プログラミング方法論

——Basicによる実現——

真　田　克　彦

(1984年10月15日受理)

Programming Methodology

——Implementation by Means of Basic——

Katsuhiko SANADA

1. はじめに

最近のプログラミング方法論の展望については，〔4〕，〔7〕，〔13〕などに詳しく述べられている。良いプログラムとは，認識し易く，誤りのない信頼性のあるプログラムのことであるといえる。さらに効率が良く，変更がし易いということも付け加えるべきであろう。これらの目的に向って，種々の方法が考えられ，開発されて来た。現在では，プログラミング方法論の原理または概念として，

モジュール化，抽象化，情報隠蔽，局所化

などが，代表としてあげられている。さらに，これらの概念の実現化のためのツールとして，抽象データタイプが有効であると認められるようになって来た。

第2章で，抽象データタイプの考えが，プログラムの設計に有効であることを，具体的な問題の例により示す。すなわち，この方法により，より良く上の4つの原理の実現に近づくことができる。

ところで，このようなプログラミング方法論の研究とは，無縁といってよいところにあるのが，Basicである。しかるに，現実には，パーソナルコンピュータの普及と共に，Basicによりプログラミングをする人口は急激に増加しており，著者自身もBasicでプログラムを作ることが非常に多い。このような現状から，Basicで良いプログラムを作る方法を考えてみた。その1つの試みが，第3章で述べたSp-Basicである。これは前処理プログラムにより，Basicの機能の拡張をはかったものである。

第4章では，第2章で述べた設計法をSp-Basicで実現した例を示した。

2. 抽象データタイプ (abstract data type)

2. 1 抽象データタイプ

大規模なプログラムの設計における鍵は，その問題の複雑さと細部の情報を減少させることである。そのための手段として現在最も有効と考えられているのは，問題の分割（モジュール化）と抽象

化である。問題によっては分割された部分（モジュール）が，なお複雑過ぎる場合も多く，この種の複雑さを減少させるのが抽象である。抽象化は問題のもっている諸属性の中から１つまたは幾つかの属性を思惟の対象として抽出し，それら以外の属性を捨象することによって，一時に理解しなければならない対象の複雑さを減少させる。

プログラムの設計において，抽象データタイプの考えが定着した概念となりつつある。抽象データタイプは，その実現法とは独立なデータタイプであって，そのタイプの上での演算（opperation）の集まりによって定義される。

抽象データタイプは，そのタイプの定義とそのタイプすべての演算が，プログラムの１つの部分に局所化できるという意味で，データタイプをモジュール化する。したがって，抽象データタイプを変更する場合には，その１つのモジュールのみを変更すればよい。さらに，その抽象データタイプが定義されている外では，その抽象データタイプを原始データタイプ（整数型，実数型など）と同じように扱うことができる。

抽象データタイプの実現には，使用するプログラム言語の基本データタイプ（array など）を用いることになる。その言語の原始データタイプや演算が，抽象データタイプのモデルや演算に非常に近い言語であることが望ましいことは当然である。

抽象データタイプの例として，stack データタイプを図１に示す。抽象データタイプは pushdown list とか last in-first out とも呼ばれるデータ構造で，すべての挿入と消去は，top という一方の端の場所で行う。図１の５つの演算（operation）により定義される。S は stack を示し，x は element type すなわち，原始データタイプか，または他の抽象データタイプの要素である。Boolean（true or false）も原始データタイプである。

タイプ　stack
記　号
　S—stack
　x—element type
　b—Boolean
演算
1. INIT(S)⟶S
   stack S を空 stack にする。
2. TOP(S)⟶x
   stack S の top の要素を返す。
3. POP(S)⟶S
   stack S の top の要素を消去する。
4. PUSH(S)⟶S
   stack S の top に要素 x を挿入する。このとき前の要素は押し下げられる。
5. EMPTY(S)⟶b
   Sが空 stack なら true を，そうでなければ false を返す。

図１　stack 抽象データタイプ

大低のプログラム言語は，基本データタイプとして，array データタイプを備えている。array データタイプにより stack データタイプは実現できる。その Pascal による例が〔1〕の中に見られる。〔1〕の中には，他にも list, queue, tree などの抽象データタイプの Pascal による実現の例が示されている。〔3〕の中では，array データタイプの仕様記述，およびそれを用いた stack データタイプの実現（implementation）が示されている。

2. 2　index プログラム

2. 2. 1　問題の分析

本節では index プログラムという例題によって，抽象データタイプを用いたプログラムの設計方

法について述べる。(2. 2節の作成には〔10〕を参考にした)

index プログラムとは，ある文書が与えられたとき，その文書に現れた各単語（word）と，その現れた位置を示すプログラムである。

ここでは与えられた文書（input file と呼ぶ）は，行の列で構成され，各行は行番号で始まり，その後に単語（word）の列がある。各単語はアルファベット文字（alphabetic character）の空でない列であり，隣り合う単語は1つ以上の空白（space）またはアルファベットでない文字，すなわち句読点，改行文字などで区切られている。index の結果の出力（output file）は，文書中の単語をアルファベット順に並べ，各単語の文書中に出現した行番号を付したものとする。input file と output file の例を図2，図3に示す。

index プログラムは，問題を考慮すると，input file と output file というデータ構造があり，プログラムはその間のデータの変換を行うことになるが，その変換の際に単語と行番号を記録するための内部テーブル（inner table）としてのデータ構造が必要とされる。

input file の要素上で実行される演算（opperation）の集まりを定義することにより，Input タイプの抽象データタイプを定義することができる。

```
100 THIS IS A PIECE OF TEXT
200 FOR DESIGNING PROGRAM. THE FILE
300 IS A SEQUENCE OF LINES AND EACH LINE
400 IS A LINENUMBERED FOLLOWED BY A
500 SEQUENCE OF WORDS.
```

図2

すべての input process が終了するまでは，output process を始めることはできない。例えば，input file の最終行に単語“A”が含まれていることもある。ところが“A”は，output file の最初に出力させなければならないからである。

A	400	400	300	100
AND	300			
BY	400			
DESIGNING	200			
EACH	300			
FILE	200			
FOLLOWED	400			
FOR	200			
IS	400	300	100	
LINE	300			
LINENUMBERED	400			
LINES	300			
OF	300	100		
PIECE	100			
PROGRAM.	200			
SEQUENCE	500	300		
TEXT	100			
THE	200			
THIS	100			

図3

したがって内部テーブルは，入力されるすべての単語と行番号を記録するための大規模なデータ構造となる。その構造は array, record, tree, その他プログラマーによって選ぶことができる。(勿論，使用するプログラム言語による制限はある。) しかし，それらのテーブルの操作は，データ上の演算の集まりとして定義できるので，Inner Table タイプの抽象データタイプを定義する。

output file もデータ構造であり，index された単語と行番号の，ある目的向き（プリント向き，デイスクファイル向きなど）の format 化されたデータの構造が考えられる，出力操作を出力要素上の演算の集まりとして定義できるので，output タイプの抽象データタイプを定義することができる。

結果として3種類の抽象データタイプをモジュール化することになる。

(1) Input タイプ

(2) Inner Table タイプ

(3) Output タイプ

プログラムの設計の第1段階は3つの抽象データタイプの定義の作成から始めなければならない。

2. 2. 2 Input タイプの定義

Input タイプの例を図4から図6までに示した。

図4－1は第1の定義の試みである。記法は，例えば

LINE——→1行分のデータ (line data)

において，英大文字で示された LINE は演算 (operation) の名前を示す。演算のための入力パラメータが必要な場合には，名前の右に括弧でくくって示す。——→の右は演算の結果の出力を示す。この場合，LINE という演算により，1行分のデータ，すなわち line data が得られることを示している。

Input 1 タイプの抽象データタイプ（図4－1）を定義した場合，そのモジュールを利用したプログラムの一部の概略を図4－2に示す。

Input 1 タイプの場合，1行分のデータ (line data) の中には単語と単語の間の区切り記号(スペースや句読点など）を含んだままである。したがって図4－2の行50と90に示したように，Input 1 タイプを使う場合，line data から行番号と各単語に分解する作業をしなければならない。しかしながら，index プログラムの output に必要な情報は行番号と各単語だけで，区切り記号は必要ないわけで，Input 1 モジュールは余分な情報を外に出していることになり，不適当である。そこで，その点を改良して，Input 2 タイプ（図5－1）を作成した。演算 WORD により，現在行

Input 1　タイプ

```
LINE——→1行分のデータ (line data)
EOF——→end-of-file (true or false)
```

図 4—1

```
10  while
20      not EOF
30  do begin
40      LINE——→line data
50      {line data より行番号をとり出す}
60      while
70          {line data は空でない}
80      do begin
90          {line data より次の単語をとり出す}
100         {単語と行番号の処理}
110     end
120 end
```

図 4—2

Input 2　タイプ

```
LNUMBER——→行番号 (linenumber)
WORD  ———→次の単語 (next word)
EOL   ———→end-of-line (true or false)
EOF   ———→end-of-file (true or false)
```

図 5—1

```
10  while
20      not EOF
30  do begin
40      LNUMBER——→行番号
50  while
60      not EOL
70      do begin
80          WORD——→次の単語
90          {単語と行番号の処理}
100     end
110 end
```

図 5—2

Input 3　タイプ

```
GETWL——→単語と行番号
EOF  ——→end-of-file (true or false)
```

図 6—1

```
10  while
20      not EOF
30  do begin
40      GETWL——→単語と行番号
50      {単語と行番号の処理}
60  end
```

図 6—2

の現在処理中の単語の，次の単語が返される。Input 2 タイプに対応するプログラムの概略（図5—2）において，内部ループ50～100行は，行の終りを見つけるまで実行される。すなわち，Input 2 モュールを用いる場合には，input file が行の列になっていることを知っていなければならない。したがって，各単語と行番号を結びつけることを常に意識していなければならない。input file が行の列であることは，Input モジュールの外では不必要な細部の情報であるので，Input モジュールの外に出すべきではなく，内部に隠すべき情報である。

Input 3 タイプ（図6—1）では，細部の情報を隠すように設計した。GETWL 演算では，次の単語とその位置を示す行番号が返される。これにより，input file の構成に関する情報は外部に出てこない，これを用いたプログラムの概略は図6—2のようになる。

2. 2. 3 Inner Table タイプの定義

内部テーブルのデータ構造の実現方法については，ここで触れる必要はない。しかし，input file の中の単語には当然2度以上現われるものもあり，その場合に，それらを現れる度毎に内部テーブルに格納するか，1度現われたものは重ねて格納しないかは，内部テーブルのデータ構造に関係する問題になってくるであろう。しかし，情報の経済性から当然異なる単語のみを格納すればよいと判断できる。したがって，Inner Table 1 タイプを図7—1のように設計した。

ENTWORD（単語）は括弧内の単語を内部テーブルに格納する。また ENTLINE（行番号）は括弧内の行番号を内部テーブルに格納する。Inner Table 1 タイプを用いたプログラム概略は図7—2に示した。50～100行の if 文は内部テーブルの構造に関係するもので，モジュール内に隠して，外部に出さない方が望ましい。すなわち，指定した単語がすでに内部テーブルにあるかどうかは，不必要な細部の情報であって，そのモジュールを利用する側が知る必要はない。

Inner Table 1　タイプ

```
ISINTABLE (単語) ——→b (true or false)
ENTWORD (単語) ——→内部テーブル
ENTLINE (行番号) ——→内部テーブル
```

図 7—1

```
10  while
20        not EOF
30     do begin
40        GETWL——→単語と行番号
50        if ISINTABLE
60        then  ENTLINE (行番号)
70        else begin
80           ENTWORD (単語)
90           ENTLINE (行番号)
100       end
110    end
```

図 7—2

Inner Table 2　タイプ

```
ENTWL (単語，行番号)——→内部テーブル
```

図 8—1

```
10  while
20        not EOF
30     do begin
40        GETWL——→単語，行番号
50        ENTWL(単語，行番号)
60     end
```

図 8—2

Inner Table 3　タイプ

```
ENTWL (単語，行番号)——→内部テーブル
CHANGEMODE ——→変換された内部テーブル
RETRIEVE ————→単語，行番号
EOD ————————→end-of-data (true or false)
```

図 9

したがって，ここでは図8—1の Inner Table 2 と，図8－2のプログラムの方が望ましいと考えられる。この場合，同じ単語がダブって格納されるかどうかは，Inner Table 2 モジュールの実現の仕方に関することであって，モジュール内で決定されることである。

ところで，output file はアルファベット順に出力しなければならないが，単語の並べかえ（sorting）や output 用にテーブルを変換する操作は，Inner Table モジュール内の仕事である。また output のために，内部テーブルからデータをとり出す操作も，Inner Table モジュールの仕事に含まれる。これらのことを考慮して，Inner Table 3（図9）タイプを設計した。

CHANGEMODE は，内部テーブル内にすべての入力データが格納された後に，操作されるべき演算で，sorting process や出力準備のためのテーブル変換などを行う。また RETRIEVE は単語と行番号の対を，内部テーブルより取り出す演算である。

Output タイプ

```
PRINT（単語，行番号）——→出力ファイル
```

図 10

index プログラム

```
10  begin
20    while
30          not EOF
40      do begin
50          GETWL——→単語，行番号
60          ENTWL（単語，行番号）
70        end
80    CHANGEMODE
90    while
100         not EOD
110     do begin
120         RETRIEVE——→単語，行番号
130         PRINT（単語，行番号）
140       end
150   end
```

図 11

2. 2. 4 Output タイプの定義

Output タイプ（図10）は PRINT 演算により，単語と行番号を出力ファイルに出す。どのようなメディア（プリンタか，ディスクファイルかなど）に出すか，どのような format で出すか，などの実現は，すべてプログラマーにまかせられるが，特に困難な点はない。

2. 2. 5 全体のプログラム

これまでの議論により，抽象データタイプ Input 3, Inner Table 3, Output の各タイプを用いて，index プログラムの全体の概略を示すと，図11のようになる。

出来上がったプログラムは，極めて簡単な構造であり，また各抽象データタイプはそれぞれ独立しており，それぞれ独立に変更が可能である。

3. Sp-Basic

3. 1 Basic の前処理プログラム

前章で述べた抽象データタイプの実現（implementation）は，プログラムのモジュール化によって行われる。モジュール化により演算（operator）の概念を一般化し，アルゴリズムの局所化された部分のカプセル化（encapsulation）を行うことができる。ところで，このような機能を実現するためには，使用するプログラム言語の役割は極めて大きく，実現のためには，適切なプログラム言語を選ぶことが重要なことである。〔8〕

しかし，観点を変えて，現在広く普及している言語において，前述の抽象データタイプの実現を試みてみることも興味あることであり，その言語の持つ欠点とか種々の特徴を明確にするきっかけともなる。

そこで，非常に不完全な言語である Basic によって，そのような試みを行なってみたいと考えた。Basic は不完全な言語ではあるが，パーソナルコンピュータの普及と共に広く使われており，これにプログラミング方法論を適用することも無意味とは思われない。Basic も欠陥ばかりではなく，長所も持っている。どうせ Basic を使うなら有効に使うべきであろう。

Basic の致命的な欠陥の1つは手続き (procedure) の機能を持たないことである。(他にも致命的な欠陥はあるが)。先ず，前記のモジュール化を実現するためには，Basic に手続き機能を追加する必要がある。その手段として，前処理プログラム (preprocessor) の方法を用いることにした。勿論不充分ではあるが，この方法により一応の手続き機能を持たせることができる。

拡張された機能を持つ Basic を以下 Sp-Basic と呼ぶことにする。(Sp とは special の略である) Sp-Basic で書かれたソースプログラムを，前処理プログラムによって通常の Basic の形に落し，その後は普通どおりの処理手順となる。したがって，普通の処理に比べて，前処理の段階が追加されることになる。

前処理プログラムの良い点は，言語を改良したいと思ったとき，コンパイラ（またはインタプリータ）を書かないでもすみ，現在あるものの上に乗っかって仕事をすることができることである。

Sp-Basic のここでの主目的は，Basic の手続き機能の拡張であったが，更により良い言語に仕立てるために，別の制御構造を追加することも比較的簡単にできるであろう。Sp-Basic は新しいプログラム言語を志しているのではなく，Basic の最悪の欠陥を克服し，それを幾分か使える言語にすることが，Sp-Basic の目的である。

3. 2　Sp-Basic の説明

Sp-Basic は通常の Basic*)に手続き (procedure) としてのサブルーチンの機能を付け加えた。

サブルーチンはプログラムの一部を定義するものであり，サブルーチン呼出し文によって，それが活性化できるように，サブルーチンの宣言で，その部分に名前を結び付けておく。この宣言はプログラムと同一形式をしているが，サブルーチンの頭書きで始まるその形式と働きは Fortran のサブルーチンに従ったものであり，ほとんど同じ働きを目差したものである。

付加した機能は，サブルーチンの定義，サブルーチンの呼出しそして COMMON 文であり，それらの構文規則は次のようである。

サブルーチンの定義

〈サブルーチンの定義〉::=〈サブルーチン頭書き〉〈本体〉 **SEND**

*) Basic にも多種の異形があるようであるが，ここに示すものは，パーソナルコンピュータ PC-9801 の基礎言語となっている N_{88}-BASIC を用いて作成した。したがって，この試みは，すべての Basic に適用できるわけではない。

〈本体〉::=〈宣言部〉〈文の部〉**SRETURN**〈文の部〉*)

〈サブルーチン頭書き〉::=**SUBROUTINE**〈名前〉|

SUBROUTINE〈名前〉(〈仮引数〉{,〈仮引数〉})

サブルーチンの呼出し

〈サブルーチンの呼出し文〉::=**SCALL**〈サブルーチンの名前〉|

SCALL〈サブルーチンの名前〉(〈実引数〉{,〈実引数〉})

COMMON 文

〈COMMON 文〉::=**COMMON**〈変数〉{,〈変数〉}

図12に説明のためのプログラム例を示した。

200 SUBROUTINE PLUS (R1)

は，サブルーチン頭書きである。PLUS がサブルーチンの名前で，R1が仮引数である。頭書きからSEND までがサブルーチンの定義（または宣言）であり，頭書きと SEND との間の200～250行の文の集まりが，サブルーチン本体である。本体の中には少なくとも1つの SRETURN 文を含まなければならない。それによって，呼出しへ戻ることができる。

本体内の変数C（220, 230行）は局所的変数であり，このサブルーチン本体内のみで有効である。また210行 COMMON 文に現れる変数 P1, P2 は COMMON 変数であり，メインルーチンの70行 COMMON 文の変数A, Bと共通の記憶場所を占める変数である。この COMMON 文の働きも Fortran と類似している。

PLUS サブルーチンの呼出しは，メインルーチンの

90 SCALL PLUS (WA)

によって行われる。すなわち，名前 PLUS で結び付けられサブルーチンが活性化する。このとき，実パラメータ WA と仮パラメータ R1 の間で情報の受け渡しが行われる。

3. 3 Sp-Basic のコード生成規則

本節では Sp-Basic で書かれたプログラムが，前処理プログラムによって，どのように Basic に変換されるかを示す。

図12の Sp-Basic プログラムは，図13の Basic プログラムに変換される。図12から図13への変換の例を参照しながら，コード生成規則を説明する。

サブルーチンの呼出し

90 SCALL PLUS (WA)　　(図12)

は

90 PARA1=WA : GOSUB*PLUS : WA=PARA1　　(図13)**)

に変換される。

---

*) 文の部とは，通常の文の集まりのことである。

**) N88-BASIC では，ラベルを表すのに＊を付して＊PLUS のように表す。

サブルーチンの頭書き

200　SUBROUTINE PLUS (R 1)　　(12図)

は

200　'SUBROUTINE PLUS (R 1)*)

202　*PLUS　　(図13)

204　R1=PARA 1

に変換される。変数 PARAi（ i は順次 1, 2, 3, …となる）は，実引数と仮引数の間の情報の受け渡しを媒介する変数である。したがって，Sp-Basic においては，PARAi は予約語に含まれる。

SRETURN 文

240　SRETURN　　(図12)

は

240　PARA 1=R 1 : RETURN　　(13図)

```
10 ' ファイル ネーム exampl
20 '---------- exmain ----------
30 *EXAMPL
40 '  this is an example
50 *MAIN
60 '
70 COMMON A,B
80 READ A,B
90 SCALL PLUS(WA)
100 SCALL MINUS(SA)
110 SCALL PROD(SEKI)
120 SCALL QUOT(SHO,AMARI)
130 PRINT WA,SA,SEKI,SHO,AMARI
140 LPRINT WA,SA,SEKI,SHO,AMARI
150 DATA 12.5,13.2
160 '
170 END
180 '---------- plus ----------
190 '
200 SUBROUTINE PLUS(R1)
210 COMMON P1,P2
220 C=1
230 R1=(P1+P2)*C
240 SRETURN
250 SEND
260 '---------- minus ----------
270 '
280 SUBROUTINE MINUS(R2)
290 COMMON PP1,PP2
300 ALPHA=1
310 R2=(PP1-PP2)*ALPHA
320 SRETURN
330 SEND
340 '---------- prod ----------
350 '
360 SUBROUTINE PROD(R3)
370 COMMON PA,PB
380 BETA=1
390 R3=(PA*PB)/BETA
400 SRETURN
410 SEND
420 '---------- quot ----------
430
440 SUBROUTINE QUOT(A,B)
450 COMMON PX,PY
460 C=1
470 A=PX/PY*C
480 B=PX MOD PY
490 SRETURN
500 SEND
```

図12　Sq-Basic のソースプログラム例

```
10 ' ファイル ネーム exampl
20 '---------- exmain ----------
30 *EXAMPL
40 '  this is an example
50 *MAIN
60 '
70 'COMMON A,B
80 READ CMN1,CMN2
90 PARA1=WA:GOSUB*PLUS:WA=PARA1
100 PARA1=SA:GOSUB*MINUS:SA=PARA1
110 PARA1=SEKI:GOSUB*PROD:SEKI=PARA1
120 PARA1=SHO:PARA2=AMARI:GOSUB*QUOT:SHO=PARA1:AMARI=PARA2
130 PRINT WA,SA,SEKI,SHO,AMARI
140 LPRINT WA,SA,SEKI,SHO,AMARI
150 DATA 12.5,13.2
160 '
170 END
180 '---------- plus ----------
190 '
200 'SUBROUTINE PLUS(R1)
202 *PLUS
204 R1=PARA1
210 'COMMON P1,P2
220 ZC=1
230 R1=(CMN1+CMN2)*ZC
240 PARA1=R1:RETURN
250 'SEND
260 '---------- minus ----------
270 '
280 'SUBROUTINE MINUS(R2)
282 *MINUS
284 R2=PARA1
290 'COMMON PP1,PP2
300 YALPHA=1
310 R2=(CMN1-CMN2)*YALPHA
320 PARA1=R2:RETURN
330 'SEND
340 '---------- prod ----------
350 '
360 'SUBROUTINE PROD(R3)
362 *PROD
364 R3=PARA1
370 'COMMON PA,PB
380 XBETA=1
390 R3=(CMN1*CMN2)/XBETA
400 PARA1=R3:RETURN
410 'SEND
420 '---------- quot ----------
430 '
440 'SUBROUTINE QUOT(A,B)
442 *QUOT
444 A=PARA1: B=PARA2
450 'COMMON PX,PY
460 WC=1
470 A=CMN1/CMN2*WC
480 B=CMN1 MOD CMN2
490 PARA1=A:PARA2=B:RETURN
500 'SEND
```

図13　Sp-Basic のプログラムが Basic に変換された例

*) ' は注釈行（REM行）を示す.

に変換される。

SEND 文

```
250  SEND                                    (12図)
```

は

```
250  'SEND                                   (図13)
```

に変換される。

COMMON 文

COMMON 文に現れた変数は変数 CMNi に変換される。現れた順に，i には 1, 2, 3…が入る。CMNi も Sp-Basic では予約語に含まれる。　例えば

```
70  COMMON A,B
80  READ A,B                                (図12)
```

は

```
70  'COMMON A,B
80  READ CMN1, CMN 2                        (図13)
```

に変換される。また

```
210  COMMON P1,P 2
220  C=1                                     (図12)
230  R1=(P1+P2)*C
```

は

```
210  'COMMON P 1, P 2
220  ZC=1                                    (図13)
230  R1=(CMN1+CMN2)*ZC
```

に変換される。メインルーチン内の変数Aとサブルーチン PLUS内の変数P1は共に同じにCMN1変換され，同様にBと P2 も共に CMN 2 に変換されている。

局所変数の名前には帽子（1字のアルファベット文字）を付ける，その付け方の規則は，サブルーチンの出現順（サブルーチンの名前の出現順）に，逆順のアルファベット Z, Y, X,…… C, B, A を割り当てる。したがって，図12の 220，230 行の局所変数Cには帽子Zが付けられ，図13の220, 230行では ZC となっている。同様に，300行の ALPHA にはYが付けられ YALPHA に，380行の BETA にはXが付けられ XBETA に，460行のCにはWが付けられ WC となる。これにより局所変数はそれぞれのサブルーチン内だけの変数となる。勿論，メインルーチン内での変数の名前の付け方に多少の注意を払う必要はある。すなわち，アルファベットの最後の方の文字で始まる変数名は，なるべく使用しない方が無難であろう。

上記の生成規則により，Sp-Basic ではサブルーチンが，手続きとしてメインルーチンから独立できることになる。しかし，全く完全な方法でないことは致し方ない。

配列変数の場合には，引数にすることはできないので，COMMON 文を用いる方法をとる。この方法を示す例を図14に示す。またその変換されたものを図15に示す。

```
60  COMMON MA, FD
80  DIM MA (M, M), FD(M)
```
(図14)

は

```
60  'COMMON MA, FD
80  DIM CMN1 (M, M), CMN2(M)
```
(図15)

```
10 ' ﾌｧｲﾙ ﾈｰﾑ exgas
20 '---------- lmain ----------
30 *LMAIN
40 '
50 '   main of linear system
60 COMMON MA,FD
70 READ M
80 DIM MA(M,M),FD(M)
90 '
100 '   input data
110 PRINT "matrix":LPRINT "matrix"
120 FOR I=1 TO M
130   FOR J=1 TO M
140     READ MA(I,J)
150     PRINT MA(I,J);
160     LPRINT MA(I,J);
170   NEXT J
180   READ FD(I)
190   PRINT FD(I)
200   LPRINT FD(I)
210 NEXT I
220 '
230 '   subroutine call
240 SCALL*GAUS0(M)
250 '
260 '   print of results
270 PRINT :LPRINT
280 PRINT "result":LPRINT "result"
290 FOR I=1 TO M
300   PRINT "f(";I")=";FD(I)
310   LPRINT "f(";I")=";FD(I)
320 NEXT I
330 '
340 DATA 3
350 DATA 2,1,10,49
360 DATA 3,4,5,56
370 DATA 5,3,7,71
380 END
390 '---------- gaus0 ----------
400 SUBROUTINE GAUS0(M)
410 '  solving linear equations
420 '            by Gauss' elimination method
430 COMMON A,F
440 ' dim A(M,M),F(M)
450 '   forward elimination
460 M1=M-1
470 FOR I=1 TO M1
480   I1=I+1
490   FOR J=I1 TO M
500     AP=A(J,I)/A(I,I)
510     F(J)=F(J)-AP*F(I)
520     FOR K=I1 TO M
530       A(J,K)=A(J,K)-AP*A(I,K)
540     NEXT K
550   NEXT J
560 NEXT I
570 '  backward substitution
580 F(M)=F(M)/A(M,M)
590 FOR I=1 TO M1
600   I0=M-I
610   I1=I0+1
620   FOR J=I1 TO M
630     F(I0)=F(I0)-A(I0,   F(J)
640   NEXT J
650   F(I0)=F(I0)/A(I0,I0)
660 NEXT I
670 '
680 SRETURN
690 SEND
```

図 14

```
10 ' ﾌｧｲﾙ ﾈｰﾑ exgas
20 '---------- lmain ----------
30 *LMAIN
40 '
50 '   main of linear system
60 'COMMON MA,FD
70 READ M
80 DIM CMN1(M,M),CMN2(M)
90
100     input data
110 PRINT "matrix":LPRINT "matrix"
120 FOR I=1 TO M
130   FOR J=1 TO M
140     READ CMN1(I,J)
150     PRINT CMN1(I,J);
160     LPRINT CMN1(I,J);
170   NEXT J
180   READ CMN2(I)
190   PRINT CMN2(I)
200   LPRINT CMN2(I)
210 NEXT I
220 '
230 '   subroutine call
240 PARA1=M:GOSUB*GAUS0:M=PARA1
250 '
260 '   print of results
270 PRINT :LPRINT
280 PRINT "result":LPRINT "result"
290 FOR I=1 TO M
300   PRINT "f(";I")=";CMN2(I)
310   LPRINT "f(";I")=";CMN2(I)
320 NEXT I
330 '
340 DATA 3
350 DATA 2,1.10,49
360 DATA 3,4,5,56
370 DATA 5,3,7,71
380 END
390 '---------- gaus0 ----------
400 'SUBROUTINE GAUS0(M)
402 *GAUS0
404 M=PARA1
410 '  solving linear equations
420 '            by Gauss' elimination method
430 'COMMON A,F
440 ' dim A(M,M),F(M)
450 '   forward elimination
460 ZM1=M-1
470 FOR ZI=1 TO ZM1
480   ZI1=ZI+1
490   FOR ZJ=ZI1 TO M
500     ZAP=CMN1(ZJ,ZI)/CMN1(ZI,ZI)
510     CMN2(ZJ)=CMN2(ZJ)-ZAP*CMN2(ZI)
520     FOR ZK=ZI1 TO M
530       CMN1(ZJ,ZK)=CMN1(ZJ,ZK)-ZAP*CMN1(ZI,ZK)
540     NEXT ZK
550   NEXT ZJ
560 NEXT ZI
570    backward substitution
580 CMN2(M)=CMN2(M)/CMN1(M,M)
590 FOR ZI=1 TO ZM1
600   ZI0=M-ZI
610   ZI1=ZI0+1
620   FOR ZJ=ZI1 TO M
630     CMN2(ZI0)=CMN2(ZI0)-CMN1(ZI0,ZJ)*CMN2(ZJ)
640   NEXT ZJ
650   CMN2(ZI0)=CMN2(ZI0)/CMN1(ZI0,ZI0)
660 NEXT ZI
670 '
680 PARA1=M:RETURN
690 'SEND
```

図 15

に変換される。またサブルーチン内では

```
430  COMMON A, F
500  AP=A(J, I)/A(I, I)                    (図14)
510  F(J)=F(J)-AP*F(I)
```

は

```
430  'COMMON A, F
500  ZAP=CMN1 (ZJ, ZI)/CMN1 (ZI, ZI)      (図15)
510  CMN 2 (ZJ)=CMN 2 (ZJ)—ZAP*CMN 2 (ZI)
```

に変換される。MA とAは共に同じ配列変数 CMN 1 に変換され，同様に FD とFは共に配列変数 CMN 2 に変換される。

このように，Sp-Basic では配列を使ったサブルーチンも定義できる。しかし，配列宣言 DIM 文はメインルーチンだけで行うようにする。上例のサブルーチンでは，AとFとが配列変数であることを注釈文で示しておくことが適当である。

3. 4 Sp-Basic の応用

Sp-Basic の最も有効な利用法は，出来上った Basic プログラムを Sp-Basic のサブルーチンの形にしてファイルに保存することである。特に数値計算のパッケージ保存に有効である。

保存しているファイルを利用する場合には，いくつかのサブルーチンを適当に編集してとり出す編集システムを利用する。普通の Basic では既成のプログラムを利用する場合には，使用されている変数がダブらないかどうか，調べなければならないが，Sp-Basic ではその心配がない。

Sp-Basic に更に別の制御構造を加えることも考えられるが，将来の課題である。

4. Sp-Basic による抽象データタイプの実現

第2章において，抽象データタイプを用いたプログラムの設計法について論じた。この設計法では，その実現法とは独立に設計を行うことができる。そして，その設計を実現するためには，抽象データタイプを実現するのに適した言語を用いると，その実現も容易であろうし，しかもそのプログラムは認識し易く，簡潔なものとなるであろう。しかし，高級とは云えない言語 Basic によっても不十分ながら実現に近づけることができる，ここでは，前章で説明した Sp-Basic を用いて実現した例を示す。

例題は 2. 2 で提示した index プログラムであり，そこで示された設計に従って実現する。図16-1～16-4 に，Sp-Basic によって index プログラムを実現したプログラムを示す。

図11に示された index プログラム全体の概略は図16のプログラムのメインプログラム180～290行に対応している。340～850行が Input タイプの実現である。860～1510行が Inner Table タイプの実現である。そして1520～1600行が Output タイプの実現である，各モジュールは，ほとんど独立しているが COMMON 文を通じてメインルーチンとかかわっている部分がある。

Inner Table タイプにおける内部テーブルは，Basic が array データ構造を備えているので，配列により実現した。しかし，ソーティングプログラム（980～1040行の部分）は bubble ソートを用いているが，データが多くなるとこれでは時間がかかり過ぎるので，もっと高級なソーティングの方法を用いる必要がある。その場合，勿論プログラムはもっと複雑で大きくなるであろう。

図17のような，Basic のプログラムを input file として，図16の index プログラムを実行させると図18のような結果が得られる。

5. おわりに

Sp-Basic を作成した動機は，〔6〕，〔7〕を勉強して刺激を受けたことにある。両著の著者達に敬意と感謝の意を表したい。

現在，Sp-Basic により数値解析関係のプログラムを作成している。パーソナルコンピュータには益々愛着を感じてきた次第である。

```
10 ' ﾌｧｲﾙ ﾈｰﾑ index2
20 '---------- index program -----------
30 '***      main control      ***
40 '
50 COMMON TBL$,TBL%,NTC,INL$,ALS$
60 '   TBL$:word table
70 '   TBL%:linenumber table
80 '   NTC :word table counter
90 '   INL$:line input data
100 '   ALS$:characters contained in words
110 DIM TBL$(1000),TBL%(1000)
120 INL$="":NTC=0
130 ALS$="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789%#$!."
140 '
150 INPUT " file name of program ";NM$
160 OPEN NM$ FOR INPUT  AS #1
170 '
180 WHILE NOT EOF(1)
190   SCALL GETWL(WRD$,LN)
200   SCALL ENTWL(WRD$,LN)
210 WEND
220 '
230 SCALL CHANGEMODE
240 '
250 DEF FNEOD(X)=X>0
260 WHILE FNEOD(NTC)
270   SCALL RETRIEVE(WRD$,LN)
280   SCALL WPRT(WRD$,LN)
290 WEND
300 '
310 CLOSE 1
320 '
330 END
340 '---------- INPUT ----------
350 SUBROUTINE GETWL(WD$,SN)
360 '   WD$:word
370 '   SN :line number
380 '
390   COMMON TAB$,TAB%,NT,INL$,ALS$
400 '   TAB$:word table
410 '   TAB%:linenumber table
420 '   NT  :word table counter
430 '   INL$:line input data
440 '   ALS$:characters contained in words.
450 '
460 '
470   WD$="":W1$=" "
480   WHILE W1$=" "
490     GOSUB*GETCH
500   WEND
```

図 16—1

```
510   SFLAG=INSTR(ALS$,W1$)
520   IF SFLAG=0 THEN WD$="":GOTO *LGET
530   WHILE SFLAG<>0
540     WD$=WD$+W1$
550     GOSUB*GETCH
560     SFLAG=INSTR(ALS$,W1$)
570   WEND
580 '
590 *LGET
600 SRETURN
610 '
620 *GETCH
630   IF INL$="" THEN GOSUB*LINPT
640   W1$=MID$(INL$,NCH,1)
650   NCH=NCH+1
660   IF W1$=CHR$(13) THEN INL$=""
670  RETURN
680 '
690 *LINPT
700   LINE INPUT #1,INL$
710   INL$=INL$+CHR$(13)
720   NCH=1
730   GOSUB*LNUM
740  RETURN
750
760 *LNUM
770   SN$="":W1$=""
780   WHILE W1$<>" "
790     GOSUB*GETCH
800     SN$=SN$+W1$
810   WEND
820   SN=VAL(SN$)
830  RETURN
840 '
850 SEND
860 '---------- INNER TABLE ---------
870 SUBROUTINE CHANGEMODE
880   COMMON TAB$,TAB%,NT
890 '     TAB$:word table
900 '     TAB%:linenumber table
910 '     NT  :word table counter
920 '
930   GOSUB*SORT
940   GOSUB*CHNG
950 '
960 SRETURN
970 '
980 *SORT
990   FOR I=NT TO 2 STEP -1
1000     FOR J=1 TO I
```

図 16—2

```
1010        IF TAB$(J)<TAB$(J+1) THEN SWAP TAB$(J),TAB$(J+1):
                                      SWAP TAB%(J),TAB%(J+1)
1020      NEXT J
1030    NEXT I
1040   RETURN
1050
1060 *CHNG
1070    I=NT
1080    WHILE I>1
1090      A$=TAB$(I)
1100      J=1
1110      WHILE TAB$(I-J)=A$
1120        TAB$(I-J)=""
1130        J=J+1
1140      WEND
1150      I=I-J
1160    WEND
1170   RETURN
1180 '
1190 SEND
1200 '
1210 '
1220 SUBROUTINE ENTWL(WD$,SN)
1230 '    WD$:word   SN:linenumber
1240 COMMON TAB$,TAB%,NT
1250 '    TAB$:word table
1260 '    TAB%:linenumber table
1270 '    NT  :word table counter
1280 '
1290    IF WD$="" THEN *LENT
1300    NT=NT+1
1310    TAB$(NT)=WD$
1320    TAB%(NT)=SN
1330 '
1340 *LENT
1350 SRETURN
1360 SEND
1370 '
1380 '
1390 SUBROUTINE RETRIEVE(WD$,SN)
1400 '     WD$:word   SN:linenumber
1410   COMMON TAB$,TAB%,NT
1420 '     TAB$:word table
1430 '     TAB%:linenumber table
1440 '     NT  :word table counter
1450 '
1460    WD$=TAB$(NT)
1470    SN=TAB%(NT)
1480    NT=NT-1
1490 '
```

図 16—3

```
1500 SRETURN
1510 SEND
1520 '---------- OUTPUT ----------
1530 SUBROUTINE WPRT(WD$,SN)
1540 '    WD$:word
1550 '    SN :linenumber
1560 '
1570    IF WD$<>"" THEN LPRINT :LPRINT WD$,SN; ELSE LPRINT SN;
1580 '
1590 SRETURN
1600 SEND
```

図 16—4

```
10 REM THIS IS AN EXAMPLE
20 '
30 A=12.5:B=13.5
40 A%=A:B%=B
50 SUM=A+B          :PROD=A*B
60 SUM%=A%+B%       :PROD%=A%*B%
70 PRINT SUM,SUM%,PROD,PROD%
80 '
90 END
```

図 17

```
12.5            30
13.5            30
A               50 50 40 30
A%              60 60 40
AN              10
B               50 50 40 30
B%              60 60 40
END             90
EXAMPLE         10
IS              10
PRINT           70
PROD            70 50
PROD%           70 60
REM             10
SUM             70 50
SUM%            70 60
THIS            10
```

図 18

参考文献

1) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: Data Structures and Algorithms, Addison-Wesley (1983)
2) Guttag, J.: Abstract Data Types and the Development of Data Structures, CACM 20, pp.396-404 (1977)
3) Guttag, J., Horowitz, E. and Musser, D.R.: Abstract Data Types and Software Validation, CACM 21, pp.1048-1063 (1978)
4) 稲垣，坂部：抽象データタイプの代数的仕様記述法の基礎，(1)—情報処理 Vol.25, No.1 pp.47-53 (1984), (2)—Vol.25, No.5, pp.491-501 (1984)
5) Isner, J.F.: A Fortran Programming Methodology Based on Data Abstraction, CACM 25, pp.686-697 (1982)
6) Kernighan, B. W. and Plauger, P. J.: ソフトウェア作法，共立出版 (1981)
7) 木村，米沢：算法表現論，岩波講座 情報科学—12 (1982)
8) Liskov, B. Snyder, A, Atkinson, R aand Schaffert, C: Abstraction Mechanisms in CLU, CACM 20, pp.564-576 (1977)
9) Loveman, D.B.: Program Improvement by Source-to-Source Transformation, JACM 24, pp.121-145 (1977)
10) Mitchell, R.J.: Program Design-A Practical Approach, Lec. Notes in CS 152, pp.132-145 (1982)
11) Parnas, D.L.:On the Criteria To Be Used in Decomposing Systems into Modules, CACM 15, pp.1053-1058 (1972)
12) 紫合：ソフトウェア設計法，ソフトウェア科学 Vol. 1, No.2, pp.55-68 (1984)
13) 鳥居，二木，真野：プログラミング方法論の展望，情報処理 Vol.20, No.1, pp.22-43 (1979)