# 東京大学教育用計算機センターにおける PASCALについて

東京大学工学部計数工学科和田研究室
金田　泰

教育用計算機センターにおいては，すでにPascal が使用できるようになっており，多くのユーザによって使用されているが，ここにその使用の手引きをまとめておく。



## 1　Standard Pascal との違い

教育用計算機センターのPascalコンパイラは，Standard Pascal に比べて制限された部分もかなりあり，また拡張された部分もある。また，不備な点もある。これらの点について述べる。これ以外の点については，下記の文献を参照されたい。

Jensen, K. and Wirth, N. 〔1975〕
'Pascal User Manual and Report', Springer-Verlag, New York
(上の'Report'の邦訳：和田英一訳「プログラム言語PASCALの文法」, bit 8：4, 341－366〔1976〕)

1　記号，文字の違い

1)　記号：

| Standard Pascalの記号 | Melcomにおける記号 |
|---|---|
| {, (* | (* のみ可 |
| }, *) | *) のみ可 |
| [ | (. |
| ] | .) |
| ↑ | @ |

2)　予約語は名前と同じように，普通の大文字でかく。

3)　小文字は使えない。

(以下の記述においては，Standard Pascalの記号および小文字を用いる。また，予約語はアンダーラインをつけてかく。計算機にかけるときには，上の規則に従って変換しなければならない)。

2　型に関する制約

1)　型の同一性に関して

異なる場所で宣言された名前のついていない型の変数は違う型とみなす。また，名前の違う型は異なるものとみなす。

例1)　a, b : array[ 1 . . 2 ]of integer ;
　　c　　: array[ 1 . . 2 ]of integer
　　a := bは許される。
　　a := cは許されない。 □

例2)　type t 1 = record f, g : integer end ;
　　t 2 = record f, g : integer end ;
　　var a : t 1 ;　b : t 1 ;　c : t 2 ;
　　a := bは許される。
　　a := cは許されない。□

2)　同じ型を他の名前であらわすことはできない。

例) type int = integer は許されない。 □

3)　文字列定数は使えない。ただし，write，writelnのパラメタとなっている場合は許される。また，alfa型の場合は定数定義の中を除いて許される（従って，みかけ上，文字数2～8字の文字列定数は許される。この場合，《write，writelnの引数の場合は除いて》右側に空白がつめられて，8字のalfa型定数とみなされる。）（alfaに関しては6)を参照）

例1）　var a：alfa；
　　a：＝'al'；{ a：＝'al␣␣␣␣␣␣'と等価 }
　　a：＝'a'　　{ error（　a はchar型定数だから）}　□

例2）　const　a＝'al'；{ error }
　　c＝'c'；{ OK }　□

4) 部分範囲型に関して

integer以外の型の部分範囲型は許されない。ただし，配列を宣言するとき添字型としてかくことは許される。

例）　array[ 1 . . 10 ] of 'a'. . 'z' は許されない。
　　array[ 'a'. .'z'] of integer はよい。
　　また，Scl＝( a，b，c )のとき，Sub＝a . . bは許されない。　□

5)　setに関して

setの要素は，そのordが0以上63以下でなければならない。従って，要素の数は64個以下である。

例）　var S：set of　63 . . 64は許されない。同様に，type t＝( t1，t2，…，ti，……，tn )においてord( tn )>63ならvar S：set of ti . . tnは許されない。　□

6) 予約語packedは意味をもたない。従ってpacked arrayは使えない。そのかわり，型alfaがあるが，これは8文字からなる文字列であり，Standard Pascalのpacked array[ 0 . . 7 ]of charに近く，pack，unpackはできるが，a[ i ]{a：alfa}のような形では使えない。すなわち，char型の定数または変数との間の変換をおこなうには，pack，unpackを用いるほかはない。*

7)　textとfile of charは異なる。

file of charは1文字1レコードのファイルとなるので効率が悪いうえ，これに対し，write readなどの標準手続きは使えない。

---

* 従って，このコンパイラのalfa型は，Standard Pascalのそれではなく，Wirth . N. : The Programming Language Pascal，Acta Informatica，1：1，35－63［1971］のalfaに等しい。

8)　alfaおよび要素の上限のordが32以上のsetについて

これらの型に関してはコンパイラにbugsがあるので注意

array[ type1, type2 ]of alfaは　array[ type1 ]of array[ type2 ] of alfaとかかなければ正しく処理されない。

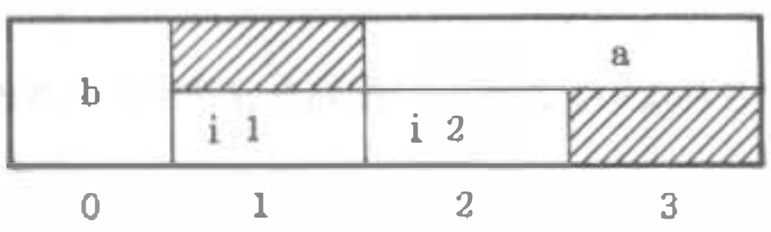
<参考>

```
r : record case b : boolean of false : ( a : alfa ) ;
                               true : ( i1, i2 : integer ) end
```

とすると，rは次のようにとられる。

倍語境界

↓




9)　scalar 型に関して

次の例は多重宣言として扱われる。

```
var  k : (male, female) ;
     l : (male, female) ;
```

すなわち，scalar型の中にあらわれる名前はすでに同じレベルで宣言されたものであってはならない。上の例では，male，femaleは2回あらわれている。*

10)　recordの可変部について

tag fieldは空であってはならない。たとえば，

record …… case　Boolean　of …… endは許されない。

field並びは空であってはならない。

```
record case b : Boolean of true : ( field : T ) ;
                           false : ( )              end
```

は

```
record case b : Boolean of true : ( field : T ) ;
                           false :                  end
```

のようにかく。（Standard Pascalでは前者のようにかくことになっているが）。

---

*　例は次の論文から引用した。この文献では，この種の問題をやや詳しく扱っている。

J.Welsh,W.J.Sneeringer and C.A.R.Hoare, 'Ambiguities and Insecurities in Pascal', Software practice and experience, vol 7, 685－695 [1977]

3　式に関して

1）あまり複雑な式は許されない。

たとえば，var　a，b，c，d：integerのときa＋(b＋(c＋(d)))は許されない。3重以上の( )は許されないし，2重でも，最も内側にsetやalfaの演算式を含むときは許されない。(従来は最も内側の括弧の外がsetやalfa以外の項からなっていれば，もっと複雑な式をかくことができたが，これは，コンパイラの虫をとるために犠牲にされた。)

2）set型の式の中に　. .　は許されない。

例）[ 0 . . 2 ]は[ 0，1，2 ]とかかなければならない。□

3）構造型の比較

同じ構造型の変数を比較(＝，＜＞)することができるが，可変部をもつrecord型であっても，bitごとの比較をするので，その点注意を要する。alfa型の変数については大小の比較もできる。

例1）　a，b：alfa

a＝b，a＞＝b，a＞b，a＜＝b，a＜b，a＜＞bは可能。

たとえば，E：＝a＞b　の意味は，

```
a, b : packed array [ 1 . . 8 ] of char
i := 1 ;
while ( a[ i ]= b[ i ]) and ( i < 8 ) do  i := i+1 ;
```

E：＝a[ i ]＞b[ i ]によって記述される。□

例2）
```
a, b : record  case tag : Boolean of
                    false : ( i : integer ) ;
                     true : ( c1, c2 : char )  end
a. tag := true ; a . c2 := 'a' ;
b := a ;  b . c2 := ' b ' ;
a . tag := false ; b. tag := false ;
a . i := 1 ; b . i . = 1 ;
```

のとき，a＝ bはfalse．□

4）論理式の評価のしかた

Boolean型の式は，全体を評価してからその値をきめる。すなわち，true or E，false and Eにおいて，それぞれEを評価する。従って，次の場合には注意が必要である。

①　Eが副作用をもつ関数を含む場合(関数は副作用をもたないことが望ましいが)。

例）
```
var a : Boolean ;
function  f : Boolean ;
begin  a := true ; f := false end ;
```

begin　a := false ; write ( f or a ) end.

は true を出力する。*　□

② Eが評価不能な場合

例）　p↑ := nil ;

while ( p = nil ) or ( p↑ = 8 ) do ……　□

4　手続き，関数

1）　alfa型定数，set型の式は実パラメタとしてかけない。

2）　関数の型として，set，alfaは使えない。（診断メッセージがでず，正しくない目的プログラムが生成される場合があるので注意を要する。）他の構造型はもちろんダメである。

3）　variableパラメタをもつ手続き，関数をパラメタとして受けわたすことはできない（これはStandard　Pascalの文法書にもかかれている）。

例）　function dum ( var i : integer ) : integer ;

begin …… end ;

procedure proc ( function dum : integer ) ;

begin …… end ;

begin proc (dum)　end.

は，最後の行で#200 の error とされる。最初の行が，

function dum( i : integer ) : integer ;

であれば許される。□

4）　関数の内側に手続きまたは関数があるとき，内側で外側の関数に対する代入をおこなってはならない。

例）　function f : integer ;

procedure p ;

---

*　次の例はさらにおそろしい副作用をもつ例である。

```
program side effect ( output ) ;
var f : text ;
function change : char ;
begin writeln ( f ) ; change := 2 end ;
begin rewrite ( f ) ; writeln( 1 , change, 3 ) ; writeln( 4 )
end .
```

このプログラムの出力は次のようになる。

outputの出力：

14

fの出力：

23

（これは，コンパイラの虫だという説がある。）

begin　f：＝1　end ;

begin …… end は許されない。

4） 標準関数について

○　round は用意されていない。

○　ord (x) ： x が文字の場合については「 11. その他 」を参照。

x が整数のときは x ＝ ord (x)。

x が実数のときは， ord (x)はその bit pattern を整数と解釈したときの値。

5　手続きの大きさ等に関する制限

1） 手続き（関数 ）の大きさについて

1つの手続きはあまり大きくてはいけない。（ 目的プログラムで， 1,500語以下 ）

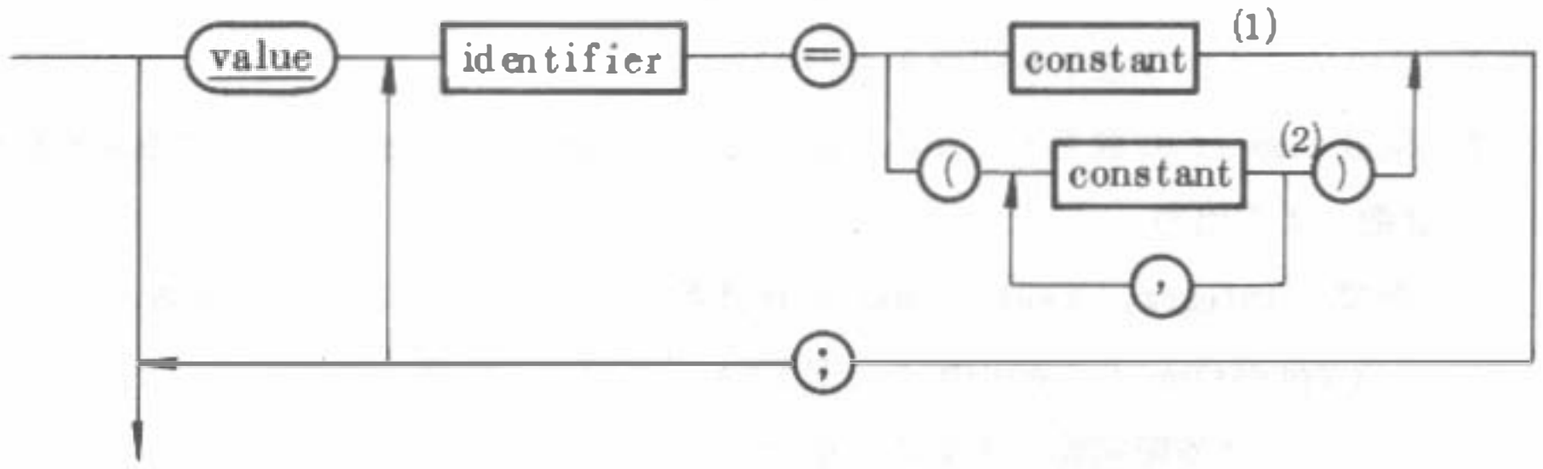
通常は，原始プログラムで200～300行くらいまでOK。しかし，出力の多い program ではもっときびしい。

2） 定 数 の 数

1つの手続きの中にあまり多くの定数をかくことは許されない。 integer，real などで 60個， alfa，最大の要素の ord が 32以上の set で 30個以下でなければならない。

6　変数の初期値設定について

大域変数に初期値を与えるための宣言ができる。変数初期値設定部は変数宣言部の次におかれ，構文は次のようである。




＜ identifier ＞ は，直前の変数宣言で宣言された変数名であり，そこで宣言された順に初期値の定義もおこなわれなければならない。単純型変数の場合は上の(1)に従い，配列型変数の場合は(2)に従って定義される。（ alfa型の変数は，単純型変数とみなされる ）。

例 1)

```
var      i : integer ;
         a : array [ 1..3 ] of real ;
value    i = 1 ;
         a = ( 0.0 , 1.0 , 2.0 ) ;
```

この例において， a＝( 0 , 1 , 2 )とかいてはならない。このようにかくと， a は

real の０，１，２ではなく，integer の０，１，２と同じ bit pattern に初期化される。これを利用して，次のようなことが可能である。

a ＝（'41100000' X, '42800000' X, '43100000' X）

（「11. その他[4]」を参照）□

例 2)

var　p : ↑ integer ;

value　p ＝'40000' X { ＝nil } ;

この定義により，pに nil を初期値として与えることができる（p＝nil とかくことはできない）。

すべての pointer 型に対する nil の値は等しい（'40000' X で nil にできる）。□

7　入　出　力

1）数の入力誤り

数を入力するとき数でないものがあらわれると，'＊＊＊ integer expected ……' のようなメッセージを印刷し，次に誤りのあったカードを印刷し，'@' で誤りの位置を示した後，実行を中断する。

実行を中断する理由は，このような誤りの多くは，操作の誤りにより，データとしてプログラムをよんでしまうことによっておこり，[*1] このような場合には，実行を継続することはむだだと考えられるからである。

2）printer control character は存在しない。

例）write（'1'）は 1 を印刷する。□

3）標準関数 page は引数をもたない。従って，この操作は output に対してのみ可能である。

4）16進数による出力

次の形で，integer，real，char を16進数として出力することができる。

expression { : width } ␣ HEXA [*2]

{ {…} は省略可能であることを示す }

width は幅指定。

width の default は 8 である。

1 ≦ width ≦ 7 とすると，上位桁の有効，無効にかかわらず，下位 width 桁が出力される。

---

＊1 プログラムとデータは同じDCB M:SI からよむ。従って，プログラムをファイルから，データをカードからよむときなどには注意を要する（操作法を参照されたい）。

＊2 名前 'hexa' は，この位置にあらわれたときのみ特殊な意味をもつ。'forward' 同様通常の名前としても使用できる（'Pascal', 'Fortran' も同様 ―「separate compiling に関して」参照）。

5) alfaの出力幅

alfa型の変数をwrite, writelnによってかくときは，幅を7以下に指定すると左から指定された幅だけ出力される。

6) 幅指定のないときの数の出力

幅指定せず続けてintegerをwriteすると，ライン・プリンタの場合1行に12個ずつそろえて出力されるが，realのときはそろわないので注意。

7) writelnは改行しない場合がある。

writelnをつづけて2つ以上おこなっても改行は1度しかおこらない（とくに，まだなにも書かれていないtext fileに対して何度writelnをおこなっても改行されない。空行をおきたいときは，たとえばwrileln（' '）のようにする。*

例） writeln(1); writeln ; writeln(2)は

```
1
2                    ←空行なし
```

を出力する。□

8) eofになっても次の場合は，fileに対する書きこみはできない。

getのくりかえしの後にeofになり，そのまま（rewriteせず）fileに書き足そうとする場合（任意の型のfileについて）。

例） var f : text

while not eof (f) do get (f);

writeln ( f, ..... )

は，fを変化させない。□

この理由により，file inputへの出力は許されない。

8 プログラム・パラメタの意味および外部定義

プログラム・パラメタは，このPascalにおいては，プログラムの外部で宣言され，そのプログラムで使うものをかく。text型またはその他のfileの変数は，プログラムの中で宣言される（input, outputを除き）から，プログラム・パラメタとしてはかかない（かいてはならない）。これをプログラム・パラメタとしてかくと，それは外部にあるものとみなされるから，外部にそれがないときには，load時にerror（PRIMARY REFFRENCE NOT LOCATED）となる。input, outputは外部（PASCLIB）にあるので，プログラム頭部にかくが，これは省略することもできる。省略しても同じように扱われる。従って，プログラム・パラメタの中にinputをかかず，read（input ……），get（input）を用いなくても，inputは自動的にresetされる。

---

* Standard Pascalでは，writelnをおこなった回数だけ改行される。

プログラムの中で宣言し，それを外部で用いる変数および手続き，関数は，その名前の前に'def'をつけて宣言する（宣言のときだけこのようにする）。

例1）
```
procedure  def  external ;
    begin ……… end  □
```

例2）
```
var  def  ext 1,  def、ext 2,  int : T
    { int は外から見えない }  □
```

この機能のため，defは手続き，関数，変数の名前として使うことはできない。

9　separate compilingに関して

このPascalコンパイラは，separate compilingの機能をもつ。これによりoverlayが可能である。また，アセンブリ言語でかかれたプログラムと接続することもできる。

例1）
```
program main ( p , q , input , output ) ;
    ……
procedure p ; Metasym ; {本体なし}
  { pはMetasymbol（Assembly言語）でかかれたものであることを示す }
function  q ( a : T ; var  b : Y ) ; Pascal ; {本体なし}
  { qはPascalでかかれた別のprogram内で外部定義されたfunctionであることを
    示す }
      ⋮
begin { main } ……… end.


program  def  sub ( input,  output ) ;
      ⋮
function  def  q ( a : T ; var  b : Y ) ;
  { body }
      ⋮
end ; { main  programなし } ■
```

上の例において，型T，Yは2つのプログラムで全く同じに定義されていなければならない。同じ変数を使うときには，やはり同じに宣言されていなければならない。

この例において'Pascal'のかわりに'Fortran'とかくこともできるが，Fortranでかかれたプログラムと自動的にlinkできるわけではない。

上のプログラムをコンパイルして作ったROMをlinkするときには，次の点に注意しなければならない。（ROM名もMAIN，SUBであるとする）。

```
! LOAD (EF, (SUB), (MAIN)), (UNSAT, (PASCLIB))
                ↑
          MAIN MODULEを最後にかく
```

例2）

3つのプログラムを overlay する場合の例

```
! LOAD (EF,(SUB1),(SUB2),(MAIN),(UNSAT,(PASCLIB)),;
!      (LMN,LOADMODULE),(PERM),(BREF)
! TREE  MAIN-(SUB1,SUB2)
```

これで，次の形の loadmodule が，LOADMODULEという名のファイルにできる。


（MAIN，SUB1，SUB2はROMの名前である）□

10　診断メッセージ

1）印刷されない（コンパイル時）診断メッセージについて

番号の大きいerrorに対しては，適切な診断メッセージが印刷されない。すべてのerrorの listを付録に示す。（Standard　Pascalと番号がちがうので注意されたい）。

2）実行時errorのおこった場所を知る方法

コンパイラ出力するリストの左から2番目の数（16進数）は，その行の原始プログラムに対応する目的プログラムの相対番地をあらわす（およそ行のはじめの部分に対応する番地をあらわす。）。実行時errorにより実行が中断されたときは（致命的なerrorでない限り）* 'RELATIVE　ADDRESS='の次に相対番地が示されるので，errorがおこったおよその位置を知ることができる。ただし，errorは必ずしもユーザ・プログラムの中でおこるとは限らないし，**2つ以上のプログラムをlinkして作ったload　moduleにおいては，奇妙な番地が印刷されることがある。

3）実行時errorに関して

①　入力のerrorに関しては，一部すでに述べた。

eofになっているのにgetしようとしたときは，*** END OF FILE *** の

---

*　致命的なerrorとは，すなわち操作システムに制御を奪われてしまうようなerrorをさす。

**　ユーザ・プログラムの相対番地の最大値より大きな値が示されているときは，Pascal monitorの中でerrorが検出された場合である。Pascal　monitorは，ユーザ・プログラムの実行を助けるためにユーザ・プログラムにlinkされるが，ここでerrorがおこるのはたいがい入出力のときである。

errorとなるが，このときはRELATIVE ADDRESSが正しく表示されない。従って，いずれのreadがerrorをひきおこしたかということは，これだけではわからない。

② VALUE OUT OF RANGEのerrorは，expressionの値が期待される範囲にはいっていないときにおこる。

例1）

```
var  ch : char
```

ch := 'ア'は，このerrorをおこす。なぜなら，

not ( ch in [ chr(0) .. chr(63) ] ) * であるから。□

例2）

```
var   i : 0 .. 10
      i := 11
```

は，このerrorをおこす。□

この検査をはずすには，option A－を使う。（操作法　参照）

② INDEX OR CASE VAR OUT OF RANGEのerrorは，配列のindexが宣言された範囲をこえたとき，もしくはcase文のcaseの次の式の値が，case名札の中にない値をとったときにおこる。たとえば，

例1）

```
var   a : array [ 1..5 ] of integer ;
      i : integer
i := 6 ;…… a [ i ] ……□
```

例2）

```
case 0 of
  1 : S1 ;
  2 : S2
end □
```

この検査をはずすには，option X－を使う。

11 そ の 他

1) ' 'はletterである。

characterのsetは次のようである。

（各機はiとchr(i)の組である。）

---

* このPascalコンパイラはこの記法を知らない。念のため。この検査をすることが期待されるところで常に，コンパイラが検査コードを生成しているとは限らないので注意されたい。

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | — | 1 | A | 2 | B | 3 | C | 4 | D | 5 | E | 6 | F | 7 | G |
| 8 | H | 9 | I | 10 | J | 11 | K | 12 | L | 13 | M | 14 | N | 15 | O |
| 16 | P | 17 | Q | 18 | R | 19 | S | 20 | T | 21 | U | 22 | V | 23 | W |
| 24 | X | 25 | Y | 26 | Z | 27 | 0 | 28 | 1 | 29 | 2 | 30 | 3 | 31 | 4 |
| 32 | 5 | 33 | 6 | 34 | 7 | 35 | 8 | 36 | 9 | 37 | + | 38 | − | 39 | ＊ |
| 40 | ／ | 41 | ( | 42 | ) | 43 | $ | 44 | = | 45 | | 46 | , | 47 | . |
| 48 | ' | 49 | ¢ | 50 | ! | 51 | : | 52 | # | 53 | % | 54 | \| | 55 | & |
| 56 | @ | 57 | '' | 58 | < | 59 | > | 60 | ? | 61 | ¬ | 62 | ∧ | 63 | ; |

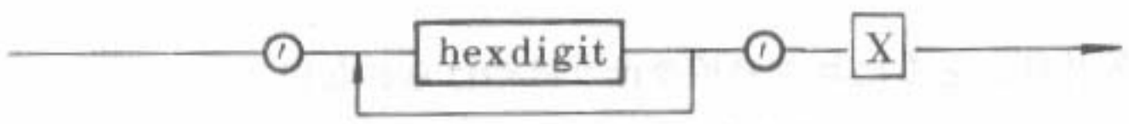
端末によっては，chr（0），chr（49）は特殊文字となる場合がある。

2）mod の定義

a　mod　b = abs（a）− abs（a）div　b＊b≧0 *

3）pointer 型の変数に値を代入せずに使うと何がおこるか全く予期できないので注意されたい。

4）プログラム中での 16 進数の記法




＜hexdigit＞::＝0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
A | B | C | D | E | F

16 進数は integer として扱われる。

5）特殊な名前について

'def' は手続き，関数，変数およびプログラムの名前としては（def　def という場合以外）使えない。

次の名前が標準の名前である。

get , put , reset , rewrite , pack , unpack , insert , append , read , write,
new , readln , writeln , odd , ord , chr , eof , abs , sqr , trunc , pred ,
succ , eoln , alfaleng , input , output , alfa , real , char , Boolean ,
false , true , integer , sin , cos , exp , ln , sqrt , arctan , time , page,
date

このうち，

① procedure　insert（a, b : integer ; var　c : integer）は，a を左へ b bits shift したものと c と bit ごとの論理和をとり，結果を c に入れる（a，b はかわらな

---

＊　Standard Pascal においては，

m　mod　n＝m−（（m　div　n）＊n）

が成りたつとされているが，このコンパイラにおいては，この関係はなりたたない。

い)。

② procedure append (var n : integer ; b, c : integer)は，aを左へb bits shiftして，cとbitごとの論理和をとり，結果をaに入れる(b, cはかわらない)。

これら2つの手続きはPascalコンパイラで使用するために組みこまれたもので，一般のユーザの使用を目的としていない。

③ time, dateはimplementされていない。

6) case名札に関する制約

このPascalにおいては，符号なし定数でなければならない。

例)

```
var  i  :  integer
case  i  of
+1 : S1
-1 : S2
end
```

の2つのcase名札は，ともにコンパイラはうけつけられない。□

7) プログラムが途中で切れている場合*

この場合は，コンパイラはただちにコンパイルを中断する。通常の終了とちがう点は，1)プログラムの最後の行の次にあたかも空行があるかのようにみえること，2)たとえerrorが検出されていなくても，

* ABOVE ERROR(S)……

のメッセージがでることである。

8) 極端に大きな配列をとると，linking(！LOAD)のときにerrorをおこす。

コンパイラは術あふれがおこらない限りどんな大きな配列をも許すが，LOADERは，"illegal org"というメッセージをだしてlinkingを中止してしまうことがある。

9) ある種の構文誤りは検出されない。たとえば，case名札並びに関する診断はかなりあまくなっている。おそらく実害はないだろうが注意されたい。

10) (goto)名札の数は20以下である。

11) case名札として極端に絶対値の大きな値は許されない。また，差が極端に大きいcase名札が1つのcase文の中にあらわれてはならない，(最大でも数100どまり，一般には数10以下におさえるべきである)。

なお，record等に関しては，まだ虫があると思われる。ほかにもみつかっていない虫があると思われる。もし疑問の点があれば，プログラム指導員金田に教えていただきたい。

---

* たとえばプログラムのおわりの'.'がぬけている場合

## 2　操　作　法

### 1　プログラム，データをカードで与えるとき

カードは次のように並べる。


*　やや大きいプログラムのとき、大きな配列があるときなどは
もっとCOREを大きくとる。
(ORDER)は不要である。

**　これらのカードは、データの入力を要しないときには不要。

### 2　TSSにおける使用法

1)　標準的な使用法

TSSでは，次のようにして使うことができる。

! PASCAL␣Sourcefile *　+　↵

! LYNX␣¥␣ON **␣loadmodule: .PASCLIB　↵

! loadmadule. ␣datafile ++　↵

---

*　プログラムはあらかじめファイルにつくっておく。
(直接入力することもできるが，あまりすすめられない)。

**　loadmoduleと同名のファイルが存在しないときは，" ON "でよいが，すでにある同名のファイルをこわしてかくときには" OVER "を用いる。

+　従来ROMをgo file(¥)上に作ろうとしてもできないことがあったが，1978年9月以降できるようになった。

++　data fileはinputからよむデータがはいったファイルである。これを指定しないと端末から入力することになる。inputを使っていないつもりでも，自動的にinputはresetされるので，1行だけは常によみこまれる。すなわち，data fileを指定しないときは，

! loadmodule.

のあと　CR　を2回押さなければプログラムの実行は開始されない。

上の記述において，sourcefile，loadmoduleは，使用者が勝手に名前をつけてよい。すなわち，たとえば，

```
! PASCAL␣MYS●URCE ↵
! LYNX␣￥␣OVER␣PASCAL ;. PASCLIB ↵
! PASCAL . ↵
```

のようにできる。

2) 短　縮　形

```
! PASCAL␣sourcefile ↵
! RUN ;. PASCLIB ↵
```

データがない，または端末からよむときは上のように短縮することができる。

3) TSSでの限界

accountのclassがB，Tの場合には，小さなプログラム（100～200行程度）ならTSSでコンパイルできるが，class Aではいかなるプログラムもコンパイルできない。

また，class B，Tで，プログラムが大きすぎてコンパイルの途中で放棄するときは，

STACK OVERFLOW

などのメッセージが印刷される。

3　ファイルからのプログラムの入力方法

（データをカードから入力する場合）

この場合，まちがえる人が多いので，特に記しておく。

```
! JOB ･･･
! ASSIGN␣M:SI, (FILE, ･･･ )
! PASCAL
! LOAD␣(GO), (UNSAT, (PASCLIB))
! ASSIGN␣M:SI, (DEVICE, CR)
! RUN
! DATA
  :
```

これを忘れる（入れない）人が多い。

4　input，output以外のファイルに関して

このPascalにおいては，すべてのファイルはプログラムの外に作られ，ユーザはこれをはっきり操作することが必要である。しかしながら，ファイル名はプログラム・パラメタとして書いてはならない（「Standard Pascalとの違い」の「プログラム・パラメタの意味および外部定義」参照）。

はっきり操作することが必要であるという意味は，次のような制御カードを！RUNカー

ドの前（もしくは！PASCALーカードの前）に入れる必要があるという意味である。

！ASSIGN F：ファイル名*，（FILE，名前）

ここで，「ファイル名」は，プログラム内で宣言した名前であり，「名前」はそれと等しくなくてもよい，適当な文字列である。

複数のファイルを用いるときには，それぞれに対して，このようなカードを必要とする。このカードを入れないと，ファイルに書きこんだつもりでも実際には書かれない（このカードを入れてあれば，プログラムの中におけるファイルに対する最初の操作がresetであろうとrewriteであろうと，　正しく処理される）。

例）

```
! JOB ･･･
! ASSIGN␣F：F，（FILE，FILE：F）
! PASCAL
  PROGRAM TEST（●UTPUT）；
  VAR F：TEXT；
  BEGIN REWRITE(F)；WRITELN（"WRITTEN？"）END.
! LOAD␣(GO)，(UNSAT，(PASCLIB))
! RUN □
```

！ASSIGNーカードは，この場合物理的なファイルを論理的なファイルに割りつける働きをする。従って，上の例においては"FILE：F"という名前の物理的なファイルが作られる。上の場合，jobの実行後もこのファイルは存在しつづけるので，このようなjobを，いちいち物理的なファイルの名前をかえて何度も実行すると，物理的なファイルが使い果たされて，もはやjobの実行ができなくなるような事態が生じうる。

もう1つ注意すべき点は，同じ名前をもつ物理的なファイルは1つの登録番号につき1つしか存在しえないので，同じ名前のファイルに対する！ASSINGーカードをもつjobが，同じ登録番号のもとで，同時に実行されると，そのうちの一方が割りつけに失敗して放棄されることである。

なお，！ASSIGNーカードに関して必要ならば，「MELCOM UTS／VSバッチ処理システム使用の手引き」（NMーUG00ー03Aを参照されたい。

5　Options

●ptionsを！PASCALーカードの上にかくことはできない。●ptionはすべて，プログラムの中の注釈として，次の形でかく。

（＊¥□▲，□△，･･･，□△　任意の注釈＊）

---

＊";"前後に空白があってはならない。

ここで，□は下に示す文字のうちのいずれかで，△は"＋"または"－"である。たとえば，

（＊¥A－＊）　　（＊¥B＋，X－Comment＊）

□に入れる文字と，その意味は次のとおりである。

A：expressionの値の検査をする（＋）かしない（－）か。defaultはA＋。たとえば，

type char = '_' .. ';'　（＊ord（'_'）＝0，ord（';'）＝63＊）

であるから，chr（65）は，A＋のときerrorとされる。

B：目的プログラム（ROM）をM:GOからかく（－）か，M:B●からかく（＋）か。default は B－。目的プログラムを保存したいときにはB＋を指定し，制御カードを次のようにする。

```
! J●B ···
! LIMIT␣(CO, 30)
! ASSIGN␣M:BO, (FILE, NAME)
! PASCAL
    :
{ ! LOAD␣(EF, (NAME)), (UNSAT, (PASCLIB)) }
```

ただし，通常は load module の形でとっておく方がよい。

C：object listをとる（＋）かとらない（－）か，defaultはC－。

E：原始プログラム・リストのあと，診断メッセージを印刷する（＋）か否（－）か。default は＋。

L：原始プログラム・リストをとる（＋）か，とらない（－）か，defaultはL－。

O：stack overflowの検査をする（＋）か否（－）か。defaultはO＋，O－にすることはすすめられない。

T：複合option．T＋が指定されると，A＋，D＋，O＋，R＋，X＋が指定されたのと同じ効果をもち，T－が指定されると，A－，D－，O－，R－，X－が指定されたのと同じ効果をもつ。

X：arrayの indexおよび，case expressionの値を検査する（＋）かしない（－）か。defaultはX＋，たとえば，

1)　arr：array［0..1］of integer；b：integer；b：＝2；arr［b］:＝···

2)　case 3 of
　　1：···；4：···
　　end

のような場合は，X＋が指定されているとerrorとされる。

P：post mortem dumpをおこなう（＋）かおこなわない（－）か，defaultはP－。ただ

し，P＋を指定しても，post motem dumpが正しくimplementされていないため，特に効果はない。

D，R：効果はない。

• Standard Pascalとはoptionsの内容が異なるので注意されたい。

付　録

## 診断メッセージ

1. scalar type expected
2. integer too large
3. error in constant
4. = expected
5. field name declared twice
6. bad range
7. tagfield type bad
8. name declared twice
9. ) expected
10. : expected
11. identifier expected
12. identifier not declared
13. index must be of scalar type
14. of expected
15. variable type should be a type identifier
16. procedure declared twice
17. end expected
18. error in type declaration
19. error in variable declaration
20. , or) expected in value list
21. error in procedure declaration
22. value should be used at main level
23. parameter list ignored
24. error in declaration part
25. lowbound > highbound
26. not a variable identifier
27. value parameters should appear in their order of declaration
28. symbolic subrange type not allowed
29. parameters missing in function declaration
30. component type is file
31. undeclared identifier
32. variable or field identifier expected
33. expression too complicated - register stack overflow
34. type of variable should be array
35. type of expression must be scalar
36. conflict of index type with declaration
37. ] expected

38. type of variable should be record
39. no such field in this record
40. type of variable should be pointer or file
41. field name expected
42. illegal symbol in expression
43. undefined label
44. illegal type of parameter in standard function or standard procedure
45. type identifier in statement part
46. procedure used as function
47. type of standard function parameter should be integer
48. ) expected
49. identifier expected
50. illegal type of operand
51. or cannot be used as monadic operator
52. := expected
53. assignment not allowed
54. illegal symbol in statement
55. type or constant identifier
56. then expected
57. type of expression is not Boolean
58. ; expected
59. do expected
60. illegal parameter substitution
61. label expected
62. illegal type of expression
63. constant expected
64. : expected
65. of expected
66. tagfield missing for this variant
67. until expected
68. end expected
69. loop control variable must be simple and local or global
70. to/downto expected
71. too many cases in case statement
72. number of parameters does not agree with declaration
73. mixed types
74. too many labels in this procedure
75. too many (long) constants or yet undefined labels in this procedure
76. depth of procedure nesting too large

77. label defined more than once
78. too many exit labels
79. ( expected
80. , expected
81. assignment to formal function identifier illegal
82. too many nested with statements
83. standard inline procedure/function used as actual parameter
84. too many (long) constants in this procedure
85. assignment to function identifier must occur in function itself
86. actual parameter must be a variable
87. packed field not allowed here
88. operators <and> are not defined for powersets
89. redundant operation on powersets
90. procedure too long
91. too many exit label or forward procedures
92. too many class or file variables
93. bad function type
94. =, <> not allowed here
95. bad file declaration
96. type declared twice
97. end. encountered
98. [ expected
99. index out of range
100. label too large
101. value is out of range
102. division by zero
103. parameter procedure has more than 17 parameters
104. packed set of more than 32 elements forbidden
105. body of external procedure must be 'PASCAL' or 'metasymbol'
106. external definition must be global
108. object is externally defined and referenced
120. illegal character
121. forward reference not allowed in record definition
122. file in separated program must be external
123. 'reset' and 'rewrite' don't apply to 'input' and 'output'
124. text file expected
125. pointer must be bound to a record
126. no case provided for this value
127. file not allowed in pointer declaration

128. alfa constant not allowed here
200. value parameter expected