じ)から要注意である。これは他の制御書式についても同じである。この点FORTRANとは異なる。FORTRANでは、

```
      WRITE(6, 100)A
100   FORMAT(F7.3///)
```

とすると、Aを印刷してから3行改行される。PL/Iで、こうしたい場合は、

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

とすればよい。又、改頁だけさせたければ、

```
PUT PAGE;
```

とすればよい。すなわち、オプションを使う場合はEDIT以降がなくてもよいのである。これはオプションの存在理由の1つと言えよう。勿論、オプションの存在理由はこれが主ではなく、後述するLIST型やDATA型のように書式を書かない入出力文では必要不可欠なものなのである。

オプションを使うときの注意として、1つのPUT文に同時に指定できるのはPAGEオプションとLINEオプションだけであり、SKIPオプションはPAGEオプションやLINEオプションとは共存できない。すなわち、

```
PUT PAGE LINE(5) EDIT(A) (F(7, 2));
```

は許される。この場合、順序を逆にして、LINE(5) PAGEとしても最初に改頁が行われ、次に5行目へ行送りされる。しかし、PAGE SKIPや、LINE(n) SKIPの組合せは許されない。

<b>LIST型出力文

LIST型入出力文は前述したようにJISを越えたFORTRANにあるリスト指示入出力文(暗黙書式付き入出力文)に相当するものである。FORTRANのリスト指示入出力文は、

ACOS−6 FORTRANの場合、

```
WRITE(6, LIST)A, B, K
```

などと書く。すなわち、FORMAT文番号を書く所にLISTと書くだけで、FORMAT文は不要である。(これは書式なし入出力文とは違うので誤解のないように!)。FORMATはデータの型に応じてシステムが選択する。つまりシステムおまかせ型のFORMATである。しかし互換性はなく、FACOM OSⅣ/F4 FORTRAN HE(GE)では、

```
WRITE(6, *)A, B, K
```

と、LISTではなく星印を用いる。ACOS−6では複素数を除いてはこの形式も許しているの

で、複素数を出力する場合以外は星印を用いた方が互換性が保てる。

以上はＦＯＲＴＲＡＮの話であるが、ＰＬ／Ｉでは上例は、

ＰＵＴ　ＬＩＳＴ（Ａ，Ｂ，Ｋ）；

と書く。すなわち、ＥＤＩＴ型の場合と比べると、ＥＤＩＴと書いた所がＬＩＳＴとなり、書式を並べる２番目のカッコがいらない。

ＦＯＲＴＲＡＮのリスト指示入出力文では改頁とか改行等の書式制御はできなくて、いつもベタ打ちとなるが、ＰＬ／Ｉでは前述したオプションを使っていくらでもできる。例えば、

ＰＵＴ　ＳＫＩＰ（２）　ＬＩＳＴ（Ａ，Ｂ，Ｋ）；

などと書ける。又、文字列を混ぜて、

ＰＵＴ　ＰＡＧＥ　ＬＩＳＴ（Ｍ，'ＧＡＴＳＵ'，Ｎ，'ＮＩＣＨＩ'）；

のような書き方もできる。

ＬＩＳＴ型出力文は使い方が大変簡単なので、初心者や不精者におあつらえ向き、もってこいの出力文である。しかし、もっと不精者向きの出力文がある。それが次に述べるＤＡＴＡ型出力文である。

＜ｃ＞ＤＡＴＡ型出力文

ＤＡＴＡ型入出力文は形式的には、ＪＩＳを越えたＦＯＲＴＲＡＮにあるネームリスト入出力文にやや類似しているが、ネームリスト入出力文は使い方がややこしいのでベテラン向きであるが、ＰＬ／ＩのＤＡＴＡ型入出力文はＬＩＳＴ型入出力文と同様、非常に簡単に使えるので初心者やＦＯＲＭＡＴを考えるのが面倒な不精者に最適である。

ＬＩＳＴ型出力文では印刷はおまかせ型ＦＯＲＭＡＴであったが、ＤＡＴＡ型出力文ではそれに加えて、ごていねいに変数名と＝を印刷してくれるのである。すなわち、

変数名＝値

の形式で出力してくれる。例えば、Ｋ＝１２３４，Ｘ＝１．２３×10^8 だとすると、

ＰＵＴ　ＤＡＴＡ（Ｋ，Ｘ）；

とすれば結果は、

Ｋ＝１２３４　　Ｘ＝１．２３Ｅ＋０８；

という形式で印刷される。（最後に；が印刷される。又、実際には書式の関係で＝のあとに空白ができるかも知れない）。文の形式はＬＩＳＴ型の場合のＬＩＳＴをＤＡＴＡに変えるだけで、ＬＩＳＴ型と全く同じである。オプションも使え、

ＰＵＴ　ＳＫＩＰ（３）　ＤＡＴＡ（Ａ，Ｂ，Ｃ，Ｌ）；

などとしてもよい。しかし、ＬＩＳＴ型のように文字列をそのまま印刷させることはできない。例

えば、

PUT DATA (M, 'YEN') ;

などとすると誤りとなる。しかし、文字型変数を印刷させることはできる。この場合、LIST型と多少相異が生じる。例えば文字型変数NAMEの内容が文字列 'TARO' だとすると、

PUT LIST (NAME) ;

の結果は、

TARO

となるが、

PUT DATA (NAME) ;

の結果は、

NAME= 'TARO' ;

と、DATA型の場合は ' (引用符) もそのまま印刷される。

このように、DATA型出力文は変数名をそのつど印刷してくれるので、変数名をつけるとき、その変数の意味内容をそのまま表わす名前をつけておけば便利である。幸いPL/IではFORTRANの6文字と違って、もっと長い名前も自由に付けられる。しかし、DATA型入出力文は便利なだけに、EDIT型やLIST型入出力文に比べて効率が悪く、処理時間が長くかかり、サブセット的なPL/Iでは割愛されている。(ACOS-6や、OSⅣ/F4では勿論使用できる)。

(8-2) 入力文 (カードリーダ)

入力文は原則的には前述した出力文のPUTという箇所をGETに変えればよいのである。しかし、出力文の場合とは異なる点もあるので注意が必要である。EDIT型入力文, LIST型入力文, DATA型入力文の3つのタイプの入力文に共通していることは、出力文の場合とオプションが異なることである。出力文のところで述べたPAGEオプション, SKIPオプション, LINEオプションのうち、入力文でも使えるのはSKIPオプションだけで、PAGEオプションやLINEオプションは使えない。その理由はカードから入力することを考えればすぐ分るだろう。SKIP (n) はFORTRANのREAD文のFORMATにおける/(スラッシュ)と同じでカード送りを意味する。又、出力文にはなくて入力文特有のオプションで、COPYオプションというのがある。これは読み込んだデータをカードイメージそのままの形でラインプリンタに出力する機能であり、例えば、

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

と書くと、入力と同時にカードにパンチされている形式のままでAの値を印刷する。

＜a＞ＥＤＩＴ型入力文

出力の場合と異なって制御書式として、Ｘ書式，ＳＫＩＰ書式は許されるが、ＰＡＧＥ書式，ＬＩＮＥ書式はオプション同様許されない。又、前述したようにＣＯＰＹオプションはあるが、ＣＯＰＹ書式というのはない。

Ｆ書式ではカードにパンチされているデータとそれを読み込む書式が一致せず、いずれか一方に小数点があり、他方にないときは、小数点のある方が優先される。例えば、カード上に１８．７とパンチされていて、これをＦ（４）で読んでも１８．７と入力される。００１８や００１９にはならないし、エラーにもならない。逆に、１８７とパンチされたものをＦ（５，１）で読むと、18.7と入力され、１８７とはならない。又、データと書式の双方に小数点があるが位置が一致していないときはデータの方が優先される。例えば、１２．３４をＦ（５，１）で読み込んでも１２３．４とはならないで、１２．３４と入力される。

Ｅ書式についても同様である。例えば、１２３４Ｅ２とパンチされているデータを、Ｅ（６，２）なる書式で読み込むと、書式の方が小数点をもつので優先されて、１２．３４×１０^{2} として入力され、データ通りに１２３４×１０^{2} とはならない。同じデータをＥ（６，０）で読み込むと、データと書式が一致して、１２３４×１０^{2} として入力される。又、１２．３４Ｅ２なるデータをＥ（７）なる書式で読み込むと、データの方が小数点をもつので優先されて、１２．３４×１０^{2} として入力され、書式に合せて、１２３４×１０^{2} とはならない。データも書式も小数点をもつ場合、例えば、１２．３４Ｅ２なるデータを、Ｅ（７，１）なる書式で読み込むと、データの方が優先されて、１２．３４×１０^{2} として入力される。書式に合せて、１２３．４×１０^{2} とはならない。

Ｃ書式の場合はデータに虚数単位Ｉを付けてはいけない。あくまで２つの実数（整数を含む）のペアと考える。

以上のルールはＦＯＲＴＲＡＮの場合と大体類似している。しかし、ＰＬ／Ｉで注意すべき重要な点がある。それは、ＦＯＲＴＲＡＮの入力文（ＲＥＡＤ文）では、１つのＦＯＲＭＡＴ文はカード送りを示す／がない限り、１枚のカードに対応している。例えば、

```
        READ（5，100）A
  100   FORMAT（F7．3）
```

を１回実行し、もう１回この文を実行すると、２回目には次のカードの１カラム目から読むことになる。しかしＰＬ／Ｉでは、

```
  GET  EDIT（A）（F（7，3））；
```

を１回実行し、もう１回この文を実行すると、２回目は１回目と同じカードの第８カラムから７桁を読み込むことになる。だから次のカードの１カラム目から読ませたいときは２回目のＧＥＴ文を、

```
  GET  EDIT（A）（SKIP，F（7，3））；
```

とするか、１回目と２回目を同じ文にしたいならば、

ＧＥＴ　ＥＤＩＴ（Ａ）（（ＣＯＬＵＭＮ（１），Ｆ（７，３））；

とすればよい。

以上の事柄と関連して、もう１つ注意すべきことは出力文の項でも少し触れたが、制御書式以外の書式（Ｆ，Ｅ，Ｃ，Ａ，Ｂ書式）の最後尾にあるものより更に後にある制御書式は無視される。例えば、

ＧＥＴ　ＥＤＩＴ（Ａ，Ｋ）（Ｆ（８，１），Ｘ（５），Ｆ（７），Ｘ（６０））；

とすると、Ｘ（５）は機能するが、最後のＸ（６０）は無視される。つまり、ＡをＦ（８，１）で読み、５桁おいてＫをＦ（７）で読み終えるとあとは見向きもしないで次の文に移るのだと考えてよい。従ってもう１回この文を実行すると２回目は同じカードの第２１カラムから読み始める。

＜ｂ＞ＬＩＳＴ型入力文

ＬＩＳＴ型入力文はＬＩＳＴ型出力文のＰＵＴをＧＥＴに変えるだけでよい。データをカードにパンチするときは、１つずつカンマ又は空白で区切ればよく、書式には気を使わなくてよい。又、複素数は複素形式で入出力される。例えばカード上に、

１．２３，７５　６－８Ｉ　'ＪＡＰＡＮ'，'１１０１'Ｂ

とパンチされているデータを、

ＧＥＴ　ＬＩＳＴ（Ａ，Ｋ，Ｃ，Ｓ，Ｔ）；

で読み込むと、Ａ＝１．２３，Ｋ＝７５，Ｃ＝６－８ｉとなり、Ｓには文字列ＪＡＰＡＮ，Ｔにはビット列１１０１が代入される。（この場合Ｃは複素数型、Ｓは文字型、Ｔはビット型の宣言がされているものとする）。又、カード上に、

－６３．８，，４．３

とパンチされているデータを、

ＧＥＴ　ＬＩＳＴ（Ｘ，Ｙ，Ｚ）；

で読み込むと、Ｘ＝－６３．８，Ｚ＝４．３となるがＹには何も代入されず、変化しない。すなわち、カンマが２つ続くとその間に対応する変数には何も代入されない。

＜ｃ＞ＤＡＴＡ型入力文

ＤＡＴＡ型入力文もＤＡＴＡ型出力文のＰＵＴをＧＥＴに変えるだけでよい。データをカードにパンチするときはやはり１つずつカンマ又は空白で区切ればよく、書式には気を使わなくてよい。但し、ＬＩＳＴ型のときと違って、変数名＝値　の形式でパンチしなくてはいけないし、１つのＧＥＴ文で読み込む一組のデータの終りには；（セミコロン）を打たなくてはいけない。又、カー

ド上のデータの順序はＧＥＴ文中の順序と違っていてもよい。例えばカード上に、

Ｂ＝０．３＋５．６Ｉ，Ａ＝２３．１　Ｃ＝'ＤＡＹ'；

とパンチされているデータを、

ＧＥＴ　ＤＡＴＡ（Ａ，Ｂ，Ｃ）；

で読み込むと、Ａ＝２３．１，Ｂ＝０．３＋５．６ｉ，Ｃは文字列ＤＡＹとなる。（但し、Ｂは複素数型、Ｃは文字型の宣言がされているものとする）。又、カード上に、

Ｘ＝－２３．３，Ｙ＝０．８；

とパンチされているデータを、

ＧＥＴ　ＤＡＴＡ（Ｘ，Ｙ，Ｚ）；

で読み込むと、Ｘ＝－２３．３，Ｙ＝０．８となるが、Ｚには何も代入されず変化しない。又、この逆に、カード上に、

Ｓ＝１．８，Ｔ＝３，Ｕ＝５．５；

とパンチされているデータを、

ＧＥＴ　ＤＡＴＡ（Ｓ，Ｔ）；

で読み込むと、Ｕの入る場所がないので誤り（正確にはＮＡＭＥ条件の発生）となる。

以上で、ラインプリンタを対象とした出力文およびカードリーダを対象とした入力文の解説を終る。ラインプリンタやカードリーダ以外の入出力装置（パーマネントファイルや磁気テープ、カード出力等）を対象とした入出力文については後章のファイルに関連した部分で解説する予定である。なるべく肩のこらない文章にするべく努力しているが、内容が細部に入ってくるとなかなかそんなことも言っておれなくなる点、御許し頂きたい。次回は代入文，メインプログラムとサブプログラム，初期値設定等について解説する予定である。次回（８月）につづく。

（プログラム相談員）