コンピューター言語と思考について

植松茂暢・福田弘之

1. 初めに

LSIの使用によって，ここ数年のコンピュータの進歩と普及には目覚ましいものがあります。電卓を用いる小学生，マイコンに熱中する中学生をよく見かけます。5，6年前まで教育工学センターのミニコンを用いて，シィミィレーション装置の20台のVTRを駆使するためのプログラムをアセンブラーで組んでいて，紙テープの孔を銀紙を切って帖ってふさいだり，また孔をあけたり，悪戦苦闘したのは，本当に夢のようです。

ここに，一般教育科目の「マイコンとBASIC言語」という講義で学生を指導したり，自分の研究生を指導したりして感じたことを2，3報告します。

2. 思考の繰り返しと言語について

コンピュータが威力を発揮するのは，プログラムの面で言えば

（1） データを変えると同一プログラムで，何回でも繰り返し計算すること

（2） プログラムの中に繰り返しを作りプログラムをコンパクトに作れること

によるものと思います。ここでは，この後者のものについて考察します。

プログラムの繰り返しには，複雑さの度合によって，つぎのような段階があります。

(A) サブルーチン

プログラムの中に何個所かで同じ働きをする部分があるとき，それをサブルーチンとして取り出し別に書いておいて，その働きが必要な都度，そのサブルーチンに行き，その働きを実行して帰って来ます。

この働きをアセンブリ言語のレベルで見ますと，サブルーチンの頭の部分に帰り番地を自動的に記入して，サブルーチンから帰る場合には，それを見て帰

ります。アセンブリ言語では，サブルーチンの中で帰り番地に変更を加えるなど面白いプログラムが作れますが，FORTAN 言語など高級言語では，そのようなきめ細かなプログラムは作れません。

(B) IF 文または FOR (DO) 文による繰り返し

これはプログラムの一部を繰り返えすために制御変数をきめ，プログラムのある部分を実行すると，その変数を変え，その変数がある条件を充たすまでそれを繰り返えします。この IF 文または FOR 文は何重にも出来て便利なものであって，これを修得することは FORTRAN 言語，BASIC 言語など高級言語をマスターする鍵であると思います。

FOR 文によるループが n 重になっているときは，$n-1$ 重ループまでの変数を定めたとして，n 重目のループの様子をプログラムすればよいのですが，学習者にはなかなか理解されません。特に中学生のプログラミングを見ていると，一重のループでも各ステップでの変数の変化を追って見なければ納得しないようです。これはピアジェの言う形式的操作期に入ったばかりであるからでしょうか。

(C) 再帰的定義

一般にあるものを定義するとき，定義の中に定義する対象を用いることはしません。しかし，数学的帰納法では，これを許します。

たとえば $n!$ (n の階乗) は

$$n!=n\times(n-1)\times\cdots\times 2\times 1$$

のことですが，これは n を整数として

$n=1$ のときは $f(n)=1$

$n>1$ のときは $f(n)=n\times f(n-1)$

と定義されます。これと同じような定義を再帰的定義といいます。

ある言語で再帰的定義が可能であるかどうかは，その言語をアセンブリ言語のレベルに落した段階で，サブルーチンに飛ぶとき，帰り番地，変数の値をどのように管理してくれるかによります。FORTRAN 言語，ALGOL 言語，COBOL 言語などでは，帰り番地を 1 つ管理するだけなので，本質的に再帰的

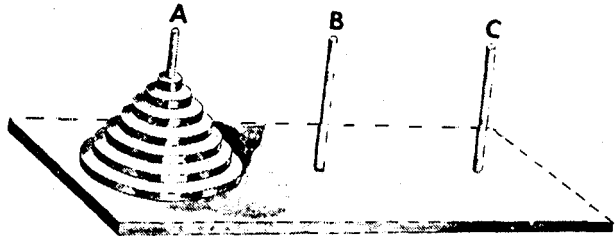
定義はできません。しかしBASIC言語では，帰り番地をたくさん管理するので，変数の管理をプログラマーがすれば，再帰的用法が可能です。第1図は，有効な使用法ではありませんが，n の階乗の計算で，サブルーンチを再帰的に使用した例です。

```
10 INPUT "N=";N
20 F=1
30 GOSUB 1000
40 PRINT F
50 END
1000 IF N=1 THEN RETURN
1010 F=F*N
1020 N=N-1
1030 GOSUB 1000
1040 RETURN
```

第1図

再帰的定義を最も美しい形で可能にするのはLISP言語です。LISP言語の再帰的用法をHANOIの塔で説明してみます。

HANOIの塔は図2のようなパズルのオモチャです。Aの棒につきささっている何枚か（n 枚）の円板を一枚づつ，棒Bを用いて，棒Cに移すのですが，円板は一度に一枚づつしか移動できないし，また，小さい円板の上へ大きい円板を乗せてはいけません。



第2図

この問題の解法をHANOI（nABC）とすれば，そのSTRATEGYは

① A にある n 枚の円板のうち上の円板 1～($n-1$) を棒Cを用いて棒Bに移す。（この解法はHANOI（$n-1$ A B C）である）

```
(DEFPROP HANOI
 (LAMBDA (N X Y Z)
  (PROG NIL
        (COND ((ZEROP N) NIL) ·····n=0になると移動しない
              (T (HANOI (SUB1 N) X Z Y) ·····n>0のとき HANOI(n-1 xzy)をし
                 (PRINT (LIST N X Z)) ·····移動する円板の番号とどこからどこへ
                                           移動するかをプリントし
EXPR)                 (HANOI (SUB1 N) Y X Z)))))···HANOI(n-1 yxz)を実行する

NIL
```

第3図

② A にある1番下の円板 n を棒 C に移す。

③ 棒 B に移した円板 1～$n-1$ を棒 A を用いて棒 C に移す。(この解法は HANOI $(n-1)$ B A C である)

この STRATEGY ①, ③で解法を再帰的に用いて定義している。LISP 言語では, この STRATEGY をそのままプログラムします。第3図はプログラムで, 第4図は $n=3$ の実験例です。

```
*(HANOI 3 "A "B "C)

(1 A C)
(2 A B)
(1 C B)
(3 A C)

(1 B A)
(2 B C)

(1 A C)
NIL
*
```

第4図

LISP の再帰的定義を HANOI の塔の解法で説明しましたが, このように, LISP 言語では数学的な思考をそのままプログラムできます。LISP 言語は(計算を主としない方面の) 数学, 言語学などの研究には, 最も適した言語であると思います。

3. プログラミングの巧拙について

現在マイクロコンピュータで使える高級言語は FORTRAN, PL/I, PASCAL, LISP, COBOL, BASIC, FORTH 等数多くあります。COBOL については大型コンピュータで走るものと同程度のレベルのものが使える様になったそうです (CIS-COBOL ver 5)。これらの高級言語を用いると, 小さなプログラムでは問題を分析したりフロチャートを書かなくても直接キーボードに向ってプログラムを打ち込むことが出来るほどです。通常マイクロコンピュータでは BASIC が使えます。BASIC で小さなプログラムを作っているときは問題はあまり生じませんが少し長いプログラムになると, それまでの無計画な方法では作れないことに気がつきます。たとえ作ることが出来てもそのプログラムを後から修正しようと思うと作った本人も含めて全く手が出せない程難解なものとなってしまっています。この原因はプログラムを作る基本的な態度と問題がある様です。ところで我々数学屋から見ると美しいプログラムとそうでないプログラムの区別がつきます。プログラムを作るのは数学においては定理を分析し

て証明を組立てる作業に，よく似ています。定理を証明するためによく分析し，いい準備（Lemma）を用意すれば証明は透明で美しいものになります。我々数学屋は美しい定理とエレガントな証明に出合うといたく感動します。同様に透明で美しいプログラムを見るとそのプログラムを作った人に数学的なセンスを見い出してうれしくなります。我々が見て美しいプログラムとは次の様なものです。まず最初の数行でプログラムの全体がどの様な構造になっているか分かる様に書いてあって，プログラムが作業ブロックごとによく整理されているものです。あまり技巧を凝らしたものより自然な流れのある方が美しく感じます。

この様なプログラムを書く訓練は高級な言語でプログラムを作っているうちはなかなか上達しない様です。一度アセンブラを用いてプログラムを作りトップ・ダウン方式を身につけるのが良いかと思いますが何よりもまず美しいプログラムとそうでないものをはっきり区別出来る心を養うことが大切だと思います。