九州大学学術情報リポジトリ
Kyushu University Institutional Repository

PASCAL入門 : PASCAL8000に沿って

疋田, 輝雄
東京大学理学部情報科学科

安村, 通晃
東京大学理学部情報科学科

石畑, 清
東京大学理学部情報科学科

https://doi.org/10.15017/1472538

出版情報：九州大学大型計算機センター広報. 11 (3), pp.133-190, 1978-09-01. 九州大学大型計算機センター
バージョン：
権利関係：

PASCAL入門

―― PASCAL8000に沿って ――

正田　輝雄*，安村　通晃*，石畑　清*

目　　次

はじめに …… 134

1　PASCAL事始め …… 134
2　ブロック構造 …… 139
3　宣　言 …… 142
4　データ型――基本型 …… 144
5　文 …… 148
6　配列型 …… 155
7　レコード型 …… 157
8　集合型 …… 159
9　ファイル型 …… 160
10　ポインタ型 …… 162
11　手数きと関数 …… 165
12　入出力 …… 171
13　プログラム例 …… 173

付1　PASCAL8000とStandard PASCALとの言語仕様上の相違 …… 178
付2　標準手続きおよび関数 …… 179
付3　演算子 …… 181
付4　システムで定義されている語 …… 183
付5　カタログドプロシジャ …… 184
付6　PASCAL8000システムの使用法 …… 185
付7　コンパイラオプション …… 186
付8　ファイルの使用法 …… 187
付9　エラーメッセージ …… 188

参考文献 …… 190

*東京大学理学部情報科学科

はじめに

PASCALは，スイスの連邦工科大学のWirthによって設計されたALGOL系のプログラミング言語です。データの構造を記述する機能が豊富なこと，繰返しなどのプログラムの流れを記述しやすいこと，言語仕様が簡潔で理解しやすいことなどが特徴です。structured programmingとの関連で注目を集め，教育用，研究用を中心にして広く使われています。

PASCALのコンパイラは，数多く作られていますが，九州大学の大型計算機センターのPASCALは，東京大学のセンターのHITAC 8800／8700で最初に開発されたものを移植したもので，PASCAL 8000と呼ばれます。PASCAL 8000の言語仕様は，Standard PASCAL(文献(1), (2))とほぼ同じですが，いくつか変更した点があります。この入門では，Standard PASCALに沿って解説し，PASCAL 8000の変更点は付録1にまとめてあります。

1. PASCAL 事始め

本節は1つのプログラム例にそって，PASCALプログラムで基本となる，行と文，変数の宣言，名前とシンボル，および簡単な入出力等について述べようと思います。

1.1 プログラム例

年率7％，8％または10％でインフレが進行するとして，円の価値が10年の間でどう変わっていくかを求めるプログラムをPASCALとFORTRANで書いてみました。（プログラム1および2）

```
LOG    16104 A003501         0035 : START    TIME=17.16.41
LOG    16104 A003501         0035 : END      TIME=17.16.47

      OSIV/F4  E20C  V08L06            <<<  JCL STATEMENTS AND SYSTEM MESSAGES LIST  >>>      DATE  07/21/78   TIME  17:16

//A003501  JOB F0035,#####,REGION=1024K,CLASS=A
//  EXEC PASCAL
//*PASSRCE DD *
JDJ142I - STEP WAS EXECUTED - COND CODE 0000
JDJ373I STEP /PASCAL  / START 78202.1716
JDJ374I STEP /PASCAL  / STOP  78202.1716 CPU    0MIN 00.24SEC STOR VIRT   256K
JDJ375I  JOB /VA003501/ START 78202.1716
JDJ376I  JOB /VA003501/ STOP  78202.1716 CPU    0MIN 00.24SEC
     1    (*ASSUMING ANNUAL INFLATION RATES OF 7, 8, AND 10 PER CENT,
     2      FIND THE FACTOR BY WHICHI THE YEN WILL HAVE BEEN DEVALUED
     3      IN 1, 2, ...N YEARS.*)
     4    PROGRAM INFLATION(OUTPUT);
     5    CONST N = 10;
     6    VAR  I : INTEGER; W1,W2,W3 : REAL;
     7    BEGIN  I := 0;  W1 := 1.0;  W2 := 1.0;  W3 := 1.0;
     8       REPEAT  I := I+1;
     9          W1 := W1 * 1.07;
    10          W2 := W2 * 1.08;
    11          W3 := W3 * 1.10;
    12          WRITELN(I:2,'   ',W1:13,'   ',W2:13,'   ',W3:13)
    13       UNTIL I=N
    14    END.

 1     1.07000E+00     1.08000E+00     1.10000E+00
 2     1.14490E+00     1.16640E+00     1.21000E+00
 3     1.22504E+00     1.25971E+00     1.33100E+00
 4     1.31080E+00     1.36049E+00     1.46410E+00
 5     1.40255E+00     1.46933E+00     1.61051E+00
 6     1.50073E+00     1.58687E+00     1.77156E+00
 7     1.60578E+00     1.71382E+00     1.94872E+00
 8     1.71819E+00     1.85093E+00     2.14359E+00
 9     1.83846E+00     1.99900E+00     2.35795E+00
10     1.96715E+00     2.15892E+00     2.59374E+00


     PASCAL TIMING SUMMARY:

         COMPILE TIME =       0.15
         EXECUTE TIME =       0.04
```

プログラム1

```
FACOM OSIV/F4 FORTRAN IV (GE)  V03L03            DATE 78.07.21  TIME 17.23.53

      C
      C  ASSUMING ANNUAL INFLATION RATES OF 7, 8, AND 10 PER CENT,
      C  FIND THE FACTOR BY WHICH THE YEN WILL HAVE BEEN DEVALUED
      C  IN 1, 2, ... N YEARS.
      C
000001        N = 10
000002        W1 = 1.0
000003        W2 = 1.0
000004        W3 = 1.0
000005        DO 10 I=1,N
000006        W1 = W1 * 1.07
000007        W2 = W2 * 1.08
000008        W3 = W3 * 1.10
000009        WRITE(6,100) I,W1,W2,W3
000010    100 FORMAT(1H ,I2,3X,E12.6,3X,E12.6,3X,E12.6)
000011     10 CONTINUE
000012        STOP
000013        END

  1   0.107000E+01   0.108000E+01   0.110000E+01
  2   0.114490E+01   0.116640E+01   0.121000E+01
  3   0.122504E+01   0.125971E+01   0.133100E+01
  4   0.131079E+01   0.136049E+01   0.146410E+01
  5   0.140255E+01   0.146933E+01   0.161051E+01
  6   0.150072E+01   0.158687E+01   0.177156E+01
  7   0.160577E+01   0.171382E+01   0.194872E+01
  8   0.171818E+01   0.185092E+01   0.214359E+01
  9   0.183845E+01   0.199900E+01   0.235795E+01
 10   0.196714E+01   0.215892E+01   0.259374E+01
```

プログラム2

1.2　行と文

2つのプログラムを比べてみて最初に気づくことは，通常のFORTRANでは1行に1つの文しか書けないのに対し，PASCALではセミコロン(;)を間に挿むことにより，1行の中にいくつも文字が書けるということです。またFORTRANでは通常，文は第7コラムから書かなければなりませんが，PASCALではどのコラムからはじめてもかまいません。これはFORTRANがもともとカードを基本とした言語であるのに対し，PASCALがテープを基本とした言語ともいえることからきた相違でしょう。PASCALプログラムの実行結果のあとには，コンパイルおよび実行に要した時間が秒単位で表示されています。

1.3　変数の宣言

FORTRANには暗黙の型宣言というものがあり，IからNまでの英字で始まる名前は整数，それ以外の英字で始まる名前は実数の型であるということが決まっていて，この暗黙の型宣言に従うものは宣言しなくてもよいことになっていますが，PASCALではプログラムの中で使う名前（ identifier ）はすべてあらかじめ宣言しなければなりません。このうち，値の変わらない定数（ constant）は CONST 以下でその値を，変数（ variable）は ＶＡＲ以下でその型（ type） を宣言します。プログラム例ではNを値が10の整数の定数として,また，Iを整数（ integer), W1,W2,W3を実数（ real ） の型の変数として宣言しています。

1.4　プログラムの構造

PASCALのプログラムは，データを表わす宣言部（ declarations）と，文の集まりで実際に実行される実行部（ statements ）とに分れます。この2つを合わせてブロック（block ）とよびます。

宣言部には，上で述べた定数（CONST ）や変数（VAR）の宣言の他に，ラベル(LABEL)や型（TYPE）の宣言と，FORTRANのサブルーチンや関数に相当する手続き（PROCEDURE）や関数（FUNCTION）の宣言があります。

実行部はBEGINとENDの中にいくつかの文を含んだものです。各文の種類や機能については次節以降に述べます。

プログラムは，一つのブロックの前にプログラム名やファイルを宣言するプログラム頭部(program heading)を，そして最後にピリオド(.)を加えたものです。

以上大雑把に述べたプログラムの構造を構文図式(syntax diagram)とよばれるもので厳密に表現すると，図1のようになります。

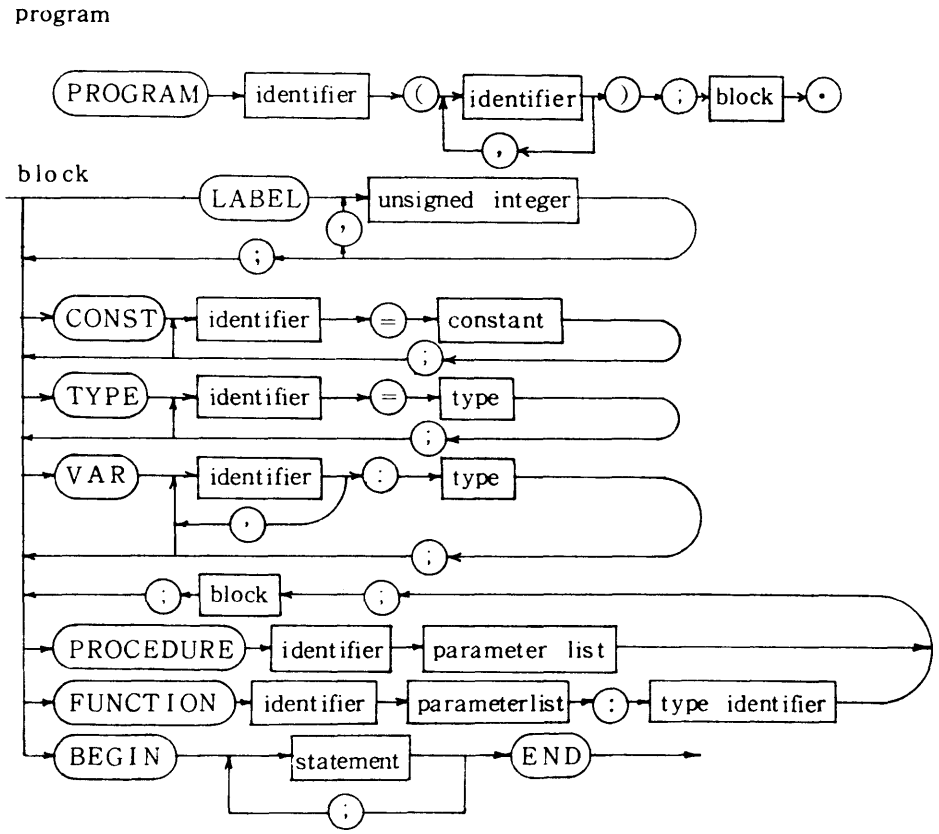


図1　プログラム及びブロックの構文図式

1.5　名前(identifier)と予約語(reserved word)

FORTRANでは通常名前は英字で始まる英数字6文字以内で，文字と文字との間に空白がはいってもかまいません。それに対してPASCALでは名前は英字で始まる英数字で，8文字をこえてもかまいませんが，その分は無視されます。たとえば

JUGEMJUGEMGOKONOSULIKIRE

というのは正しい名前ですが，

JUGEMJUGEM2

という名前と同じであるとみなされます。

次に，予約語としてPASCAL プログラムの中で演算子やプログラムの構文の区切り符号などの特別なきまった意味をもつものはユーザが定義する変数名，定数名，手続き名などとして使うことはできません。たとえば例にあげたPASCALプログラムの中では，PROGRAM, CONST, VAR, BEGIN, REPEAT,

UNTIL, END が予約語です。

また，標準手続き，標準関数，標準型などの名前としてあらかじめ定義されている名前もあります。例にあげたプログラムでは

OUTPUT, INTEGER, REAL, WRITELN

があらかじめ定義された名前です。これらの予約語やあらかじめ定義された名前の一覧表は付録4にあります。

ちなみに，REPEAT…UNTIL というのは，繰り返し文といって，…の部分を UNTIL の後に示された条件が満たされるまで繰り返せ，という実行を指示する文です。この例の場合は FORTRANではDO文で表わされます。FORTRANのDO文に対応する，PASCAL の繰り返しを示す文としては，このrepeat 文の他に for 文，while 文があります。

1.6 PASCALの用語

PASCAL の中で使われる英文字 (letter) と数字 (digit)，および特別なシンボル (specialsymbol) をBNF (Backus-Naur Form) という方法で表現すると下の表の通りです。（表1）

例にあげたプログラムの中で，代入の記号として：＝が使われていることに気づかれたでしょう。PASCAL では，値の比較の場合に用いる等号（＝）と区別するために，代入には：＝ (colon - equal あるいは becomes と読む）を使います。

また，この例ではでてきませんが，PASCAL 8000 では Standard PASCALの仕様と異なり角カッコの代用記号として，〔のかわりに (.を，〕のかわりに.）を用います。

＜ letter ＞ ::＝A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

＜ digit ＞ ::＝0|1|2|3|4|5|6|7|8|9

＜ special symbol ＞ ::＝+|−|＊|／|＝|＜＞|＜|＞|＜＝|＞＝|(|)|(.|.)|(＊|＊)|
:＝|.|,|;|:|▼|@|
div|mod|in|or|and|not|if|then|else|case|of|repeat|until|while|
do|for|to|downto|begin|end|with|goto|const|var|type|array|record|
set|file|function|procedure|label|packed|program

表1　PASCAL シンボル一覧表

1.7 コメント（注釈）

プログラム例の最初の部分のように，（＊と＊）とではさまれた文字列全体がコメントになります。FORTRAN では第1コラムにCをかくとその行全部がコメントになりますが，PASCAL ではコメントは空白の許されるところなら行のどの部分においても，また2行以上にわたってもかまいません。

1.8 入出力

PASCAL の入出力は，FORTRAN と違い FORMAT 文を伴うことはありません。入出力する変数の型（実数型，整数型，あるいは文字列型など）により自動的に変換が行なわれます。出力の場合けた数はコロン(:)のあとの数字によって指定します。この数は変数や式でもかまいません。この指定を省略すると，整数型の時は12けた，実数型の時は24けたを指定したことになります。例では整数は2けた，実数

は13けた出力させています。

文字列型というのは，この例では3ケタの空白を出力させるのに使われていますが，一般に引用符(▼)ではさまれた文字列(string)のことをいいます。文字列中に現われる文字はPASCAL の文字セットの中のものなら何でもかまいません。引用符自身を含めたい時はそれを2つ重ねてかきます。(たとえば▼DON▼▼T▼というようにかきます。)プログラム例でできた文字列型の出力は，FORTRAN でいえばH変換に相当します。

出力に使われる標準手続きにはWRITEとWRITELNの2種類があります。前者を使うと，FORTRANと違って一回の出力のあと改行しません。一行分の出力を何回にも分けて行なうことができます。後者では引数をすべて出力したあと改行を行ないます。プログラム例ではWRITELNが使われています。

2. ブロック構造

PASCAL プログラムは，その中で使われる名前 (identifier) の宣言部と，実行文の集まりである実行部とからできています。このような，宣言部と実行部とからできているものをブロック (block) とよびます。その宣言部の中で，手続き (procedure) や関数 (function) を宣言することができますが，これら手続きや関数それ自体も，宣言部と実行部とからなるブロックでできています。このように１つのプログラムはブロックの入れ子 (nesting) の構造をもっています。

図２で四角で囲った部分が１つのブロックです。

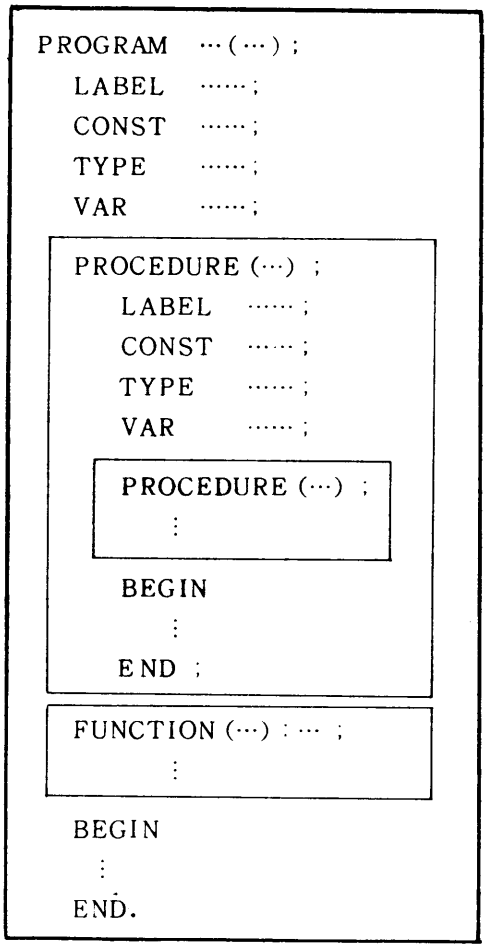


図２　プログラムの構造

１つの手続きや関数の宣言部で宣言された名前は，その手続きや関数に対してローカルであるといい，これらの名前はこの手続きや関数の中でのみ意味をもちます。この意味をもつ範囲を，その名前のスコープ (scope) といいます。ある手続きや関数に対して，その外側で宣言された名前はグローバルであるといいます。

図３を例にとって，あるブロックで宣言された名前が意味をもつブロックを示しますと，次のようになります。

名前の宣言されたブロック	その名前が意味をもつブロック
M	M, P, A, B, Q, R, S,
P	P, A, B
A	A, B
B	B
Q	Q, R, S
R	R
S	S

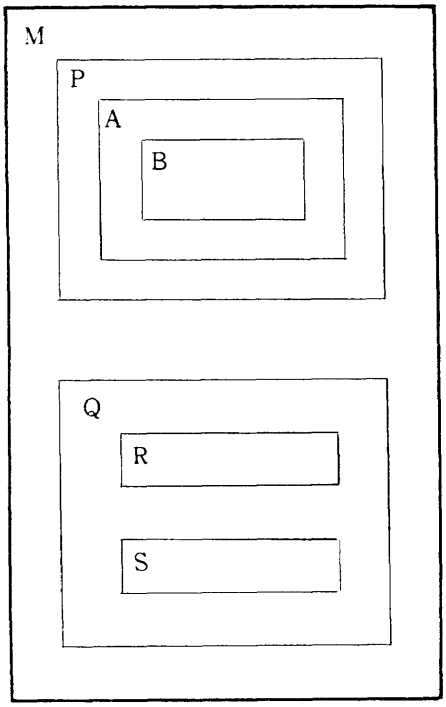


図3　ブロック構造

同じ名前が2つ以上のブロックで宣言されている時は，あるブロックに現われた名前の定義は，そのブロックはその名前の宣言に従い，それより外側のブロックでのその名前の宣言とは関係ありません。たとえば，次のプログラムの手続きBの中で変数H，I，J，K，Lが使われていますが，このうちJ，K，はこの手続きBにローカルな変数で，変数Iは手続きAで宣言されたグローバルな変数，変数Hは主プログラムで宣言されたグローバルな変数で，これらはいずれもそれぞれの変数のスコープの中にはいっていて有効です。しかし変数Lは，手続きCで宣言されていてそのスコープにはいっていないので，未定義のエラーとなります。

```
PROCRAM… ;
  VAR H, I: INTEGER;
  PROCEDURE A;
   VAR I, J: INTEGER;
  PROCEDURE B;
   VAR J, K: INTEGER;
   BEGIN H:=I+J;K:=H*L;…
   END;
  PROCEDURE C;
   VAR L: INTEGER;
   BEGIN…
   END;
  BEGIN…
  END;
 BEGIN…
 END.
```

3. 宣　言

PASCAL のプログラムに現われる宣言にはプログラム頭部とブロックの中の宣言部とがあります。プログラム頭部は，プログラム名と，外部とやりとりするファイルの名前とを宣言するものです。たとえば

```
PROGRAM FILEPROCESS(F, OUTPUT);
```

のようにかきます。プログラム名は他の部分との関係は全くなく，なにをかいても自由です。ファイル名に関しては付録8を参照して下さい。

ブロックの中の宣言部は，前節に述べたように，次の各部分からできています。

(1) ラベル宣言部
(2) 定数定義部
(3) 型定義部
(4) 変数宣言部
(5) 手続きと関数宣言部

各部分は，必要ない時は欠けてもかまいませんが，順序はこの順をかえてはいけません。以下各部分について説明します。

3.1 ラベルの宣言

ここでは，そのブロックの実行部で，文の前にたてられるラベル（整数）を宣言します。PASCALでは使われるラベルはすべて宣言しなければなりません。宣言は次のようにします。

```
LABEL 1, 2, 100;
```

3.2 定数の定義

これは，プログラムの中で，定数を用いるかわりにそれに名前をつけてその名前をかわりに用いようというものです。こうすることによってプログラムが見やすくなり，また定数の値自身を変更する際に1ヶ所だけ変えればよいことになります。

定数は数か，文字列か，すでに定数として定義された名前（の前にさらに符号がついてもよい）です。その定義はたとえば次のようになります。

```
CONST   PI=3.14159265;
        MG=8000;
        FK=-MG ;
        FIN='THE END';
```

3.3 型の定義

PASCAL においては，基本的なデータの型の他に，それらからより複雑な型を構成していく手段がいろいろ用意されています。また，そうして構成した型にユーザが名前をつけて，変数の宣言の時などにその名前を用いることができます。これはPASCAL の特徴の1つです。

型の定義は次のようにします。

```
TYPE   T1= ARRAY(.1..10.) OF INTEGER;
       T2= ARRAY(.0.. 5.) OF REAL
```

ARRAYはFORTRANのDIMENSIONにあたります。型T1をこのように定義しておけば，変数の宣言において，

```
VAR M: ARRAY(.1..10.)OF INTEGER;
```

とかくかわりに（これはFORTRANのINTEGER M(10)にあたります），

```
VAR M: T1;
```

ですみます。

ARRAYの他にもRECORD, SET, FILE, POINTER等，型を構成していく方法がありますが，これらについては後で詳しく説明します。

3.4 変数の宣言

変数はすべて，変数宣言部で宣言します。宣言の例をあげますと，

```
VAR ROOT1, ROOT2, ROOT3,: REAL;
COUNT: INTEGER;
FOUND: BOOLEAN;
FILLER: CHAR;
```

同じ型の変数は，間にコンマ(,)をはさんで並べておけばよいのです。コンロ(:)の後に変数の型をかきます。上の例では標準型といわれるあらかじめ用意された型ばかり現れていますが，もちろんここにユーザが定義した型をかいてもよいわけです。

3.5 手続きと関数の宣言

手続きや関数は，それらが実行部でよばれる前に宣言しておかなければなりません。（ただしFORWARDの宣言をしておくと，プログラム上でよばれる後に定義することができます。）詳しい宣言の仕方の説明は後にまわすことにします（第11節）。

4. データ型 —— 基本型

RASCAL はデータ型の種類の豊富なことが大きい特徴で，基本的な（簡単な）データ型から構造をもった型をいろいろ構成していくことができます。構成していく方法（data structuring）として，ARRAY，RECORD, SET, FILE, POINTER がありますが，それらの説明は後に譲って，ここでは基本的な型（unstructuctured type）について述べます。

基本的な型には　標準的な（あらかじめ定められた）型（standard scalar types）が4つ，すなわち論理型，整数型，実数型，文字型と，ユーザが定義するスカラー型（scalar type）と部分範囲型（subrange type）とがあります。後の2つはPASCAL独特の型であるといえます。

一般にそれぞれの基本型の要素の間には大小関係が定められていて，比較の演算子＝，＜＞，＜，＜＝，＞＝，＞を用いることができます。比較の演算子は論理型の値をもちます。それぞれの意味は自明でしょう。（＜＞は not equal の意味です。）また各型の変数や定数の間の演算のためにいろいろの演算子や標準関数が定義されています。

4.1 論理型（Boolean type）

値としてTRUEかFALSE をとるデータ型を論理型といいます。この型の変数の宣言は

```
VAR P1,P2,P3 : BOOLEAN;
```

のように書きます。

論理型の変数や定数（TRUEとFALSE）の間の演算子として，AND, OR, NOT があり，それぞれFORTRANの．AND．，．OR．，．NOT．にあたります。意味は自明でしょう。

TRUEとFALSE との間の大小関係はFALSE＜TRUEと定められています。従って論理型の変数や定数の間に大小関係の演算子＝，＜＞，＜，＜＝，＞＝，＞を用いることができます。

これらを用いると，たとえば論理変数P，Qの

同値（equivalence）は　　P＝Q

排他的OR（exclusive or）は　P＜＞Q

で表わすことができます。

論理型の値をもつ標準関数としては

ODD (I) 整数型の変数Iに対して，Iが奇数の時TRUE，偶数の時FALSE の値をとります。
たとえばODD(8)の値はFALSE，ODD(32767)はTRUE です。

EOLN(F) ファイル　Fが end of line の状態にあるか否か。

EOF (F) ファイル　Fが end of file の状態にあるか否か。

の3つがあります。

4.2 整数型（integer type）

これはFORTRANの整数型と同じものです。整数型の変数の宣言は次のようになります。

（先頭の文字がI～Nでも宣言は必要です。）

```
VAR I1, I2 : INTEGER;
```

我々のPASCAL システムでは整数は単精度だけですから，取扱える値の範囲は次の通りです。

$-2^{31}=-2147483648 \sim 2^{31}-1=2147483647$

整数型の変数，定数に対する演算子としては，

＊ DIV MOD ＋ －

があります。ここでDIVは割算で，FORTRANでは整数型に対しても実数型に対しても／を用いますが，PASCALでは整数型割算にはDIVを用います。またMODは剰余を示します。

つまり

A MOD B＝A－(A DIV B)＊B

次に大小関係の演算子として（FORTRANの.EQ.,.NE.等にあたるものとして），＝，<>，<，<=，>=，>があり，その結果は論理型です。

整数型の値をとる標準関数としては次の4つがあります。

ABS(I)	Iの絶対値
SQR(I)	Iの2乗
TRUNC(X)	Xは実数型で，TRUNC(X)の値はXの整数部分。たとえばTRUNC(3.7)=3. TRUNC(−3.7)= −3
ROUND(X)	実数Xを四捨五入した値

4.3 実数型（real type）

これもFORTRANとかわりのない型で，宣言は

VAR X：REAL;

のようにします。精度は我々のシステムでは倍精度（FORTRANのREAL＊8）になっています。

実定数は，次の3つのうちのどれか1つの形です。

3.14　　5.772E−1　　1E−10

たとえば10.はいけません。

実数型の定数や変数に対する演算子としては

＊　／　＋　－

があり，これらの意味は自明でしょう。（FORTRANのべき乗＊＊に相当するものはありません）

値が実数型の標準関数としては

ABS(X)	Xの絶対値
SQR(X)	Xの2乗
SIN(X)	正弦関数
COS(X)	余弦関数
ARCTAN(X)	逆正接関数
LN(X)	自然対数
EXP(X)	指数関数
SQRT(X)	平方根

があります。はじめの2つを除いて，Xは整数型でも実数型でも結果は実数型となります。

一般にPASCALでは，実数型の式が予想されるところに整数型の式をかくことが許され，その際にその式の実数化が自動的になされます。上の標準関数（ABS，SQRを除く）の例でも，実数を引数とする

関数に整数の引数を使ってもよいわけです。

こういう整数型から実数型への型の変換は，式の内部や，代入文や，関数と手続の引数などにおいて行なわれます。従ってFORTRANのFLOAT(I)という実数化の関数は不要です。

逆の，実数の整数化は自動的にはできません。たとえばXを実数型，Iを整数型の変数として，

```
X：＝1；
```

や

```
I／X
```

は正しい例ですが，

```
I：＝X；
```

はいけません。標準関数TRUNCまたはROUNDを用いて

```
I：＝TRUNC(X)；
```

などとしなければなりません。

4.4　文字型（ character type ）

文字を値にとる変数や定数を文字型といいます。定数は，(▼)でかこんで，▼A▼，▼9▼，▼*▼のように表わします。変数は

```
VAR C：CHAR；
```

のように宣言します。

文字型の変数，定数に対して2つの標準関数があります。

ORD(C)　文字型のCに対して整数型の値をとる。
その値はEBCDICの内部表現。

CHR(I)　ORD(C)の逆関数。従ってIは整数型でCHR(I)は文字型です。

たとえば，ORD(▼A▼)＝193，　CHR(240)＝▼0▼

です。ただしこれらの値はPASCALの処理系によって異なりますから注意が必要です。文字型の値C_1とC_2の間の大小関係はORD(C_1)とORD(C_2)の間の大小関係と同じであると定められています。たとえば，▼A▼＜▼Z▼の値はTRUEです。

4.5　スカラー型（ scalar type ）

スカラー型はユーザが変数のとりえる値をすべて（有限個）並べあげることによって定義する型です。ここで値は名前(identifier)，つまり英数字の列で先頭が英字であるものです。たとえば

```
TYPE　　DAY＝(MON, TUES, WED, THUR, FRI, SAT, SUN)；
　　SEX＝(MALE, FEMALE)；
```

というように定義します。

スカラー型の値に対しては大小関係が定義における並びの順序の前の方ほど小（たとえばMON＜FRI）と定められています。従ってスカラー型の変数D1, D2, D3,を

```
VAR D1, D2, D3：DAY；
```

と宣言すれば，この変数D1, D2, D3に対して大小関係を演算子＝，＜＞，＜，＜＝，＞＝，＞が使えます。もちろんスカラー型の値の代入も可能です。たとえば

```
D1 ：＝ＳＵＮ；
D2 ：＝Ｄ１；
D3 ：＝ＴＵＥＳ；
P　：＝Ｄ２＜Ｄ３；（ここでＰは論理型の変数）
```

またスカラー型の変数に対する標準関数として

SUCC(X)	Xの次の元。たとえばSUCC(MON)=TUES
PRED(X)	Xの直前の元。たとえばPRED(SUN)=SAT
ORD(X)	スカラー型の値の並びの，前から順に0, 1,…， n−1の値をとる。 たとえばORD(SUN)=6

があります。

4.6　部分範囲型（subrange type）

この型は，その型の変数のとる範囲として，今までにでてきた標準型およびユーザの定義するスカラー型の部分範囲を指定することによって定義されます。（ただし実数型は除きます。）定義するには，その範囲の下限と上限を‥をはさんで示します。たとえば

```
TYPE WORKDAY=MON..FRI；
VAR POINT：0..100；
```

のようになります。部分範囲型の変数に対して，もとの型に対する演算子が（結果がその範囲を出ない限り）まったく同じように使えます。

5. 文

本節はプログラムの実行部の基本要素である文(staement)について述べます。文は，代入文などの基本文(simple statement)と，複合文，条件文，繰り返し文などの構造化文(stuructured statement)に大きく分けることができます。

5.1 代入文

文の中で最も基本となるのが代入文で，新しく計算された式の値を変数に代入することを指示します。代入文の形式は次の通り。

＜variable＞：＝＜expression＞

ここで代入の演算子としてFORTRANの場合と違い：＝(colon equal)を使いますが，これで比較演算子＝と区別することができます。たとえば，

A：＝B＝C；

は，変数BとCを比較してその結果の論理値を変数Aにしまうことを意味します。

代入文の右辺にあらわれる式(expression)は，定数，変数，関数名が演算子と組み合わされてできるもので，その評価の順序は，演算子間に順位がある場合にはその順位に従い，そうでない(即ち等しい順位の)場合には左から右へと評価されます。演算子順位(operator precedence)とは，演算子とオペランドの結びつきの強さを定めたもので，PASCAL では次の様になっています。

最も強い演算子	NOT
次に強い演算子	*, ／, DIV, MOD, AND
次に強い演算子	+, −, OR
最も弱い演算子	＝, ＜＞, ＜, ＜＝, ＞, IN

演算子順位

また，カッコ(parentheses)でかこまれた式は，その前後の演算子よりも先に評価されます。

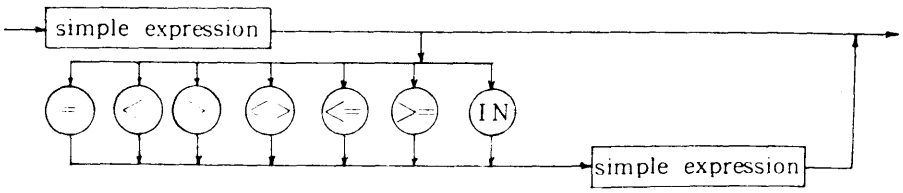
例

15 DIV (2+1) → 15 DIV3 → 5

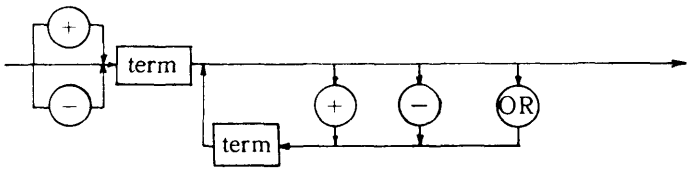
5+3*2>8 → (5+(3*2))>8 → 11>8 → TRUE

式の一般的な形式を構文図式で表現すると次のようになります。

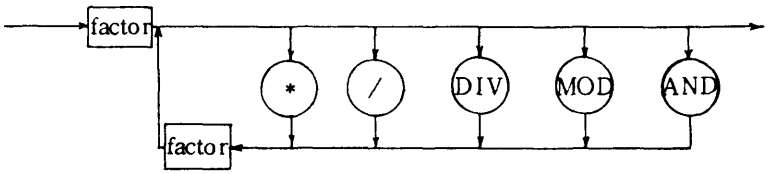
expression



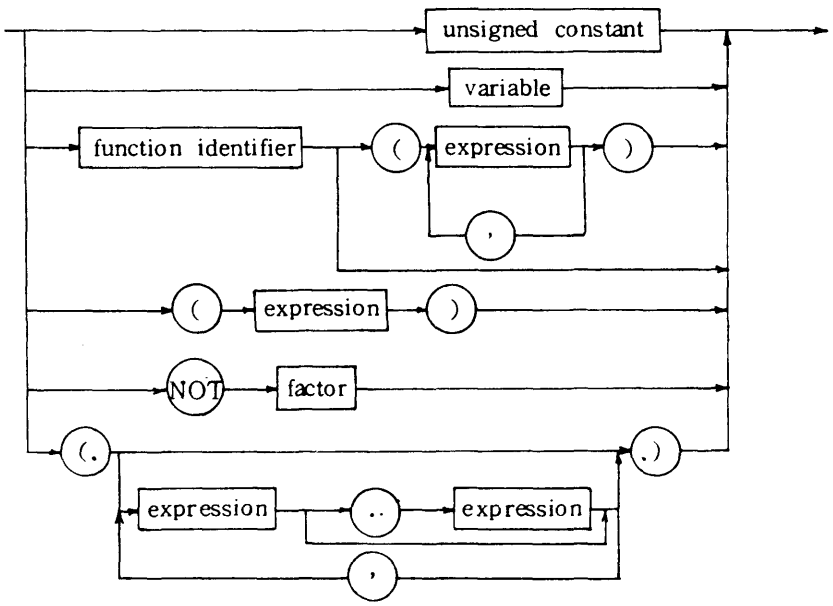
simple expression



term



factor



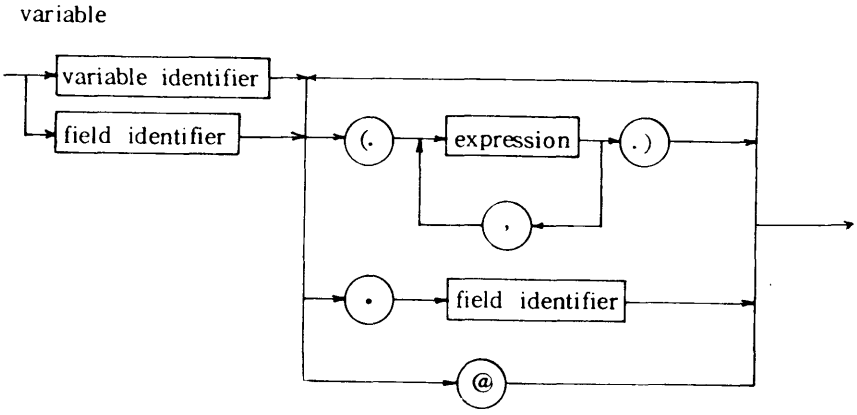


図4　式（expression）の構文図式

なお，論理型演算子のANDとORは，これらの演算子の左側のオペランドだけで全体の式の値が決定されてしまう場合でもやはり右側のオペランドの評価をおこないますから注意が必要です。たとえば

```
VAR A:ARRAY(. 1..10.)OF INTEGER;I:INTEGER;
BEGIN I:=1;
  REPEAT READ(A(. I. ));  …;I:=I+1
  UNTIL  (I>10)OR(A(. I. )=0)
END
```

というようなプログラムで，Iが10を越えたとき，ORの左側ですでにTRUEなのですが，右側も評価するので配列の添字の範囲をこえてしまいます。

代入文の左辺として許される変数の型は，左辺が実数型で右辺が整数型である場合を除き，右辺の型と一致していなければなりません。型の一致が与えられていれば，前節で述べた基本型の，スカラー型，部分範囲型，BOOLEAN, INTEGER, REAL, CHARの他，より高級な型であるARRAY, POINTER, RECORDなどの変数でも代入することができます。

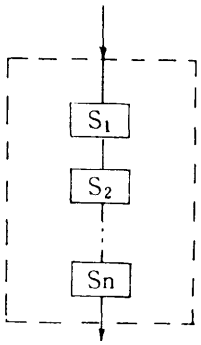
5.2　複合文

いくつかの文をまとめて1つの文として扱いたいときがあります。この場合これらの文をBEGINとENDの2つのシンボルでくくり，これを複合文といいます。

プログラムの実行部本体（body）はそれ自身複合文です。

(例)

```
BEGIN SUM:=X+Y;DIFF:=X-Y;
WRITE(SUM, DIFF)
END
```



複合文の流れ図

PASCALでは，セミコロン(;)は文の区切り記号ですから，ENDの前にはつける必要がありません。もしENDの前にセミコロンをつけた場合は，セミコロンとENDの間に空文（empty statement）があるとみなされますが，別にエラーになるわけではありません。

5.3 条件文

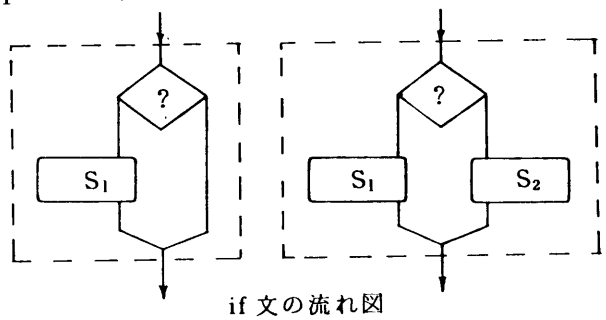
条件によって実行したい文を選択したいときは条件文（if または case 文）を使います。

5.3.1 if 文

ある条件が成り立つときだけ文を実行させることを指示するときには if 文を使います。その条件が成り立たないときに実行すべき文があるときには，ELSEのあとに書きます。次に一般形を示します。

IF＜expression＞THEN＜statement＞

または　IF＜expression＞THEN＜statement＞ELSE＜statement＞



if 文の流れ図

ここで，IFとTHENの間の式は論理型（Boolean）の式でなければなりません。一般形で示したものの前者は，後者のELSEのあとの文が空文である場合と同じとみなされます。

特に注意を要する点は，THEとELSEの後には，1つの文しか許されませんから，2つ以上の文を書きたいときは，それらをBEGINとENDでくくり複合文とすることです。この時ELSEの前にセミコロンをいれてはいけません。いれると文の区切り，すなわち if 文の終わりとみなされます。従って次の文は間違いです。

```
IF P THEN BEGIN S1 ; S2 ; S3 END; ELSE S4 ;
```

また次の文では条件Pで実行する文は空文で，その次の複合文はPの値によらず必ず実行されます。

```
IF P THEN ; BEGIN S1 ; S2 ; S3 END;
```

次に，if 文を重複して使うとき，構文上のあいまいさが生ずることがあります。

(例)

```
IF E1 THEN IF E2 THEN S1 ELSE S2
```

この場合は，「最長一致の原則」に従って，次の文と等価であると解釈されます。

```
IF E1 THEN
   BEGIN IF E2 THEN S1
         ELSE S2
END
```

(例)

```
VAR TRUMP:BOOLEAN;SUIT:(MAJOR,MINOR);SCORE:INTEGER;
   IF TRUMP THEN IF SUIT=MAJOR THEN SCORE:=30
                 ELSE SCORE:=20
   ELSE SCORE:=40
```

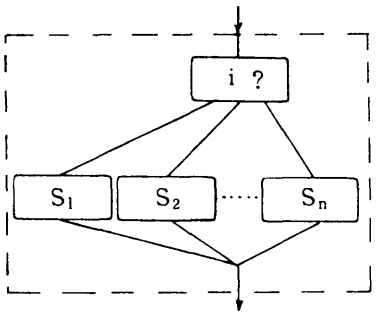
なお，FORTRANには算術ＩＦ文と論理ＩＦ文がありますが，PASCALの if 文をFORTRANであらわそうとすると一般にはGO TOを伴なうはん雑なものになります。

5.3.2 case 文

選択すべき文が多数あるときは case 文を使います。case 文は，選択子 (selector) とよばれる式と，選択子と同じ型の定数 (case label) をもつ文の例からなります。一般形は次の通りです。

```
CASE<expression>OF
<case label list>:<statement>;
        ......
<case label list>:<statement>
END
```

式の値と等しい定数を case label としてもつ文だけが選択されて実行されます。式の値がどの定数とも等しくないときはエラーになります。



case 文の流れ図

ここで式 (selector) の型は実数を除く基本型に限ります。case label はこれと同じ型の定数でなければならず，また後でのべる goto 文が参照する label とは別のものです。

(例1)

```
VAR I:INTEGER;
CASE I OF
 0:X:= 0;
 1:X:= SIN(X);
 2:X:= COS(X);
 3:X:= EXP(X);
 4:X:= LN (X)
END
```

(例2)

```
VAR CH:CHAR;
CASE CH OF
 'A','D': CH:=SUCC(CH)
 'B','E': CH:=PRED(CH)
 'C','F':(*NULL CASE*)
END
```

FORTRANでこの case 文に対応するのは計算型GO TO文ですが，計算型GO TO文では各ラベルのところへ飛んでいったあとは「あとは野となれ山となれ」なので，各文の終りにもとのプログラムの流れに復帰するためGO TO文が必要になります。

5.4 繰り返し文

繰り返しを指示する文は，FORTRANではDO文のみですが，PASCALでは，繰り返しの条件の判定法の違う repeat 文，while 文，for 文の３つがあります。

5.4.1 repeat 文

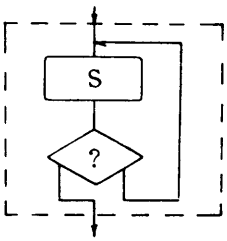
ある条件が成り立つまで (until) 繰り返すことを指示したい時は repeat 文を使います。repeat 文の一般形は，

REPEAT<statement>;<statement>;……;<statement>UNTIL<expression>

REPEATとUNTILの間の文を実行したあと，UNTILの後の式（論理型でなければなりません）の値が評価されます。この値がFALSEならREPEAT文の先頭から実行が繰り返されます。TRUEになったとき繰り返しが終わります。繰り返しの文は少なくとも一回は実行されます。
REPEATとUNTILの間にはいくつ文があってもかまいません。

(例)

```
REPEAT K:=I MOD J;I:=J;J:=K
UNTIL J=0
```



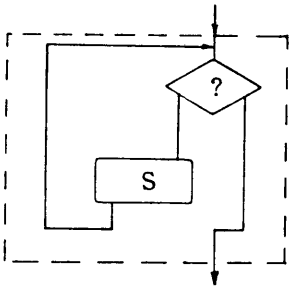
repeat 文の流れ図

5.4.2 while 文

ある条件が成り立っている間(while)繰り返すことを指示したい時はwhile文を使います。while文の一般形は，

WHILE＜expression＞DO＜statement＞

ここで式は論理型(Boolean)でなければなりません。この式は文の各繰り返しの前に評価され，値がFALSEになった時繰り返しが終わります。このため式の値が最初からFALSEである場合，文は一回も実行されません。



while 文の流れ図

```
READ(N);
WHILE N>1 DO BEGIN WRITE(N);
                   IF ODD(N) THEN N:=N*3+1
                   ELSE N:=N DIV 2
              END
```

(問) このプログラムが停止しないNの値は？

5.4.3 for 文

for文はFORTRANのDO文に対応するもので，制御変数が初期値と最終値の間の値を一回ずつとって繰り返しが行なわれます。FORTRANのDO文と違い，制御変数の値を減らしながら繰り返すこともできます。ただしその増加値（減少値）は1と－1だけです。一般形は次の通りです。

FOR＜control variable＞:=＜initial value＞TO＜final value＞DO＜statement＞

または

FOR＜control variable＞:=＜initial value＞DOWNTO＜final value＞DO＜statement＞

ここで，control variable(制御変数)，initial value(初期値)，final value(最終値)はすべて同じ種類の基本型に限られます。必ずしも整数である必要はありません。そしてこれらの値を繰り返し文の中で変更してはいけません。増加型（減少型）で，初期値が最終値より大きい（小さい）とき，繰り返し文は一度も実行されません。

(例)

```
FOR I:=1 TO 100 DO IF A(.I.)>MAX THEN MAX:=A(.I.);
```

5.5　その他の基本文

代入文と空文の他に基本文として手続き文と goto 文があります。

5.5.1　手続き文

手続き (procedure) を呼ぶときは，FORTRANと違ってその手続き名と（あれば）パラメータを並べたものを書くだけです。手続きと関数のことは後でくわしく述べる予定ですが，そのパラメータ（引数）の結合のしかたにはPASCALでは2通りの方法，即ち，値で渡す方法と，アドレスで渡す方法とがあります。（他に手続きや関数の名前を渡す方法もありますが説明は省略します。）前者 (value parameter) では，実引数は式で，後者 (variable parameter) では，実引数は変数でなければいけません。もちろん仮引数との型の一致が要求されます。

(例)

```
…; SEARCH(A, KEY, I) ; WRITE(I, A(␣I␣)); …
```

5.5.2　goto 文

プログラムの流れを変えるのに，通常は上で述べた条件文（if 文と case 文）や繰り返し文（repeat 文，while 文，for 文）を使いますが，これらで表わしきれない場合には，goto 文を使うことができます。飛び先きの場所は文の頭につけられたラベル（FORTRANと同様数字に限る）で示します。

(例)

```
BEGIN
        ……
    READ(CH); IF EOF(INPUT)THEN GOTO 1;
        ……
END;
        ……
1: WRITELN(▼END OF FILE▼);
```

5.6　その他の構造化文

その他の構造化文としてwith文があります。

5.6.1　with文

with 文はレコード型の変数のフィールドの指示を簡単にするのに使う文です。（レコードについての説明は構造化データを説明するときにします。）その一般形は，

WITH＜record variable list＞DO＜statement＞

たとえば，

PERSON␣AGE : =22; PERSON␣SEX : = FEMALE

というのをwith文を使うと次の様に書くことができます。

```
WITH PERSON DO
    BEGIN AGE: = 22; SEX: = FEMALE END
```

6. 配 列 型

前にPASCALのデータ型のうちで基本的なもの(unstructured types) 6つ—— 論理型，整数型，実数型，文字型，スカラー型，部分範囲型—— について述べました。ここではこれらから構成されるより複雑なデータ型(structured types)について説明します。この単純な型から複雑な型を構成していく機能の豊富なことがPASCALの大きな特徴の一つです。構造化データとしては，配列型，レコード型，集合型，ファイル型，ポインタ型があります。これからこの順にデータ構造について説明します。

配列型はFORTRANにもあります。(FORTRANで構造化データといえる唯一のものです。)配列とは同じ型のデータがきまった数だけ集まったもので，しかも各要素が添字(index)とよばれる変数または定数で示されるものです。PASCALの配列はFORTRANのよりも少し一般的になっているといえます。それはまず第一に，配列の要素は同じ型であるかぎりどんな型でも，それ自身構造化データであってもかまわないということです。だから配列の配列というものを考えられます(これは結局2次元の配列です)。第二に，添字の型がFORTRANでは整数(の部分範囲)ですが，PASCALでは添字として基本型のうちの次のものが使えます。

(1) 論理型

(2) 文字型

(3) ユーザの定義したスカラー型

(4) 上の3つあるいは整数型の部分範囲

添字の型をT_1,要素の型をT_2 とすると，配列型Tの定義の一般形は次のようになります。

```
TYPE T=ARRAY(. T1 .) OF T2;
```

たとえばFORTRANでの

```
INTEGER A(100)
```

という配列Aの宣言はPASCALでは

```
VAR A:ARRAY(. 1..100 .)OF INTEGER
```

となります。

配列の添字は，必ずしも1からはじまる必要はなく，

```
VAR B : ARRAY(. -10 .. 10 .) OF INTEGER;
```

も許されます。また，すでにDAYSというスカラー型がたとえば

```
TYPE DAYS=( SUN, MON, TUES, WED, THUR, FRI, SAT);
```

と定義されているとすると，

```
VAR SICK:ARRAY(. DAYS .)OF BOOLEAN;
```

などと宣言できます。

配列の要素を示すには，配列名の次に添字を(. と .)でかこんで続けます。たとえば

```
A(. I .):=A(. I-1 .)+A(. I-2 .);
```

とか

```
SICK(. MON .):=TRUE;SICK(. SAT .):=FALSE;
```

とかくわけです。

配列の要素が再び配列である場合，たとえば

```
VAR MATRIX：ARRAY(。1..10。)OF ARRAY(. 1.。10。 )OF REAL;
```

は，これを多次元の配列として簡単に

```
VAR MATRIX：ARRAY(。1.。10, 1.。10. )OF REAL;
```

とかくことができます。

この要素を示すにはMATRIX(.7,7.)などとかくことができますが，MATRIX(。7。)(. 7。)とかいてもかまいません。

一般に構造化データに対してPACKEDというシンボルを前につけて，メモリーの使用量を節約することができます。たとえば

```
TYPE VECTOR=PACKED ARRAY(。1.. 100。)OF BOOLEAN;
```

のようにかきます。

特に重要なのは，PASCALでは文字列が文字のPACKされた配列で表わされることです。すなわち，n文字の文字列は

```
PACKED ARRAY(。1.. n。)OF CHAR
```

の型に属するとされています。この型に属する値の間には大小関係が定められていて，比較の演算子＝，＜＞，＜，＜＝，＞，＞＝を使って比較することができます。

(例)

```
VAR S : PACKED ARRAY(. 1.。6.)OF CHAR;
    ......
    S :=▼TOKYO  ▼;
    ......
    IF S=▼NAGOYA▼ THEN ......
```

7. レコード型

レコード型はいくつかの必ずしも同じでない型のデータをひとまとまりとして扱うためのものです。たとえばあるものがいくつかの属性を持つとして，それらをまとめて全体としてそのものを表現するというような時に使うと便利です。

例をあげます。日付けはふつう年，月，日であらわされます。日付けをあらわす型DATEを定義してみますとたとえば次のようになります。

```
TYPE DATE=RECORD DAY:1..31;
                 MONTH:(JAN, FEB, MAR, APR, MAY, JUN. JUL, AUG,
                        SEPT, OCT, NOV, DEC);
                 YEAR:INTEGER
            END;
```

ここでDATEという型はDAY, MONTH, YEARという3つの要素をまとめて扱うためのものです。このDAYなどの要素のことをレコードのフィールドといい，それらには，それぞれDAYなどの名前(field identifier)がつけられます。フィールドの型はフィールド名の後にコロン(:)をおいてかきます。フィールドの型はあらかじめ定義された型ならどんな型をかいてもかまいません。実行部中で，あるレコードのある要素を示すには，レコード名と要素名との間にピリオド(.)をはさんでかきます。たとえば

```
VAR SOMEDAY, NEWYEAR:DATE;
```

と宣言しますと，

```
SOMEDAY. DAY:=1;
SOMEDAY. MONTH:=JAN;
SOMEDAY. YEAR:=1977;
```

というふうにレコードの要素への代入ができます。またレコードをひとまとまりのまま代入することもできます。たとえば

```
NEWYEAR:=SOMEDAY;
```

次の例として，一人の人間はその名前とか生年月日とかいろいろの属性をもちますが，このいろいろの型のデータの集まりとしての一人の人間のレコードを定義してみます。

```
TYPE ALFA=PACKED ARRAY(. 1.. 12. )OF CHAR;
     STATUS=(SINGLE, MARRIED):
     PERSON=RECORD
                NAME:RECORD FIRST, LAST:ALFA END;
                SEX:(MALE, FEMALE);
                BIRTH:DATE;
                CASE MS:STATUS OF
                  SINGLE:(    );
                  MARRIED:(NCHILDREN:INTEGER)
                END;
```

ここでCASE以下に注意して下さい。これはフィールドMSの値がSINGLEかMARRIEDかによってあとのフィールドをどのようにとるかを決めることを示します。1つのレコードの要素の数や型が，MSの値によってかわってくるわけです。レコードの中でこのCASE以下の部分を可変部分(variant part)，それ以前の部分を固定部分(fixed part)といい，1つのレコードの中に可変部分(variant part)は最後に1つだけ許されます。但し可変部分(variant part)の中でまたrecordの可変部分(variant part)が現われてもかまいません。またこのMS：のようなフィールドのことをタグフィールド(tag field)といい，どのフィールドをその可変フィールドとするかを変数として指示します。

ここでこれまで述べてきたレコード型の定義のシンタックスをまとめておきます。

＜record type＞ ::＝RECORD＜field list＞END

＜field list＞ ::＝＜fixed part＞|＜fixed part＞;＜variant part＞|＜variant part＞

＜fixed part＞ ::＝＜record section＞{;＜record section＞}

＜record section＞ ::＝＜field identifier＞{,＜field identifier＞}:＜type＞|＜empty＞

＜variant part＞ ::＝CASE＜tag field＞＜type identifier＞OF＜variant＞{;＜variant＞}

＜variant＞ ::＝＜case label list＞:(＜field list＞)|＜empty＞

＜case label list＞ ::＝＜case label＞{,＜case label＞}

＜case label＞ ::＝＜constant＞

＜tag field＞ ::＝＜identifier＞:|＜empty＞

さて，ここで変数WHOを

```
VAR WHO : PERSON;
```

と宣言したとしますと，その各フィールドを指示するのに次のように

```
WHO. NAME. FIRST := 'WILLIAM      ';
WHO. NAME. LAST := 'SHAKESPEARE   ';
WHO. SEX:=MALE;
WHO. BIRTH. DAY:=23;
WHO. BIRTH. MONTH:=APR;
WHO. BIRTH. YEAR:=1564;
WHO MS : = MARRIED;
WHO. NCHILDREN : = 3 ;
```

とかくのは面倒ですが，5節に述べられたWITH文を用いると次のように簡単にかけます。

```
WITH WHO, NAME, BIRTH DO
    BEGIN FIRST: ='WILLIAM      ';LAST:='SHAKESPEARE   ';
          SEX:=MALE;
          DAY:=23;MONTH:=APR;YEAR:=1564;
          MS:=MARRIED;NCHILDTEN:=3
    END;
```

8. 集　合　型

この型は値としてある型のベキ集合，つまりもとの型のすべての要素からなる集合の部分集合をとるものです。すなわちもとの型の値をいくつか選んで，それをまとめて一つのもの（集合）として扱おうとするものです。もとの型の値の種類がn個なら異なった集合の値の種類は2^nです。（空集合も許されます。）もとの型は実数型以外の基本型でなければなりません。そして，インプリメントの都合上64個以上の値をとらず，かつそれぞれの値（スカラーや文字型の場合，標準関数ORDを使って得られる値）も0と63の間にあるものでなければなりません。（PASCAL8000では）

この型の定義の一般形は次の通りです。

TYPE＜identifier＞＝SET OF＜base type＞;

例として

```
TYPE HAND= SET OF 1..52;
VAR MYHAND, YOURHAND:HAND;
```

としますと，変数MYHAND, YOURHANDは，1から52までの間のいくつかの数からなる集合です。集合はその要素で示されます。その示し方は，要素をコンマ(,)で区切って並べます。要素として2つの式の間に..をかくと2つの値の間のすべてを含むものと考えます。たとえば

```
MYHAND := (. 13 .);
YOURHAND := (.1, 14, 27..29, 40.);
```

まとめるとシンタックスは次のようになります。

＜set＞::=(.＜element list＞.)

＜element list＞::=＜element＞{,＜element＞}|＜empty＞

＜element＞::=＜expression＞|＜expression＞..＜expression＞

集合の間の演算子としては

＋　和　集　合

＊　共通部分

－　差　集　合

があります。また論理型の値を返す演算子として

＝，＜＞　　等しいか？　等しくないか？

＜＝，＞＝　　集合間の包含関係。2つの集合が等しい時も値はTRUEです。等しい時FALSEとする狭い意味の包含関係の演算子はありません。

IN　　要素が集合に含まれるか？

(例) 13 IN YOURHAND → FALSE

次に集合型を使ったプログラム例を示します。

```
VAR I:INTEGER;
    S:SET OF 1..50;
BEGIN S:=(. .);
    FOR I:=1 TO 50 DO
    IF ODD(I) THEN S:=S+(.I.)
END
```

9. ファイル型

PASCALのプログラムでは順編成のファイル(sequential file)を扱うことができます。しかしその基本的な考え方や扱い方はFORTRANなどとかなり異なります。

PASCALではファイルをすべて同じ型の要素の「列」であると考えます。あるファイルの中ではどれかただ一つの要素しか同時には参照できません。別の要素を参照するには，2つの間の要素を順番に読み(あるいは書き)進んでいかなければならないわけです。ここでいうファイルは配列と同じように同じ型のデータの抽象的な集まりと考えればよいわけですが，ディスクなどの二次記憶に実際にデータをおくことを考えてこのような制限がつけられているわけです。ファイルが配列と異なるのは長さ(要素の数)があらかじめ決められてなく，空でもかまわないということです。

ファイル型の変数の宣言は次のようにします。

VAR ＜identifier＞：FILE OF ＜type＞；

各ファイル変数に対して自動的にバッファ変数と呼ばれるものが決められます。変数名をFとするとバッファ変数は

F@

で表わされます。バッファ変数はファイルの現在参照可能な要素のことを意味します。ファイルの内容の読み出し，書きこみはこのバッファ変数を通してのみ可能です。またバッファ変数の位置をファイル上で1要素分進めるのと，ファイルの先頭にもどす標準手続きが用意されています。

バッファ変数F@の位置がファイルの終わりを越えているかどうかを知るために論理型の標準関数EOFがあります。EOF(F)はF@がファイルの終わりを越えているときTRUE,そうでないときFALSEを返します。

ファイルを扱うための標準手続きには次の4つがあります。

RESET (F)　F@をファイルの先頭におく。F@はファイルの先頭の要素になります。

REWRITE (F)　Fの内容がすべて消され，長さが0,空のファイルとなります。ファイルを書きかえるときに用いられます。EOF (F) はTRUE となります。

GET (F)　F@を次の要素へ進める。すなわち次の要素の値がF@にはいります。次の要素がない場合はEOF (F) がTRUEとなり，F@の値は不定となります。

PUT (F)　F@の値をファイルFの最後に書きこむ。PUTする前にはEOF (F) はTRUEでなければなりません。

ファイルを使ったプログラム例です。

```
PROGRAM  WRITESUM(F, OUTPUT);
VAR F : FILE OF REAL;
    SUM : REAL;
BEGIN
SUM:=0.0;      RESET (F);
WHILE NOT EOF (F) DO
  BEGIN SUM:=SUM+F@; GET (F) END;
```

```
    WRITELN(SUM)
END
```

この例でわかるようにRESETやGETなどの標準手続きはFORTRANのREAD文と違ってバッファの位置を動かすだけで，その内容の参照はユーザが直接バッファの中に手をつっこんで見るという形になっています。このため上のプログラムではGETがファイルの内容を使った計算のあとにくるのに比べて，FORTRANで同じことをかくとREAD文が前にでてくるのに気づくと思います。

要素が文字であるファイルをテキストファイルといいます。普通，最もよく使われるファイルですので，他のファイルと少し違った取り扱いができるようになっています。PASCALでは

```
TYPE TEXT=FILE OF CHAR;
```

で定義されるTEXTという型名があらかじめ用意されています。この型のファイルはいくつかの要素をまとめた「行」に分かれています。行を扱うために次の3つの標準手続き，標準関数があります。

WRITELN (F)	現在の行への書きこみを終える。
READLN (F)	次の行の先頭までF@の位置を進める。F@は次の行の先頭の文字になります。
EOLN (F)	F@が行の終わりの位置にあるかどうかを返す論理型の関数。 EOLN (F) がTRUEの時，F@は行の区切りの位置にあります。 この時F@の値はブランクであるとされています。

さらにFをTEXT型，CHを文字型の変数とすると

```
F@:=CH;   PUT (F)   のかわりに  WRITE(F, CH)
CH :=F@;  GET (F)   のかわりに  READ(F, CH)
```

とかくことができます。この他TEXT型のファイルに対する入出力の手続きは12節を参照して下さい。

10. ポインタ型

データのうちで，ブロックの最初で宣言されそのブロックの実行中ずっと存在するものをstaticといいます。これに対して，プログラム実行の途中で新たに（NEWという標準手続きによって）生成されるデータがあります。こういうデータをdynamicであるといいます。これらの生成されたデータは名前がついていないので，ポインタとよばれる変数（実際にはアドレスと考えてかまいません）で示されます。

いま，Tをすでに定義されている型もしくはこれから定義する型として，

```
VAR P: @T
```

とポインタ変数Pを宣言するとします。そうするとプログラムの実行部中において

```
NEW (P)
```

という文を実行すると，型Tのデータが一つ生成されます。（正確にいうと，型Tのデータ1つ分の領域がメモリ中にとられますが，その値はまだ与えられていません。）そしてポインタ変数Pがそのデータを「指し」ます。そしてこのデータは

```
P@
```

で示されます。この状態でさらにもう一度

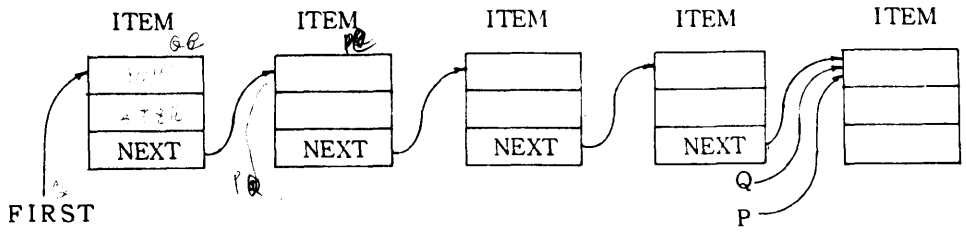
```
NEW (P)
```

を実行すると，もう一つ，型Tのデータが生成され，ポインタ変数Pはこの新しいデータの方を指します。（従って前に生成されたデータは「行方不明」になります。）

次に具体的に例を示します。

```
TYPE LINK=@ITEM;
     ITEM=RECORD
            NAME : PACKED ARRAY(.1..8.)OF CHAR;
            SIZE : REAL;
            NEXT : LINK
          END;
VAR FIRST, P, Q:LINK; I:INTEGER;
BEGIN NEW(FIRST);Q:=FIRST;
      FOR I:=1 TO 4 DO
       BEGIN
        NEW(P);
        Q@.NEXT:=P;
        Q:=P
       END
END
```

型 LINK は ITEMというレコード型のデータをさすポインタ型として宣言されています。このプログラムの実行終了時には5つのITEMの次図のような「鎖」ができ上ります。



同じ型を指す2つ以上のポインタ変数の間で値の代入が自由にできることに注意して下さい。ポインタ変数がどのデータも指していないことを示す時に使う特別の値をNILとかき，この値も代入することができます。また2つのポインタの値を比較演算子＝または＜＞を使って比較することもできます。

次にポインタとレコードを有効に用いたプログラム例を示します。これは有理式を微分するプログラムで，A＋B等の最簡式は1つのレコードで表現され，一般の式はそれらをポインタでつなぐことで表現されています。微分はポインタでつながれた各項ごとに行なわれます。ポインタの使い方や有理式をデータとして表現する方法が参考になると思います。

```
PROGRAM DIFF(OUTPUT);
(*SYMBOLIC DIFFERENTIATION.  M. YASUMURA.  MAY 16,1975.*)

TYPE KIND = (CONSTANT,VARIABLE,PAIR);
     CLASS = (SUM,DIFFERENCE,PRODUCT,QUOTIENT);
     PEXP =^EXPRESSION;
EXPRESSION = RECORD CASE K: KIND OF
                  CONSTANT: (NUMBER: INTEGER);
                  VARIABLE: (PRINTNAME: CHAR);
                  PAIR : (LEFT,RIGHT,DERIV: PEXP; OP: CLASS)
             END;
VAR  ONE,ZERO,X,U,Y,DY,Z,DZ: PEXP;

FUNCTION G(CL: CLASS; L,R: PEXP): PEXP;
 VAR  TEMP: PEXP;
 BEGIN  NEW(TEMP,PAIR); WITH TEMP^ DO
          BEGIN LEFT := L; RIGHT := R; DERIV := NIL; OP := CL END;
        G := TEMP
 END;
FUNCTION ASSOC(U,X: PEXP): PEXP;
 VAR  E,U,UD,V,VD,T1,T2: PEXP;
 BEGIN  E := U;  WITH E^ DO
  CASE K OF
    CONSTANT: ASSOC := ZERO;
    VARIABLE: IF E = X THEN ASSOC := ONE ELSE ASSOC := ZERO;
    PAIR: IF DERIV <> NIL THEN ASSOC := DERIV
                          ELSE
    BEGIN
     U := LEFT; V := RIGHT; UD := ASSOC(U,X); VD := ASSOC(V,X);
     CASE OP OF
       SUM: IF UD = ZERO THEN DERIV := VD
            ELSE IF VD = ZERO THEN DERIV := UD
                              ELSE DERIV := G(SUM,UD,VD);
       DIFFERENCE: IF VD = ZERO THEN DERIV := UD
                                ELSE DERIV := G(DIFFERENCE,UD,VD);
```

```
36          PRODUCT: IF UD = ZERO THEN
37                       IF VD = ZERO THEN DERIV := ZERO
38                       ELSE IF VD = ONE THEN DERIV := U
39                                       ELSE DERIV := G(PRODUCT,U,VD)
40                   ELSE IF VD = ZERO THEN
41                                IF UD = ONE THEN DERIV := V
42                                       ELSE DERIV := G(PRODUCT,V,UD)
43                   ELSE BEGIN IF UD = ONE THEN T1 := V
44                                       ELSE T1 := G(PRODUCT,V,UD);
45                              IF VD = ONE THEN T2 := U
46                                       ELSE T2 := G(PRODUCT,U,VD);
47                              DERIV := G(SUM,T1,T2)
48                           END;
49          QUOTIENT: IF VD = ZERO THEN DERIV := G(QUOTIENT,UD,V)
50               ELSE IF VD = ONE THEN DERIV := G(QUOTIENT,G(DIFFERENCE,UD,E),
51               ELSE DERIV := G(QUOTIENT,G(DIFFERENCE,UD,G(PRODUCT,E,VD)),V)
52        END (*OP*);
53        ASSOC := DERIV
54       END
55      END (*K*)
56     END (*ASSOC*);
57
58    FUNCTION VARI(C: CHAR): PEXP;
59     VAR TEMP: PEXP;
60     BEGIN NEW(TEMP,VARIABLE); TEMP@.PRINTNAME := C; VARI := TEMP
61     END;
62    FUNCTION CON(C:INTEGER): PEXP;
63     VAR TEMP: PEXP;
64     BEGIN NEW(TEMP,CONSTANT); TEMP@.NUMBER := C; CON := TEMP
65     END;
66    PROCEDURE LISTOUT(P: PEXP);
67     BEGIN WITH P@ DO
68       CASE K OF
69         CONSTANT: WRITE(NUMBER:1);
70         VARIABLE: WRITE(PRINTNAME);
71         PAIR: BEGIN WRITE('('); LISTOUT(LEFT);
72                  CASE OP OF
73                    SUM: WRITE('+');
74                    DIFFERENCE: WRITE('-');
75                    PRODUCT: WRITE('*');
76                    QUOTIENT: WRITE('/')
77                  END (*OP*); LISTOUT(RIGHT); WRITE(')')
78               END (*PAIR*)
79       END (*K*)
80     END (*LISTOUT*);
81    BEGIN  (*MAIN*)
82      WRITELN; WRITELN;
83      ONE := CON(1); ZERO := CON(0); X := VARI('X'); C := VARI('C');
84      Y := G(DIFFERENCE,G(SUM,G(PRODUCT,X,X),G(PRODUCT,X,C)),CON(5));
85      DY := ASSOC(Y,X);
86      WRITE('  Y  :='); LISTOUT(Y); WRITELN;
87      WRITE('DY/DX:='); LISTOUT(DY); WRITELN;
88      Z := G(QUOTIENT,ONE,G(SUM,X,CON(2)));
89      DZ := ASSOC(Z,X);
90      WRITE('  Z  :='); LISTOUT(Z); WRITELN;
91      WRITE('DZ/DX:='); LISTOUT(DZ); WRITELN
92    END.
```

```
  Y  :=(((X*X)+(X*C))-5)
DY/DX:=((X+X)+C)
  Z  :=(1/(X+2))
DZ/DX:=((0-(1/(X+2)))/(X+2))
```

```
PASCAL TIMING SUMMARY:

     COMPILE TIME =       0.42
     EXECUTE TIME =       0.04
```

プログラム 3

11. 手続きと関数

大きなプログラムの中で，何度も実行される部分をとり出してそれを一まとまりとして定義するという技法は，FORTRANではサブルーチンと関数という概念であらわされますが，これらはPASCALでは手続き（procedure）と関数（function）にあたります。PASCALの手続きや関数は，PORTRANのと比べて，手続きが自分自身を呼び出すことができる，手続きの定義の中にまた別の手続きの定義を書くことができる，などいくつかの点でより一般的になっています。

11.1 手続き（procedure）

手続きは，あらかじめ手続き宣言（procedure declaration）によって宣言されるもので，実行部中で必要に応じ手続き文（procedure statement）によって呼び出され実行されます。

手続き宣言は手続き頭部（procedure heading）と，本体であるブロック（bolock）とからできています。本体は，第2節に説明したプログラムの本体と同じ構造をしています。

＜ブロック＞::＝ ＜ラベル宣言部＞
＜定数定義部＞
＜型定義部＞
＜変数宣言部＞
＜手続き・関数宣言部＞
＜実行部＞

手続き頭部は，その手続きの名前と，（もしあれば）その仮引数（formal parameters）を示します。その一般形は次の通りです。

PROCEDURE ＜名前＞；

あるいは

PROCEDURE ＜名前＞（＜仮引数宣言部＞）；

仮引数宣言部で宣言された変数は，その手続きにローカルな変数であるとみなされます。従ってこれらは変数宣言をする必要はありません。仮引数宣言部の一般形は次の通りです。

＜変数名＞，…，＜変数名＞：＜型名＞

または

VAR＜変数名＞，…，＜変数名＞：＜型名＞

またはこの両者をいくつか（順序は任意）をセミコロン(;)で区切って並べたもの。

ここで前者のように宣言された引数を値引数（value parameter），後者のように宣言された引数を変数引数（variable parameter）といい，これらは手続きが呼ばれる際の実引数（actual parameter）との結合方法が異なります。すなわち，値引数では実引数の値だけが仮引数に渡されるのに対し，変数引数では実引数の番地（address）が仮引数に渡され仮引数の参照や代入はこの番地を使って間接的に行なわれます。従って，変数引数の実引数は必ず変数でなければなりませんが，値引数の実引数は一般に式（expression）になります。（ただしいずれの場合も実引数と仮引数の型の一致は要求されます。）次のプログラムは変数引数と値引数との効果の違いを示しています。

```
 1    PROGRAM PARAM(OUTPUT);
 2    VAR I,J: INTEGER;
 3    PROCEDURE P(X:INTEGER; VAR Y:INTEGER);
 4     BEGIN
 5      X:=X+1; Y:=Y+1;
 6      WRITELN(X,Y)
 7     END;
 8
 9     BEGIN
10      I:=0; J:=0;
11      P(I,J);
12      WRITELN(I,J)
13    END.
```

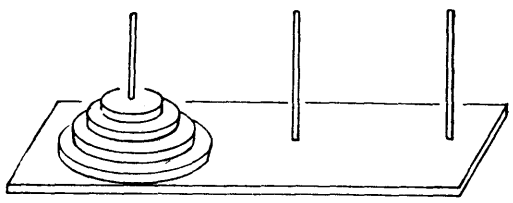
```
 1           1
 0           1
```

一般には引数が単純な型である場合には値引数のほうが効率や安全性の面からすぐれています。しかし値引数では手続きの側から呼んだ側へ値を返すことができません。このようなことをしたい場合や，大きな配列のような構造化データを引数とする場合は変数引数のほうが適しています。

次に再帰的呼び出し（recursive call）について説明します。1つの手続きの実行部中で自分自身を呼び出すことを再帰的呼び出しといい，これはFORTRANでは禁止されていますが PASCAL では許されます。プログラムしたい問題の性質によっては大変便利なものです。前節に載せた有理式の微分プログラムにもこの再帰的呼び出しが使われています。また次のプログラム例は，ハノイの塔(注)と呼ばれる有名なパズルのPASCALによる解です。

```
 1    PROGRAM HANOI(OUTPUT);
 2    PROCEDURE HANOI(N:INTEGER; X,Y,Z:CHAR);
 3     BEGIN
 4      IF N>0 THEN
 5       BEGIN
 6        HANOI(N-1,X,Z,Y);
 7        WRITELN('MOVE PIECE ',N:1,' FROM ',X,' TO ',Z,';');
 8        HANOI(N-1,Y,X,Z)
 9       END
10     END;
11
12     BEGIN
13      HANOI(4,'A','B','C')
14     END.
```

```
MOVE PIECE 1 FROM A TO B;
MOVE PIECE 2 FROM A TO C;
MOVE PIECE 1 FROM B TO C;
MOVE PIECE 3 FROM A TO B;
MOVE PIECE 1 FROM C TO A;
MOVE PIECE 2 FROM C TO B;
MOVE PIECE 1 FROM A TO B;
MOVE PIECE 4 FROM A TO C;
MOVE PIECE 1 FROM B TO C;
MOVE PIECE 2 FROM B TO A;
MOVE PIECE 1 FROM C TO A;
MOVE PIECE 3 FROM B TO C;
MOVE PIECE 1 FROM A TO B;
MOVE PIECE 2 FROM A TO C;
```



プログラム5　ハノイの塔

(注)　ハノイの塔とは，台に3本の柱があってそのうち1本に大きさの異なる円盤がn枚図のように重なっており，この円盤の山を，大きさの順を変えずに，第3の柱へ移動せよという問題です。移動は1枚ずつ行ない，第2の柱を補助として使ってよいが，移動の途中のどの時点でもどの柱も大きさの順が逆になってはいけないという条件がついています。

次に１つの制限事項を述べておきます。ある手続きの呼び出しがその手続きの宣言よりも前に行なわれることがあります。この場合，その呼び出しの前に次のようなことわり書き(forward declaration)をしておく必要があります。たとえば手続きQの中で，まだ宣言されていない手続きPを引用していると，

```
PROCEDURE P (<仮引数宣言部>);FORWARD;
PROCEDURE Q (<仮引数宣言部>);
    BEGIN……;
          P(…);
          ……
    END;
   ⋮
PROCEDURE R; (ここには仮引数をかかない)
    BEGIN
          …
    END;
```

のようにかかなければいけません。

11.2 関数 (function)

関数は手続きとほとんど同じように宣言します。関数頭部(function heading)は手続き頭部に関数の返す値の型が加わったものです。関数の頭部の一般形は次の通りです。

```
FUNCTION<名前>:<結果の型>;
```

あるいは

```
FUNCTION<名前>(<仮引数宣言部>):<結果の型>;
```

結果の型は，基本型と，ポインタ型とに限られます。

関数宣言の実行部中には，関数名への代入文が必ずなければなりません。その他再帰的呼び出しやことわり書き(FORWARD)等については手続きの場合と同じです。

関数の呼び出しは式の中に関数名とカッコでくくった引数を並べて書くことによって行なわれます。

さて，手続きや関数の中で，ローカルでない変数に値を代入することをその手続きや関数の副作用(side-effect)といい，しばしば予期せぬ悪い結果を招いたりするのでできるだけ避けた方がよいものです。次の例はその見本です。

```
1    PROGRAM SIDEEFFECT (OUTPUT);
2    VAR Z: INTEGER;
3    FUNCTION SQUARE(X: INTEGER): INTEGER;
4    BEGIN Z := Z-X;
5          SQUARE := SQR(X)
6    END;
7    BEGIN Z := 10; WRITELN(Z,SQUARE(Z),SQUARE(Z))
8    END.
```

```
10          100          0
```

プログラム 6

PASCALで用意されている標準手続きと標準関数は付録2にまとめておきます。標準手続きのうち入出力については第12節で述べます。

11.3 プログラム例　WANGのアルゴリズム

最後に，LISP 1.5のマニュアルにLISPプログラムの例としてのせられているWANGのアルゴリズムをPASCALで書いてみたものをあげておきます。これは命題論理における定理を証明するプログラムです。この例では

$$(\{P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

という命題（3段論法）が定理であることを証明しています。プログラムの詳しい説明は省きますが，レコード型やポインタ型の変数がうまく使われていて，LISPのプログラムがPASCALに見事にかきかえられているところに注目して下さい。

```
(*THE WANG ALGORITHM FOR PROPOSITIONAL CALCULUS.  M.Y.*)
PROGRAM WANG(INPUT,OUTPUT);
TYPE STRUCTURE = (MONO,COMP);
     ELEMENT = (NOTP,ANDP,ORP,IMPLIES,EQUIV,ARROW,ZP,ZQ,ZR);
     LIST = ^LST;
     LST = RECORD CASE ST: STRUCTURE OF
             MONO: (UNIT: ELEMENT);
             COMP: (LEFT,RIGHT: LIST)
           END;
VAR L: LIST;
    CH: CHAR;
    VALID: ARRAY[.CHAR.] OF BOOLEAN;

FUNCTION CONVERT(CH: CHAR): LIST;
VAR W: LIST;
BEGIN NEW(W,MONO); WITH W^ DO
   CASE CH OF
     '~': UNIT := NOTP;
     '&': UNIT := ANDP;
     '|': UNIT := ORP;
     '>': UNIT := IMPLIES;
     '=': UNIT := EQUIV;
     'A': UNIT := ARROW;
     'P': UNIT := ZP;
     'Q': UNIT := ZQ;
     'R': UNIT := ZR
   END;
CONVERT := W
END;
FUNCTION CONS(L,R: LIST): LIST;
VAR W: LIST;
BEGIN NEW(W,COMP); WITH W^ DO
   BEGIN LEFT := L; RIGHT := R END;
   CONS := W
END;
FUNCTION CAR(L: LIST): LIST; BEGIN CAR := L^.LEFT END;
FUNCTION CDR(L: LIST): LIST; BEGIN CDR := L^.RIGHT END;
FUNCTION CADR(L: LIST): LIST; BEGIN CADR := CAR(CDR(L)) END;
FUNCTION CADDR(L: LIST): LIST; BEGIN CADDR := CADR(CDR(L)) END;
FUNCTION ATOM(L: LIST): BOOLEAN;
BEGIN IF L = NIL THEN ATOM := TRUE
      ELSE CASE L^.ST OF
             MONO: ATOM := TRUE;
             COMP: ATOM := FALSE
           END
END;
FUNCTION EQUAL(L1,L2: LIST): BOOLEAN;
BEGIN IF (L1 = NIL) OR (L2 = NIL) THEN EQUAL := (L1 = L2)
      ELSE IF ATOM(L1) OR ATOM(L2)
           THEN IF ATOM(L1) AND ATOM(L2) THEN EQUAL := (L1^.UNIT = L2^.UNIT)
                                         ELSE EQUAL := FALSE
           ELSE EQUAL := EQUAL(L1^.LEFT,L2^.LEFT) AND EQUAL(L1^.RIGHT,L2^.RIGHT)
END;
```

```
54    FUNCTION MEMBER(X,U: LIST): BOOLEAN;
55    BEGIN IF U = NIL THEN MEMBER := FALSE
56          ELSE IF EQUAL(X,CAR(U)) THEN MEMBER := TRUE
57          ELSE MEMBER := MEMBER(X,CDR(U))
58    END;
59    FUNCTION INCLUDE(X,U: LIST): LIST;
60    BEGIN IF MEMBER(X,U) THEN INCLUDE := U
61          ELSE INCLUDE := CONS(X,U)
62    END;
63    FUNCTION TH1(A1,A2,A,C: LIST): BOOLEAN; FORWARD;
64    FUNCTION TH2(A1,A2,C1,C2,C: LIST): BOOLEAN; FORWARD;
65    FUNCTION TH(A1,A2,C1,C2: LIST): BOOLEAN; FORWARD;
66    FUNCTION THL(C,A1,A2,C1,C2: LIST): BOOLEAN; FORWARD;
67    FUNCTION THR(C,A1,A2,C1,C2: LIST): BOOLEAN; FORWARD;
68    FUNCTION TH1L(V,A1,A2,C1,C2: LIST):BOOLEAN; FORWARD;
69    FUNCTION TH1R(V,A1,A2,C1,C2: LIST):BOOLEAN; FORWARD;
70    FUNCTION TH2L(V,A1,A2,C1,C2: LIST):BOOLEAN; FORWARD;
71    FUNCTION TH2R(V,A1,A2,C1,C2: LIST):BOOLEAN; FORWARD;
72    FUNCTION TH11(V1,V2,A1,A2,C1,C2: LIST): BOOLEAN; FORWARD;
73
74    FUNCTION THEOREM(S: LIST):BOOLEAN;
75    BEGIN THEOREM := TH1(NIL,NIL,CADR(S),CADDR(S))
76    END;
77    FUNCTION TH1;
78    BEGIN IF A = NIL THEN TH1 := TH2(A1,A2,NIL,NIL,C)
79          ELSE IF MEMBER(CAR(A),C) THEN TH1 := TRUE
80          ELSE IF ATOM(CAR(A)) THEN
81            TH1 := TH1(INCLUDE(CAR(A),A1),A2,CDR(A),C)
82          ELSE TH1 := TH1(A1,INCLUDE(CAR(A),A2),CDR(A),C)
83    END;
84    FUNCTION TH2;
85    BEGIN  IF C = NIL THEN TH2 := TH(A1,A2,C1,C2)
86           ELSE IF ATOM(CAR(C)) THEN
87             TH2 := TH2(A1,A2,INCLUDE(CAR(C),C1),C2,CDR(C))
88           ELSE TH2 := TH2(A1,A2,C1,INCLUDE(CAR(C),C2),CDR(C))
89    END;
90    FUNCTION TH;
91    BEGIN  IF A2 = NIL
92           THEN IF C2 = NIL THEN TH := FALSE
93                ELSE TH := THR(CAR(C2),A1,A2,C1,CDR(C2))
94           ELSE TH := THL(CAR(A2),A1,CDR(A2),C1,C2);
95    END;
96    FUNCTION THL;
97    VAR W: LIST;
98    BEGIN W:= CAR(U); WITH W DO CASE UNIT OF
99          NOTP: THL := TH1R(CADR(U),A1,A2,C1,C2);
100         ANDP: THL := TH2L(CDR(U),A1,A2,C1,C2);
101         ORP : THL := TH1L(CADR(U),A1,A2,C1,C2) AND TH1L(CADR(U),A1,A2,C1,C2);
102         IMPLIES: THL := TH1L(CADDR(U),A1,A2,C1,C2) AND TH1R(CADR(U),A1,A2,C1,C2
103         EQUIV: THL := TH2L(CDR(U),A1,A2,C1,C2) AND TH2R(CDR(U),A1,A2,C1,C2)
104           END;
105   END;
106   FUNCTION THR;
107   VAR W: LIST;
108   BEGIN W := CAR(U); WITH W DO CASE UNIT OF
109         NOTP: THR := TH1L(CADR(U),A1,A2,C1,C2);
110         ANDP: THR := TH1R(CADR(U),A1,A2,C1,C2) AND TH1R(CADDR(U),A1,A2,C1,C2);
111         ORP: THR := TH2R(CDR(U),A1,A2,C1,C2);
112         IMPLIES: THR := TH11(CADR(U),CADDR(U),A1,A2,C1,C2);
113         EQUIV: THR := TH11(CADR(U),CADDR(U),A1,A2,C1,C2) AND
114                       TH11(CADDR(U),CADR(U),A1,A2,C1,C2)
115           END;
116   END;
117   FUNCTION TH1L;
118   BEGIN IF ATOM(V)
119         THEN  IF MEMBER(V,C1) THEN TH1L := TRUE
120               ELSE TH1L := TH(CONS(V,A1),A2,C1,C2)
121         ELSE  IF MEMBER(V,C2) THEN TH1L := TRUE
122               ELSE TH1L := TH(A1,CONS(V,A2),C1,C2);
123   END;
124   FUNCTION TH1R;
125   BEGIN IF ATOM(V)
126         THEN  IF MEMBER(V,A1) THEN TH1R := TRUE
127               ELSE TH1R := TH(A1,A2,CONS(V,C1),C2)
128         ELSE  IF MEMBER(V,A2) THEN TH1R := TRUE
129               ELSE TH1R := TH(A1,A2,C1,CONS(V,C2));
130   END;
131   FUNCTION TH2L;
```

```
132     BEGIN IF ATOM(CAR(V))
133           THEN  IF MEMBER(CAR(V),C1) THEN TH2L := TRUE
134                 ELSE TH2L := TH1L(CADR(V),CONS(CAR(V),A1),A2,C1,C2)
135           ELSE  IF MEMBER(CAR(V),C2) THEN TH2L := TRUE
136                 ELSE TH2L := TH1L(CADR(V),A1,CONS(CAR(V),A2),C1,C2);
137     END;
138     FUNCTION TH2R;
139     BEGIN IF ATOM(CAR(V))
140           THEN  IF MEMBER(CAR(V),A1) THEN TH2R := TRUE
141                 ELSE TH2R :=TH1R(CADR(V),A1,A2,CONS(CAR(V),C1),C2)
142           ELSE  IF MEMBER(CAR(V),A2) THEN TH2R := TRUE
143                 ELSE TH2R := TH1R(CADR(V),A1,A2,C1,CONS(CAR(V),C2));
144     END;
145     FUNCTION TH11;
146     BEGIN IF ATOM(V1)
147           THEN  IF MEMBER(V1,C1) THEN TH11 := TRUE
148                 ELSE TH11 := TH1R(V2,CONS(V1,A1),A2,C1,C2)
149           ELSE  IF MEMBER(V1,C2) THEN TH11 := TRUE
150                 ELSE TH11 := TH1R(V2,A1,CONS(V1,A2),C1,C2);
151     END;
152     FUNCTION GETIN: LIST;
153     VAR CH: CHAR;
154     BEGIN READ(CH); WRITE(CH);
155           IF  CH = '(' THEN GETIN := CONS(GETIN,GETIN)
156           ELSE IF CH = ')' THEN GETIN := NIL
157           ELSE IF VALID(.CH.) THEN GETIN := CONS(CONVERT(CH),GETIN)
158           ELSE GETIN := GETIN
159     END;
160     FUNCTION LREAD: LIST;
161     VAR CH: CHAR;
162     BEGIN READ(CH); WRITE(CH);
163           IF  CH <> '(' THEN LREAD := LREAD ELSE LREAD := GETIN
164     END;
165
166     BEGIN (*MAIN*)
167        FOR CH := CHR(0) TO CHR(255) DO VALID(.CH.) := FALSE;
168        VALID(.'¬'.) := TRUE; VALID(.'&'.) := TRUE;
169        VALID(.'|'.) := TRUE; VALID(.'>'.) := TRUE;
170        VALID(.'='.) := TRUE; VALID(.'*'.) := TRUE;
171        VALID(.'P'.) := TRUE; VALID(.'Q'.) := TRUE;
172        VALID(.'R'.) := TRUE;
173        L := LREAD;WRITELN; WRITELN(THEOREM(L));
174     END.
```

```
(* ((& (> P Q) (> Q R))) ((> P R)))
TRUE
```

```
PASCAL TIMING SUMMARY:

     COMPILE TIME =       0.82
     EXECUTE TIME =       0.06
```

プログラム7

12. 入 出 力

PASCALの入出力は付録8に示すように一般にファイルを通して行ないます。このうち特に入出力それぞれ一つずつ標準のテキストファイル，INPUTとOUTPUTが用意されています。
この2つは以下に述べるように入出力操作の時ファイル名を省略できるようになっています。またこの2つに対してRESET, REWRITEを使ってはいけません。

TEXT型のファイルに対する入出力READ, WRITEは，一文字ずつのみでなく，整数や実数も簡便に読んだり書いたりできます。また，FORTRANにおけるFORMAT文にあたるものを簡単にかけるようになっています。この節では，入出力用の5つの手続きREAD, READLN, WRITE, WRITELN, PAGEに関する規則を列記します。

12.1 標準手続きREAD

以下FをTEXT型のファイル，V1,V2,…，VNを，文字型，整数型，または実数型の変数とします。

1) READ(V1,…, VN)は
   READ(INPUT, V1,…, VN)と同じです。
2) READ(F, V1, …, VN)は
   BEGIN READ(F, V1);…; READ(F, VN) ENDと同じです。
3) READLN(V1,…, VN)は
   READLN(INPUT, V1,…, VN)と同じです。
4) READLN(F, V1,…, VN)は
   BEGIN READ(F, V1);…;READ(F, VN);
   READLN(F) END
   と同じです。VNの読まれた後に次の行の先頭までスキップされます。
5) READ(F, V)では，Vの型によって，Fからその型の値がVに読込まれます。数と数との間はブランク又は行末によって分離されていなければなりません。

12.2 標準手続きWRITE

以下，Fをファイル，P1,…，PNを後で述べる形の「パラメータ」とします。

1) WRITE(P1,…, PN)は
   WRITE(OUTPUT, P1,…, PN)と同じです。
2) WRITE(F, P1,…, PN)は
   BEGIN WRITE(F, P1);…; WRITE(F, PN)ENDと同じです。
3) WRITELN(P1,…, PN)は
   WRITELN(OUTPUT, P1,…, PN)と同じです。
4) WRITELN(F, P1,…, PN)は
   BEGIN WRITE(F, P1);…; WRITE(F, PN);WRITELN(F)END
   と同じです。
   これは，P1,…，PNをプリントし，現在の行を終えます。

5)　パラメータＰ１は次の３つの形のいずれか１つです。

Ｅ

Ｅ：Ｅ1

Ｅ：Ｅ1：Ｅ2

ここでＥ，Ｅ1,Ｅ２は一般の式です。

6)　Ｅは，プリントされる値で，文字型，整数型，実数型，論理型または文字列型のいずれかです。

7)　Ｅ１は，プリントされる文字巾の最小値を示します。この巾におさまりきらない場合は，巾はE1よりも大きくとられます。Ｅ１を省略すると各型に応じた標準値がとられます。
（PASCAL 8000では文字型：1，整数型：12，実数型：24，論理型：5，文字列：文字列の長さです。）

8)　Ｅ２はＥが実数型の時に用いられ，小数点以下の数字の数を示します。この時実数は固定小数点方式（FORTRANのＦ形式）で書かれます。Ｅ２を省略した時浮動小数点方式（FORTRANのＥ形式）です。

9)　Ｅが論理型の時はTRUEまたはFALSEと書かれます。

12.3 標準手続き PAGE

PAGE(F)はTEXT型のファイルへの書きこみの際に改ページを行ないます。

13. プログラム例

前節までで言語仕様のほぼ全部の説明がおわりました。入門としてはやや先を急ぎすぎたかもしれませんが，マニュアルとしても十分使えるものになっていると考えています。これだけの分量で一通り言語の記述がすんでしまうということがPASCALの簡潔さの一証明だといえるでしょう。一段落したところで今節は，復習もかねて，ある1つの問題をプログラムしてみようと思います。

13.1 エラトステネスのふるい(Eratosthenes sieve)

最近構造化プログラミング structured programming ということがよく話題になります。これはプログラミング方法論に関しており，いろいろの概念や考え方を含んでいて一口には要約しにくいものです。そこで，詳しくは原典を読んでいただくことにして，ここではその中で stepwise refinement という考え方を，一つの問題例をPASCALでプログラムしながら紹介したいと思います。さらに次節でそのプログラムをFORTRANとPL/Iに翻訳して，それらを比較してみることにします。

問題は，エラトステネスのふるいとよばれる素数を求めるアルゴリズムをプログラムせよというものです。このアルゴリズムはたいていの人が知っていると思いますが，それを「日本語」で記述すると次のようになるでしょう。「ふるい」をかけていく集合をSIEVE,得られた素数を入れる集合をPRIMESとして

1. 2からNまでのすべての整数をSIEVEという集合に入れよ。
2. 集合SIEVEの中の最小の数を選べ。
3. それを集合PRIMESに入れよ。
4. 集合SIEVEからその数の倍数(その数自身も含めて)をすべて取除け。
5. 集合SIEVEが空でなければ2－5.を繰返せ。空ならばおしまい。

stepwise refinement とは，ことば通り，アルゴリズムの各部分のデータや計算方法を段階的に少しずつ細かく規定し記述していって，最終的に求める「動く」プログラムを得ようという方法のことです。我々は，上のプログラム(?)を第一段階として，PASCALでかかれたプログラムを目標とします。さて第二段階にすすむために，NEXT という変数を導入していま見つかった素数を現わすことにし，PASCALふうの記法を用いて上のアルゴリズムを書直してみますと，次のようなプログラム(プログラム8)ができます。このプログラムは非常にわかりやすく，PASCALをほとんど知らない人でも何とか理解できるでしょう。(SUCC(NEXT)はNEXT+1と同じです。)

```
program ERATOSTHENES(OUTPUT);
const N = 10000;
var SIEVE, PRIMES: set of 0..N;
    NEXT, J: INTEGER;
begin (* initialize *)
  SIEVE :=(.2..N.); PRIMES :=(. .); NEXT :=0;
repeat (* find neXt prime *)
  while not (NEXT in SIEVE) do NEXT :=SUCC(NEXT);
  PRIMES :=PRIMES +(.NEXT.);
  J :=NEXT;
```

```
  while J<= N do  (* eliminate *)
    begin SIEVE :=SIEVE -(.J.);  J:=J+NE×T  end
  until SIEVE = (. .)
end .
```

プログラム 8

さて，このままではこのプログラムは動きません。PASCALの集合型の変数は機械の内部表現としては機械語のビットパターンとして表現されます。それ故集合の要素の数に制限があります（我々のシステムでは64です）。通常，集合型の変数は機械語の1語（我々の場合は2語ですが）で表され，従って要素の数の上限は機械語のビット数(WORDLENGTH)になります。（またそうすると集合間の演算が直接機械命令に対応することになり大変好都合です。）そこで，十分大きなNまでの要素を求めるためには，WORDLENGTHの大きさの集合をいくつか並べた配列を考えることになります。これに対応してNEXTという変数もレコード型にして，BIT,WORDという2つのフィールドで配列の中での現在の位置を示すことにします。この変更をプログラム8.にほどこすと最終的なプログラム9.ができあがります。

このプログラムはPASCALの異なるインプリメントから独立であることに注意して下さい。定数定義部のWORDLENGTHの値を変えるだけで，機械語の長さの異なったPASCALシステムに適用可能です。しかもインプリメントからの独立性にかかわらず高い実行効率を維持しています。

このプログラムは，はじめから数として奇数のみを考えることでかなり改良できます。その改良はプログラム9.からよりもプログラム8.からはじめる方がよいでしょう。興味をもたれた方はプログラムしてみて下さい。

```
 1  (*   ERATOSTHENES' SIEVE -- BY C.A.R. HOARE.
 2            T.HIKATA.   1975.7.18.
 3            C.F. 'STRUCTURED PROGRAMMING',  PP.129-130.   *)
 4  PROGRAM PRIME(OUTPUT);
 5  CONST WORDLENGTH = 64;  MAXBIT = 63;  (*IMPLEMENTION DPENDENT*)
 6        W = 199;
 7  VAR   SIEVE,PRIMES : ARRAY(.0..W.) OF SET OF 0..MAXBIT;
 8        NEXT : RECORD BIT,WORD : INTEGER END;
 9        J,K,S,T : INTEGER;
10        EMPTY : BOOLEAN;
11
12  BEGIN
13    (*INITIALIZE*)
14    FOR T := 0 TO W DO
15      BEGIN SIEVE(.T.) := (..);
16            FOR S := 0 TO MAXBIT DO SIEVE(.T.) := SIEVE(.T.)+(.S.);
17            PRIMES(.T.) := (..)
18      END;
19    SIEVE(.0.) := SIEVE(.0.)-(.0,1.);
20    NEXT.BIT := 0;  NEXT.WORD := 0;
21  EMPTY := FALSE;
22
23    WITH NEXT DO
24    REPEAT
25      (*FIND NEXT PRIME*)
26      WHILE NOT(BIT IN SIEVE(.WORD.)) DO BIT := BIT+1;
27      PRIMES(.WORD.) := PRIMES(.WORD.)+(.BIT.);
28      J := BIT;  K := WORD;
29      WHILE K<=W DO  (*ELIMINATE*)
30        BEGIN SIEVE(.K.) := SIEVE(.K.)-(.J.);
31              J := J+BIT;  K := K+WORD;
32              WHILE J>MAXBIT DO BEGIN J := J-WORDLENGTH; K := K+1 END
33        END;
```

```
24          IF SIEVE(.WORD.) = (..) THEN BEGIN BIT := 0;  EMPTY := TRUE  END;
35          WHILE (WORD<W) AND EMPTY DO
36            BEGIN WORD := WORD+1;  EMPTY := (SIEVE(.WORD.)=(..))  END
37        UNTIL EMPTY;
38
39        (*OUTPUT PRIMES*)
40        FOR T := 0 TO W DO
41          FOR S := 0 TO MAXBIT DO
42            IF S IN PRIMES(.T.) THEN WRITELN(T*WORDLENGTH+S)
43      END.
```

プログラム　9

13.2 PASCAL VS. FORTRAN

さて我々はこのPASCALプログラムをFORTRANとPL/Iとに翻訳してみました。それがプログラム10と11.です。はじめからアルゴリズムをこれらの言語で記述してもほぼ同じようなプログラムができたと思います。プログラム9. 10. 11.を比較してみて下さい。プログラムの内容について何も知らない人がこれらを見たとすると，プログラム9.は，決してやさしいものではありませんが，理解の容易さはプログラム10.とは比較にならないと思います。ではどのあたりが違うのでしょうか。両者の相違点を少し細かく見てみましょう。

```
      INTEGER SIEVE(400),PRIMES(400)
      INTEGER BIT,WORD
      INTEGER J,K,S,T,OUT
      LOGICAL EMPTY
      DATA SIEVE /Z7FFFFFFE,399*Z7FFFFFFF/
      DATA PRIMES /400*0/
      BIT=1
      WORD=1
      EMPTY=.FALSE.
    1 IF(IAND(2**(BIT-1),SIEVE(WORD)).NE.0) GO TO 2
      BIT=BIT+1
      GO TO 1
    2 PRIMES(WORD)=IOR(PRIMES(WORD),2**(BIT-1))
      J=BIT
      K=WORD
    3 IF(K.GT.400) GO TO 5
      SIEVE(K)=IAND(SIEVE(K),ICOMPL(2**(J-1)))
      J=J+BIT
      K=K+WORD-1
    4 IF(J.LE.31) GO TO 3
      J=J-31
      K=K+1
      GO TO 4
    5 IF(SIEVE(WORD).NE.0) GO TO 6
      BIT=1
      EMPTY=.TRUE.
    6 IF((WORD.GE.400).OR..NOT.EMPTY) GO TO 7
      WORD=WORD+1
      EMPTY=SIEVE(WORD).EQ.0
    7 IF(.NOT.EMPTY) GO TO 1
      DO 9 T=1,400
      DO 8 S=1,31
      IF(IAND(2**(S-1),PRIMES(T)).EQ.0) GO TO 8
      OUT=(T-1)*31+S
      WRITE(6,100) OUT
  100 FORMAT(I10)
    8 CONTINUE
    9 CONTINUE
      STOP
      END
```

プログラム　10

第一に気づくのは定数の定義です。PASCALプログラムではWORDLENGTHやWという名前を具体的な数値のかわりに使っています。この方が実際の数を用いるよりもプログラムの中での意味がはっきりします。また、求めたい素数の範囲を増やす等これらの値を変更したい時，PASCALプログラムでは定数定義部一ヶ所の修正ですみますが，FORTRANプログラムでは何ヶ所かの変更場所をさがし出さなければなりません。

第二の点としては，集合の型や演算がPASCALにはあるのに対し，FORTRANではそれらを表現するのに苦労していることです。つまりSIEVE, PRIMESの宣言をINTEGERで行なっていることや，ビットの位置を示すのに２＊＊（BIT−1）という表現を使わざるを得ないことなどです。（そのため機械語32ビットの最上位の符号ビットは使えません。）また，集合の演算に使ったマスキング演算子　IAND, IOR 等は，幸いに我々のFORTRANにはありましたが，JIS FORTRAN(水準7000)にはないものです。

第三に，制御文 repeat, while 等の使用によってプログラム 9.は制御の流れが見やすくなっているのに対し．FORTRANではその代用としてGO TO文を乱発せざるをえなくなっていて，どの部分がプログラムの意味上のサブユニットをなすのかわかりにくくなっています。

第四に，プログラムの書式やコメントの書き方が，FORTRANでは決められているのに対して,PASCALは自由書式なので，たとえば行の頭下げ (indentation) を効果的に行なっています。

```
    /*   ERATOSTHENES' SIEVE -- BY C.A.R. HOARE.
                 I.H.   1975.8.15.
                 C.F. 'STRUCTURED PROGRAMMING',  PP.129-1309   */

 1  SIEVE:PROC OPTIONS(MAIN);

 2  DCL (SIEVE(0:399),PRIMES(0:399)) BIT(31) ALIGNED,
        1 NEXT,
          2 BIT BIN FIXED(31),
          2 WORD BIN FIXED(31),
       (J,K,S,T,OUT) BIN FIXED(31),
        EMPTY BIT(1) ALIGNED,
        (BITG,JG,SG) BIN FIXED(31),
       (BITP,JP,SP) BIT(31) ALIGNED,
        ZERO BIT(31) ALIGNED;

    /*  INITIALIZE  */
 3  ZERO=REPEAT('0'B,30);
 4  DO T=0 TO 399;
 5    SIEVE(T)=REPEAT('1'B,30);
 6    PRIMES(T)=REPEAT('0'B,30);
 7  END;
 8  SIEVE(0)=REPEAT('1'B,28)||'00'B;
 9  NEXT.BIT=0;  NEXT.WORD=0;
11  EMPTY='0'B;

    /*  FIND NEXT PRIME AND ELIMINATE ITS MULTIPLES  */
12  L1:  ;
13  BITG=2**NEXT.BIT;  BITP=BITG;
15  DO WHILE (BOOL(BITP,SIEVE(WORD),'0001'B)=ZERO);
16    NEXT.BIT=NEXT.BIT+1;  BITG=2**NEXT.BIT;  BITP=BITG;
19  END;
20  PRIMES(WORD)=BOOL(PRIMES(WORD),BITP,'0111'B);
21  J=NEXT.BIT;  K=NEXT.WORD;
23  DO WHILE (K<=399);
24    JG=2**J;  JP=JG;
26    SIEVE(K)=BOOL(SIEVE(K),BOOL(JP,JP,'1100'B),'0001'B);
27    J=J+NEXT.BIT;  K=K+NEXT.WORD;
29    DO WHILE (J>30);  J=J-31;  K=K+1;  END;
33  END;
34  IF SIEVE(NEXT.WORD)=ZERO THEN DO;  NEXT.BIT=0;  EMPTY='1'B;  END;
38  DO WHILE ((NEXT.WORD<399)&(EMPTY='1'B));
39    NEXT.WORD=NEXT.WORD+1;
40    IF SIEVE(NEXT.WORD)=ZERO THEN EMPTY='1'B;  ELSE EMPTY='0'B;
42  END;
```

```
43    IF EMPTY='0'B THEN GO TO L1;

      /*  OUTPUT PRIMES  */
44    DO T=0 TO 399;
45      DO S=0 TO 30;
46        SW=2**S;  SP=SW;
48        IF BOOL(SP,PRIMES(T),'0001'B)¬=ZERO THEN DO;
49          OUT=T*31+S;
50          PUT FILE(SYSPRINT) EDIT(OUT) (F(10)) SKIP;
51        END;
52      END;
53    END;
54    END;
```

プログラム 11

プログラム11.についても上のような比較，考察をＰＬ／Ｉを御存知の方それぞれにやっていただきたいと思いますが，大雑把にいえば，上の第一，二の点ではFORTRANに近く，第三，第四の点ではPASCALに近いといえそうです。

この例はPASCALの強力なデータ構造のよさがよくあらわれた例でした。もちろんPASCALは何にでも向くというわけではありませんが，そのよさのいくらかはこの例で了解されると思います。上のように同じアルゴリズムを異なった言語で記述し比較することはなかなかおもしろいことです。

参考のために最後にそれぞれのプログラムのコンパイルおよび実行のCPU時間の値（計算時間のみで入出力の時間は含みません）を表にしておきます。アセンブラ言語でもプログラムしてみましたので，その結果も付加えてあります。時間はそれぞれ２回の平均をとりました。

	コンパイル時間	実行時間
プログラム－ 9（PASCAL）	0.24 sec	0.51 sec
プログラム－10（FORTRAN）	0.25 sec	1.09 sec
プログラム－11（PL／I）	0.56 sec	1.06 sec
アセンブラ	2.14 sec	0.67 sec

付1　PASCAL 8000 と Standard PASCAL との言語仕様上の相違

これまでほぼStandard PASCALにそってPASCALの解説をしてきました。ここではPASCAL 8000に新たにつけ加えられた機能を簡単に説明します。詳しくは文献〔3〕を参照して下さい。なお，付2以下の内容はPASCAL 8000の仕様にもとづいています。

1　定数宣言部で配列やレコード，集合などの型に属する定数を定義できます。たとえば

```
CONST  ALLZERO = (＃0, 0, 0＃):ARRAY(．1..3．) OF  INTEGER;
       LASTDAY = (＃31, DEC, 1976＃):DATE;
       ALLCARDS = (．1..52．);
```

のようにかきます。

2　メインプログラムの変数宣言部で宣言された変数に初期値を与えることができます。それには変数宣言部の後に初期値設定部をおきます。たとえば

```
VAR  TABLE  :  ARRAY(.1 ..12．)OF  INTEGER;
VALUE  TABLE:=(＃1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1＃);
```

のようにかきます。

3　FORALL文　制御変数がある集合の要素を順番に値としてとり一つの文が繰り返し実行されます。たとえば

```
FORALL  I  IN(．1, 3, 4, 6．) DO
          FLAG(．I．): =TRUE
```

4　LOOP文。繰り返しの文の途中からぬけだすことを許すためのものです。たとえば

```
LOOP  I:=I + 1 ;
      IF  A(．I．) = 0  THEN  EXIT;
      J : = J − 1
END
```

付2　標準手続きと関数

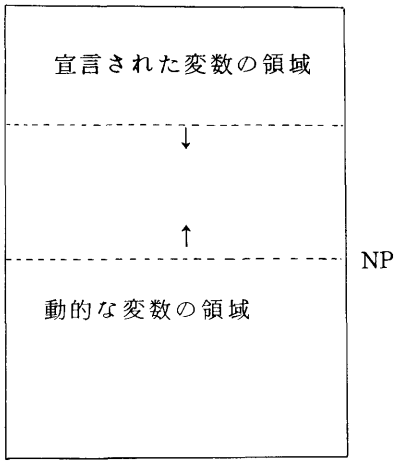
付2.1　標準手続き

(1)　NEW

ポインタ変数によって指される変数を実行時に作りだします。一般形は

NEW(P)

なお，システムの記憶領域の割り当ては下の図のようになっており，NEWでは図のNPの値を所定の数だけ上方へ移動してPに代入することが行なわれます。



(2)　MARK, RELEASE

それぞれ，ポインタ変数を引数とします。

MARK(P)はNPの値をPに代入し，RELEASE(P)は逆にPの値をNPに代入する働きをします。これらはプログラムのある時点から以後に作られる動的な変数が別の時点以後にはすべて不要になることがわかっている場合に使われます。

RELEASEを行なうと，NPの値がMARKの時点の値に戻るので，以後の動的な変数は，MARKとRELEASEの間につくられた変数と同じ領域に作られることになります。PASCALシステムでは，ゴミ集め(garbage collection)を自動的に行なう機能が備わっていないので，大きなプログラムを作るときはこれを有効に使うことが必要になることがあります。

(3)　ファイル関係の手続き

RESET	ファイルを初期状態にする。
REWRITE	ファイルの内容をすべて捨て，新しくデータを書きこめるようにする。
GET	ファイルから要素をひとつよみだしてバッファ変数に入れる。
PUT	バッファ変数にはいっている値をファイルに書き出す。

(4)　入出力の手続き

READ, READLN, WRITE, WRITELN, PAGE, 詳細は12節を参照して下さい。

(5)　その他

TIME(S)　Sに現在の時刻を▼hhmmsstt▼の形で入れて返す。hは時間，mは分，sは秒，tは1／100秒を表わす。

DATE(S)　Sに日付をいれて返す。

いずれもSはPACKED ARRAY(.1..8.) OF CHARの型に属する変数とします。

付2.2　標準関数

(1) 算術関数

ABS	絶対値
SQR	平　方
SIN	正弦関数
COS	余弦関数
ARCTAN	逆正接関数
EXP	指数関数
LN	自然対数
SQRT	平方根

引数はいずれも整数型，または実数型で，結果はABSとSQRが引数と同じ型で，その他は実数型です。

(2) 変換関数

TRUNC　実数の整数化（切り捨て）。

ROUND　実数の整数化（四捨五入）。

ORD　論理型，文字型を含む基本型およびポインタ型の順序数。数え上げで定義された型の場合，最初の項は0,以下それぞれに1, 2…の値が対応づけされ，これがORDの値になります。

例)　TYPE　SCALAR=(A, B, C)の場合
ORD(A)= 0　ORD(C)= 2です。

例)　ORD(FALSE)=0,　ORD(TRUE)=1

CHR　引数は整数型で結果は対応する文字型。

(3) その他の関数

ODD　整数型引数が奇数ならTRUE,偶数ならFALSE。

SUCC　基本型の後続をかえす。つまりORD(SUCC(X))＝ORD(X)＋1です。

例)　TYPE　SCALAR＝(A, B, C)の場合SUCC(A)＝Bです。

PRED　SUCCの反対。

EOF　引数のファイルがend－of－file状態にあるときTRUE,そうでないときEALSE

EOLN　引数のテキストファイルがend－of－lineの状態にある時TRUE, そうでない時FALSE

CLOCK　無引数関数でジョブ開始時からのCPU時間を1／1000秒単位の整数で返す。

付3　演　算　子

PASCALの演算子には次のようなものがあります。

(1) 算術演算子

＋　加算。引数は整数型または実数型で，結果は引数が両方とも整数型の場合整数型，その他の場合は実数型です。

－　減算。型については加算と同じ。

＊　乗算。型については加算と同じ。

DIV　整数型の除算。FORTRANの整数型の除算と同じ規則で答が与えられます。両引数，答とも整数型です。

MOD　剰余。両引数，答とも整数型で，A MOD　B＝A－B＊（A DIV B）の関係が成立します。

／　実数型の除算。引数は整数型または実数型で，答は実数型です。

(2) 比較演算子

次の6つの演算子は，基本型，集合型，ポインタ型と文字列に適用されます。おのおのの型に対する演算子の意味は次のとおりです。

基本型，ポインタ型　　普通の意味の数値の比較。

集合型　　＝，<>は普通の意味。<=，>=は集合の包含関係です。

文字列　　EBCDICにもとづく辞書式の比較

比較の演算子は，2つの引数の型が同じでなければなりません。結果は論理型です。

演算子	適用される型
=, <>	基本型，集合型，ポインタ型，文字列
<, >	基本型，文字列
<=, >=	基本型，集合型，文字列

(3) 論理演算子

NOT

OR

AND

いずれも論理型の引数と結果をもつ演算子で，意味は自明でしょう。

(4) 集合に対する演算子

＋　和集合（union）

＊　積集合（intersection）

－　差集合（difference）

この3つはいずれも，同じ型の集合の間の演算です。

IN　第1引数は第2引数の集合の構成要素の型です。第1引数が第2引数の要素であるかどうかを示し，結果は論理型です。

演算子の評価の順位は次のとおりです。上の方ほど先に評価されます。

1　NOT

2　＊　／　DIV　MOD　AND

3　＋　－　OR

4　＝　<>　<　>　<=　>=　IN

カッコを使って評価の順序を指定できるのはFORTRANなどと同様です。

ここで注意する必要があるのは，演算子の評価の順序がFORTRANやALGOLなどと違っているところがあることです。（特に論理演算子で）。たとえばFORTRANでは

A．EQ．B．AND．C．EQ．D

と書けますが，PASCALでは，これは

（A＝B）　AND　（C＝D）

とカッコでくくらなければなりません。

付4　システムで　義されている語

A．標準名

(1) 定数

FALSE, TRUE, MAXINT, NIL

(2) 型

INTEGER, BOOLEAN, REAL, CHAR, TEXT

(3) ファイル

INPUT, OUTPUT

(4) 関数

ABS, ARCTAN, CHR, CLOCK, COS, EOF, EOLN, EXP, LN, ODD
ORD, PRED, ROUND, SIN, SQR, SQRT, SUCC, TRUNC

(5) 手続き

DATE, GET, MARK, NEW, PAGE, PUT, READ, READLN, RELEASE,
RESET, REWRITE, TIME, WRITE, WRITELN

以上はシステムで用意された名前ですが，これと同じ名前をあるブロックの中で別の意味に定義して使うことは可能です。（お奨めしませんが。）そのブロックの中ではユーザの定義の方が優先します。

B．特殊シンボルとして区切りに使われる語

AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNTO, ELSE,
END, FILE, FOR, FORALL, FUNCTION, GOTO, IF, IN, LABEL,
LOOP, MOD, NOT, OF, OR, PACKED, POSTLUDE, PROCEDURE,
PROGRAM, RECORD, REPEAT, SET, THEN, TO, TYPE, UNTIL,
VALUE, VAR, WHILE, WITH

これらの語は本来の意味以外で使うことはできません。

付5　カタログドプロシジャ

PASCA8000システムをユーザは自由に利用することができます。センターのシステムには次のカタログドプロシジャが登録されているので，容易にPASCALを使うことができます。

プロシジャ名	記号パラメータ	プロシジャステップ名
PASCAL	〔, SYSOUT=$\left\{\begin{matrix}A\\K\\R\end{matrix}\right\}$〕 〔, PARM=▼パラメータ列▼〕	PASCAL

機能：PASCALプログラムのコンパイルおよび実行を行ないます。

パラメータによっていくつかのオプションを選択することができます。

記号パラメータ：

(1) SYSOUT　出力クラスを指定します。省略時はAがとられます。

A …… LP,　K …… カナ付ＬＰ,　R …… 端末出力

(2) PARM　オプションを指定します。パラメータ列は，以下に述べるパラメータをコンマで区切って並べたものです。指定がない時はそれぞれ標準値がとられます。また，（　）内は，オプションの省略形で，これを指定することもできます。

パラメータの種類

(i) {GO(G)|NOGO(NG)}

コンパイルしたオブジェクトコードを実行するかしないかを指定します。標準値はGOです。コンパイル時にエラーが検出された場合にはGOであっても実行はされません。

(ii) {TIME(T)|NOTIME(NT)}

コンパイルと実行の所要時間をプリントするかしないかを指定します。標準値はTIMEです。

(iii) {LIST(L)|NOLIST(NL)}

ソースプログラムのリストを出力するかしないかを指定します。標準値はLISTです。なおリスト出力については付7も参照して下さい。

(iv) CSIZE(CS)＝4096以下の整数

コンパイル時の作業用領域の大きさをキロワード単位で指定します。標準値はCSIZE＝36です。

注)

(v) GSIZE(GS)＝4096以下の整数

ユーザプログラムを実行する際の作業領域の大きさをキロワード単位で指定します。標準値はGSIZE＝36です。

なおPASCALプログラムを実行させるためにはサポートプログラムの領域を含めて，リージョンサイズは{（CSIZEまたはGSIZE）＋20}×4キロバイト程度必要です。

注) 1キロワード＝4キロバイト

関連ＤＤ名

¥PASSRCE(必須) …… ソースプログラム用

¥PASDATA　…… 実行時の入力データ用

付6　PASCAL 8000 システムの使用法

PASCALプログラムのコンパイルと実行を行うためのジョブ制御文は以下のようになります。

```
//ジョブ名    JOB  課題名,パスワード
//      EXEC  PASCAL
//¥PASSRCE DD *
```

ソースプログラム

```
//¥PASDATA DD *
```

実行時の入力データ

```
//
```

ただし，実行時の入力データがなく，かつ，プログラム頭部で標準ファイルINPUTを指定していない場合//¥PASDATA DD * は不要です。

ソースプログラムや実行時の入力データを，データセットから読み込む場合は以下のようになります。

```
//ジョブ名    JOB    課題名,パスワード
//      EXEC PASCAL
//¥PASSRCE DD DSN=データセット名, DISP=SHR
                        (ソースプログラム用)
//¥PASDATA DD DSN=データセット名, DISP=SHR
                        (データ用)
//
```

付7　コンパイラオプション

コマンドのパラメータの他に，プログラムのテキスト中にコンパイラに対していくつかのオプションの指定を挿入することができます。これらはプログラムの中の挿入された場所以降で有効で，従って一つのプログラムの中で何度でもオプションを切換えることができます。オプションの指定はコメントの一部を使って行なわれます。一般形は次の通りです。

(＊¥X{＋／－}，Y{＋／－}…＜任意のコメント＞　＊)

ここでX，Yはオプションの種類を示す文字で，後にあげる4種類のうちいずれかです。次に続く文字が＋の場合はそのオプションで指定した動作の開始を，－の場合は終了を示します。なお¥（一部のTSS端末では＼（逆スラッシュ）になります）の前後，コンマの前後などにブランクをいれることはできません。

オプションの種類

1) C　コンパイルされて生成されたオブジェクトコードのリストをアセンブラの形式で出力することを指定します。標準値は－です。

2) L　ソースプログラムをリストすることを指定します。標準値は＋です。

3) T　次の3種類のチエックを実行時に行なうコードを生成することを指定します。標準値は＋です。
　○スカラー型および部分範囲型変数に対する代入時の値の範囲のチエック
　○ array のインデックスおよびポインタの値の範囲のチエック
　○ case　文のセレクタの値の範囲のチエック
　これらのチエックは実行効率を落とすのでデバックが終わったプログラムにはT－を指定して行なわないようにしておくことが望ましいのですが，デバックには役立ちます。

4) S　1コラムから72コラムの間だけをデータとして扱う。標準値は－です。

オプション指定の例

```
(＊¥T－，C＋，L－　　COMPILER OPTIONS ＊)
(＊¥L－＊)
```

付8　ファイルの使用法

PASCALプログラムにおけるファイル型変数には2種類の使い方があります。1つはプログラムの中だけで使われるファイルでプログラム頭部で宣言されていないものです。この種類のファイルに対するデータセットのわりあておよび解除はPASCAL8000システムによって自動的に行われるのでDD文は不要です。

もう一つはプログラム頭部でプログラムに対するパラメータとして宣言されているファイルです。この宣言は外部のファイルをPASCALのプログラムで取り扱うためのもので，宣言された各ファイルに対してプログラム内のファイル型変数の名前と同じDD名をもつDD文で，データセットをわりあててやる必要があります。

DCBパラメータはDASCALシステムによって決められるのでユーザが指定する必要はありません。

例　プログラムXが，INFILEからデータをよみこんでOUTFILEに結果を出力するときのDD文は次のようになります。

```
//    EXEC  PASCAL
// ¥PASSRCE DD DSN=F0001.X  ,DISP=OLD
// INFILE DD DSN= 0001.XDATA1, DISP=OLD
// OUTFILE DD DSN=F0001, XDATA2,
// SPACE=(TRK, (10, 10)  DISP=(NEW, CATLG), UNIT=PUB
//                                  注) 下線部はユーザが任意に指定
```

また，このプログラムのプログラム頭部は，PROGRAM X(INFILE, OUTFILE, OUTPUT)

注意

1. 標準ファイルのOUTPUTはプログラム頭部で宣言されなくてはなりません。標準ファイルINPUTとOUTPUTに対しては変数宣言不要であり，DD文を与える必要もありません。
2. 標準手続きGET, PUTを呼び出す時にはあらかじめ同じファイルに対して標準手続きRESET, REWRITEをそれぞれ呼び出しておかなければなりません。(ただし，INPUTとOUTPUTを除く)
3. PASCAL8000ではファイル変数はメインプログラムの変数宣言部で宣言されていなければなりません。

付9　エラーメッセージ

```
  1: error in simple type
  2: identifier expected
  3: 'proqram' expected
  4: ')' expected
  5: ' ' expected
  6: illegal symbol
  7: error in parameter list
  8: 'of' expected
  9: '(' expected
 10: error in type
 11: '(.' expected
 12: '.)' expected
 13: 'end' expected
 14: ';' expeeted
 15: integer expected
 16: '-' expected
 17: 'oegin' expected
 18: error in declaration part
 19: error in field-list
 50: error in constant
 51: ':=' expected
 52: 'then' expccted
 53: 'until' expected
 54: 'do' expected
 55: 'to'/'downto' expected
 58: error in factor
 59: error in variable
 60: 'in' expected
101: identifier declared twice
102: low bound exceeds high bound
103: identifier is not of appropriate class
104: identifier not declared
105: sign not allowed
106: number expected
107: incompatible subrange types
108: file not allowed here
109: type must not be real
110: tagfield type must be scalar or subrange
111: incompatible with tagfield type
112: index type must not be real
113: index type must be scalar or subrange
114: base type must not be real
115: base type must be scalar or subrange
116: error in type of standard procedure parameter
117: unsatisfied forward reference
119: forward declared: repetition of parameter list not allowed
120: function result type must be scalar, subrange or pointer
121: file value parameter not allowed
122: forward declared function: repetition of result type not allowed
123: missing result type in function declaration
124: f-format for real only
125: error in type of standard function parameter
126: number of parameters does not agree with declaration
127: illegal parameter substitution
128: result type of parameter function does not agree with declaration
129: type conflict of operands
120: expression is not of set type
131: tests on equality allowed only
132: strict inclusion not allowed
133: file comparison not allowed
134: illegal type of operand(s)
135: type of operand must be boolean
```

136: set element type must be scalar or subrange
137: set element types not compatible
138: type of variable is not array
139: index type is not compatible with declaration
140: type of variable is not record
141: type of variable must be file or pointer
142: type conflict on parameters
143: illegal type of loop control variable
144: selector type must be scalar or subrange
145: type conflict with control variable
146: assignment of files not allowed
147: label type incompatible with selecting expression
148: subrange bounds must be scalar
149: index type must not be integer
150: assignment to standard function is not allowed
151: assignment to formal function is not allowed
152: no such field in this record
153: type error in read
154: actual parameter must be a variable
155: control variable must not be formal
156: multidefined case label
158: missing corresponding variant declaration
159: real or string tagfields not allowed
160: mismatch to forward declaration
161: again forward declared
164: substitution of stantard proc/func not allowed
165: multidefined label
166: multideclared label
167: undeclared label
168: undefined label
169: error in base type
170: procedure/function parameter must have value parameters only
172: undeclared external file
175: missing file 'input' in program heading
176: missing file 'output' in program heading
180: too long source line
181: tagfield value out of range
182: assignment to subordinate function name not allowed
183: file variable must be global
184: too long file component
185: read must not be applied to packed type
186: mismatch to procedure skeleton
187: packed variable is not allowed in variable parameter
201: error in real constant: digit expected
202: string constant must not exceed source line
203: integer constant exceeds range
220: variabe initialization allowed only in main program
221: type conflict in variable initialization
222: number of components of struetured constant does not agree with declaration
223 type of components of structured constant does not agree with declaration
224: illegal format in structured constant
225: '#)' expected
226: unallowed type in structured constant
227: record with variants not allowed in structured constant
250: too many nested scopes of identifiers
251: too many nested procedures and/or functions
253: procedure too long
255: too many errors on this source line
261: too many procedures or long jumps
280: event name declared
281: no postlude statement allowed for the event 'exit'
282: multidefined postlude statement
302: index expression out of bounds
303: value to be assigned is out of bounds

304: element expression out of range
399: not implemented
400: compiler error

参考文献

〔1〕 K.Jensen and N.Mirth, PASCAL-User Manual and Report, Springer, 1974; Second Edition, 1975.

〔2〕 N. Wirth, 和田訳, プログラム言語PASCALの文法, bit, vol. 8, no.4, pp. 341－366, 1976年4月。

〔3〕 T, Hikita and K.Ishihata, PASCAL 8000 Reference Manual, Dept.of Info.Sci., Univ. of Tokyo, March 1976.

〔4〕 K.Ishihata and T.Hikita, Bootstrapping PASCAL Using a Trunk, idid., March 1976.

〔5〕 石畑，疋田，安村，石田，PASCALコンパイラの開発，東京大学大型計算機センター年報，vol.6, pp. 123－134, 1976年.

〔6〕 疋田，コンパイラのキットを用いたPASCALの移植，日経エレクトロニクス，no. 149, pp. 100－131, 1976 12－13.