OUKA
The University of Osaka
Institutional Knowledge Archive

Title	FORTRANとの比較によるPL/Ｉ入門 (3)
Author(s)	塩野, 充
Citation	大阪大学大型計算機センターニュース. 1980, 38, p. 79-95
Version Type	VoR
URL	https://hdl.handle.net/11094/65456
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（3）

大阪大学工学部　塩　野　　充

今回は代入文、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しよう。

第9章　代　入　文

（9－1）代入文の形式

ＰＬ／Ｉの代入文はＦＯＲＴＲＡＮと殆んど同じである。ただ、忘れてはならないが忘れやすいのが終りにつける；（セミコロン）である。しかしこれは代入文に限ったことではなく、ＰＬ／Ｉのステートメントを書くときは必ずつけなければならないものである。この、セミコロンというしろもの、ＦＯＲＴＲＡＮに慣れた者にとっては実にわずらわしく、うるさいものである。「五月蠅い」と書いて「うるさい」と読むそうだが、まさにこのセミコロンは蠅のようなものである。セミコロン1つ忘れただけでエラーになる。つまり、

Ａ＝１．２３

は誤りで、

Ａ＝１．２３；

としなければならない。セミコロンの効用は1行（すなわちカードで言えば1枚）にいくつもの文が書けることであり、セミコロンがない限り、行（カード）を改めても自動的に継続行と見なされるので、ＦＯＲＴＲＡＮのように第6カラムにこだわる必要がないことである。しかし、実際に1行にいくつもの文を詰めて、例えば、

Ａ＝１．２３；Ｂ＝０．８；Ｃ＝２．５；Ｋ＝１０；Ｌ＝２３；Ｘ＝１１．９……

などと書くと、ソースリストが見にくく、間違いの元となりやすい。やはり少々ぜい沢（？）なようでもＦＯＲＴＲＡＮのように1行1文というのが見やすくて良いようである。セミコロンを忘れないための1つのアイデア（？）としては、使用するカード全部にあらかじめ７２カラム（すなわちプログラムの書ける最後のカラム）にセミコロンをデュープ（複写キー）で打っておいてから、プログラムをセミコロンなしでパンチする、という手もあるが、果たしてどちらが能率的か……？。

ＡＣＯＳ－６　ＦＯＲＴＲＡＮではＰＬ／Ｉと同様にセミコロンで区切って1行にいくつもの文が書ける〝多重文〟という機能があるが、実際使ってみるとソースリストが見にくく、又、他機種との互換性もないので、カード枚数を極力少なくしたいというような特殊な場合以外はお勧めしかねる。

（9－2）FORTRANとの差異

話を代入文に戻そう。PL／Iの代入文で、FORTRANとは異なる点、又、FORTRANにはない機能について説明しよう。

まず、加減乗除の演算子はFORTRANと同じく、＋，－，＊，／，となる。べき乗も同じく、＊＊であるが、この場合は少し注意を要する。それは、例えばFORTRANで、

A＊＊2＊＊N

と書くと（このような書き方はJIS FORTRANでは許されていないが）、処理系によっては左から右へ計算されて、

（A＊＊2）＊＊N

と同じ意味になる場合もあるが、PL／Iでは右から左へ計算されて、

A＊＊（2＊＊N）

と同じ意味になる。（但し、ACOS－6 FORTRANや、FACOM OSⅣ FORTRANではPL／I同様、右から左へ計算される）。従って、例えばA＝10，N＝3とすると、A＊＊2＊＊Nの値は左から右へ計算されると、

（10＊＊2）＊＊3＝100＊＊3＝1000000

となるが、右から左へ計算されると、

10＊＊（2＊＊3）＝10＊＊8＝100000000

となり、全然違った結果となる。従ってこのようにべき乗が続く場合は、自分の意図している演算が確実に行われるように、カッコできっちりと包んでおく方が無難である。

加減乗除の場合は優先順位もFORTRANと同じで、計算は左から右へ行われる。PL／IではFORTRANと違って、右辺や左辺、あるいは右辺の式の中で、実数型や整数型、複素数型等が入り混じっていてもよい。すなわち混合演算が許されている。これはデータ型の変換機能があるためであり、人間が筆算で計算する場合と同じである。人間が普通に計算を行う場合は実数とか整数の区別は殆んど意識していない。100円をいちいち100．0円などと言う人はいないであろう。STAR TREKに出てくる宇宙人のスポックなら話は別だが……。ところで話を元に戻して、右辺の計算の結果は左辺のデータの型に合わされる。例えば、

K＝12．3＋5．8；

では、K＝18となる。

A＝28－11；

では、A＝17．0となる。この程度の、右辺の結果を左辺の型に合わせる変換はJISを越えたFORTRANでも行われている。

四則演算で注意すべきことが1つある。例えば、FORTRANで、

M＝N／２＊２

という代入文を考えよう。今、N＝３ならば、

M＝（３÷２）×２

＝ｔｒｕｎｃ（１．５）×２

＝１×２

＝２

と計算される。ここで、ｔｒｕｎｃは小数以下の切り捨てを意味する。これがＰＬ／Ｉでは、

M＝（３÷２）×２

＝１．５×２

＝３

と計算される。つまり、同じ式でも違った値になる。ＰＬ／Ｉでは途中のｔｒｕｎｃが行われない。ＦＯＲＴＲＡＮではこの場合、右辺のオペランド（３とか２とかの計算される数値）が整数なので結果も整数であり、全て整数演算が行われる。すなわち、まず３÷２を行い、その結果の１．５も整数化して１としてから２をかけて、結果は２となる。ところがＰＬ／Ｉでは混合演算が許されているので、結果が整数であるなしにかかわらず、右辺の計算は実数で行われる。従って、３÷２の結果の１．５に２をかけて結果は３．０すなわち３となる。

ＦＯＲＴＲＡＮでは左辺に変数名（又は配列要素名）は１つしか書けなかったが、ＰＬ／Ｉでは２つ以上書いてもよい。すなわち、

Ａ，Ｂ，Ｃ＝Ｘ；

というような書き方ができる。これは、

Ａ＝Ｘ；

Ｂ＝Ｘ；

Ｃ＝Ｘ；

と書いたのと同じである。

ＦＯＲＴＲＡＮでは例えば次のような、

Ａ－－－Ｂ

という式は許されない。しかしＰＬ／Ｉではこれは、

Ａ－（－（－Ｂ））

と同じと解釈される。すなわち、＋や－の演算子がくっついて続いていると、一番左のものが演算子（二項演算子）と見なされ、それ以外は正負の符号（単項演算子）と見なされる。次のように＊や／とくっついた書き方も許される。

Ａ／－Ｂ　，　Ａ＊＊－Ｂ

これらはそれぞれ、A／（－B），A＊＊（－B）の意味となる。従って、

A－＊B ，　A＋／B

などは－や＋が符号ではなくて二項演算子となり、＊や／が意味を失うので誤りとなる。

配列名どうしや、配列名とスカラとの加減乗除は配列要素毎の演算を表わす。例えば、3つの配列A（2，3），B（2，3），C（2，3）について、PL／Iでは、

A＝B＋C；

と書くと、FORTRANの、

```
      DO  1  I＝1，2
      DO  1  J＝1，3
1     A（I，J）＝B（I，J）＋C（I，J）
```

に相当する。これらのことは第3章にも書いたので、そちらを参照されたい。

PL／Iでは、

A＝B＝C；

という文は誤りではない。この結果は、B＝Cが真ならば、Aは '1' Bとなり、B＝Cが偽ならば、Aは '0' Bとなる。すなわち、左の＝は代入記号であり、右の＝は比較演算子である。もしこれをFORTRANで書くと、

A＝B．EQ．C

となる。論理型に慣れていない人のためにもっと説明的に書くと、

```
IF（B．EQ．C）　A＝．TRUE．
IF（B．NE．C）　A＝．FALSE．
```

となる。

PL／Iでは、文字型としての数字の列を算術演算できる。例えば、

```
DCL  M  CHAR（4）；
M＝'1980'；
N＝M＋20；
```

とすると、Nは2000となる。これもデータ型の変換機能により文字型が算術型に変換されたわけである。文字型としての空白が算術型に変換されると0になる。このようなことはFORTRANでは不可能である。

以上述べたようなことがPL／Iの代入文の特徴である。

第１０章　メインプログラムとサブプログラム

（１０－１）メインプログラム

ＦＯＲＴＲＡＮではサブプログラムは、ＳＵＢＲＯＵＴＩＮＥ文、ＦＵＮＣＴＩＯＮ文、ＢＬＯＣＫ　ＤＡＴＡ文のいずれかで始まるが、そうでなくていきなり宣言文や実行文で始まるのがメインプログラムである、という区別があった。又、メインプログラムにはプログラマが名前（プログラム名）を付けることができなかった。ＡＣＯＳ－６　ＦＯＲＴＲＡＮでは。。。。。。，ＯＳⅣ／Ｆ４　ＦＯＲＴＲＡＮではＭＡＩＮという名前をシステムが自動的に付けている。

ＰＬ／Ｉではメインプログラムにもプログラマが名前（ＰＬ／Ｉでは手続き名や外部名と呼ばれる）を付けなければならない。手続き名の長さはＡＣＯＳ－６ではＴＹＰＥ（Ａ），（ＡＢ）共に６文字以内、ＦＡＣＯＭ　ＯＳⅣでは７文字以内である。だからＦＯＲＴＲＡＮのサブルーチン名と同じ６文字以内と覚えておけばよい。

メインプログラムは、例えば名前をＲＥＩＤＡＩとすると、

ＲＥＩＤＡＩ：ＰＲＯＣ　ＯＰＴＩＯＮＳ（ＭＡＩＮ）；

という文（プロシージャ文）から始まる。ＰＲＯＣはＰＲＯＣＥＤＵＲＥの省略型である。手続き名はこのようにＰＲＯＣの前に書き、手続き名とＰＲＯＣの間に：（コロンであり、セミコロンではないことに注意！）を書く。ＯＰＴＩＯＮＳ（ＭＡＩＮ）という語句がメインプログラムであることを表わしている。メインプログラムの最後は、

ＥＮＤ　ＲＥＩＤＡＩ；

と、ＥＮＤの後に空白を置いて手続き名を書き、；で終る。ＦＯＲＴＲＡＮのようにＳＴＯＰ文は不要である。（書いてもよいが書く必要はない）。

（１０－２）サブルーチン

＜ａ＞パラメータのあるサブルーチン

サブルーチンは、例えば名前をＳＵＢ１，仮引数をＡ，Ｂ，Ｃとすると、

ＳＵＢ１：ＰＲＯＣ（Ａ，Ｂ，Ｃ）；

という文で始まる。すなわち、メインプログラムのときと比べると、ＯＰＴＩＯＮＳ（ＭＡＩＮ）という語句がなくなって、代りにカッコで包まれた仮引数が続く。最後はやはりメインプログラムのときと同様に、

ＥＮＤ　ＳＵＢ１；

となる。ＥＮＤの直前のＲＥＴＵＲＮは書いても書かなくてもよい。しかしＥＮＤの直前以外の場所でＲＥＴＵＲＮさせたいときは当然書かなければならない。

このサブルーチンを引用するにはＦＯＲＴＲＡＮと全く同じで、ＣＡＬＬ文を使えばよい。例え

ば、実引数をX，Y，Zとすると、

CALL SUB1（X, Y, Z）；

とする。但し、FORTRANのときと同じように仮引数A，B，Cと、実引数X，Y，Zの個数、順序、データの型は必ず一致させておかなければならない。特にデータの型は一致していなくてもエラーにならず、一見うまくいったかのように見えて実はオカシナ結果になる場合があるので気を付けなければならない。

サブプログラム（サブルーチンや関数）を引用する方のプログラムでは、

DCL SUB1 ENTRY；

というように、SUB1はサブプログラム名でありますよというENTRY宣言をしておかなければならない。これはFORTRANとは異なる点である。

ところでFORTRANでの仮引数、実引数という言葉はそれぞれ英語の、dummy argument，actual argument の和訳であるが、PL／Iの場合、英語では仮引数に相当するのはparameter，実引数に相当するのはargumentとなっている。従って大抵のPL／Iのマニュアル類では、FORTRANのように仮引数と実引数という言葉を使わないで、英語のまま、パラメータとアーギュメントとして使っている。従って本稿でも以降はパラメータとアーギュメントを使う。すなわち、サブルーチンの頭のPROC文に書くのがパラメータ、メイン（とは限らないが、サブルーチンを呼ぶ方のプログラム）のCALL文に書くのがアーギュメントである。

余談だがPL／Iではdummy argument は仮アーギュメントと訳され、全く別の意味に用いられている。これはパラメータとアーギュメントのデータの型が一致しないとき、両者の橋渡し的役目をする記憶域のことであるが、話が複雑になるので省略する。

＜b＞パラメータのないサブルーチン

仮引数すなわちパラメータのないサブルーチンのときは、

SUB1：PROC；

とすればよい。パラメータのないサブルーチンというのはFORTRANではCOMMON文によってデータの受け渡しを行う場合であるが、PL／Iでこれに相当するのは、EXTERNAL宣言である。

FORTRANで、例えばメインプログラムで、

COMMON／BLK／X, Y, Z

CALL SUB1

となっており（BLKは共通領域名）、サブルーチンで、

SUBROUTINE SUB1

COMMON／BLK／A，B，C

となっているのを、ＰＬ／Ｉで書くと、メインプログラムで、

```
DCL （X，Y，Z） EXT；
DCL SUB1 ENTRY；
CALL SUB1；
```

となり、サブルーチンで、

```
SUB1：PROC；
DCL （X，Y，Z） EXT；
```

となる。ＥＸＴはＥＸＴＥＲＮＡＬの省略型である。ここで注意すべきことは、ＦＯＲＴＲＡＮではメインで、Ｘ，Ｙ，Ｚという名の変数がサブでは、Ａ，Ｂ，Ｃと、違った名前になっていても、ＸがＡに、ＹがＢに、ＺがＣに順番に対応付けられた。しかしＰＬ／Ｉでは違った名前になることは許されない。（パラメータとアーギュメントを用いる場合は勿論違った名前になっていてもよい）。その代り、並ぶ順番は違っていてもよい。つまり今の例のサブルーチンで、

```
SUB1：PROC；
DCL （Y，Z，X） EXT；
```

となっていてもよい。（パラメータとアーギュメントを用いる場合は勿論並ぶ順番が違ってはダメである）。

上述のＰＬ／ＩのＥＸＴＥＲＮＡＬ宣言はこの場合、ＦＯＲＴＲＡＮのＥＸＴＥＲＮＡＬ文とは全く意味が異なり、似ても似つかないものなので要注意。なお、１つのＰＲＯＣ文に書けるパラメータの個数はＴＹＰＥ（Ａ），（ＡＢ）共に１２８個となっており、十分すぎるであろう。

＜ｃ＞再帰的呼び出し

サブプログラムの再帰的呼び出しについては第３章で階乗計算を例にとり説明した。すなわち、サブプログラムが自分自身を呼び出すことである。この再帰的呼び出しを行うには、プロシージャ文で、

```
SUB1：PROC（A，B，C） RECURSIVE；
```

と書かなければならない。但し、ＴＹＰＥ（Ａ），（ＡＢ）では共に、サブプログラムは全て再帰的であると設定されているので、ＲＥＣＵＲＳＩＶＥと書いても書かなくても再帰的呼び出しができる。ＯＳⅣでは書かなければ再帰的呼び出しはできない。

＜ｄ＞整合配列

ＦＯＲＴＲＡＮにおける整合配列、例えばメインで、

```
DIMENSION A（2，3）
CALL SUB1（A，2，3）
```

となっており、サブで、

```
SUBROUTINE SUB1 (A, M, N)
DIMENSION A (M, N)
```

となっているのを、そのままPL／Iに書き直して、メインで、

```
DCL A (2, 3) ;
DCL SUB1 ENTRY;
CALL SUB1 (A, 2, 3) ;
```

サブで、

```
SUB1 : PROC (A, M, N) ;
DCL A (M, N) ;
```

とするのは大きな誤りである。PL／Iでは整合配列という用語はない。しかしこれと同じことはできる。次のようにすればよい。メイントで、

```
DCL A (2, 3) ;
DCL SUB1 ENTRY;
CALL SUB1 (A) ;
```

サブで、

```
SUB1 : PROC (A) ;
DCL A (*, *) ;
```

とする。すなわちサブプログラム中にある、大きさが可変の配列の大きさは、＊（星印）で表わせばよい。これはむしろ整合配列よりも便利である。

（10－3）関　数

関数についてもサブルーチンと大体同じことが言えよう。ただ少し異なる点があるので説明しよう。FORTRANの関数の例を1つ考えよう。メインで、

```
A＝1．0
B＝2．0
X＝FC1 (A, B)
```

関数で

```
FUNCTION FC1 (A, B)
FC1＝A＊＊2＋B＊＊2
RETURN
END
```

としよう。これをPL／Iで書き直すと、メインでは、

```
DCL FC1 ENTRY RETURNS(FLOAT DEC);
A=1.0;
B=2.0;
X=FC1(A, B);
```

関数では、

```
FC1:PROC(A, B) RETURNS(FLOAT DEC);
RETURN(A**2+B**2);
END FC1;
```

となる。すなわち、FORTRANの関数では最後に

関数名＝返される値

としたのに対し、PL／Iの関数では、

```
RETURN(返される値);
```

となる。従って必ずRETURN文が要る。又、関数の場合はサブルーチンのときと違って、PROC文にRETURNS属性を書かなければならない。前述のRETURNと違って、RETURNSと、Sが付いているので、混同しないようにしよう。ここには返される値のデータの型をRETURNSの後のカッコ内に書くのである。又、同時にメインのENTRYの後にも同じRETURNS属性を書かなければならない。このことは、FORTRANでも返される値（関数値）のデータの型によって、

```
INTEGER FUNCTION ○○○
```

とか、

```
REAL FUNCTION ○○○
```

等と書くことに相当する。又、FORTRANの場合でも、関数名がI－Nルール（暗黙の型宣言）に従うときはこのように書く必要はなく、ただ、

```
FUNCTION ○○○
```

と書くだけでよい。これと同様にPL／Iでも関数名がI－Nルールに従うときはRETURNS属性は省略してもよい。但しTYPE（A）ではI－Nルールがないので RETURNS属性は省略できない。

例として正整数Nの階乗を求める関数KAIJOを示そう。

```
KAIJO:PROC(N) RECURSIVE;
IF N=1 THEN RETURN(N);
       ELSE N=N*KAIJO(N-1);
RETURN(N);
```

END KAIJO;

関数名KAIJOはI－Nルールに従って整数型となるのでRETURNS属性は省略している。書くとすれば、

KAIJO：PROC（N） RECURSIVE RETURNS（FIXED BIN）；

となる。再帰的手続きであることを示すRECURSIVEは前述したようにTYPE（A），（AB）では書く必要はない。

（10－4）外部手続きと内部手続き

FORTRANのサブプログラムにはサブルーチンと関数の他に、BLOCK DATA文で始まる初期値設定副プログラムというのがあるが、PL／Iには存在しないし、又その必要もない。

PL／Iでは1つのプログラム単位を、ブロックと呼ぶ。今まで述べたメインプログラムも1ブロックであり、サブプログラムも1ブロックである。これだけでは別にFORTRANと変りがないが、PL／Iではブロックの入れ子、すなわちブロックの中にブロックを含むことができる。外側のブロックを外部ブロック、内側に含まれているブロックを内部ブロックという。内部ブロックの中に更に孫、ひ孫という風に内部ブロックが含まれていてもよい。たとえて言えば、カバンという外部ブロックの中に弁当箱という内部ブロックが入っており、更にその弁当箱の中にオカズ入れという内部ブロックが入っていると考えればよい。この場合、弁当箱をカバンから取り出して外に並べればカバンと弁当箱は共に外部ブロックであり、オカズ入れだけが弁当箱の内部ブロックとなる。

内部ブロックの形のプログラムを内部手続き、外部ブロックの形のプログラムを外部手続きと呼ぶ。

今まで述べたサブルーチン、関数は、FORTRANとの対応にポイントを置いていたので全て外部手続きであった。しかし、サブルーチン、関数は内部手続きの形で書くこともできる。（メインプログラムは外部手続きでなければならない）。FORTRANに文関数というのがある。これは非常に簡単な形の内部手続き関数であると考えることもできる。内部手続きのサブルーチンに類するものはFORTRANには存在しないが、もしあるとして文関数から類推してこじつけの名前を付けるとすれば、“文サブルーチン”（?）とでもなろうか。

いずれにしてもPL／Iの内部手続きはFORTRANの文関数のような簡単なものではなく、外部手続きと全く対等である。従って、任意のサブルーチンや関数は外部手続き、内部手続き、どちらの形式ででも書くことができる。しかし、FORTRANに慣れた者にとっては外部手続きの方が書き易いようである。しかし計算時間に関しては内部手続きの方が速いといえる。

例としてカードから球の半径を読み込み、その体積を求めて半径と共に印刷するプログラムを外

部手続きと内部手続きの両方で書いてみよう。

まず外部手続きの場合である。

```
/*GAIBU TETSUZUKI NI YORU TAISEKI KEISAN*/
REI1:PROC OPTIONS (MAIN);
 DCL VLM ENTRY;
 GET LIST(X);
 CALL VLM(X, Y);
 PUT LIST('HANKEI=', X, 'TAISEKI=', Y);
END REI1;
VLM:PROC(R, V);
 V=4*3.14159*R**3/3;
END VLM;
```

次に内部手続きの場合である。

```
/*NAIBU TETSUZUKI NI YORU TAISEKI KEISAN*/
REI2:PROC OPTIONS(MAIN);
 GET LIST(X);
 CALL VLM(X, Y);
 PUT LIST('HANKEI= ', X, 'TAISEKI=', Y);
VLM:PROC(R, V);
 V=4*3.14159*R**3/3;
END VLM;
END REI2;
```

この例から分かるように内部手続きの場合にはＥＮＴＲＹ宣言は要らない。又、内部手続きは、この例では含まれる手続きの終りに入っているが、冒頭でもよいし、どこに入れてもよい。

ところで、内部ブロックには、上述したようにＰＲＯＣ文で始まる内部手続きの他に、ＢＥＧＩＮブロックと呼ばれるものがある。これはサブルーチンや関数のように、どこか他の箇所から引用されるという性質のものではなく、プログラムの流れがそこへ来てそのまま通り過ぎてゆくというもので、さしたる意味のあるものではない。強いて言えばＦＯＲＴＲＡＮのオープンサブルーチン（普通のサブルーチンはクローズドサブルーチンと呼ばれる）に類するものである。

（１０－５）組込み関数

ＦＯＲＴＲＡＮではコンパイラに備わっている機能として、組込み関数と基本外部関数がある。

両者の違いは前者の方がやや算術的で、後者がやや数学的という程度である。ＰＬ／Ｉではこのような区別はなく、全て組込み関数と呼ばれている。ＦＯＲＴＲＡＮでは、例えば最小値を求める関数でも、ＭＩＮ０，ＭＩＮ１，ＡＭＩＮ０，ＡＭＩＮ１のように引数や関数値の型により種々変えて使わねばならないが、ＰＬ／Ｉではデータ型の変換機能があるので、ＭＩＮという１つの関数ですませている。ＰＬ／Ｉでは多くの組込み関数が用意されているが、それらの１つ１つをここで説明する紙数はないので、詳しくはマニュアル等で御覧頂きたい。

第１１章　初期値の設定

（１１－１）数値型の場合

ＦＯＲＴＲＡＮで変数又は配列に初期値を設定するには、ＤＡＴＡ文を用いる。但し、名前付きＣＯＭＭＯＮ文に出てくるものに対しては、ＢＬＯＣＫ　ＤＡＴＡ文で始まる初期値設定副プログラムを使わなければならない。前述したようにＰＬ／Ｉではそのような区別はなく、全て宣言文において初期値の設定ができる。

例えば、ＦＯＲＴＲＡＮで、

```
DATA  A／１．２３／
```

なる文をＰＬ／Ｉで書くと、

```
DCL  A  INIT（１．２３）；
```

となる。ＩＮＩＴはＩＮＩＴＩＡＬの省略型である。この場合、Ａのデータ型はＩ－Ｎルールに従うものとして省略してあるが、書くとすれば、

```
DCL  A  FLOAT  DEC  INIT（１．２３）；
```

というようにＩＮＩＴを一番後に書く。変数が２つ以上ある場合も同様で、ＦＯＲＴＲＡＮで、

```
DATA  A／１．０／，K／５／，M／４HABCD／
```

なる文をＰＬ／Ｉで書くと、

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

となる。

配列の場合、例えばＦＯＲＴＲＡＮで、

```
DIMENSION  X（２０）
DATA  X／２０＊０．０／
```

なる文をＰＬ／Ｉで書くと、

```
DCL  X（２０）  INIT（（２０）０．０）；
```

となる。すなわち、ＦＯＲＴＲＡＮでの、

```
２０＊  ０．０
```

が、ＰＬ／Ｉでは、

（２０）０．０

となる。次のようなＦＯＲＴＲＡＮの例、

```
INTEGER F(20,20)
DATA F/400*0/
```

をＰＬ／Ｉで書くと、

```
DCL F(20,20) FIXED BIN INIT((400)0);
```

となる。

（１１－２）文字型の場合

文字型の場合、少し気を付けねばならない点がある。例えば文字列の長さ５０の文字型変数Ｍに、５０個の＋（プラス記号）を入れたい場合は、

```
DCL M CHAR(50) INIT ((50)'+');
```

とすればよい。ここでＭは配列ではないことに注意されたい。もし、Ｍが配列Ｍ（５０）で、各配列要素の文字列は長さ３とすれば、

```
DCL M(50) CHAR(3) INIT((50)(3)'+');
```

と書かねばならない。あるいは次に示す２つのいずれかでもよい。ＩＮＩＴの部分だけを書くと、

```
INIT((50)(1)'+++')
INIT((50)('+++'))
```

となる。すなわち、全部で３通りの書き方がある。少しややこしいので一般的に書くと、要素数ｋの文字型の配列Ｍ（ｋ）があり、各要素の文字型としての長さはℓのとき、各要素にℓ個の文字Ｘを入れたい場合、

```
DCL M(k) CHAR(ℓ)
```

のあと、次の３通りの書き方がある。

```
INIT((k)(ℓ)'X');
```

```
INIT((k)(1)'XX……X');   ('XX……X' : ℓ個)
```

```
INIT((k)('XX……X'));    ('XX……X' : ℓ個)
```

以上が文字型配列の場合の注意点である。

（１１－３）やや複雑な場合

ＦＯＲＴＲＡＮのＤＡＴＡ文で、ＤＯ型並びを伴なうものがある。例えば、

```
DIMENSION A(15)
DATA (A(I), I=1, 10), A(11), (A(J), J=12, 15)
/10*1. 0, 2. 0, 4*3. 0/
```

ではＡ（１）からＡ（１０）に１．０，Ａ（１１）に２．０，Ａ（１２）からＡ（１５）に３．０が入る。これをＰＬ／Ｉで書くと、

```
DCL A(15) INIT((10) 1. 0, 2. 0, (4) 3. 0);
```

となる。ＦＯＲＴＲＡＮで、

```
DIMENSION B(100)
DATA (B(I), I=51, 100)/50*1. 23/
```

とすれば、Ｂ（１）からＢ（５０）は初期値設定されず、Ｂ（５１）からＢ（１００）には１．２３が入る。これをＰＬ／Ｉで書くと、

```
DCL B(100) INIT((50) *, (50) 1. 23);
```

となる。すなわち、初期値設定しないときは＊（星印）で表わす。

（１１－４）ＡＵＴＯＭＡＴＩＣとＳＴＡＴＩＣ

ＦＯＲＴＲＡＮのＤＡＴＡ文で注意すべきことは、サブプログラムの中のＤＡＴＡ文は初回に引用されたときのみ機能して、２回目以降の引用では機能しないことである。例えば、

```
SUBROUTINE SUB1(X)
DATA P/3. 14/
P=P**X
X=X*P
RETURN
END
```

というサブルーチンでは、初回にＣＡＬＬされたときはＤＡＴＡ文が機能してＰ＝３．１４であるが、２回目以降はＤＡＴＡ文が機能せず、Ｐの値は変化しているので３．１４ではない。２回目以降もＰを初期値設定したければＤＡＴＡ文ではなく代入文を使うしかない。（上例はプログラムとしては多少不自然だが、言わんとする所は分かって頂けるだろう）。

ＰＬ／Ｉではサブプログラムにおける初期値設定をＦＯＲＴＲＡＮのように初回の引用時のみ行うか、２回目以降も行うかを選択できる。例えばサブルーチンＳＵＢ１において変数Ｘを初回のみ０．０に初期化したければ、

```
SUB1：PROC（A, B, C）；
DCL X STATIC INIT（0．0）；
```

という風にＳＴＡＴＩＣという属性を付け加えればよい。この逆に変数Ｘを引用の度に毎回０．０に初期化したければ、ＤＣＬ文を、

```
DCL X AUTO INIT（0．0）；
```

という風にＡＵＴＯという属性を付け加えればよい。ＡＵＴＯはＡＵＴＯＭＡＴＩＣの省略型である。

ＳＴＡＴＩＣ属性はＸという変数のメモリがそのジョブの開始から終了まで確保されたままの状態で続くことを表わす。これはＦＯＲＴＲＡＮと同じである。ＡＵＴＯ属性はサブルーチンＳＵＢ１が引用されたときだけＸという変数のメモリが確保され、ＳＵＢ１の引用が終るとメモリは解放される。再びＳＵＢ１が引用されると又新たにＸのメモリが確保されるという具合に、ＳＵＢ１が引用される度にメモリの確保と解放が繰り返される。従って当然初期値設定も毎回行われるわけである。たとえてみれば、ＡＵＴＯ属性の変数は、仕事の度に雇われるアルバイトで、雇う度毎に人（メモリ）が変るので、説明（初期値設定）をしなければならないのであるが、ＳＴＡＴＩＣ属性の変数は本雇いの正社員で、最初だけ説明（初期値設定）をすればよい。

メモリの使用効率から見ればＡＵＴＯ属性では使用されていないメモリは常に解放されているので、ＳＴＡＴＩＣ属性よりも効率が良い。従って、ＡＵＴＯともＳＴＡＴＩＣとも宣言されていない変数（配列も含む）に対しては省略時解釈としてＡＵＴＯと見なされる。但し、前述のＥＸＴＥＲＮＡＬを宣言している場合にはＳＴＡＴＩＣと見なされる。ＥＸＴＥＲＮＡＬと宣言している限りは当然ＳＴＡＴＩＣでなければならないことは、ＥＸＴＥＲＮＡＬがＦＯＲＴＲＡＮのＣＯＭＭＯＮ文に相当していることを考えれば容易に納得できるだろう。ＥＸＴＥＲＮＡＬでかつＡＵＴＯというのは矛盾している。ＥＸＴＥＲＮＡＬの逆のＩＮＴＥＲＮＡＬという宣言がある。これは、その手続き（ブロック）の内部でだけ通用する名前であることを意味する。これはたとえて言えば、家族やグループのいわゆる身内でだけ通用する愛称やアダ名のようなもので、外部の人には通用しない名前である。ＩＮＴＥＲＮＡＬと宣言して、ＳＴＡＴＩＣともＡＵＴＯとも宣言しない場合はＡＵＴＯと見なされる。しかし、ＩＮＴＥＲＮＡＬでかつＳＴＡＴＩＣという宣言も可能である。

以上が特にサブプログラムにおける初期値設定の注意事項であるが、やや古いＰＬ／ＩではＳＴＡＴＩＣでなければＩＮＩＴによる初期値設定ができないものもある。勿論、ＴＹＰＥ（Ａ），（ＡＢ），ＯＳⅣではそういう心配はない。

（１１－５）ＩＮＩＴ　ＣＡＬＬ

ＰＬ／Ｉには以上の他にＣＡＬＬ文による初期値設定法がある。これはサブルーチンによって初

期値設定する方法である。

```
DCL X(50) INIT CALL SUB2(A, B);
```

この例ではSUB2は配列Xの初期値設定を行うサブルーチンである。A, BはSUB2へ渡すアーギュメントで、これ以前に値が定義されているものとする。この例でXがアーギュメントに入っていないのはSUB2が内部手続きであるゆえ、SUB2の中でXを再宣言しない限り、メインのX(50)がそのままSUB2の中でも通用しているからである。これを、INIT CALLによる初期値設定という。このINIT CALLは、TYPE(AB)とOSⅣでは使用できるが、TYPE(A)では使用できない。又、INIT CALLは、STATIC属性をもつ変数には適用できないから要注意である。

(11−6) DEFINED

最後にFORTRANのEQUIVALENCE文に相当するものを説明しよう。

FORTRANで例えば、

```
DIMENSION A(10), B(10)
EQUIVALENCE (A(1), B(1))
```

という文をPL/Iで書くと、

```
DCL A(10);
DCL B(10) DEFINED A;
```

となる。すなわち、配列Aと同じ記憶領域をさす別の名前である。1つのメモリを2つ以上の名前で呼びたいときに用いられる。人間で言えば本名と芸名のようなものであろう(?)。

上述の例は分かり易くするため2つのDCL文で書いたが、勿論1つにまとめて、

```
DCL A(10), B(10) DEFINED A;
```

としてもよい。なお、FORTRANのEQUIVALENCEでは、AとBは対等であったが、PL/Iの場合、この文例ではAが主、Bが従の関係となり、AはINITによって初期値設定できるがBはできない。

ここに述べた例はDEFINED属性の中の単純定義と呼ばれるものである。DEFINED属性の中にはこの他に、iSUB定義, 列重ね型定義というのがあるがここでは省略する。

今回は代入文に始まり、メインプログラムとサブプログラム、そして初期値設定法について解説した。段々と話が複雑になっていくのでサーッと読んだだけでは分かりにくい点もあるかも知れないが、繰り返し読んで頂けたらお分かりになるのではないかと希望的観測をしている。それでも分からないときは文書等で御質問下されば、できる限りの御答えはしたいと思っている。しかし筆者

自身も一体どれ程分かっているのか疑問である。それ程ＰＬ／Ｉは大きな言語であり、それゆえ重い言語とけなす人もいる。その意味では確かにＦＯＲＴＲＡＮは軽い言語であるというメリットはある。しかしＦＯＲＴＲＡＮには、できないことや制限が多すぎる。プログラムの初心者はＦＯＲＴＲＡＮでできることがそのままコンピュータでできることの全てであると思いがちなものである。そういう人達に本稿を通じて、コンピュータではＦＯＲＴＲＡＮでできること以外に、例えば再帰的呼び出しやビット処理、あるいは人間のする筆算と同じような混合演算ができるということを知ってもらえれば幸いである。

次回はＤＯ文、ＩＦ文、ＯＮ文、ファイル入出力文等について解説する予定である。

（プログラム相談員）