



Title	FORTTRAN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	<a href="https://hdl.handle.net/11094/65456">https://hdl.handle.net/11094/65456</a>
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るもの、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ブ(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 OSM 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、

$$\begin{aligned} M &= (3 \div 2) \times 2 \\ &= \text{trunc}(1.5) \times 2 \\ &= 1 \times 2 \\ &= 2 \end{aligned}$$

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、

$$\begin{aligned} M &= (3 \div 2) \times 2 \\ &= 1.5 \times 2 \\ &= 3 \end{aligned}$$

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 $3 \div 2$ 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、 $3 \div 2$ 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、

$$\begin{aligned} A &= X; \\ B &= X; \\ C &= X; \end{aligned}$$

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。

$$A / -B, \quad A * * -B$$

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。．．．．．，OSV/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 OSV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、代わ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、両者の橋渡しの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。OS IV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 〇〇〇

```

とか、

```

REAL FUNCTION 〇〇〇

```

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 〇〇〇

```

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られることもできる。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を一番後を書く。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 $\ell$ のとき、各要素に $\ell$ 個の文字Xを入れたい場合、

DCL M(k) CHAR( $\ell$ )

のあと、次の3通りの書き方がある。

INIT((k) ( $\ell$ ) 'X');

INIT((k) (1)  $\overbrace{'XX\cdots X'}^{\ell\text{個}}'$ );

INIT((k) ( $\overbrace{'XX\cdots X'}^{\ell\text{個}}'$ ));

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を初回のみの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が続くことを表わす。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)が変わるので、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属性よりも効率がよい。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SUB 2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SUB 2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SUB 2が内部手続きであるゆえ、SUB 2の中でXを再宣言しない限り、メインのX(50)がそのままSUB 2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つ以上の名前と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、i SUB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)