[03_05]九州大学大型計算機センター広報 : 3(5)

https://doi.org/10.15017/1467970

出版情報：九州大学大型計算機センター広報. 3 (5), pp.1-42, 1970-10-12. 九州大学大型計算機センター
バージョン：
権利関係：

FORTRANを基礎にしたプログラム言語について（I）

牛島和夫*

1．はじめに

FORTRANを基礎にした言語はいわゆる汎用のものだけでもSLIP[1]、DYSTAL[2]、FLIP[3]、GASP[4]その他多数開発されている。ここでは、そのような言語が開発されて利用されている背景、開発上の問題点、利用上の問題点などを探ってみよう。

2．FORTRANの発展と標準化

1940年代の半ばに、数値計算を高速かつ自動的に行なうという要求を満たすために、電子計算機が誕生した。当時、計算機と人間をつなぐ言語は、機械語に限定されていたために、機械語の壁を突破してでも計算機をぜひ使わなければならないという人だけに、利用者は限定されていた。ところでたくさんの問題をプログラムしてゆくと、問題の本質には関係なく、手続きとして共通な機械語の組合わせがあらわれてくる。そのような手続きを機械語に翻訳する仕事は計算機が自動的にできるはずである。

そういった考えのもとに、プログラムを数式と英語に似た言葉で記述し、自動的に機械語に翻訳する自動プログラム用言語が1950年代に入っていくつか開発されていった。その中の1つとしてFORTRANはIBM－704という特定の計算機のために仕様が作られて（1954年）開発された。FORTRAN語の目的をそのマニュアルから引用してみると、

〝FORTRAN語は数値計算のあらゆる問題を表現できることを意図したものである。特に多くの公式と変数を含む問題を容易に取扱い、各変数は3個までの独立した添字を持つことができる。しかし機械語が数値的な意味ではなく論理的な意味をもつ場合にはFORTRAN語は不満足であり、このような問題はまったく表現できないこともあるけれども、FORTRAN語で直接表現できない論理操作もライブラリルーチンを組み込めるようにしておくことによって可能になる。〟（この引用の訳はJIS規格票[5] P45による）

これらの自動プログラム用言語の開発と発展によって、問題をもった人が自分の問題を直接プログラム言語で表現して計算機を利用することができるようになり、利用者の増大に大きな役割を果した。このことは、計算センターのような計算機利用方式にも大きな影響をもたらし、いわゆるopen shop方式かclosed shop方式かの議論が盛んにおこった。しかし、現在ではご承知のように、普通の科学技術計算では、九大の大型計算機センターでも実施しているように、プログラムは問題をもった人

＊九州大学工学部通信工学科

がopenに、操作はセンター専任者がclosedで実施しているのである。

さて、年代をもとにもどしてIBM704 FORTRAN 以後開発され、市場に出される各計算機に対しても、各社が（Sales promotion上からも）FORTRANコンパイラをつけるようになってきた。その場合、各機種の性能上の特色を十分発揮できるような言語仕様が独自の立場で加えられていったので、１００種近くのFORTRAN言語ができあがってしまった。同じ会社製のものでも、機種によって異っているものまでできてしまったのである。計算機の進歩と適用業務の拡大によって次々と新しいシステムを採用してゆく際に、旧いシステムで利用していたプログラムが新しいシステムでそのまゝ動かなければ都合が悪い。FORTRANで書かれたプログラムに限ってみても、新しい機械に同じプログラムをかけて必ずしもうまくゆくとは限らない。実際、我々の中にも大型計算機に限っていえば、HITAC 5020からFACOM230－60へのプログラムのうつしかえにかなりの時間をとられたものもあったのである。これはreprogramming，transferability　として計算機の関係者の間でも大きな問題の１つとなっている。

このようにして、FORTRAN語の互換性が重大な問題になり、1962年米国のＡＳＡ（標準局、現在はＡＮＳＩ）でFORTRANの標準化の作業が開始されて、FORTRAN、Basic FORTRANの２つの言語水準としてまとめられた。[6]　その後、ヨーロッパからこの中間の水準の案が出されて、ISO（国際標準化機構）ではこの３水準をFORTRAN語の推薦規格案とした（1965年）。我国でもこれに若干の修正を行なって、1967年JIS FORTRAN水準7000、5000、3000が制定された。（この順でISOの案と対応している。FACOM230－60 FORTRANは水準7000を含んでいるといってよい。）

JISの制定後は、各所で作られるFORTRANはこの規格を１つの標準に作成され、各社のFORTRANの解説書もJIS規格との比較を明確にしているので、言語の互換性の上でこの標準化作業は非常に大きな役割を果したことになる。

3．ＦＯＲＴＲＡＮにおけるデータの構造

前節のIBM 704 FORTRANの引用文に述べられているように、FORTRANで処理の対象となるデータの内容は、数値的が論理的である。これはあくまでも内容であって本質的には、１記憶単位に１データ（数値か論理値）ということである。FORTRANのデータの種類としては、定数、単純変数、添字付変数があって、それぞれのデータ間の意味的な関係はプログラムが定めている。一方、構造的な関係は、添字付変数の場合だけにあらわれ、非常に簡単なものでしかない。すなわち、前後関係をあらわすベクトル、前後左右の関係をあらわす行列、それに上下の関係を加えた３次元配列、あとは次元をふやすだけの拡張された配列。数値計算を行っていく上では、この３次元配列までのデータ構造があれば十分であろう。ところが計算機の処理の対象は単なる数値計算だけではないし、数値計算は科学技術計算の非常に重要な部分であるが全部ではないわけである。

４．鎖構造をもったデータの処理

次のような例を考えてみよう。

切符を売る窓口があって、客がそこに並び、順番がきたら目的地を告げて切符を買う。切符を買ったら窓口をはなれる　（図１）。

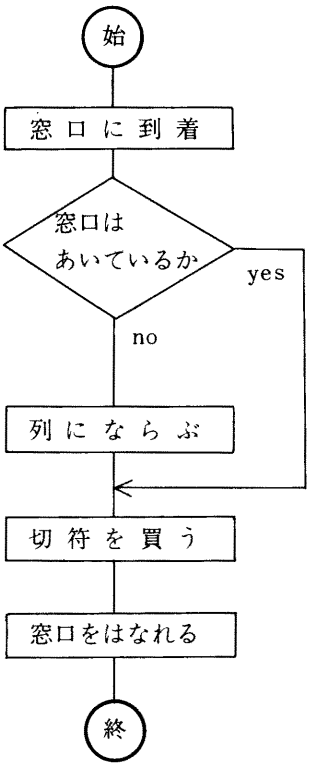


図１

このプロセスを計算機でシミュレートする問題を考えてみる。このシミュレーションを行うために計算機は次のようなものを用意しなければならない。

１）客が窓口に到着するありさまを表現する確率変数の発生機構

２）到着した客の区別とその客の目的地の情報を保存し、到着した順に待ち行列に並べる機構

３）客の目的地に応じて切符を販売する機構

４）プロセス全体の統計をとる機構（行列の長さの最大値、平均値、客の平均待時間、総売上げ、etc．）

ここではシミュレーション技術を解説するのが目的ではなく、データ構造が議論の中心であったから、1)、3)、4)は省略して2)について少し詳しく考えてみることにする。

処理する対象となる情報は客の区別と目的地であったから２次元の整数型配列

```
INTEGER A(100,2)
```

を用意する。今、窓口に３人の客が並んでいるとすると配列Ａの内容は図２のようになる。客が次々と到着するとその情報は、Ａ（４，・）Ａ（５，・），・・・の順に格納されていく。ところで、新しく到着した客の情報を配列Ａのどの場所に格納したらよいかという情報を確保するために

```
INTEGER  TAIL
```

を用意しなければならない。これは、つまり待ち行列の最後の場所を示す変数で図２ではＴＡＩＬ＝３である。そして客が到着する毎に

```
TAIL = TAIL + 1
```

となるべきものである。このようにして 100人の客が窓口に到着すると、もう配列Ａは一杯になるので（ＴＡＩＬ＝ 100となっている）

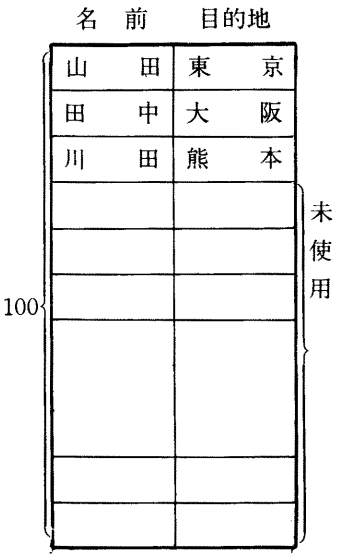


図　２

101人目からの客の情報を格納することができない。ところが、実際に切符を買った客は次々と待行列からはなれてゆく。つまりサービス

を受けた客の情報は不要になるので、A(1, ・), A(2, ・), …の順で配列Aは空いてゆく。したがって、101番目の客はA(1, ・)、102番目の客はA(2, ・)と順送りにすればよい。これは、待ち行列の長さが 100人を越えさえしなければ、いつでも行える。ところで、客をサービスする窓口の側からみると（あるいは待ち行列の長さが 100を越えないようにチェックをするために）、待ち行列の先頭の情報がどこにあるかを知らなければならない。そこで、TAILのほかに

```
INTEGER  HEAD
```

を用意して、サービスが終って客が窓口をはなれるたびに

```
HEAD = HEAD + 1
```

としておけばよい（TAILもHEADもmoduloを100とした加算である。IF(HEAD. EQ. 101) HEAD=1)。結局、待ち行列に並んでいる客の情報は、A(HEAD, ・), A(HEAD＋1, ・), …, A(TAIL, ・)のように一次元的に格納されている（ただし、HEAD≦TAIL≦100の場合）。

さて、実際に待ち行列に並んでいる客が15人とすれば、配列Aの中で使用されている部分は30であるのに、あらかじめDIMENSIONで200個の領域が確保されているので、170個は無駄に遊んでいることになる。そこで客の情報を格納するのに別の方法を考えてみる。さきに述べたようにサービスを受けた客は、窓口をはなれて、配列Aの中から消去されてゆくので、空いた場所を新たに到着した客の情報を格納するために用いることにする。先の方法では、I＋1番目に到着した人の情報は必ずI番目の次に格納されていたのだが、新しい方法で格納すると、到着順序が不明になり、到着順のサービスのシミュレーションができなくなってしまう。そこで到着順序の情報を保存するために、記憶場所を1つ増して

```
INTEGER  A(100,3)
```

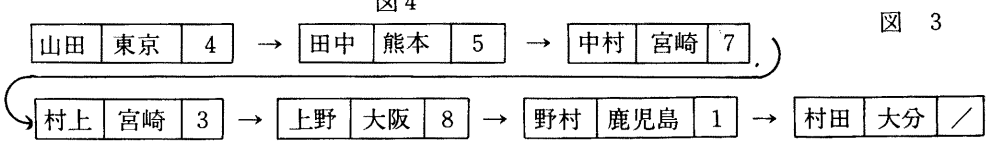
その場所を次はどこという情報の保存のために使う。このようにすると、ある時点で配列Aの中は図3のようになっているであろう。図3を順序通りに分解して図示すると図4のようになって鎖状になるので、このようなデータの構造を**鎖構造**と呼んで、データ構造のもっとも基本的なものの一つである。なお、窓口に並んだ客のサービスのように、最初に到着した客から順にサービスするような方式を**FIFO方式**(first in first out)と呼んでいる。

1	村　田	大　分	
2	山　田	東　京	4
3	上　野	大　阪	8
4	田　中	熊　本	5
5	中　村	宮　崎	7
6	○	○	○
7	村　上	宮　崎	3
8	野　村	鹿児島	1
9	○	○	○
10	○	○	○

HEAD＝2　TAIL＝1

図　3

図4



今述べた方法では、客が去って空いた場所を探す手続きを述べなかったが、空になったものは内容を0にしておいて

```
   DO 1 I = 1, 100
   IF( A(I,1) ) 1,2,1
 1 CONTINUE
        .
        .
        .
 2 A(I,1) = name
   A(I,2) = destination
   A(TAIL,3) = I
   TAIL = I
```

一方、いわゆる未使用領域をあらかじめ鎖でつないでおいて

```
   INTEGER FIRST
   DO 1 I = 1,99
 1 A(I,3) = I + 1
   A(I,100) = 0
   FIRST = 1
```

客が到着すると未使用領域の先頭から3ツ組の記憶単位をとりだして、客の情報を格納する。
ＦＩＲＳＴは未使用領域の先頭を示す変数である。

```
   I = FIRST
   FIRST = A(I,3)
   A(I,1) = name
   A(I,2) = destination
   A(I,3) = 0
   A(TAIL,3) = I
```

一方、客が窓口からはなれると、その情報を格納していた3ツ組領域は未使用領域に返す。未使用領域への返し方は、2通り考えられる。1つは未使用領域の先頭にとりつける方法で、Ｉ番目の3ツ組が不要になったものとすると

```
   A(I,3) = FIRST
   FIRST = I
```

これは未使用領域というのは待ち行列とちがって、必要なときどれがとり出されてもよいので、もっとも新しく未使用領域に入れられたものが最初にとり出されるようになっている。これを**LIFO 方式**（Last In First Out）と呼んでいる。また、このように入れ口ととり出し口が同じような鎖構造を**push down stack**と呼んでいる。

もう１つの方式は、未使用領域を待ち行列と同じように考えて、不要になったＩ番目の３ツ組を未使用領域の最後にとりつける。最後にとりつけるためには未使用領域の最後を示す変数ＬＡＳＴが必要になる（LIFO方式の場合はFIRSTだけで十分であった）。したがって、不要な３ツ組を未使用領域にもどす手続きは

```
A(LAST,3) = I
A(I,3) = 0
LAST = I
```

となる。未使用領域からとり出すときは、LIFO方式と同様に先頭からとり出すのでFIFO方式となり、このような鎖構造を**queue**と呼んでいる。

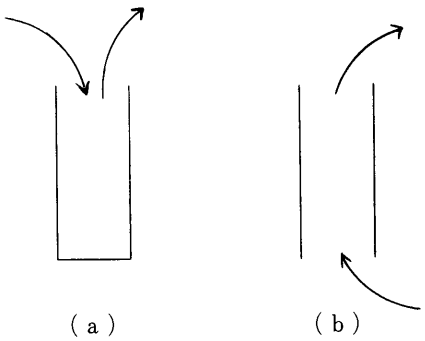


図５ push down stack(ａ)とqueue(ｂ)

このほか鎖構造一般に対するデータの処理として、行列への割込み（データの**挿入**）、あまり待時間が長くなって、急用ができたので帰ってしまった（データの**削除**）、待つ人が交代する（データの**交換**）が基本的なものである。（理解を助けるためこれらの処理をFORTRAN語で記述してみるとよい）。

鎖構造のほかに基本的なデータ構造として**木構造**がある。例えば石油の製品の系統図、生物の進化の木などの情報の相互関係は木構造をなしている。また、FORTRANコンパイラが算術式

Ａ＊Ｂ＋（Ｃ－Ｄ）／Ｅ

を解釈して、機械語の命令を生成する過程で計算機の中では、この数式を木構造に分解する作業が行われる。この算術式の意味は

〔ＡにＢを乗じたものに、ＣからＤを引いたものをＥで除したものを加える〕

と読める。この日本語をそのまゝの順で記号であらわすと

ＡＢ＊ＣＤ－Ｅ／＋

これは、カッコを除いた数式の表現方法で、逆ポーランド記法というものであるが、演算子＋－＊／は、前二つの被演算子に作用する。作用した結果は一かたまりになって被演算子になる。この被演算子と演算子の関係を図式化すれば図６のような木構造が得られる。FORTRAN コンパイラはこのような木構造を計算機の中に作って、それを参照しながら機械語の命令を作り出してゆくわけである。

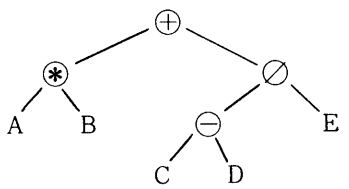


図６ 算術式Ａ＊Ｂ＋(Ｃ－Ｄ)／Ｅの木構造による表現

さて、複雑なデータ構造のうちでも最も簡単な鎖構造をもったデータの処理をFORTRAN語で記述し

てみたが、かなり面倒なことが知れるであろう。このようなデータ処理の立場からFORTRAN語 の限界をあげてみる。

1）鎖構造や木構造など構造をもったデータ処理をFORTRANで行うには繁雑な手続きが必要である。

2）動的な記憶領域の使用が不可能である。

3）データの存在場所を変数名、配列名など名前を通してしか知ることができない。

4）データの記憶の単位が1記憶単位で自由な長さのデータの取扱いはむずかしい。

5．FORTRANの言語の拡張機能

4節で述べたような限界があるにもかゝわらず、1節にあげたような言語がFORTRAN を基礎に開発され、利用されている大きな理由として次の3つがあげられるだろう。

1）FORTRAN はほとんどの機種で使用可能であり、デバッグの利便も一応整っている。さらに、利用者も圧倒的に多いので、例えば計算センターのサービス体制などが好むと好まざるとにかかわらずFORTRAN型になっている。

2）FORTRAN で記述されたプログラムは2節に述べたような多少の互換性を欠いたとしても、高級言語であるから、documentation 、transferability 、maintenance が容易である。

3）FORTRAN語の基本的な語彙に、FUNCTIONによって定義された語彙を追加することによって言語を拡張する機能を有している。

この節では、3）の拡張機能を調べてみよう。次のような例を考えてみよう。

FORTRANにも含まれている加減演算子（+，－）をFUNCTIONであらわしてみる。

```
FUNCTION  ADD(A,B)
ADD=A+B
RETURN
END

FUNCTION  SUB(A,B)
SUB=A-B
RETURN
END
```

これを

```
1 C = ADD(A,B)
2 D = SUB(A,B)
3 E = ADD(C,D)
4 F = ADD(E,C)
```

として3のCとDにそれぞれ1、2を代入すれば

```
3 E = ADD( ADD(A,B), SUB(A,B) )
```

だから、もしC、D、Eの値には関心がなく単にFの値だけを得たいのであれば、関数を重畳的に使用して

```
4 F = ADD( ADD( ADD(A,B), SUB(A,B)), ADD(A,B) )
```

と1 statementで書きあらわすこともできる。このような用法をSakoda[2]は関数の重畳使用（nesting of functions）、Barron[7] はfunctional programmingと呼んでいる。

さて、前節に述べた鎖構造の処理も、挿入、削除、入れ換え、付加といったそれぞれの手続きの集合をそれぞれFUNCTIONにして宣言しておけば、FORTRAN 語にそのような機能が追加されて言語機能が拡張されたと見なすことができるわけである。この拡張は非常に軽微なものであるけれども、もっと系統的、合目的的に行えば、FORTRANを基礎とした新しいプログラム言語ができあがったと考えてよいだろう　（挿入、削除などの処理をFUNCTIONで記述してみよ）。

6．副プログラムのパラメータ

JIS規格票[5]では、副プログラムの実引数として使用してよいものを表1のように規定している（P36）。

副プログラムの引数の性格は、副プログラムの実行に必要なデータを与える入力用パラメータと、副プログラムの実行の結果を呼び出したプログラムに返す出力用パラメータに大きくわけることができる。これはALGOLにおける値とりのパラメータ（call by value）と名前変えのパラメータ（call by name）とにほゞ対応しているといってよい。ＡＬＧＯＬでは、値とりのパラメータは値とりであることを値の部によって宣言しておかなければならないが、FORTRANでは、入力パラメータであることを示す宣言を行う文は存在しないので、コンパイラは、副プログラムに使われている引数が入力用であっても出力用であっても、支障のないように目的プログラムを作り出さなければならない。すなわち、実引数が定数、単純変数、添字付変数のときは、それが割当てられた記憶場所（番地）を副プログラムに引き渡す。また、実引数が配列名のときは、1次元の場合はA(1)、2次元ではB(1,1) などの番地を引き渡す。実引数が式の場合には、式の値を計算した結果を一たん作業用の記憶場所に退避して、その番地を引き渡す。したがって、プログラムした人が、そのパラメータが入力用か出力用かわきまえて利用しないと、とんでもない結果を引き起すこともある。たとえば、実引数が式の場合は、これは、たしかに入力用としか考えられないが、副プログラムの中で、そのパラメータに出力結果を与えたりすると、式の結果が退避されている場所にその値が格納さ

SUBROUTINE	FUNCTION
変数名	変数名
配列要素名	配列要素名
配列名	配列名
その他の式	その他の式
外部手続の名前	外部手続の名前
文字定数	×

表1　副プログラムの実引数

れて、無意味となってしまう。また、同一の引数を入力用にも出力用にも使うことができる。広報Vol 2　No.3（1969－8）P33の例3をもう一度考えてみよう。最初のCALL文でSUBが呼ばれるとき、Nは10という値を持っているが、さきに述べたようにサブルーチンは10という値をもらうのではなくて10という定数の記憶場所が引き渡される。したがって、サブルーチンの中で

```
N = N * N
```

を実行すると定数10の記憶場所の内容が100にかわってしまい、2番目のCALL文でSUBを呼ぶとき、10の入っているはずの記憶場所の内容 100がサブルーチンに引き渡されて、意図したこととは全く違う結果をまねくことになる。

```
      DIMENSION A(10)
      READ(5,100) A
  100 FORMAT(10F8.2)
      CALL SUB(A,B,10)
      CALL SUB(A,B,10)
      WRITE(6,101) A,B
  101 FORMAT(1H ,11F8.4)
      STOP
      END


      SUBROUTINE SUB(A,B,N)
      DIMENSION A(N)
      B=0
      DO 10 I=1,N
      A(I)=A(I)**2
      B=B+A(I)
   10 CONTINUE
      N=N*N
      B=B/N
      RETURN
      END
```

ＪＩＳ規格票[5]ではこのことを考慮して、〝実引数に対応する仮引数が、引用される副プログラム内で定義されたり、再定義されたりしている場合にはその実引数は、変数名、配列要素名、あるいは配列名でなければならない。〟と断っている。しかしＦＯＲＴＲＡＮのコンパイルはプログラム単位毎に行われるので、コンパイル段階では仮引数の定義再定義と実引数との対応について検査のしょうがなく実行時に実際に実引数と仮引数の結合の時点で検査するほかない。しかし、このような検査は目的プログラムの実行速度を著しく妨げるので、実際には行わないのが普通である。そこでプログラムを書くものが自分で上のような注意をしなければならないわけである。

さて、前節の例題の関数をもちいて

```
X = ADD(A,B) * SUB(A,B)
IF( ADD(A,B) ) 1,2,3
```

のプログラムは、このように関数を2度呼び出すか

```
D = ADD(A,B)
X = D * SUB(A,B)
IF( D ) 1,2,3
```

のように文を1つふやさなければならない。これに1つの出力パラメータを追加して、次のような関数を作る。

```
FUNCTION  ADD(A,B,C)
C=A+B
ADD=C
RETURN
END

FUNCTION  SUB(A,B,C)
C=A-B
SUB=C
RETURN
END
```

この関数をもちいると、上記のプログラム例は

```
X = ADD(A,B,D)*SUB(A,B,E)
IF( D ) 1,2,3
```

と書き換えることができる。このような用法が模範的プログラムというわけではないが、次のような例題でその効用が知れよう。

例、実数A、B、Cを係数とする　$Ax^2+Bx+C=0$　の根を求めるサブルーチンは以下のように書ける。

```
      SUBROUTINE SQUARE(A,B,C,X1,X2,D)
      D=B*B-4.0*A*C
      R=SQRT(ABS(D))
      S=-B/(2.0*A)
      T=R/(2.0*A)
      IF(D) 1,2,2
1     X1=S
      X2=T
      GO TO 3
2     X1=S+T
      X2=S-T
3     CONTINUE
      RETURN
      END
```

ここでA、B、Cは係数の値、Dは判別式の値、X1、X2はD≧0のとき2実根、D＜0のときは根X1 ± i X2のX1が実部、X2が虚部である。すなわち、このサブルーチンSQUAREに対してA B、Cが入力用パラメータ、X1、X2、Dが出力用パラメータとなっている。これを用いて

```
CALL SQUARE(1.0,-0.5,2.0,X1,X2,D)
IF( D ) 1,2,3
```

とすることもできるが、SUBROUTINEを関数に書き直して

```
      FUNCTION SQUARE(A,B,C,X1,X2,D)
      D=B*B-4.0*A*C
      R=SQRT(ABS(D))
      S=-B/(2.0*A)
      T=R/(2.0*A)
      IF(D) 1,2,2
    1 X1=S
      X2=T
      GO TO 3
    2 X1=S+T
      X2=S-T
    3 CONTINUE
      SQUARE=D
      RETURN
      END
```

として

```
      IF( SQUARE(1.0,-0.5,2.0,X1,X2,D) ) 1,2,3
```

としてもよい。

7．FACOM 230−60 FORTRANにおける副プログラムの呼出し

JISでは副プログラムの呼出しについて次のような規定をしている。

1）外部関数の引用　　外部関数は算術式または論理式中で、1次子としての関数の引用を用いることによって引用する。引数の並びに書かれている実引数は、対応する仮引数と順序、個数および型が一致しなければならない（JIS P34）。

2）サブルーチンの引用　　サブルーチンはCALL文により引用する。CALL文において、引数の並びを構成している実引数は、対応する仮引数と順序、個数および型が一致しなければならない。実引数として文字定数が使える（JIS P36）。

FACOM 230−60 FORTRANでは、外部関数をCALL文で用いたり、サブルーチンを外部関数のように式の中に1次子として用いた場合、コンパイル時にはいずれの場合もエラーも警告も発せられない。それは当然で、FORTRANは、プログラム単位毎に独立しているので、あるプログラム単位内でサブルーチンか関数かに確定しさえすればよい。それぞれのプログラム単位がLIEDで結合されて、実行段階でもし上のような使い方がされているとパラメータの個数が合わないというエラーとして検出されて、以後実行が打切られる。

ところで、前節の例でFUNCTION SQUARE（A，B，C，X1，X2，D）を定義しておき、2次方程式の根はあらかじめ実根であることがわかっていれば、この関数の関数値を判定する必要はない。一般に関数値に関心がなければ、関数を

```
      CALL SQUARE(A,B,C,X1,X2,D)
```

としてしまうこともあるだろう。参考のために各大学大型計算機センターで用いられている3つの機

種のFORTRANの副プログラムの呼出し系列を表2にあげてみた。

HITACやNEACでは、関数値がAccumulatorの上につくられるので、関数を上のようにＣＡＬＬ文で用いても、関数値がおきざりになるだけでエラーとしては検出されないし、サブルーチンを関数の１次子に用いても、Accumulator上の全く無意味な値を関数値とみなして、エラーとして検出せずに計算してしまう（もちろん結果はデタラメであろう）。一方、FACOMでは、関数値を指定した番地を通じて引き渡しているので、さきに述べたようなパラメータの数が合わないようなエラーを検出する。サブルーチンを関数の１次子にするという例はほとんどないだろうが、関数をＣＡＬＬ文で呼ぶ例はＨＩＴＡＣやＮＥＡＣでは度々あるので、プログラムの書き換えに際して注意すべきことである。

実際、JISではHITACやNEACのような用法は認めてはいないので、FACOMのやり方がＪＩＳから見れば正しいのであるが、ＦＯＲＴＲＡＮでは慣用的に行われてきているので微妙な問題である。

表2　副プログラムの呼出し系列

機種名	ＣＡＬＬ文 CALL SUB ($a_1, a_2, \cdots, a_n$)	関数の引用 FUNC($a_1, a_2, \cdots, a_n$)	備考
FACOM 230－60 FORTRAN	LAI　n SXJ, 7　SUB NOP　a_1, , b NOP　a_2, , b ⋮ NOP　a_n, , b	LAI　$n+1$ SXJ, 7　FUNC NOP　return, , b NOP　a_1, , b NOP　a_2, , b ⋮ NOP　a_n, , b	文献 8) p. 93, 94 n は引数の個数 SUB, FUNCは副プログラムの先頭番地 a_1, a_2, ‥‥, a_nは実引数の番地 b はベースレジスタ指定 returnは，関数値を受け渡す番地
HITAC 5020 HARP	ICA　08, a_1 ICAR　08, a_2 ICA　09, a_3 ⋮ JS　00, SUB	左に同じ JS　00, FUNC	文献 9) p. 95 08, 09はAレジスタの番地 a_1 | a_2 | a_3 | a_4 …として実引数の番地 が副プログラムに引き渡される 関数副プログラムの中で計算された関数値はAレジスタ8番地にはいってもどる
NEAC 2200／500 FORTRAN－L	DCSW　SUB DCW　a_1 DCW　a_2 ⋮ DCW　a_n DCP　@N@	DCSW　FUNC 左に同じ	文献10) p. 154 関数値の引き渡しについて文献10) には記述がないが，テストプログラムの結果から，ＡＣＣによるものと推定される …実引数の終わりを示すレコードマーク

そこで、FACOM 230－60 FORTRANでは関数をCALL文で使うような例はダミー変数をもうけ

```
DUMMY = SQUARE(A,B,C,X1,X2,D)
```

のようにすればよい。整数型関数の場合には、換算関数が自動的に挿入されてしまうので、それを嫌うむきには

```
IDUMMY = IADD(A,B)
```

のようにすればよい。このようにしておけば、逆にHITACやNEACでもそのまゝ通用する。

以上述べたような事項を基礎にして、以下にFORTRAN based languagesの具体例を示そう。

（以下次号）

文　献

1) J. Weizenbaum: Symmetric List Processor. Comm. ACM, Vol 6, 9p 524(1963)

2) J. M. Sakoda：DYSTAL Manual－Dynamic Storage Allocation Language in FORTRAN, Brown Univ. (1965), pp 308

3) 中村康弘：ページメモリーをもったリスト処理言語FLIPⅢ、第10回プログラミングシンポジウム報告集 (1969)

4) A. Pritsker, P.J. Kiviat：Simulation with GASPII. Prentice Hall pp332(1969)

5) 電子計算機プログラム用言語FORTRAN (水準7000) ほかJISC 6201－1967、日本能率協会 pp62 (1967)

6) S. Gorn (ed)：FORTRAN vs.Basic FORTRAN. Comm. ACM, Vol 7, 10 p 591 (1964)

7) D. W. Barron：Recursive Technique in Programming. Macdonald pp 64 (1968)

8) FACOM 230－60 FORTRAN 解説編 (II)

9) HITAC5020, 5020E／F FORTRAN (HARP)

10) NEACシリーズ2200オペレーティングシステムMODⅢ／MODⅣFORTRANコンパイラ説明書