JAERI－M
4762

GPL—GENKEN Programming Language

1972年3月

浅井　清・富山峯秀

日本原子力研究所
Japan Atomic Energy Research Institute

GPL-GENKEN Programming Language

日本原子力研究所東海研究所原子炉工学部計算センター
浅井 清・富山峯秀

(1972年2月10日受理)

ソフトウエア記述言語GPLの機能について報告する。GPLは順位関数をもつ順位文法から導出される言語である。GPLコンパイラの第1版はおよそ8000枚のFORTRAN IV語で書かれており、このコンパイラはFACOM230-60計算機の相対形式オブジェクト・プログラムを作り出す。コンパイラはGPL語のソース・カードを1分間に1000~2000枚処理する。このコンパイラは他機種にも適用できるように設計されている。

GPL-Genken Programming Language

Kiyoshi ASAI and Mineyoshi TOMIYAMA
Div. of Reactor Engineering, Tokai, JAERI

(Received 10 February 1972)

The features of a system writing language named GPL (Genken Programming Language) are described. The GPL is derived from a precedence grammar with precedence functions. The first version of the compiler is written in FORTRAN IV language and consists of about 8000 statements. It produces object programs in relocatble binary form on the FACOM230-60 computer. It also compiles about 1000 to 2000 GPL source cards per minute, designed for easy modification on other computer systems.

目　次

1. はじめに …… 1
2. GPLの概要 …… 1
3. GPLで使用できる文字 …… 6
4. 予約語 …… 6
5. 定数 …… 8
6. 名まえ，変数，文，レジスタ …… 11
7. 演算 …… 12
8. 式 …… 13
9. 代入文 …… 13
10. 空文（null 文） …… 15
11. 手続き参照文（procedure 文） …… 15
12. 場合文（case文） …… 18
13. goto 文 …… 20
14. if 文 …… 20
15. while 文 …… 21
16. for文 …… 21
17. セル宣言 …… 22
18. 同義語宣言 …… 24
19. 手続き宣言 …… 25
20. 関数宣言 …… 27
21. 領域宣言 …… 29
22. ブロック …… 30
23. ブロック・レベルとネスト …… 31
24. 謝辞 …… 32
25. 付録 …… 34

（GPLの文法と言語，解析方法）

1. はじめに

いままで日本原子力研究所ではアセンブラ語，FORTRAN 語などを使つてソフトウエアの開発がおこなわれてきたが，ユーザの立場からすると，これらの言語はソフトウエア記述言語としては不適当である。FORTRAN 語では計算機システムのもつハードウエア，ソフトウエアの機能を十分に生かすことができず，アセンブラ語ではコーディングと虫取り作業に時間をとられるために，プログラムが完成して使用可能となつたときに新しい計算機が導入されることになる。そのうえアセンブラ語では計算機間の互換性もない。最近開発されているソフトウエアは，その規模が非常に大きくなつているので，そのソフトウエアのどの程度の部分が機種の変更に耐えて生き残れるかが大きな問題となつてきたが，これはよいソフトウエア記述言語の開発によつて解決することができる。そのようなソフトウエア記述言語の１つとしてN. Wirth によつて提案されたSystem 360 のための言語PL 360 1) は，その構成が単純簡潔でコンパイラの作成と維持が容易であるという点でわれわれの要求に合つている。ただしPL360 はSystem 360 のために作られたものであるから他の計算機でこの言語を使用するためには，その大部分の仕様を変更しなければならない。またユーザが開発するソフトウエア記述言語は，メーカから供給されるオペレイティング・システムのもとで動くことが望ましいから，問題向き言語（例えばFORTRAN 語）との実行時における結合可能性なども問題となつてくる。GPLはPL360 の考えを基本としこれらの点を考慮して作られたソフトウエア記述言語である。GPL はFORTRAN，ALGOL， アセンブラ語の機能を取り入れており，また機能の変更にもある程度は耐え得るように設計されているので，これらのどれか１つの言語の立場からみると，GPLのもつ機能は不徹底の感はまぬがれない。しかしGPLは，これらのどの言語よりも使いやすく，そのうえハードウエアを意識してコーディングするとアセンブラ語に近いオブジエクトを作ることができる。

2. GPLの概要

2.1 GPLの文法と言語

GPLは順位関数をもつ順位文法2),3)，によつて定義されており，言語の解析は順位関数をつかつておこなう。順位関数が存在すれば，N×Nの順位関係の行列（Nは文法要素の数）は2N個の関数値に縮約され，プログラムの速い解析を実現することができる。一般には順位関数は存在しないが，任意の順位文法についてこれと同値な順位文法で順位関数をもつものが存在する。GPLはこの事実を利用している。GPLの文法と順位関数，およびその解析法については巻末の付録に掲げる。ここでプログラムの例をあげておこう。

1. はじめに

いままで日本原子力研究所ではアセンブラ語，FORTRAN 語などを使つてソフトウエアの開発がおこなわれてきたが，ユーザの立場からすると，これらの言語はソフトウエア記述言語としては不適当である。FORTRAN 語では計算機システムのもつハードウエア，ソフトウエアの機能を十分に生かすことができず，アセンブラ語ではコーディングと虫取り作業に時間をとられるために，プログラムが完成して使用可能となつたときに新しい計算機が導入されることになる。そのうえアセンブラ語では計算機間の互換性もない。最近開発されているソフトウエアは，その規模が非常に大きくなつているので，そのソフトウエアのどの程度の部分が機種の変更に耐えて生き残れるかが大きな問題となつてきたが，これはよいソフトウエア記述言語の開発によつて解決することができる。そのようなソフトウエア記述言語の1つとしてN. Wirth によつて提案されたSystem 360 のための言語PL360 1) は，その構成が単純簡潔でコンパイラの作成と維持が容易であるという点でわれわれの要求に合つている。ただしPL360 はSystem 360 のために作られたものであるから他の計算機でこの言語を使用するためには，その大部分の仕様を変更しなければならない。またユーザが開発するソフトウエア記述言語は，メーカから供給されるオペレイティング・システムのもとで動くことが望ましいから，問題向き言語（例えばFORTRAN 語）との実行時における結合可能性なども問題となつてくる。GPLはPL360 の考えを基本としこれらの点を考慮して作られたソフトウエア記述言語である。GPL はFORTRAN，ALGOL， アセンブラ語の機能を取り入れており，また機能の変更にもある程度は耐え得るように設計されているので，これらのどれか1つの言語の立場からみると，GPLのもつ機能は不徹底の感はまぬがれない。しかしGPLは，これらのどの言語よりも使いやすく，そのうえハードウエアを意識してコーディングするとアセンブラ語に近いオブジエクトを作ることができる。

2. GPLの概要

2.1 GPLの文法と言語

GPLは順位関数をもつ順位文法2)，3)，によつて定義されており，言語の解析は順位関数をつかつておこなう。順位関数が存在すれば，N×Nの順位関係の行列（Nは文法要素の数）は2N個の関数値に縮約され，プログラムの速い解析を実現することができる。一般には順位関数は存在しないが，任意の順位文法についてこれと同値な順位文法で順位関数をもつものが存在する。GPLはこの事実を利用している。GPLの文法と順位関数，およびその解析法については巻末の付録に掲げる。ここでプログラムの例をあげておこう。

```
OPT    0    1    0    1    0    0
.BEGIN
 PROCEDURE TSORT3(M*N)  *
  COMMENT  ALGORITHM 245, TREESORT3 BY ROBERT W. FLOYD.
  THE COMM. OF THE ACM., DEC., 1964.  *
 BEGIN ARRAY 100 REAL M* INTEGER N*
  PROCEDURE EXCHANGE(X,Y)*
   BEGIN REAL X,Y,T*
   T=X* X=Y* Y=T*
  END *
  PROCEDURE SIFTUP(I,N)*
   BEGIN INTEGER I,N,J,K* REAL COPY*
    K=I* X1=K* COPY=M(X1)*
LOOP.. J=2*K* X1=K* X2=J*
   IF J.LE.N THEN
   BEGIN IF J.LT.N THEN
    BEGIN IF M(X2+1).GT.M(X2) THEN X2=X2+1* END*
    IF M(X2).GT.COPY THEN
    BEGIN M(X1)=M(X2)*  K=X2* GOTO LOOP* END*
   END*
   M(X1)=COPY*
  END *
  INTEGER I*
  I=N/2 *
L0.. SIFTUP(I,N)* I=I-1* IF I.GE.2 THEN GOTO L0 *
  I=N *
L1.. BEGIN SIFTUP(1,I)* X3=I* EXCHANGE(M(1),M(X3) )*
  END*
  I=I-1* IF I.GE.2 THEN GOTO L1 *
 END*
 END .
*
```

fig. 1　GPLプログラムの例

```
        FACOM 230-60    FORTRAN D  -710110- 0009-03               COMPILATION    72.01.17    PAGE   1

* SOURCE STATEMENT *
                          C
                          C      TEST PROGRAM FOR TREESORT3
                          C
                    1            DIMENSION M(10)
                    2            REAL M
                    3            READ(5,10) N,(M(I),I=1,N)
                    4         10 FORMAT(I2,7F10.4)
                    5            WRITE(6,20) N,(M(I),I=1,N)
                    6         20 FORMAT(1H1//10X,'THE INPUT DATA ... ' 2X,I5, 'ELEMENTS AS FOLLOW'
                               1 /' (2X'10F12.5))
                    7            CALL TSORT3(M,N)
                    8            WRITE(6,30) N,(M(I),I=1,N)
                    9         30 FORMAT(1H0,10X,'THE RESULTING ARRAY'/,(2X,I5,10F11.4))
                   10            STOP
                   11            END
```

fig. 3　TSORT3を呼び出すFORTANプログラム

```
      THE INPUT DATA ...        7ELEMENTS AS FOLLOW
   5.00000    3.00000    6.00000    1.00000    5.00000    4.00000    2.00000

       THE RESULTING ARRAY
   7    1.0000     2.0000     3.0000     4.0000     5.0000     5.0000     6.0000
```

fig. 4　入力データと結果

```
FDICT
   100000000220  031000000001  100100100100  100100100100  100100100100  000000000222  021400000001  343342326331
   343363100100  100100100100  000000000220  000000000000

INGEN HAS PASSED
                               G
   000001    000001000000  000000000000  000000000001  000000000043  000000777777  000000000001    024040404040
                                                       FL            FST           FL
   000007    000000000002  000001000006  000000000015  010510000000  023410000013  010510000000    400203404140
             FST           FL            FST           LX            AXI           JUMP
   000015    023410000000  010510000013  023410000000  012710000012  406700000002  400017000000    404140414040
             LA            SIAD          STAD          AXI           LA            STAD
   000023    010317000000  044110000026  044110000023  406700000001  010317000000  044110000030    404141404041
             STAD          JUMP          STX           JUMP          LA            STA
   000031    044110000025  400010000023  045710000012  400010000034  010310000000  023110000017    414141414041
             LX            FL            FST           LA            LR            MA
   000037    012210000017  010512000000  023410000020  010310000005  010410000000  007010000017    414041414141
             LARA          SIA           LX            LX            LA            TH
   000045    004300000043  023110000016  012210000017  012310000016  010310000016  070010000000    404141414140
             JUMP          LA            THE           JUMP          FL            FS
   000053    400010000105  010310000016  070110000000  400010000075  010513000000  006113000000    414140414040
             JNAO          JZA           AXI           FL            FS            JNAO
   000061    400110000075  410010000075  406300000001  010513000000  006110000020  400110000105    414140404141
             JZA           FL            FST           STX           JUMP          FL
   000067    410010000105  010513000000  023412000000  045310000017  400010000053  010510000020    414040414141
             FST           LX            AXI           JUMP          LA            STAD
   000075    023412000000  012710000015  406700000002  400017000000  010317000000  044110000046    404140404041
             AXI           LA            STAD          STAD          JUMP          STX
   000103    406700000001  010317000000  044110000066  044110000063  400010000046  045710000015    404041414141
             JUMP          LA            RAHA          DA            STA           LAI
   000111    400010000112  010310000000  004200000043  007610000005  023110000021  402300000002    414040414140
             SXJ           NOP           NOP           LA            S             STA
   000117    025710000121  500000000021  500000000000  010310000021  011010000004  023110000021    414140414141
             LA            TL            JUMP          JUMP          LA            STA
   000125    010310000021  170110000005  400010000142  400010000127  010310000000  023110000021    414141414041
             LAI           SXJ           NOP           NOP           LX            STX
   000133    402300000002  025710000121  500000000004  500000000021  012410000021  045410000006    404141414141
             LXI           AX            STX           LAI           SXJ           NOP
   000141    404700000000  014710000006  045710000160  402300000002  025710000044  500000000000    404141404140
             NOP           LA            S             STA           LA            TL
   000147    500004000000  010310000021  011010000004  023110000021  010310000021  170110000005    404141414141
             JUMP          JUMP          LX            AXI           JUMP          LA
   000155    400010000170  400010000144  012710000010  406700000002  400017000000  010317000000    414141404040
             AND           S             STAD          STAD          STAD          STAD
   000163    010610000003  011010000004  044110000160  044110000152  044110000106  044110000102    414141414141
             STAD          SIAD          STAD          STAD          LA            STAD
   000171    044110000101  044110000075  044110000071  044110000051  010317000000  044110000157    414141414041
             LA            SIAD          AXI           LA            STAD          STAD
   000177    010317000000  044110000070  406700000001  010317000000  044110000142  044110000132    404140404141
             STAD          JUMP          STX           JUMP
   000205    044110000123  400010000123  045710000010  400010000173  000000000000  000014000006    414141410011
                                                                                     1
   000213    400000001704  304000010275  700000000000  010000006112  200000777766  000070000003    101000000020
```

fig. 2　翻訳されたプログラムTSORT3（FACOM230－60）

2.2 異種言語プログラムとの結合

GPLで書かれたプログラムをFORTRANやアセンブラのCALL 文を使つて呼び出すことができる。またGPLの手続き参照文によつてFORTRANやアセンブラ語で書かれたプログラムを呼び出すこともできる。いずれの場合にも命令の中断(interruptions,またはtraps)に対する措置はプログラマの責任においておこなわなければならない。ひとつのジョブの主プログラムがFORTRAN語で書かれているときには，中断に対する措置はFORTRANシステムの仕様にしたがい，主プログラムがGPLで書かれているときには，中断に対する措置はプログラマがおこなわなければならない。

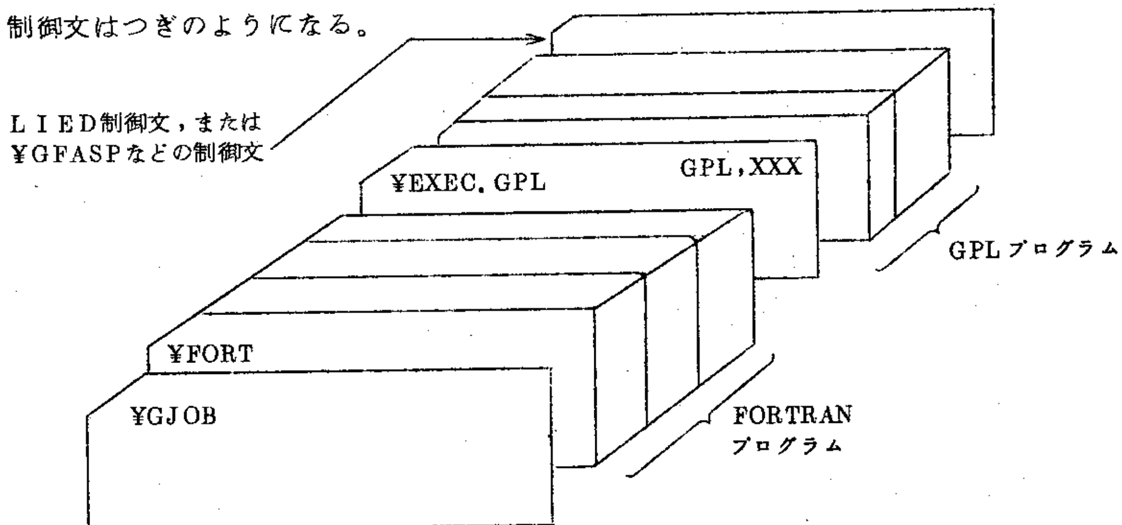
2.3 GPLコンパイラ

GPLコンパイラ(第1版)は命令部分の大きさが約25K語のFORTRANプログラムである。作業領域を含めて約65K語のGPLコンパイラは，約500枚のソース・カード(これをFORTRAN語で書くと約1000~1500枚のソース・カードとなる)を主記憶内(in core)で翻訳することができる。

作業領域をふやして98K語とすれば，約2000~25000枚のソース・カードを翻訳することができる。これらはデータを主記憶に常駐させるone pass方式でおこなわれる。このコンパイラは現在はFACOM230-60の相対形式プログラムを作り出すが，これ以外の機種についても大きな変更なく適用できるように工夫されている。

2.4 GPLの使用法

GPLコンパイラの出力は，FORTRANコンパイラの出力と同じくRELBIN なるFD名をもつファイルに相対形式で書かれる。したがつて現在原研の計算センタに使用されているマクロ化された制御文はGPLの出力に対しても有効である。GPLコンパイラはディスクに実行形式で保存されているので，FORTRAN語とGPLで書かれたプログラムが混在するジョブの制御文はつぎのようになる。



ここでXXXはGPLnn とし，ひとつのジョブのなかで1回以上GPLコンパイラを呼び出すときは，nn は00から順番に番号をつけてゆく。この例ではFORTRANプログラムが先になつているが，これはGPLプログラムが先に翻訳されてもよい。

2.5 翻訳(コンパイル)単位

GPLのコンパイラの一貫した処理の対象となるGPLのプログラムを翻訳単位とよぶ。GPLプログラムは, 文.(・はピリオド)の形式であるが,このプログラムがコンパイラの正常な処理の対象となるためには,この文は **begin** ではじまり, **end** で終らなければならない。また翻訳単位の前後はGPLの制御カード(第1行に *)が続くから,GPLの翻訳単位はつぎの形式となる。

```
┌ *GPL          OPTIONS                    ┐
│                                          │
│  ・ begin     ---------------  end.      │ 翻訳単位
└ *                                        ┘
```

ここでOPTIONSは指定しなくてもよいが,FASP と書くと翻訳されたアセンブラ語が出力される。NOLISTと書くとソース・リストは出力されない。

ERRn (n=1,2)と書くと入力されたソース・プログラムについてそれぞれレベル1と2のエラー・チエックをおこなう。

BLK・0 と指定するとブロック内で宣言された変数に対する番地の割付けがALGOL 流となる。

この翻訳単位(プログラム)が

・begin　　procedure　　name　　・・・

と手続き宣言ではじまるときは,この手続き名name はサブルーチンとして登録され,外部(この翻訳単位以外のプログラム)からの参照が可能となる。手続き宣言ではじまらないときは,この翻訳単位はメイン・プログラムとして登録される。そのエントリ名はGPMAINである。

2.6 翻訳の速さ

GPLコンパイラはFACOM230-60のFORTRAN語で書かれているが,このGPLコンパイラが入力を処理する速さは,1分間に1000~2000 枚である。付録にあげた例(Treesort 3)の処理には,コンパイラの初期設定の時間約0.3秒を除いて約1.5秒かかる。このうち約0.3秒は入力されたソース・プログラムを印刷するために使われる。(1行の印刷に約10 msec. が必要である。)1.5秒から逆算するとTreesort 3は1200 枚/分で処理されることになる。

3. ＧＰＬで使用できる文字

ＧＰＬではコード系は２６系を使用し次の文字が使用できる。

1. 英大文字

   Ａ，Ｂ，Ｃ，－－－Ｘ，Ｙ，Ｚ

2. 数字

   ０，１，－－－８，９

3. 特殊文字

   ␣ （空白を示す）

   ＝

   ＋

   －

   *

   ／

   （

   ）

   ，

   ．

   ¥

   ▼

4. 区切り記号（予約語）

ＧＰＬの文法で意味が固定されていて，他の意味に使用することを許さない語や記号を区切り記号（delimiters，またはreserveed words）という。
区切り記号には次のものがある。

ARRAY
ABS
ADCON
BEGIN
BASE
BIT
BYTE
CASE
CHARACTER
COMMENT

3. GPLで使用できる文字

GPLではコード系は26系を使用し次の文字が使用できる。

1. 英大文字

   A，B，C，－－－X，Y，Z

2. 数字

   0，1，－－－8，9

3. 特殊文字

   ␣（空白を示す）

   =

   +

   −

   *

   /

   (

)

   ,

   .

   ¥

   ▼

4. 区切り記号（予約語）

GPLの文法で意味が固定されていて，他の意味に使用することを許さない語や記号を区切り記号（delimiters，またはreserveed words）という。

区切り記号には次のものがある。

ARRAY

ABS

ADCON

BEGIN

BASE

BIT

BYTE

CASE

CHARACTER

COMMENT

DO

ELSE

END

EXIT

FOR

FUNCTION

GOTO, または GO TO

IF

INTEGER

LOCAL

LOGICAL

LONGREAL, または LONG REAL

NEG

NEGABS

NULL

OF

OVERFLOW

PROCEDURE

REAL

REGISTER

RETURN

SEGMENT

STEP

STOP

SYN

SHLA

SHRA

SHLL

SHRL

SHORTINTEGER, または SHORT INTEGER

THEN

UNTIL

WHILE

·LT·

·LE·

·GT·

·GE·

·EQ·

·NE·

OR

AND

NOT

XOR

+

−

*

/

++

−−

=

(

)

((

))

·

··

▽

▽▽

¥

5. 定　　数

GPLで使用できる定数には正負の符号のつかない，整定数，実定数，8進化定数，文字定数がある。

禁止事項

+，−の記号は実数を表示する場合に，指数部の正負の符号として用いられる場合以外はすべて演算子記号である。したがつて数値の正負を指定する符号として用いてはならない（第1版）。

5.1 整数

(1) 非負整定数

英字，特殊文字を除く数字の並びで10桁*1 以内の10進数である。

例

0

1234567890

・NE・

OR

AND

NOT

XOR

+

−

*

/

++

−−

=

(

)

((

))

・

‥

▽

▽▽

¥

5. 定　　数

GPLで使用できる定数には正負の符号のつかない，整定数，実定数，8進化定数，文字定数がある。

禁止事項

+，−の記号は実数を表示する場合に，指数部の正負の符号として用いられる場合以外はすべて演算子記号である。したがつて数値の正負を指定する符号として用いてはならない（第1版）。

5.1 整数

(1) 非負整定数

英字，特殊文字を除く数字の並びで10桁*1 以内の10進数である。

例

0

1234567890

誤りの例

−１２３　（符号付定数は許さない。）

１２　３　（空白*2 を途中に入れてはならない。）

なお負整定数については注意＊３を参照のこと。

注意

＊１　計算機により異なる。FACOM２３０−６０では-2^{35}以上$2^{35}-1$以下の数である。

＊２　空白は区切記号であるため特に認めるもの以外は，数字又は文字の列はそこで終つたものとみなす。

5.2　実定数

(1)　非負実定数

基本非負実定数は非負整数部と小数点，および小数部をこの順に書いたものである。非負整数部と小数部は両方とも数字の列であらわされ，これらの列の一方は空であつてもよいが，両方とも空であつてはならない。小数点は・であらわす。指数部は文字Eとそれに続く正，負の符号又は空白と非負整定数とで書かれ，定数は０〜７６*4までの数である。空白は正の符号とみなす。

非負実定数は，基本非負実定数か基本非負実定数の後に指数部を書いたものか，または非負整定数の後に指数部を書いたものとする。

例

n，m，E$\overset{\sqcup}{\pm}$Sをそれぞれ整数部，小数部，指数部とすると

n・m　　n・　　・m

nE＋S　（nE␣Sでもよい。）

n・mE＋S

n・mE−S

誤りの例

＋n・m　（符号付実定数は認めない。）

n・m　E−S（この位置で空白は許されない。）

なお負実定数については下の注意を参照のこと。

注意

＊３　負の整数，または実数を使用する場合は，単項演算子（7.演算を参照）を用いるか，あるいは負整定数の場合

I＝０−非負整定数

負実定数の場合

A＝０・−非負実定数

の演算結果としてもとめる（第１版）。

＊４　計算機により異る。FACOM２３０−６０では実定数は符号を含めて数値部２７ビット，指数部９ビットから構成されている。したがつて仮数部は１０進数にして約7.8桁，指数部は約±７６の範囲である。

5.3 8進化定数

0から7までの数字の列の後に$\overline{O}$をつけて表現した12桁*5 の数

例

　2 3 4 5 6 7 1 $\overline{O}$

　　　7 7 7 7 $\overline{O}$

誤りの例

　1 2 3␣3 4 5 $\overline{O}$　（数字の並びの中の空白は許されない。）

　−7 7 7 7 7 7 $\overline{O}$　（符号付定数は許されない。）

　$\overline{O}$ 7 7 7 7 0 0 0　（$\overline{O}$は数字の並びの最後に書く。）

　　7 8 9 $\overline{O}$　（8，9は許されない。）

＊5　計算機により異なる。FACOM230−60では12桁までの数である。

5.4 文字定数

▼▽と▽▼の間に書かれた文字は文字定数とみなす。文字定数は4文字までで空白も文字とみなす。またこの中に書かれた¥記号には特別な意味を持たせない。文字定数として▼▼は認めない。

例

　▼▽ABCD▽▼

　▼▽A␣▽$▽▼

誤りの例｛ ▽　　▼　（▼▼で囲まなければならない。）
　　　　　▼▽ ▼▽ ▽▼　（▼▼は文字定数として許されない。）

6. 名　　前

変数，名札，配列，手続，関数にあたえられる２０文字までの英数字の並びで，頭文字が英字のもの，この並びの中に空白を入れてはならない。外部で参照される共通領域名，手続き名などで結合プログラム（linkage editor）の処理の対象となる名前は６文字以内でなければならない。

例

GENKEN

U235

MATRIX

誤りの例

GEN KEN　　（空白を入れてはならない。

2H2O　　（頭文字は英字でなければならない。）

6.1 変数

数を保存する計算機の主記憶の番地に名まえを与え，その名まえを変数という。定数も変数のの１種であるが，特にことわらない限りは，変数というときには定数を含めない。主記憶の１語はセル（cell）とも呼ばれ，変数を特にセル変数とよぶことがある。ひとつのブロックのなかで宣言された変数は，その型と名まえが，そのブロックにおいては一意的に定められている。したがつて，同一のブロックでは同じ名まえをもつ変数を２度宣言することはできない。

変数は添字をもつことができるが，そのときは X_i をインデックス・レジスタ名，nを正整数として $X_i \pm n$ かnの形式の添字のみが許される。配列は１次元の配列のみが許される。したがつて，Aを配列名とすると A(Xi±n)，A(n) の形式の変数が許される。

6.2 文

GPLにおいて文は実行時の動作を記述するためにある。文の区切りは通常は¥記号によるが，else とまえの文の間に¥記号をおいてはならない。

例　　a = b + c　¥

　　if　a・EQ. d　then　a = a + d　else　a=a−d¥

6.3 レジスタ

FACOM230−60用GPLでプログラマが指定できるレジスタは，インデックス・レジスタX０，…，X６とベイス・レジスタB０のみである。インデックス・レジスタX０〜X６はハードウエアのインデックス・レジスタX１〜X７に対応する。これらのレジスタを使つて乗算，除算は許されない。正負の符号のない加減算のみが許される。B０のうえでは加減乗除算ともに認められない。

7. 演　　算

レジスタ，またはセルに対する演算にはつぎの種類がある。

7.1 算術演算

加算(+)，減算(-)，乗算(*)，除算(/)，非正規化加算（unnomalized）（++），非正規化減算（--）。算術演算はレジスタに対して，この演算記号の右側にある変数（定数を含む）が作用する形式でおこなわれる。したがつて結果はつねにレジスタに残つている。演算は左から右へとおこなわれるから，演算の結果，すなわち式の値はFORTRANなどの式と異なることに注意しなければならない。

例　　1 + 3 * 9 + 5

これは定数の演算であるが，計算の過程を図で示すとつぎのようになる：

演算レジスタ　Ro	式	意味
1	1	Ro ⇐ 1
4	+ 3	Ro ⇐ Ro + 3
36	* 9	Ro ⇐ Ro * 9
41	+ 5	Ro ⇐ Ro + 5

したがつて代入文A = 1 + 3 * 9 + 5でAに保存される値は33ではなく41である。

7.2 論理演算

論理演算には論理積（ビット単位のand），論理和（ビット単位の or），排他論理和（ビット単位のxor）がある。演算の方法，順序などは算術演算と同じである。

例　　A　and　B　or　C

7.3 シフト演算

シフト演算には左論理シフト（shla），右論理シフト（shra），左倍長算術シフト（snll），右倍長算術シフト（shrl）がある。単精度シフトでは論理シフト演算となるから注意しなければならない。（FACOM230-60 ではA - overfeow がおきない。）またシフト演算子の右にくるのは非負の整数でなければならない。

例　　K　shla　8

7.4 単項演算

単項演算には絶対値をとる演算（abs），逆符号数をとる演算（neg），逆符号数の絶対値をとる演算（negabs）がある。これらの演算はひとつの変数（定数を含む）に対しておこなわれるので単項演算の名がある。これらの演算は1つの式のなかでは，ただ1回だけ，しかも式のはじめにのみ使用することができる。

例　　abs　A

注意

(1) いずれの演算においても演算子の左右の変数（定数を含む）の型が異つていてはならない。

(2) 変数はレジスタ名であつてはならない。

8. 式

つぎにのべるセル式とインデックス・レジスタ式とを合せて式という。式のなかでは型の異る変数（定数を含む）の混合演算は許されない。

8.1 セル式

上の(1)～(4)の組合せで得られる演算のつながりをセル式という。(4)の演算は式の先頭でのみ定義することができる。セル式では，式中に出現するのはセル変数（定数を含む）である。

例

```
abs  FIND  and   MASK + 3
K  shla 18  or   ADDRESS
```

8.2 レジスタ式

X_i をインデックス・レジスタとするとき，

$X_i \pm n$（nは非負の整数）

の演算形式をインデックス・レジスタ式という。インデックス・レジスタが出現する式はつねにこの形式でなければならない。

9. 代 入 文

代入文にはつぎのようなものがある。

9.1 セル代入文

これは等号の右辺の演算結果を左辺の変数番地に格納する操作を示す文である。

形式

変数＝式

禁止事項

(1) 変数はレジスタであつてはならない。

(2) 変数は同義語（SYNONYM） 宣言でレジスタと結びつけられていてはならない。

注意

(1) いずれの演算においても演算子の左右の変数（定数を含む）の型が異つていてはならない。

(2) 変数はレジスタ名であつてはならない。

8. 式

つぎにのべるセル式とインデックス・レジスタ式とを合せて式という。式のなかでは型の異る変数（定数を含む）の混合演算は許されない。

8.1 セル式

上の(1)〜(4)の組合せで得られる演算のつながりをセル式という。(4)の演算は式の先頭でのみ定義することができる。セル式では，式中に出現するのはセル変数（定数を含む）である。

例

```
abs  FIND  and  MASK + 3
K  shla 18  or  ADDRESS
```

8.2 レジスタ式

X_i をインデックス・レジスタとするとき，

$X_i \pm n$ （nは非負の整数）

の演算形式をインデックス・レジスタ式という。インデックス・レジスタが出現する式はつねにこの形式でなければならない。

9. 代入文

代入文にはつぎのようなものがある。

9.1 セル代入文

これは等号の右辺の演算結果を左辺の変数番地に格納する操作を示す文である。

形式

変数＝式

禁止事項

(1) 変数はレジスタであつてはならない。

(2) 変数は同義語（SYNONYM） 宣言でレジスタと結びつけられていてはならない。

9.2 レジスタ代入文

これは符号の右辺の演算結果を符号の左辺のレジスタへ格納する操作を示す文である。

形式

レジスタ名＝式

例

X1 ＝ I ＋ J ＊ K

禁止事項

左辺がベイス・レジスタB_iであるときには，右辺の式はセルでなければならない。

9.3 多重代入文

これは等号の右辺の変数（定数を含む）の値を左辺のいくつかの変数に移す操作を示す文である。

形式

((変数，－－－，変数)) ＝変数

例

((A（X0 ＋1），B（X0），－－－，C))＝D（X3）

このとき等号の左辺と右辺の変数（定数を含む）の型は一致していなくてはならない。

禁止事項

等号の右辺にレジスタ名を書いてはならない（第1版）。等号の左辺にレジスタ名を書いてはならない。

9.4 ベクトル代入文

これは等号の右辺の変数の並びの値を左辺の変数を先頭とする連続した番地へ移す操作を示す文である。

形式

変数＝((変数，－－－，変数))

このとき等号の左辺と右辺の型は一致していなくてはならない。

例

P（X0）＝((A(X0)，B，C(X1)))

この例の文によつて引きおこされる操作は，つぎの文の列の操作と同値である：

P（X0）＝A（X0） ￥ P（X0 ＋1）＝B￥ P（X0＋2）＝C（X1）。

左辺の変数が配列でないときは，型宣言でとられた連続する番地に右辺の値が移される。

例

real P1，P2，P3，P4 ￥

P1 ＝((A（X0），B，C（X1）))

は

P1 = A (X 0)　¥　P2 = B　¥　P3 = C (X1)

と同じ操作を意味する。

禁止事項

等号の左辺，右辺の変数はレジスタ名，名札（レイベル）であつてはならない。

10. 空(null) 文

1つの空文に対して1つの no operation 命令が作り出される。したがつて空文は no operation の命令を作り出す働きをもつ。

形式

```
null
```

例

```
goto      JUMP ¥
     .
     .
     .
JUMP.. null  ¥
     A = B ¥
```

この例でプログラムの論理的な流れは

JUMP .. A = B ¥

と同じであるが，null 文によつて no operation 命令が作られている点が異る。

11. 手続き参照(procedure) 文

手続き(procedure)文は，この文で指定された手続き(proedure) の本体(procedure body)を起動する働きをもつ。本体の実行後プログラムの制御は，このprocedure 文のつぎの文に移る。ただし，起動された本体が，goto 文，function文，あるいは他の手続き(procedure)文を実行することでプログラムの制御を放棄するときはこの限りではない。

形式

```
手続き名
手続き名（実パラメータ，－－－，実パラメータ）
```

手続き名(procedure name）は，それが手続き参照文によつて呼び出されるときは，他の場所で手続き宣言(procedure declaration) によつて定義されていなければならない。手続き名が参照されるときは，参照するプログラムと参照される手続きとの間の情報の受け渡

は

P1 = A (X 0)　¥　P2 = B　¥　P3 = C (X1)

と同じ操作を意味する。

禁止事項

等号の左辺，右辺の変数はレジスタ名，名札（レイベル）であつてはならない。

10. 空(null) 文

1つの空文に対して1つのno operation 命令が作り出される。したがつて空文はno operationの命令を作り出す働きをもつ。

形式

```
null
```

例

```
    goto        JUMP ¥
                  .
                  .
                  .
JUMP.. null  ¥
        A = B ¥
```

この例でプログラムの論理的な流れは

JUMP.. A = B ¥

と同じであるが，null 文によつてno operation 命令が作られている点が異る。

11. 手続き参照(procedure) 文

手続き(procedure)文は，この文で指定された手続き(proedure) の本体(procedure body)を起動する働きをもつ。本体の実行後プログラムの制御は，このprocedure 文のつぎの文に移る。ただし，起動された本体が，goto 文，function文，あるいは他の手続き(procedure)文を実行することでプログラムの制御を放棄するときはこの限りではない。

形式

```
手続き名
手続き名（実パラメータ，－－－，実パラメータ）
```

手続き名(procedure name) は，それが手続き参照文によつて呼び出されるときは，他の場所で手続き宣言(procedure declaration) によつて定義されていなければならない。手続き名が参照されるときは，参照するプログラムと参照される手続きとの間の情報の受け渡

は

P1 = A (X 0)　¥　P2 = B　¥　P3 = C (X1)

と同じ操作を意味する。

禁止事項

等号の左辺，右辺の変数はレジスタ名，名札（レイベル）であつてはならない。

10. 空(null) 文

1つの空文に対して1つのno operation 命令が作り出される。したがつて空文はno operationの命令を作り出す働きをもつ。

形式

```
null
```

例

```
goto      JUMP ¥
  .
  .
  .
JUMP.. null ¥
       A = B ¥
```

この例でプログラムの論理的な流れは

JUMP .. A = B ¥

と同じであるが，null 文によつてno operation 命令が作られている点が異る。

11. 手続き参照(procedure) 文

手続き(procedure)文は，この文で指定された手続き(proedure) の本体(procedure body)を起動する働きをもつ。本体の実行後プログラムの制御は，このprocedure 文のつぎの文に移る。ただし，起動された本体が，goto 文，function文，あるいは他の手続き(procedure)文を実行することでプログラムの制御を放棄するときはこの限りではない。

形式

```
手続き名
手続き名(実パラメータ，－－－，実パラメータ)
```

手続き名(procedure name)は，それが手続き参照文によつて呼び出されるときは，他の場所で手続き宣言(procedure declaration) によつて定義されていなければならない。手続き名が参照されるときは，参照するプログラムと参照される手続きとの間の情報の受け渡

しは，つぎのいずれかの方法による：

(1) パラメータの受け渡し（例1）。

(2) 領域定義文（segment文）によつて定義された領域の共有（例2）。

(3) 参照される手続きを含むブロックの変数を共用する （参照される手続きが，この手続きを参照するプログラムに含まれているとき。）（例3）。

(4) レジスタの共用

(5) モニタの特定領域，ハードウエアの特定の機能（PSW，program status word など），システム・ルーチンなどの利用。

上記の方法で翻訳，実行の効率ともにすぐれているものからならべると，(4)，(3)，(2)，(1)，(5)の順となる。レジスタに多くの情報をのせることはできないから，実用的な見地からいえば，(3)，(2)，(1)の方法が一般的といえる。プログラム単位（翻訳単位）を異にする手続きとの情報の受け渡しは(2)の方法によるのが理想的である。

注意

(1) FACOM230−60GPLではFACOM230−60FORTRANとの結合をはかるために，手続きを参照する直前にA−レジスタにパラメータ数をのせる。したがつてA−レジスタを利用して情報の授受はできない。

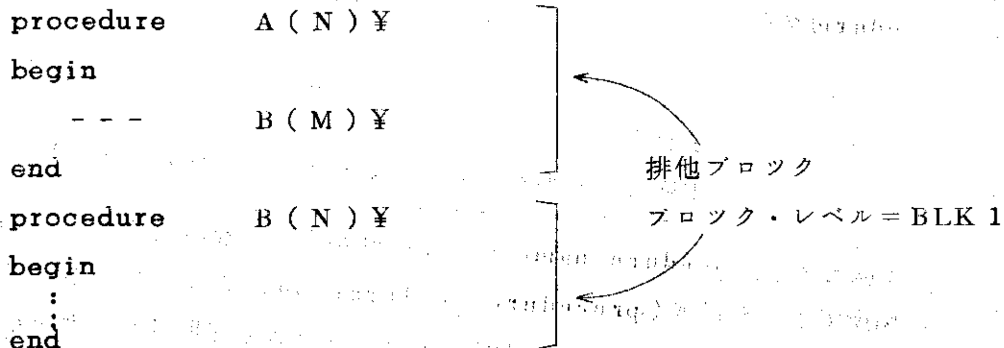
(2) パラメータのベイス・レジスタは，参照された手続きのなかで指定された（仮パラメータの）ベイス・レジスタが使用される。

(3) FACOM230−60用GPLの手続き参照文proc(A_1，---，A_n）の翻訳された命令の列はつぎのようになつている：

```
LAI     n
SXJ,7   PROC
NOP     A1
  :
NOP     An
```

(4) GPLの手続き宣言で定義された手続きが参照されたときは，その手続きはパラメータ数のチエックをおこなわない。

(5) 下図のように互いに排他的なブロックをなす手続き宣言によつて定義された手続きAは，Bを参照することができる。ただし，このときこれらの宣言を含む翻訳単位のブロック・レベルは制御カード＊GPLでBLK0 と指定されていてはならない。



(6) 手続き参照によつて現在実行中のプログラムから参照された手続きに制御が移されるときには，インデックス・レジスタX 6が実行中のプログラムの命令番地を保存する。

禁止事項

(1) パラメータに手続き名を指定してはならない。

(2) パラメータにレジスタ名を指定してはならない。

(3) パラメータに領域（セグメント）名を指定してはならない。

(4) パラメータに変数以外の式を指定してはならない。

(5) パラメータに名札（レイベル）を指定してはならない（第1版）。

(6) パラメータに4文字を超える文字定数を指定してはならない。

(7) 再帰的な手続き参照は許されない（例4）。

例1.
```
procedure    exchange ( X , Y ) ¥
begin  real   X , Y , T ¥
  T = X ¥   X = Y ¥ Y = T ¥
end ¥
exchange( M(1) , M ( X 3 ) ) ¥
```

例2.
```
  procedure exchange  ¥
begin segment base   //  ¥
begin real  X , Y , T ¥  end  ¥
  T = ¥   X = Y ¥   Y = T ¥
end  ¥
```

この手続きexchangeと翻訳単位を異にするプログラムで
```
begin segment base   // ¥
  begin real  X , Y , T ¥  end
      :
exchange  ¥
```

となつていれば，領域名//（ブランク・コモン）でX，Y，Tを共有している。

例3.
```
begin real  X , Y , T ¥
procedure exchange  ¥
begin
  T = X ¥   X = Y ¥   Y = T ¥
end  ¥
```

となつているときは，exchangeを含むプログラム単位でX，Y，Tが定義されている。

例4.
```
procedure  A  ¥
  begin  - - -  B ( N ) ¥ - - - end ¥
procedure  B ( M ) ¥
```

begin - - - A ¥ - - - end ¥

このときは手続きAとBは互いに参照し合つており参照が再帰的となる。

12. case 文

(1) 場合(case)文は，begin と end でくくられたいくつかの文のうちから，インデックス・レジスタの内容に対応するひとつの文を選び出して実行する。選び出された文が実行されたあとのプログラムの制御は，endのあとに続く文に移る。

形式

```
  case  Xi     of    begin
 第1文  ¥
 第2文  ¥
   ⋮
 第n文  ¥            end
```

インデックス・レジスタXi の内容は整数1からnまでの範囲の数でなければならない。インデツンデックス・レジスタXi の内容が整数jのときは，第j番目の文が実行され，その後プログラムの制御は end のあとに続く文に移る。実行された文が goto 文を含んでいると，その goto 文によつて制御が他の文に移ることもある。

例

```
            X1 = 2 ¥
            case X1   of   begin
LABE1 ··     goto    ASTER ¥
LABE2 ··     begin   I=J*2+K¥  L=M¥
                goto LABEL1 ¥  end ¥
                            end ¥
```

この例では，X1の値が2であるから，まず文LABEL2 ·· begin - - - end が実行される。この文はLABEL1 への goto 文を実行し，制御はASTER なる名札を持つた文に移る。

(2) コンパイルの方法

case 文の翻訳された結果はつぎのようになつている。

```
        ⋮
    jump  A , Xi
 S1 ·· 第1文の命令群
    Jump  C
        ⋮
 Sn ·· 第n文の命令群
```

begin - - - A ¥ - - - end ¥

このときは手続きAとBは互いに参照し合つており参照が再帰的となる。

12. case 文

(1) 場合(case)文は，begin と end でくくられたいくつかの文のうちから，インデックス・レジスタの内容に対応するひとつの文を選び出して実行する。選び出された文が実行されたあとのプログラムの制御は，endのあとに続く文に移る。

形式

```
  case  Xi    of   begin
 第1文  ¥
 第2文  ¥
   ⋮
 第n文  ¥          end
```

インデックス・レジスタXi の内容は整数1からnまでの範囲の数でなければならない。インデックス・レジスタXi の内容が整数 j のときは，第 j 番目の文が実行され，その後プログラムの制御は end のあとに続く文に移る。実行された文が goto 文を含んでいると，その goto 文によつて制御が他の文に移ることもある。

例

```
            X1 = 2 ¥
            case X1  of  begin
LABE1 ··     goto   ASTER ¥
LABE2 ··     begin  I=J*2+K¥  L=M¥
               goto LABEL1¥  end ¥
                         end ¥
```

この例では，X1の値が2であるから，まず文LABEL2 ·· begin - - - end が実行される。この文はLABEL1 への goto 文を実行し，制御はASTER なる名札を持つた文に移る。

(2) コンパイルの方法

case 文の翻訳された結果はつぎのようになつている。

```
        ⋮
   jump  A , Xi
 S1 .. 第1文の命令群
   Jump  C
        ⋮
 Sn ·· 第n文の命令群
```

J ump C

A., J ump S_1

⋮

J ump S_n

C.,

禁止事項

(1) X i にインデックス・レジスタ以外のレジスタ名，変数名，名札などを指定してはならない。

13. goto 文

goto 文は，この文で指定された名札をもつ文にプログラムの制御を移す働きをもつ。

形式

```
goto    L
```

注意

(1) goto 文で指定された名札は，label宣言されていなくてもよいが，同一のプログラム（翻訳）単位のなかで定義されていなければならない。

禁止事項

(1) 名札以外の変数（定数を含む），式，または文を名札として使つてはならない。

14. if 文

if 文について述べるまえに，if 文を構成する重要な要素のひとつである「条件」について説明しよう。

(1) A，Bをそれぞれ変数（定数を含む），Rを集合{.EQ.，.NE.，.LT.，.LE.,.GE.，.GT.}の1つの要素とするとき，ARBを単純条件と呼ぶ。

(2) C1，C2が単純条件であるとき，C1 and C2とC1 or C2を複合条件と呼ぶ。

(3) C1が複合条件，C2が単純条件であるとき，C1 and C2とC1 or C2も複合条件と呼ぶ。

(4) 単純条件，複合条件を合せて条件と呼ぶ。

(5) 条件はつねに真か偽かのいずれかの値をとる。

(6) 複合条件はつねに左から右へと計算される。

if 文にはつぎの2つの形式がある：

```
if  条件  then  文                              (t1)
if  条件  then  無条件文  else  文               (t2)
```

形式（t1）のとき，条件が真の値をとればthenのつぎの文が実行される。条件が偽の値をとればthenのつぎの文は実行されず，プログラムの制御はこの文のつぎの文に移る。

形式（t2）のとき，条件が真の値をとればthenのつぎの無条件文が実行される。条件が偽の値をとればelseのつぎの文が実行される。いずれの場合にも該当する文の実行後のプログラムの制御は，このif 文のつぎの文に移る。

注意

(1) 複合条件

C_1 and C_2 and … C_{n-1} and C_n

13. goto 文

goto 文は，この文で指定された名札をもつ文にプログラムの制御を移す働きをもつ。

形式

```
goto    L
```

注意

(1) goto 文で指定された名札は，label宣言されていなくてもよいが，同一のプログラム（翻訳）単位のなかで定義されていなければならない。

禁止事項

(1) 名札以外の変数（定数を含む），式，または文を名札として使つてはならない。

14. if 文

if 文について述べるまえに，if 文を構成する重要な要素のひとつである「条件」について説明しよう。

(1) A，Bをそれぞれ変数（定数を含む），Rを集合{.EQ.，.NE.，.LT.，.LE.，.GE.，.GT.}の1つの要素とするとき，ARBを単純条件と呼ぶ。

(2) C1，C2が単純条件であるとき，C1 and C2とC1 or C2を複合条件と呼ぶ。

(3) C1が複合条件，C2が単純条件であるとき，C1 and C2とC1 or C2も複合条件と呼ぶ。

(4) 単純条件，複合条件を合せて条件と呼ぶ。

(5) 条件はつねに真か偽かのいずれかの値をとる。

(6) 複合条件はつねに左から右へと計算される。

if 文にはつぎの2つの形式がある：

if	条件	then	文			(t1)
if	条件	then	無条件文	else	文	(t2)

形式（t1）のとき，条件が真の値をとれば then のつぎの文が実行される。条件が偽の値をとれば then のつぎの文は実行されず，プログラムの制御はこの文のつぎの文に移る。

形式（t2）のとき，条件が真の値をとれば then のつぎの無条件文が実行される。条件が偽の値をとれば else のつぎの文が実行される。いずれの場合にも該当する文の実行後のプログラムの制御は，この if 文のつぎの文に移る。

注意

(1) 複合条件

C_1 and C_2 and … C_{n-1} and C_n

は，すべての単純条件Ci が真であるときに真となる。

複合条件

C_1 or C_2 or … C_{n-1} or C_n

は，左から右へと単純条件Ci を判定してゆき，どれかひとつの単純条件Ci が真であるときに真となる。

(2) 複合条件はつねに左から右へと判定されるから，

C_1 or C_2 and C_3 and C_4 or C_5

の計算の順序をカッコで示せば，

((((C_1 or C_2) and C_3) and C_4) or C_5)

となる。

例

```
if  MAX.GE.X3  then  X3=X3+1
if  MAX.GE.X3  or  MAX.GE.K  then  X3=X3+1
else  begin  MAX=MAX+1 ¥ X4=X4-1 ¥  end
```

(3) FACOM230－60 版GPLでは条件の値はどのようなレジスタにも保存されず，プログラムの流れとして表現されるから，ひとつの条件の値を他の場所において参照することはできない。

15. while 文

while 文はこの文の複合条件が真の値をとつている間は，do のあとに続く文を繰り返し実行する。while 文を使用すると，if 文と for文の組合せでは簡潔にかけない演算を簡単に記述することができる。

形式

while 複合条件 do 文

注意

(1) 複合条件の計算方法は if 文と同じである。

(2) while 文は条件が満たされている限りはdo のあとの文を繰り返して実行するから，この文中に条件の成立を変化させる要因が存在しなければならない。

例

```
while  X1.LT.5  do  begin  A(X1+1)=0¥  X1=X1+1  ¥  end
```

16. for 文

for 文では，for のあとに続くインデックス・レジスタのX_i の内容が，その初期値か

は，すべての単純条件Ci が真であるときに真となる。

複合条件

C_1 or C_2 or … C_{n-1} or C_n

は，左から右へと単純条件Ci を判定してゆき，どれかひとつの単純条件Ci が真であるときに真となる。

(2) 複合条件はつねに左から右へと判定されるから，

C_1 or C_2 and C_3 and C_4 or C_5

の計算の順序をカッコで示せば，

((((C_1 or C_2) and C_3) and C_4) or C_5)

となる。

例

```
if  MAX.GE.X3  then  X3=X3+1
if  MAX.GE.X3  or  MAX.GE.K  then  X3=X3+1
else  begin  MAX=MAX+1 ¥ X4=X4-1 ¥  end
```

(3) FACOM230-60 版GPLでは条件の値はどのようなレジスタにも保存されず，プログラムの流れとして表現されるから，ひとつの条件の値を他の場所において参照することはできない。

15. while 文

while 文はこの文の複合条件が真の値をとつている間は，do のあとに続く文を繰り返し実行する。while 文を使用すると，if 文と for文の組合せでは簡潔にかけない演算を簡単に記述することができる。

形式

while　複合条件　do　文

注意

(1) 複合条件の計算方法は if 文と同じである。

(2) while 文は条件が満たされている限りはdo のあとの文を繰り返して実行するから，この文中に条件の成立を変化させる要因が存在しなければならない。

例

```
while  X1.LT.5  do  begin  A(X1+1)=0¥  X1=X1+1 ¥  end
```

16. for 文

for 文では，for のあとに続くインデックス・レジスタのX_i の内容が，その初期値か

は，すべての単純条件Ci が真であるときに真となる。

複合条件

C_1 or C_2 or … C_{n-1} or C_n

は，左から右へと単純条件Ci を判定してゆき，どれかひとつの単純条件Ci が真であるときに真となる。

(2) 複合条件はつねに左から右へと判定されるから，

C_1 or C_2 and C_3 and C_4 or C_5

の計算の順序をカッコで示せば，

((((C_1 or C_2) and C_3) and C_4) or C_5)

となる。

例

```
if  MAX.GE.X3  then  X3=X3+1
if  MAX.GE.X3  or  MAX.GE.K  then  X3=X3+1
else  begin  MAX=MAX+1 ¥  X4=X4-1 ¥  end
```

(3) FACOM230-60 版GPLでは条件の値はどのようなレジスタにも保存されず，プログラムの流れとして表現されるから，ひとつの条件の値を他の場所において参照することはできない。

15. while 文

while 文はこの文の複合条件が真の値をとつている間は，do のあとに続く文を繰り返し実行する。while 文を使用すると，if 文と for文の組合せでは簡潔にかけない演算を簡単に記述することができる。

形式

while　複合条件　do　文

注意

(1) 複合条件の計算方法は if 文と同じである。

(2) while 文は条件が満たされている限りはdo のあとの文を繰り返して実行するから，この文中に条件の成立を変化させる要因が存在しなければならない。

例

```
while  X1.LT.5  do  begin  A(X1+1)=0¥  X1=X1+1 ¥  end
```

16. for 文

for 文では，for のあとに続くインデックス・レジスタのX_i の内容が，その初期値か

らstep に続く定められたきざみによつて増加し，until に続く上限に到達するまでdo に続く文が繰り返して実行される。Xi の内容が上限を越えると，この文は実行されず，プログラムの制御はこの文のあとに続く文に移される。

形式

```
for  Xi=式  step  正整数  until  上限  do  文
```

注意

(1) 上限には正整数，インデックス・レジスタ名，セル変数を指定することができる。

(2) Xi と上限との内容の比較は文の実行が始まる前におこなわれる。

(3) 式は正の整数値をとらなければならない。

禁止事項

(1) 上限に負の値をもつものを指定してはならない（第1版）。

(2) until に続くきざみに負の整数を指定してはならない。

(3) forのあとに続く変数にインデックス・レジスタ名以外の変数（定数を含む）を指定してはならない。

例

```
for X1=I+1  step  2  until  K   do
  begin  integer  K,L ¥
     K=M(X1) ¥ M(X1)=N(X2) ¥ L=X2 ¥
        PROC(L)¥
     end ¥
```

この例でfor文の上限を与える変数Kと，begin のあとに新しく定義された変数Kとは番地も内容も異る。（この点については 2.2ブロックを参照のこと。）

17. セル宣言(Cell Declarations)

主記憶に番地をもち，そこに数値化された内容を保存する変数をセル変数と呼び，これをレジスタ名，レイベル名，関数名，手続き名などと区別する。セル変数は，それが使用される以前に整数，浮動小数点数などの型と配列についての宣言が必要である。また，これらの変数に初期値を設定することもできる。

形式

$$\left\{\begin{array}{l}\text{integer}\\ \text{real}\end{array}\right\}\ \alpha_1,\ \cdots\ ,\alpha m \qquad (t1)$$

$$\text{array}\ \ n\left\{\begin{array}{l}\text{integer}\\ \text{real}\end{array}\right\}\ \alpha_1,\ \cdots\ ,\alpha m \qquad (t2)$$

らstep に続く定められたきざみによつて増加し，until に続く上限に到達するまでdo に続く文が繰り返して実行される。Xi の内容が上限を越えると，この文は実行されず，プログラムの制御はこの文のあとに続く文に移される。

形式

```
for  Xi=式  step  正整数  until  上限  do  文
```

注意

(1) 上限には正整数，インデックス・レジスタ名，セル変数を指定することができる。

(2) Xi と上限との内容の比較は文の実行が始まる前におこなわれる。

(3) 式は正の整数値をとらなければならない。

禁止事項

(1) 上限に負の値をもつものを指定してはならない（第1版）。

(2) until に続くきざみに負の整数を指定してはならない。

(3) forのあとに続く変数にインデックス・レジスタ名以外の変数（定数を含む）を指定してはならない。

例

```
for X1=I+1   step  2  until  K    do
  begin  integer  K,L ¥
      K=M(X1) ¥ M(X1)=N(X2) ¥ L=X2 ¥
          PROC(L)¥
      end  ¥
```

この例でfor文の上限を与える変数Kと，begin のあとに新しく定義された変数Kとは番地も内容も異る。（この点については 2.2 ブロックを参照のこと。）

17. セル宣言(Cell Declarations)

主記憶に番地をもち，そこに数値化された内容を保存する変数をセル変数と呼び，これをレジスタ名，レイベル名，関数名，手続き名などと区別する。セル変数は，それが使用される以前に整数，浮動小数点数などの型と配列についての宣言が必要である。また，これらの変数に初期値を設定することもできる。

形式

$$\left\{\begin{array}{l}\text{integer}\\ \text{real}\end{array}\right\}\ \alpha_1,\ \cdots\ ,\alpha_m \qquad (t1)$$

$$\text{array}\ \ n\left\{\begin{array}{l}\text{integer}\\ \text{real}\end{array}\right\}\ \alpha_1,\ \cdots\ ,\alpha_m \qquad (t2)$$

$$\left\{\begin{matrix}\text{integer}\\ \text{real}\end{matrix}\right\} \quad \alpha_1 = V_1 , \quad \cdots \quad , \alpha m = V_m \qquad (t3)$$

$$\text{array } n \left\{\begin{matrix}\text{integer}\\ \text{real}\end{matrix}\right\} \quad \alpha_1 = (V_1, \ \cdots , \ V_i), \cdots, \ \alpha_m = (V'_1, \cdots, V'_j) \qquad (t4)$$

ここで $\left\{\begin{matrix}A\\B\end{matrix}\right\}$ とあれば，AまたはBのいずれかを選択することを意味する。

（t1）の宣言では，変数 α_1, … , αm が整数（integer)，または浮動小数点（real）の型（type）をもつことを意味する。

（t2）の宣言では，変数 α_1, … ， αm が整数，または浮動小数点の型をもつ大きさnの1次元の配列として定義される。

（t3）の宣言では，整数，または浮動小数点数の型をもつ変数 α_1, … ， αm が，それぞれ初期値 V_1, … ， Vm で定義される。

（t4）の宣言では，整数，または浮動小数点数の型をもつ配列変数 α_1, … ， αm が，それぞれ初期値 $\alpha_1(1)=V_1$, $\alpha_1(2)=V_2$, … , $\alpha_1(i)=V_i$ ($i \leq n$) , … , $\alpha m(1)=V'_j$, … $\alpha m(j)=V'_j$, ($j \leq n$) で定義される。

注意

(1) （t3),（t4）の宣言で

$$\left\{\begin{matrix}\text{integer}\\ \text{real}\end{matrix}\right\} \quad \alpha_1 = V_1, \ \cdots \ , \alpha_i , \ \alpha_{i+1}, = V_{i+1}, \ \cdots \qquad (t3)$$

$$\text{array } n \left\{\begin{matrix}\text{integer}\\ \text{real}\end{matrix}\right\} \ \alpha_1 = (V_1, \cdots , V_i), \ \cdots \ , \alpha_i, \ \cdots \qquad (t4)$$

と初期値をもたない変数も同時に定義することができる。

(2) それぞれの形式は，それが1体となつて宣言されなければならない。たとえば（t2）で array n A ¥ integer A ¥ と分割して宣言すると誤りとなる。初期値の設定についても同じである。

(3) 初期値として文字定数，負の符号のない整数，正負の符号のない浮動小数点数（8進数は整数）を指定することができる。

(4) 左辺の型と一致しない型をもつ初期値を与えることができるが，その値は左辺の型に合うようには変換されない。演算は左辺の変数の型でおこなわれる。

(5) ブロックが多層構造になつているとき，なかのブロックで初期値の設定をおこなうと，コンパイラはこの操作をおこなうための命令群を作り出す。これに反して一番外側のブロックでの初期値の設定は，結合プログラム（linkage editor）によつてプログラムのローディング時におこなわれる。（ブロック構造を参照のこと。）

(6) 宣言の終りは¥記号がくる。

例

```
integer   I , J , K ¥
integer   N= 0  ¥
array 3 real F=( 1.0 , 2.0 , 3.0 ) ¥
integer   A=77$\overline{0}$ , B=76$\overline{0}$ , C=1  ¥
```

18. 同義語宣言 (Synonym Declarations)

同義語宣言は，この宣言にあらわれた変数名とこの宣言がおこなわれるまえに宣言されている変数の属性を結びつける働きをする。

形式

array n {real / integer} 変数名 syn {変数 / 定数} ,………,

………, 変数名 syn {変数 / 定数} (t1)

変数名を配列として宣言するなら array n が必要である。

{real / integer} {adcon / slcon} 変数名 syn 変数 , ………,

………, 変数名 syn 変数 (t2)

(t1) の宣言で変数名 syn 変数と宣言されたときは，変数名はこのとき変数がもつ番地を共有する。

(t1) の宣言で変数名 syn 定数と宣言されたときは，変数名の番地は定数で示された番地に同じである。すなわち，この定数は機械語の番地そのものを示している。したがつて，この定数はベイス・レジスタの番号とそのレジスタからの変位(displacement)を示している。定数がレジスタ番号を含んでいなければ，この変数名のベイス・レジスタはB0 (FACOM 230-60の場合)が与えられる。

(t2) の宣言では，変数が変数名を別名とする番地定数(address constant)，またはセグメント・ロード定数(segment load constant)であることを示す。

例

```
real  A  syn  B (1)  ¥
integer  I  syn  C (5)  ¥
array  5  integer  D syn  K  ¥
```

(6) 宣言の終りは¥記号がくる。

例

```
integer   I , J , K ¥
integer   N= 0 ¥
array 3 real F=( 1.0 , 2.0 , 3.0 ) ¥
integer   A=77$\overline{0}$, B=76$\overline{0}$, C=1 ¥
```

18. 同義語宣言(Synonym Declarations)

同義語宣言は,この宣言にあらわれた変数名とこの宣言がおこなわれるまえに宣言されている変数の属性を結びつける働きをする。

形式

array n {real / integer} 変数名 syn {変数 / 定数} ,………,

………, 変数名 syn {変数 / 定数} (t1)

変数名を配列として宣言するなら array n が必要である。

{real / integer} {adcon / slcon} 変数名 syn 変数,………,

………, 変数名 syn 変数 (t2)

(t1) の宣言で変数名 syn 変数と宣言されたときは,変数名はこのとき変数がもつ番地を共有する。

(t1) の宣言で変数名 syn 定数と宣言されたときは,変数名の番地は定数で示された番地に同じである。すなわち,この定数は機械語の番地そのものを示している。したがつて,この定数はベイス・レジスタの番号とそのレジスタからの変位(displacement)を示している。定数がレジスタ番号を含んでいなければ,この変数名のベイス・レジスタはB0(FACOM 230-60の場合)が与えられる。

(t2) の宣言では,変数が変数名を別名とする番地定数(address constant),またはセグメント・ロード定数(segment load constant)であることを示す。

例

```
real   A   syn   B(1)   ¥
integer   I   syn   C(5)   ¥
array   5   integer   D syn   K   ¥
```

integer　adcon　M　syn　ECB(3)　¥

注意

(1) 変数名と変数が結びつけられるときに両者の型は異つていてもよい。

(2) 同義語宣言では初期値は設定されない。

(3) 番地定数として定義された変数の番地の部分に他の変数の所在を示す番地を入れる宣言はない。たとえば，TACOM230−60　FASP　の文

A　ADCON　FCB

と同じ定義はつぎのようになる：

integer adcon B syn A ¥

:

A＝FCB or A ¥

この例からわかるように，実行時にFCBの番地（すなわちベイス・レジスタBi からの変位）を求めて変数Aとの論理和を取ることでAが正しい番地定数となる。この場合Aはすでにベイス・レジスタの番地を持つていると仮定する。

(4) プログラムの命令中に番地定数を定義するときはこのような操作は関数を使つておこなう。（関数の定義と参照の項を見よ。）

禁止事項

(1) 上の(t1),(t2) いずれの宣言においてもA(Xi)，A(Xi±C) の形式の変数を指定してはならない。

19. 手続き宣言（Procedure Declarations）

手続き宣言は，begin で始まりendで終る一群の宣言と文のまとまり（これを手続きの本体という）に1つの名まえ（手続き名）を与え，この手続き名を参照することで手続きの本体の実行を可能とするためにある。

形式

procedure	手続き名　¥文	(t1)
procedure	手続き名（仮パラメータ，…，仮パラメータ） ¥　文	(t2)

(t1) の形式では宣言された手続きはパラメータを持たないから，参照するプログラムと参照される手続きとの間の情報の授受は，インデックス・レジスタや共通領域などを通して行なわれることになる。

(t2) の形式では宣言された手続きはパラメータを持つから，情報の授受はこのパラメータを通しても行なわれる。

手続き宣言の本体は，begin ……end のブロックでなければならない。

integer adcon M syn ECB(3) ¥

注意

(1) 変数名と変数が結びつけられるときに両者の型は異つていてもよい。

(2) 同義語宣言では初期値は設定されない。

(3) 番地定数として定義された変数の番地の部分に他の変数の所在を示す番地を入れる宣言はない。たとえば，TACOM230－60 FASP の文

A ADCON FCB

と同じ定義はつぎのようになる：

integer adcon B syn A ¥

⋮

A＝FCB or A ¥

この例からわかるように，実行時にFCBの番地（すなわちベイス・レジスタBi からの変位）を求めて変数Aとの論理和を取ることでAが正しい番地定数となる。この場合Aはすでにベイス・レジスタの番地を持つていると仮定する。

(4) プログラムの命令中に番地定数を定義するときはこのような操作は関数を使つておこなう。（関数の定義と参照の項を見よ。）

禁止事項

(1) 上の（t1)，(t2) いずれの宣言においてもA（Xi），A(Xi±C) の形式の変数を指定してはならない。

19。 手続き宣言（Procedure Declarations）

手続き宣言は，begin で始まりendで終る一群の宣言と文のまとまり（これを手続きの本体という）に1つの名まえ（手続き名）を与え，この手続き名を参照することで手続きの本体の実行を可能とするためにある。

形式

procedure	手続き名 ¥文	(t1)
procedure	手続き名（仮パラメータ，…，仮パラメータ） ¥ 文	

(t1) の形式では宣言された手続きはパラメータを持たないから，参照するプログラムと参照される手続きとの間の情報の授受は，インデックス・レジスタや共通領域などを通して行なわれることになる。

(t2) の形式では宣言された手続きはパラメータを持つから，情報の授受はこのパラメータを通しても行なわれる。

手続き宣言の本体は，begin ……end のブロックでなければならない。

例1.

```
procedure  EXCHANGE(X,Y)¥
begin real  X,Y,T¥
   T=X¥  X=Y¥Y=T¥    end
```

例2.

```
procedure  EXCHANGE ¥
begin segment base /COMMON/¥
real  X , Y , T¥  sogment base  local¥
T=X¥  X=Y¥  Y=T¥    end
```

例3.

```
procedure  IFX ¥
if P(X3).GT. P(X4 begin) then begin A(X3)=P(X4) ¥
X3=X3+1¥  end else  X4=X4+1 ¥ end
```

この例3では変数PとAはこの手続き宣言を含むプログラムで定義されていると仮定されている。

注意

(1) 手続き宣言は，他の手続き宣言の本体のなかで定義することができるが，このように1つのブロックのなかで宣言された手続きは，このブロックの外側のブロックで参照することはできない。（ブロック構造を参照のこと。）

(2) 1つのブロックのなかでいくつかの手続きを宣言することができる。それらの手続きは他の手続きを参照することができるが，そのとき参照が再帰的であつてはならない。すなわち，直接的，または間接的に自己を参照してはならない。ひとつの手続き宣言中で他の手続きを参照するときは，それらの手続き宣言は排他的，かつそれらを含む翻訳単位のブロック・オプションが1でなければならない（下図参照）。

```
*GPL        BLOCK=1  ――――――――  ブロック・オプション=1
 begin
   :
 procedure   A   ¥            ┐
   begin  ……     end  ¥      ┘ 並列，排他的
 procedure   B   ¥            ┐ 手続き宣言
   begin  ……  ¥  A¥  ……  end ┘
```

(3) 仮パラメータに対して手続きの本体で型宣言をおこなわなければならない。したがつて仮パラメータをもつ手続きの本体は必らず**begin** で始まり**end**で終る。すなわち本体はブロックであつて，型宣言は本体の初めの部分に含まれている。

(4) 手続き宣言は，他の宣言と同じくプログラムのなかでは実行文のまえに定義されていなければならない。

(5) 手続きの実行を終つて，この手続きを呼び出したプログラムに制御を返すときは，この

手続きの本体の一番外側の endに制御を移せばよい。

例

```
proceure  P(X) ¥
begin
      goto  FIN ¥
FIN ・・ end
```

この例では goto FIN によつて、この手続きP (X)を呼び出したプログラムに制御が移る。

(b) 副プログラムとしての手続きは。begin のあとに procedure ABC …と宣言された手続き名 ABC…が結合プログラム(linkage editor)の処理の対象となる。

禁止事項

(1) 仮パラメータに指定できるのはセル変数のみである。インデックス・レジスタ名，定数，手続き名，関数名，レイベル(レイベルについては第1版のみ)などを指定してはならない。

20. 関数宣言(Function Declarations)

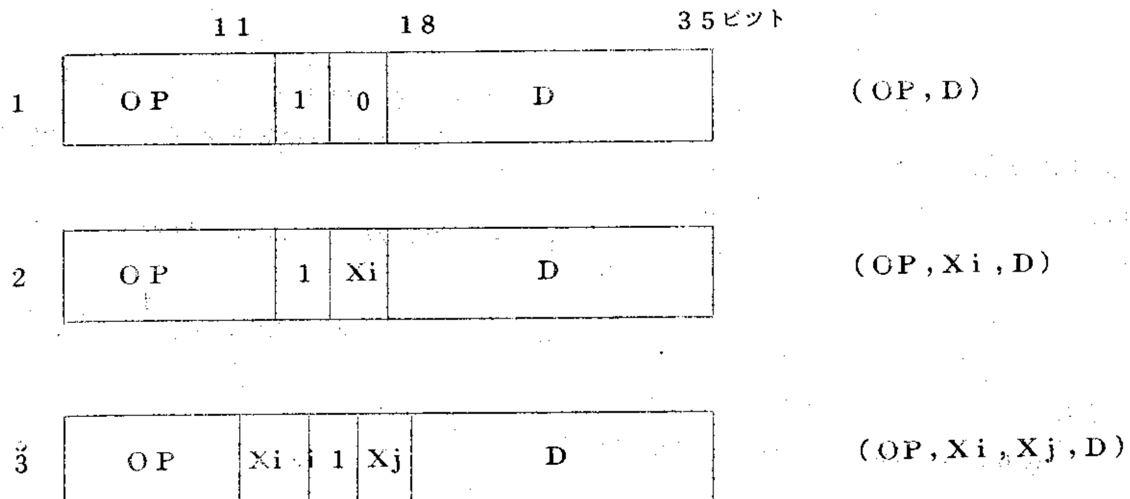
関数宣言は代入文や手続き文では利用できない計算機の命令を使うためにある。この機能はGPLが実現された計算機に依存している。ここではFACOM230−60版GPLについてのべる。

形式

function 関数名(Ti,Ii), … 関数名(Tn,In)

ここで Ti はFig. 5 にかかげた命令の形式を示す1から6までの整数である。Ii は命令の形式に応じて長さの定まるFACOM230−60の命令を8進数で与える。関数名は何であつてもよいが，計算機の命令と直接結びついた名前がよい。

形式



手続きの本体の一番外側のendに制御を移せばよい。

例

```
proceure  P(X) ¥
begin
      goto  FIN ¥
FIN ‥ end
```

この例ではgoto FINによつて、この手続きP(X)を呼び出したプログラムに制御が移る。

(b) 副プログラムとしての手続きは。begin のあとに procedure ABC …と宣言された手続き名 ABC…が結合プログラム(linkage editor)の処理の対象となる。

禁止事項

(1) 仮パラメータに指定できるのはセル変数のみである。インデックス・レジスタ名，定数，手続き名，関数名，レイベル(レイベルについては第1版のみ)などを指定してはならない。

20. 関数宣言(Function Declarations)

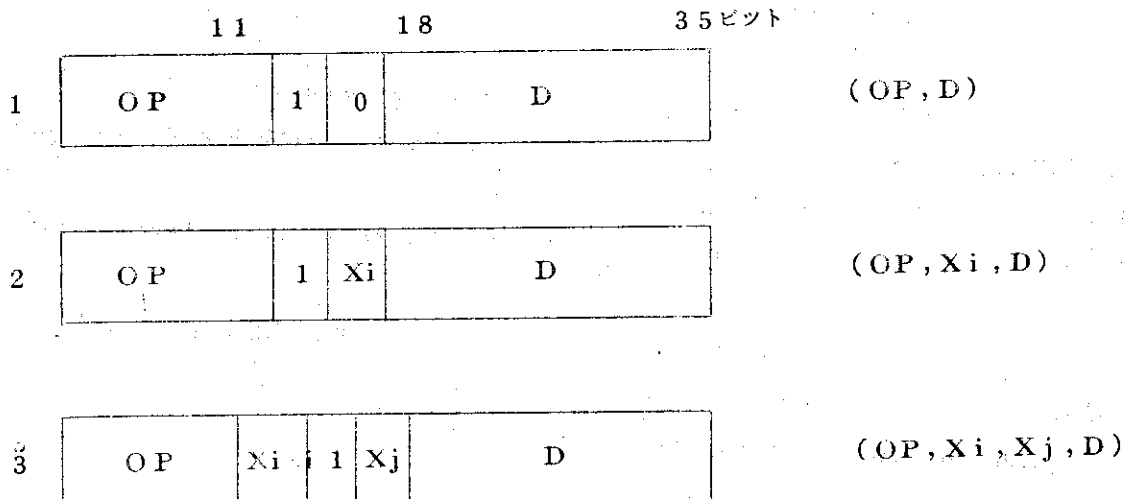
関数宣言は代入文や手続き文では利用できない計算機の命令を使うためにある。この機能はGPLが実現された計算機に依存している。ここではFACOM230−60版GPLについてのべる。

形式

```
function  関数名(Ti,Ii),  …  関数名(Tn,In)
```

ここでTiはFig.5 にかかげた命令の形式を示す1から6までの整数である。Ii は命令の形式に応じて長さの定まるFACOM230−60の命令を8進数で与える。関数名は何であつてもよいが，計算機の命令と直接結びついた名前がよい。

形式



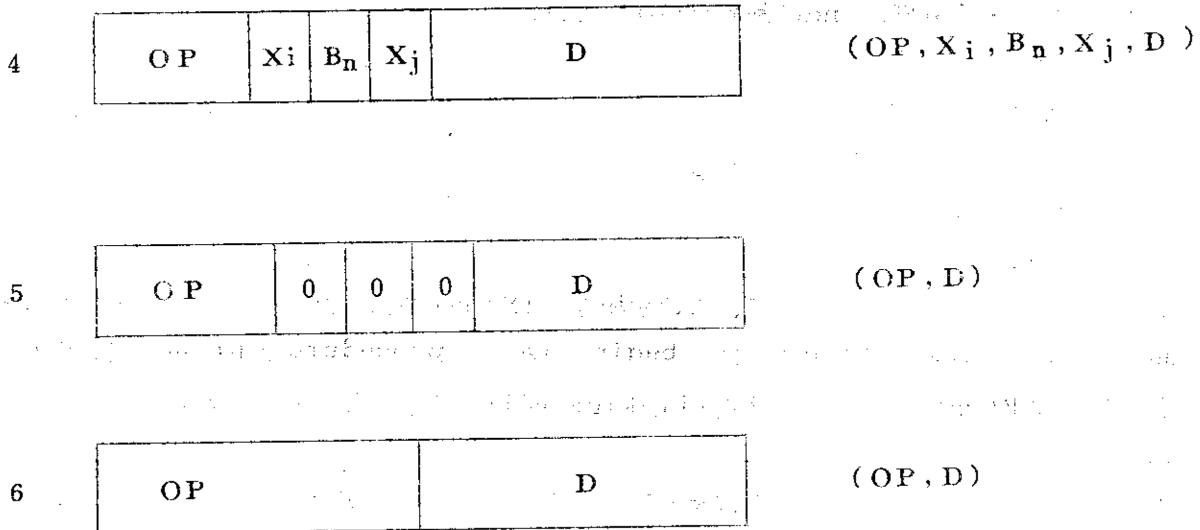


Fig. 5　function の形式（FACOM230－60）

例

```
function LR(1,0104Ō),TRP(1,6000Ō),J(1,4000Ō),
         ADCON(4,777Ō),BP(5,000Ō) ¥
```

この宣言はつぎのように関数文で使われる。

```
begin label ADR1,ADR2,ADR3 ¥
        LR(ADR1) ¥
        TRP(S.OPEN) ¥
        J(ADR3) ¥
ADR1‥ ADCON(0,1,0,ADR2) ¥¥
ADR2‥ ADCON(3,1,0,FCB(1)) ¥
        BP(0) ¥
ADR3‥ Comment NEXT STATEMENT ¥
```

これはFACOM230－60システム・マクロ「オープン・ファイル」の展開形である。

注意

(1)　8進数で3ケタの命令777は番地定数(address constant)を示す。この777はコンパイラによつて8進数000に置きかえられ，そのかわりに結合プログラム(linkage editor)のためのコントロール・ビットが作られる。

(2)　8進数で3ケタの命令776はセグメント・ロード定数(segment load constant)を示す。

(3)　形式(2)～(4)の関数の変数のX_i，B_nはそれぞれのレジスタ名を指定してもよいし，またそれぞれのレジスタに対応する番号を指定してもよい。

(4)　形式(6)は(1)～(5)のあらゆる場合を含むことができる。

禁止事項

(1)　B_nはユーザがその使用法を理解している場合を除いてはB_1以外を指定してはならない。

21. 領域宣言(Segment Base Declarations)

領域宣言によつて，この宣言がおこなわれたあとで定義される変数やプログラムの命令が宣言で指定された領域に含まれることになる。領域宣言は異つたプログラム(コンパイル)単位の間で領域を共有する必要があるときに特に有用である。

形式

segment base Bi	(t1)
segment base /領域名/	(t2)
segment base //	(t3)
segment base local	(t4)

(t1) の宣言がおこなわれると，これ以後に定義される領域(変数，プログラムの命令など)はすべてベイス・レジスタ B_i を基礎番地にもつとして翻訳される。この宣言がなければ通常はベイス・レジスタ B_1 が仮定されて翻訳される。

(t2) の宣言がおこなわれると，これ以後に定義される変数はこの宣言で指定された領域内で定義される。変数の領域内の番地は，変数の定義された順序にしたがう。この領域とFORTRAN のCOMMON /領域名/，…で定義された領域(レイベル付コモン)とは同じであるから，GPLで書かれたプログラムとFORTRAN で書かれたプログラムとは領域を共有することができる。

(t3) の宣言は(t2) の宣言と同じであるが名前だけが違い，これはFORTRAN のCOMMON……で定義された領域(ブランク・コモン)と同じである。

(t4) の宣言は(t1) から(t3) までの宣言を取消す働きをもつている。この宣言がなされたあとは，翻訳される命令と変数はすべてこのプログラム単位の領域内にとられる。

例

```
segment base /COM/ ¥ real A1,B1,C1 ¥
segment base /COM/ ¥
  integer I,J,K ¥  segment base // ¥
  real A,B,C ¥ array 10 real D ¥
  segment base local ¥
  real X,Y,Z ¥
```

上の例では，領域名COM(FORTRAN のレイベルつきコモン/COM/と同じ)に6個の変数A1,B1,C1,I,J,K の番地がとられる。領域名//(FORTRAN のブランク・コモンと同じ)で13個の変数の番地がとられる。segment base localのあとのこのプログラムに依存した局所的な変数X，Y，Zの番地は，このプログラムの変数の領域にとられるが，これらの局所変数の領域名は存在しないから，局所変数を他のプログラム単位から呼び出すことはできない。

注意

(1) どのプログラム単位でも，翻訳の初めには

segment base B_1

segment base local

が仮定されているから，この2つを宣言しなくてよい。

(2) segment base ／領域名／（／／を含む）で変数を定義したあとで局所変数を定義したり，実行文を定義するとき，そのまえに必らず segmnt base local の宣言がいる。

(3) 1度定義された（t1）の宣言文をリセットするのは，新しい（t1）形式の宣言による。（t2），（t3）の宣言をリセットするのは，新しい（t2），（t3）の形式の宣か（t4）の宣言による。

(4) 同一ブロック内で同じ名まえをもつ領域宣言があるときは，それらの宣言はひとつのつながりと見なされ，全体の大きさで領域の大きさが決まる。

(5) ひとつのブロック内で定義された領域は，そのブロックが終了したときに，そのブロック内で定義されたその領域の変数に相当する番地が消去される。ただし領域名は消去されない。

(6) 領域の大きさは，プログラム単位の翻訳時にその領域が取る最大の大きさとして定義される。

22. ブロック

ブロックはおもにつぎの2つの目的のために使われる：

(1) 一連の文をまとめて1つの無条件文とする。

(2) 新しい量を導入し，これに名まえを与える。この名まえは外側のブロックでは未定義である。

形式

begin D ¥ ··· ¥D ¥S¥ ··· S¥ end	(t1)
begin S ¥ ··· S¥ end	(t2)

ここでDは宣言，Sは文を示す。ブロックも1つの文であるから，Sはブロックであつてもよい。

ブロック内では宣言Dは文Sのまえになければならない。ブロック内の文Sは左から右へと実行される。

（t1）の形式では宣言Dによつて新しい量が導入される。この量に与えられた名まえは，このブロックの外側のブロックから呼び出すことはできない。また外側のブロックと同じ名まえが定義されると，このブロック内では新しく定義された名まえが意味をもち，その名まえのもつ量が使われる。このブロックの外側では，それまで外側で定義された名まえと量が意味をも

注意

(1) どのプログラム単位でも，翻訳の初めには

segment base B_1

segment base local

が仮定されているから，この2つを宣言しなくてよい。

(2) segment base ／領域名／（／／を含む）で変数を定義したあとで局所変数を定義したり，実行文を定義するとき，そのまえに必らず segmnt base local の宣言がいる。

(3) 1度定義された（t1）の宣言文をリセットするのは，新しい（t1）形式の宣言による。（t2），（t3）の宣言をリセットするのは，新しい（t2），（t3）の形式の宣か（t4）の宣言による。

(4) 同一ブロック内で同じ名まえをもつ領域宣言があるときは，それらの宣言はひとつのつながりと見なされ，全体の大きさで領域の大きさが決まる。

(5) ひとつのブロック内で定義された領域は，そのブロックが終了したときに，そのブロック内で定義されたその領域の変数に相当する番地が消去される。ただし領域名は消去されない。

(6) 領域の大きさは，プログラム単位の翻訳時にその領域が取る最大の大きさとして定義される。

22. ブロック

ブロックはおもにつぎの2つの目的のために使われる：

(1) 一連の文をまとめて1つの無条件文とする。

(2) 新しい量を導入し，これに名まえを与える。この名まえは外側のブロックでは未定義である。

形式

```
begin  D ¥ ··· ¥D  ¥S¥ · · · S¥  end        (t1)
begin  S  ¥ ··· S¥  end                      (t2)
```

ここでDは宣言，Sは文を示す。ブロックも1つの文であるから，Sはブロックであつてもよい。

ブロック内では宣言Dは文Sのまえになければならない。ブロック内の文Sは左から右へと実行される。

（t1）の形式では宣言Dによつて新しい量が導入される。この量に与えられた名まえは，このブロックの外側のブロックから呼び出すことはできない。また外側のブロックと同じ名まえが定義されると，このブロック内では新しく定義された名まえが意味をもち，その名まえのもつ量が使われる。このブロックの外側では，それまで外側で定義された名まえと量が意味をも

つ。

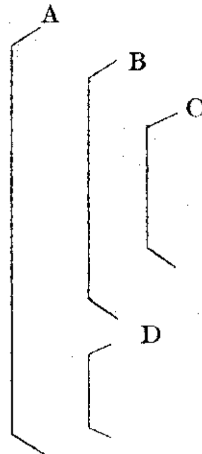
(t2) の形式では宣言がないから，このブロックで使用されるレイベル宣言，変数などはすべてこのブロックを含む外側のブロックで定義されていなければならない。

23. ブロック・レベルとブロックのネスト

23.1 ブロックのネストとブロックのレベル

いくつかのブロックがたがいに含む，含まれるの関係にあるとき，これらのブロックはネストをなすという。

つぎの図でブロックA，またはB，またはCを基準にすると，ブロックA，B，Cはネストをなしており，ブロックA，またはDを基準にするとブロックA，Dはネストをなしている。



ブロックBとDはネストを成さない。このBとDはAについて同列(並列)なブロックであるという。ブロックBが他のブロックAに含まれているとき，BはAよりも高い(より深い)ブロック・レベルをもつという。

23.2 宣言ブロック

ブロックはつねに**begin** ではじまり，**end** で終る。ブロックには**begin** のあとに宣言が続くものと，実行文が続くものとがある。前者を宣言ブロック，後者を無宣言ブロックとよぶ。宣言ブロックは翻訳時に宣言を処理するための初期設定が必要となるので，ややコンパイル時間を消費する。無宣言ブロックについてはこのようなことはない。

23.3 ブロックとレイベル

ブロック内で定義されたレイベルは宣言とは見なされない。レイベルは，レイベル名にピリオドを2つ続けることによつて定義される。レイベルの定義とレイベル宣言とは別の概念であるから，この2つを混同してはならない。無宣言ブロック内でレイベルの定義が存在しても，そのブロックは宣言ブロックとはならない。

宣言ブロック内で定義されたレイベルを，このブロックの外側から参照することはできない。

つ。

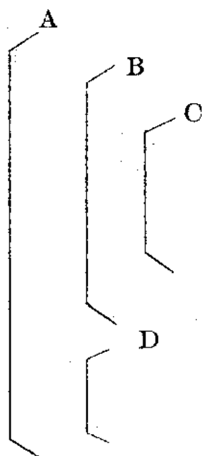
(t2)の形式では宣言がないから，このブロックで使用されるレイベル宣言，変数などはすべてこのブロックを含む外側のブロックで定義されていなければならない。

23. ブロック・レベルとブロックのネスト

23.1 ブロックのネストとブロックのレベル

いくつかのブロックがたがいに含む，含まれるの関係にあるとき，これらのブロックはネストをなすという。

つぎの図でブロックA，またはB，またはCを基準にすると，ブロックA，B，Cはネストをなしており，ブロックA，またはDを基準にするとブロックA，Dはネストをなしている。



ブロックBとDはネストを成さない。このBとDはAについて同列（並列）なブロックであるという。ブロックBが他のブロックAに含まれているとき，BはAよりも高い（より深い）ブロック・レベルをもつという。

23.2 宣言ブロック

ブロックはつねに**begin** ではじまり，**end** で終る。ブロックには**begin** のあとに宣言が続くものと，実行文が続くものとがある。前者を宣言ブロック，後者を無宣言ブロックとよぶ。宣言ブロックは翻訳時に宣言を処理するための初期設定が必要となるので，ややコンパイル時間を消費する。無宣言ブロックについてはこのようなことはない。

23.3 ブロックとレイベル

ブロック内で定義されたレイベルは宣言とは見なされない。レイベルは，レイベル名にピリオドを2つ続けることによつて定義される。レイベルの定義とレイベル宣言とは別の概念であるから，この2つを混同してはならない。無宣言ブロック内でレイベルの定義が存在しても，そのブロックは宣言ブロックとはならない。

宣言ブロック内で定義されたレイベルを，このブロックの外側から参照することはできない。

これに反して無宣言ブロック内で定義されたレイベルはこのブロックの外側で参照することができる。

例1.

```
begin  label  L ¥            ┐
                             │  宣言ブロック
  L‥  X3=X3+1  ¥             │
                             │
end                          ┘
```

例2.

```
begin  integer  I ¥          ┐
                             │  宣言ブロック
  L‥  X3=X3+1  ¥             │
                             │
end                          ┘
```

例3.

```
begin                        ┐
                             │  無宣言ブロック
L‥ X3=X3+1  ¥                │
                             │
end                          ┘
```

例1，2のレイベルLは，このブロックの外側から参照することはできない。これに反して例3のレイベルは，このブロックの外側から参照することができる。

2.3.4　ブロツク・オプション

ブロックAで宣言された変数は，このブロックの外側では参照できないから，その変数に割当てられた番地は，ブロックAに一時的に割当てられたものと解することができる。したがつて，ブロックAと並列なブロックBで宣言された変数は，ブロックAで宣言された変数と同じ番地を割当てられてもよいことになる。これはALGOLの考え方である。GPLではブロック・オプションBLK0と制御カードに指定されているときに，このような番地の割当てがおこなわれる。これに反し，一度割当てられた番地は，再び他の変数に割当てられることはないように翻訳することもできる。これはFORTRANの考え方である。GPLではブロック・オプションBLK1制御カードに指定されているときに，このような番地の割当てがおこなわれる。GPLとBLK0の指定がなければ，BLK1が仮定される。

謝　辞

GPLコンパイラを作成するにあたつて原研計算センタの岡田，山崎，次田の諸氏には，GPLで書かれた解釈ルーチンをFORTRAN語に書き換えていただいた。富士通株式会社SEの安村，大浦，井坂の諸氏には必要な資料を取り寄せていただき，またプログラムのデイバッグでもお世話になつた。上記の方々および，GPLについて検討，支援していただいた原研計算センタの諸氏に感謝の意を表したい。

これに反して無宣言ブロック内で定義されたレイベルはこのブロックの外側で参照することができる。

例1.

```
begin  label  L ¥            ┐
                             │  宣言ブロック
  L··  X3=X3+1  ¥            │
                             │
end                          ┘
```

例2.

```
begin  integer  I ¥          ┐
                             │  宣言ブロック
  L··  X3=X3+1  ¥            │
                             │
end                          ┘
```

例3.

```
begin                        ┐
                             │  無宣言ブロック
L·· X3=X3+1  ¥               │
                             │
end                          ┘
```

例1，2のレイベルLは，このブロックの外側から参照することはできない。これに反して例3のレイベルは，このブロックの外側から参照することができる。

2.3.4　ブロツク・オプション

ブロックAで宣言された変数は，このブロックの外側では参照できないから，その変数に割当てられた番地は，ブロックAに一時的に割当てられたものと解することができる。したがつて，ブロックAと並列なブロックBで宣言された変数は，ブロックAで宣言された変数と同じ番地を割当てられてもよいことになる。これはALGOLの考え方である。GPLではブロック・オプションBLK0と制御カードに指定されているときに，このような番地の割当てがおこなわれる。これに反し，一度割当てられた番地は，再び他の変数に割当てられることはないように翻訳することもできる。これはFORTRANの考え方である。GPLではブロック・オプションBLK1 制御カードに指定されているときに，このような番地の割当てがおこなわれる。GPLとBLK0の指定がなければ，BLK1が仮定される。

謝　辞

GPLコンパイラを作成するにあたつて原研計算センタの岡田，山崎，次田の諸氏には，GPLで書かれた解釈ルーチンをFORTRAN語に書き換えていただいた。富士通株式会社SEの安村，大浦，井坂の諸氏には必要な資料を取り寄せていただき，またプログラムのデイバッグでもお世話になつた。上記の方々および，GPLについて検討，支援していただいた原研計算センタの諸氏に感謝の意を表したい。

参考文献

1) Wirth,N. PL360, A Programming Language for the 360 Computers, J. ACM, Vol・15, No. 1, Jan・1968, pp・37−74.

2) 浅井，稲見，藤村，変形PL360言語の文法とコンパイラ，第12回プログラミング・シンポジウム報告集，情報処理学会，1971年1月・

3) 藤村，浅井・PRECEDENCE：Precedence Grammarsを求めるプログラム，JAERI−memo。4003，日本原子力研究所，1970年5月。

付　　　録

ここではＧＰＬの構成についてすこしふれておく。Fig. 6 はPL360 とＧＰＬとの比較表である。ＧＰＬはPL360から出発したが，その機能や内容はほとんど異っている。Fig･7はＧＰＬの解析の基礎をなす順位文法と順位関数を計算するプログラム3)の機能を示したものである。Fig･8 はこうして得られた関数値を実際のコンパイラが使用する方法を，Fig･9とFig･10はＧＰＬで書かれたＧＰＬコンパイラの１部を示している。Fig･11とFig･12 はＧＰＬの文法と順位関数である。

項目	PL 360	GPL
目標	システム記述言語	同左
計算機	IBM System360	FACOM230−60，IBM360，CDC6600
言語形式	ALGOL60−like 言語	同左
文法	順位文法	同左
解析	順位関数	同左
翻訳	文法生成規則に対して解釈規則を与えておこなう。	同左
命令生成	IBM System360	仮想的な計算機PM 1の命令を生成し，その後その命令を対象計算機の命令に変換。
プログラムのロード	簡単なセルフ・ローダを使用	OSのLinkage Editorを使用。
他言語との結合	できない	アセンブラ言語，FORTRAN 語で書かれたプログラムとの結合可能。

Fig･6　PL360とＧＰＬ

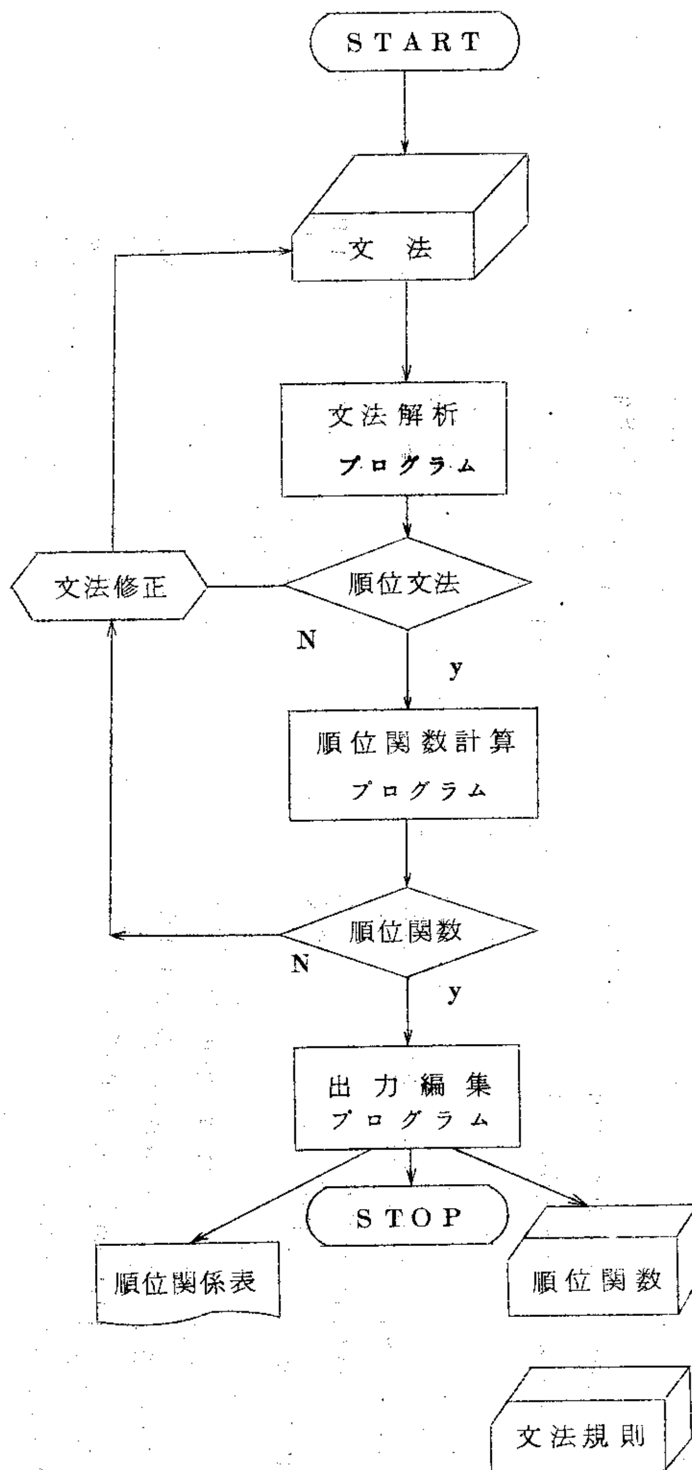


Fig. 7　　順位関数の計算手順

生成規則の左辺の検索

生成規則 $S_k \longrightarrow S_i \cdots\cdots S_j$ の左辺 S_k の検索は次の 1.～4.の手順でおこなう。

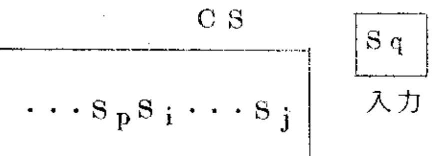
1. コントロール・スタックCSのなかに置かれている $S_i, \cdots\cdots, S_j$ などはすべて数値化されている。

2. 右図のCSの文法要素と入力要素 S_q との間に次の順位関係があるとき左辺の検索が始まる。

3. 数値 $S_i \cdots\cdots S_j$ の和を求める。その和 ℓ は左辺の表LPのひとつの番地を与える。
   もしLP (ℓ)＞0ならば，LP (ℓ)が求める左辺である。

4. もしL (ℓ)＜0ならば，和 ℓ を持つ右辺は複数個存在する。このとき，これらの生成規則は表MULTRに保存されている。|LP (ℓ)| はMULTRの該当する生成規則の最初の番地を与える。

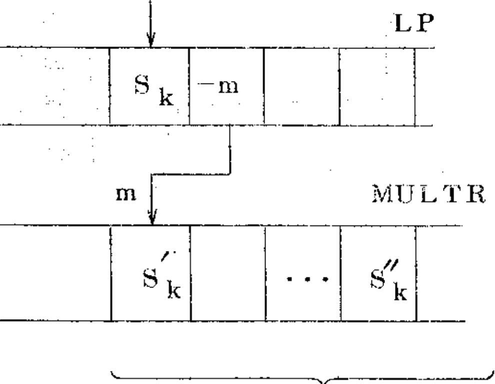
5. GPLの場合のLP，MULTRの大きさは，それぞれ200バイトと450バイトである。



$$S_p \lessdot S_i, \quad S_j \gtrdot S_q,$$

$$S_i \doteq \cdots \doteq S_j.$$

$$\ell = S_i + \cdots + S_j.$$



Fig・8　順位文法の解析

日立高速納

```
                                                         PAGE    6

 GLOBAL PUSH DOWN STACKS
 ==========================================
* CS ... CONTROL STACK (GLOBAL)
      THIS STACK STORES SYNTACTIC ELEMENTS IN ANALYSIS. IF DECLARA-
     TION IS PARSED, THIS STACK CONTAINS SYNTACTIC ELEMENT 'DECL'
     AS ITS TOP ELEMENT. IF A PROGRAM HAS BEEN PARSED BY THE
     PARSER, THE STACK CONTAINS 'PROGRAM' AS ITS TOP ELEMENT.
  ICS ... INDEX TO CS

* VS ... VALUE STACK (GLOBAL)
      THIS STACK CONTAINS VALUES WHICH CORRESPOND TO SYNTACTIC EL-
     MENTS IN CONTROL STACK CS. IN MOST CASES THE VALUE IS AN
     ADDRESS OF AN IDENTIFIER OR AN OPERATOR.
  IVS ... INDEX TO VS

* BS ... BLOCK STACK (GLOBAL)
      THIS STACK CONTAINS TYPE OF BLOCKS(E.G. PROCEDURE,SYN DECL,
     PROGRAM, ETC) IN ANALYSIS. THE TOP OF THE STACK CONTAINS THE
     TYPE OF BLOCK BEING ANALYZED.
  IBS ... INDEX TO BS

* BNS ... BLOCK NUMBER STACK (GLOBAL)
      THIS STACK CONTAINS BLOCK NUMBERS OF BLOCKS IN ANALYSIS.
     EVERY BLOCK WHICH IS PROCESSED BY THE PARSER HAS A SEQUENTIAL
     NUMBER IN ORDER OF OCCURRENCES.
  IBNS ... INDEX TO BNS

* PAS  ... TARGET PROGRAM ADDRESS (GLOBAL)
      THIS STACK CONTAINS GENERATED TARGET CODE ADDRESSES OF
     SYNTACTIC ELEMENTS IN CS.
  IPAS  ... INDEX TO PAS

* LDS ... LABEL DEFINITION STACK (GLOBAL)
      THIS STACK CONTAINS LABELS AND THEIR LOCATIONS DEFINED BY
     'LABEL DEF'.. . EACH CELL OF THE STACK IS OF THE FORM
     .BOTTOM OF STACK
     .....(0,0)(PTR,DL) ... (PTR,DL)(0,BLOCKNO)(PTR,DL) ...
      WHERE PTR IS A POINTE.. FOR OF LABEL IDENTIFIER AND
            DL IS A DEFINED LOCATION OF THE LABEL.
  ILDS ... INDEX TO LDS

* RLS ... REFERENCED LABEL STACK (GLOBAL)
      THIS STACK CONTAINS REFERENCED LABELS, LOCATIONS REFERENCED.
      EACH CELL OF THE STACK IS OF THE FORM
     BOTTOM (0,0)(PTR,RL) ...(PTR,RL)(0,BLOCKNO)(PTR,RL) ...
      WHERE PTR IS A POINTER OF REFERENCED LABEL IDENTIFIER,
            RL IS A LOCATION WHERE THE LABEL IS REFERENCED AND
            BLOCKNO IS THE BLOCK NO. IN WHICH THE LABEL IS
           REFERENCED.
      AT AN END OF A BLOCK, LDS STACK IS SEARCHED TO FILL A PROPER
```

fig. 9　GPLコンパイラのスタック

```
                                                         PAGE   40

      * 'COMP AOR'     ..= 'COMP COND' OR                            *
      *                                                              *
      ****************************************************************

       X0=X0+3 * PROG(X0)=TRA * PROG(X0+1)=00 * PROG(X0+2)=00*
        COMMENT TRANSFER IF TRUE(I.E., TRA).
                                      THIS CONDITION MEANS THAT THE
               PRECEDING CONDITION IS TRUE. TPL MEANS TRANSFER IF
               TRUE. 00 MUST BE FILLED WITH A LOCATION OF NEXT
               'SIMPLE ST'. REMEMBER CURRENT X0 IN TTS STACK. PUP UP
               TFS STACK IF IT IS ALREADY DEFINED BY 'COMP COND' AND*
       ITTS=ITTS+1 * X2=ITTS * TTS(X2)=X0 *  X2=ITFS *
       IF TFS(X2).EQ.0 THEN GOTO  AOR1 *
                IP=X0 * X0=TFS(X2) * PROG(X0+1)=IP+3 * X0=IP *
       ITFS=ITFS-1 *
   AOR1.. ITFS=ITFS-1*
       GOTO PPARSE *

 S067..
 COMMENT**************************************************************
      *                                                              *
      * 'COND THEN'    ..= 'COMP COND' THEN                          *
      *                                                              *
      ****************************************************************

       X0=X0+3 * PROG(X0)=TRA * PROG(X0+1)=00 * PROG(X0+2)=00 *
       ITFS=ITFS+1 * TFS(X2)=X0 *
        COMMENT  CLOSE TTS STACK OF THIS 'COMP COND' *
       X2=ITTS * IP=X0 *
  CTH1.. IF TTS(X2).EQ.0 THEN GOTO  CTH2 *
              X0=TTS(X2) * PROG(X0+1)=IP+3 * X2=X2-1 * GOTO CTH1 *
  CTH2.. X2=X2-1 * ITTS=X2 * X0=IP *
       GOTO PPARSE *

 S068..
 COMMENT**************************************************************
      *                                                              *
      * 'TRUE PART'    ..= 'SIMPLE ST' ELSE                          *
      *                                                              *
      ****************************************************************

          COMMENT GENERATE A TRANSFER INSTRUCTION TO SKIP A STATEMENT
                FOLLOWING ELSE *

        X0=X0+3 * PROG(X0)=TRA * PROG(X0+1)=00 * PROG(X0+2)=0 * IP=X0*
        COMMENT CLOSE TFS STACK OF THIS 'COMP COND' *
        X2=ITFS *
  TPR1.. IF TFS(X2).EQ.0 THEN GOTO TPR2 *
        X0=TFS(X2) * PROG(X0+1)=IP+3 * X2=X2-1 * GOTO TPR1 *
  TPR2.. X2=X2-1 * ITFS=X2 *
```

fig. 10　GPLコンパイラの1部

```
              ----- EACH OPTION OF OUTPUT -----

          0       0       0       0       1

                    NUMBER OF CHARACTER IN A WORD =  4


        *** WRITE OUT OF INPUT RULES (FORM 2)    ***


LEFT PART BEGINS WITH  11 COLUMN
RIGHT PART BEGINS WITH 28 COLUMN

 1           'ID'            ..= IDI
 2           'T NUMBER'      ..= IDT
 3           'STRING'        ..= IDS
 4           'X REG'         ..= IDX
 5           'FUNC ID'       ..= IDF
 6           'OPARAM ID'     ..= IDD
 7           'PROC ID'       ..= IDP
 8           'B REG'         ..= IDB
 9           'LABEL ID'      ..= IDI
10           'EXTERN ID'     ..= EXTERNAL-BASE-ID
11      2    'T CELL ID'     ..= 'ID'
12      5    'T CELL'        ..= 'T CELL ID'
13      6                        'T CELL1'  )
14      7                        'T CELL2'  )
15      8    'T CELL1'       ..= 'T CELL2' 'ARITH OP' 'T NUMBER'
16      9                        'T CELL3' 'T NUMBER'
17     10    'T CELL2'       ..= 'T CELL3' 'X REG'
18     11    'T CELL3'       ..= 'T CELL ID'  (
19     15    'ARITH OP'      ..= +
20     16                        -
21     17                        *
22     18                        /
23     19                        ++
24     20                        --
25     61    'REL OP'        ..= .LT.
26     62                        .EQ.
27     63                        .GT.
28     64                        .LE.
29     65                        .GE.
30     66                        .NE.
31     21    'LOG OP'        ..= AND
```

fig. 11　GPLの文法　(1)

```
32     22                        OR
33     23                        XOR
34     24    'SHIFT OP'      ..= SHLA
35     25                        SHRA
36     26                        SHLL
37     27                        SHRL
38     12    'UNARY OP'      ..= ABS
39     13                        NEG
40     14                        NEGABS
41           'EQU'           ..= =
42           'X REG EXP1'    ..= = 'X REG'
43           'X REG EXPR'    ..= 'X REG EXP1'
44                               'X REG EXP1' 'ARITH OP' 'T NUMBER'
45           'X REG ASS'     ..= 'X REG' 'X REG EXPR'
46                               'X REG' 'T CELL EXP*'
47           'T CELL EXP1'   ..= = 'T CELL'
48                               = 'T NUMBER'
49                               = 'STRING'
50           'T CELL EXP2'   ..= = 'UNARY OP' 'T CELL'
51                               = 'UNARY OP' 'T NUMBER'
52                               = 'UNARY OP' 'STRING'
53                               = A 'T CELL'
54           'T CELL EXPR'   ..= 'T CELL EXP1'
55                               'T CELL EXP2'
56                               'T CELL EXPR' 'ARITH OP' 'T CELL'
57                               'T CELL EXPR' 'ARITH OP' 'T NUMBER'
58                               'T CELL EXPR' 'LOG OP' 'T CELL'
59                               'T CELL EXPR' 'LOG OP' 'T NUMBER'
60                               'T CELL EXPR' 'SHIFT OP' 'T NUMBER'
61           'T CELL EXP*'   ..= 'T CELL EXPR'
62           'T CELL ASS'    ..= 'T CELL' 'T CELL EXP*'
63     44    'FUNC1'         ..= 'FUNC2' 'T NUMBER'
64                               'FUNC2' 'X REG'
65     46                        'FUNC2' 'T CELL'
66     47                        'FUNC2' 'STRING'
67                               'FUNC2' 'LABEL ID'
68                               'FUNC2' 'PROC ID'
69     48    'FUNC2'         ..= 'FUNC ID' (
70     49                        'FUNC1' ,
71           'PROC1'         ..= 'PROC2' 'T CELL'
72                               'PROC2' 'T NUMBER'
73                               'PROC2' 'STRING'
74           'PROC2'         ..= 'PROC ID' (
75                               'PROC1' ,
76           'MULT ASS1'     ..= ((  'T CELL'
77                               ((  'T NUMBER'
78                               ((  'STRING'
79           'MULT ASS2'     ..= 'MULT ASS3' ,
80           'MULT ASS3'     ..= 'MULT ASS1'
81                               'MULT ASS2' 'T CELL'
82                               'MULT ASS2' 'T NUMBER'
83                               'MULT ASS2' 'STRING'
84           'MULT ASS4'     ..= 'MULT ASS3' ))
```

fig. 11　(2)

```
 85        'MULT ASS'     ..= 'MULT ASS4' 'T CELL EXP1'
 86        'VECTOR EXPR'  ..= = 'MULT ASS4'
 87   50   'CASE SEQ'     ..= CASE 'X REG' OF BEGIN
 88   51                      'CASE SEQ' 'STATEMENT' ;
 89        'SIMPLE ST'    ..= 'T CELL' 'X REG EXPR'
 90                           'T CELL' 'EQU' 'B REG'
 91                           'B REG' = 'T CELL'
 92                           'T CELL ASS'
 93   53                      'X REG ASS'
 94                           'MULT ASS'
 95                           'T CELL' 'VECTOR EXPR'
 96   54                      NULL
 97   55                      GOTO 'ID'
 98   56                      'PROC ID'
 99                           'PROC1' )
100   57                      'FUNC ID'
101   58                      'FUNC1' )
102   59                      'CASE SEQ' END
103   60                      'BLOCKBODY' END
104   67   'NOT'          ..= .NOT.
105        'CONDITION'    ..= 'X REG' 'REL OP' 'T CELL'
106                           'X REG' 'REL OP' 'T NUMBER'
107                           'T NUMBER' 'REL OP' 'X REG'
108                           'T CELL' 'REL OP' 'X REG'
109                           'T CELL' 'REL OP' 'T CELL'
110                           'T CELL' 'REL OP' 'STRING'
111                           'T CELL' 'REL OP' 'T NUMBER'
112                           'T NUMBER' 'REL OP' 'T CELL'
113   72                      OVERFLOW
114   74                      'T CELL'
115   75                      'NOT' 'T CELL'
116   76   'COMP COND'    ..= 'CONDITION'
117   77                      'COMP AOR' 'CONDITION'
118   78   'COMP AOR'     ..= 'COMP COND' AND
119   79                      'COMP COND' OR
120   80   'COND THEN'    ..= 'COMP COND' THEN
121   81   'TRUE PART'    ..= 'SIMPLE ST' ELSE
122   82   'WHILE'        ..= WHILE
123   83   'COND DO'      ..= 'COMP COND' DO
124        'ASS STEP'     ..= 'X REG ASS' STEP 'T NUMBER'
125        'LIMIT'        ..= UNTIL 'X REG'
126   86                      UNTIL 'T CELL'
127   87                      UNTIL 'T NUMBER'
128   88   'DO'           ..= DO
129   89   'STATEMENT*'   ..= 'SIMPLE ST'
130   90                      IF 'COND THEN' 'STATEMENT*'
131   91                      IF 'COND THEN' 'TRUE PART' 'STATEMENT*'
132   92                      'WHILE' 'COND DO' 'STATEMENT*'
133   93                      FOR 'ASS STEP' 'LIMIT' 'DO' 'STATEMENT*'
134   94   'STATEMENT'    ..= 'STATEMENT*'
135   95   'SI T TYPE'    ..= SHORT INTEGER
136   96                      INTEGER
137   97                      LOGICAL
```

fig. 11 (3)

```
138   98                      REAL
139   99                      LONG REAL
140  100                      BYTE
141  101                      CHARACTER
142                           LABEL
143  102   'T TYPE'       ..= 'SI T TYPE'
144  103                      ARRAY 'T NUMBER' 'SI T TYPE'
145  104   'T DECL1'      ..= 'T TYPE' 'ID'
146  105                      'T DECL2' 'ID'
147  106   'T DECL2'      ..= 'T DECL7' ,
148  107   'T DECL3'      ..= 'T DECL1' = 'LP'
149  108   'T DECL4'      ..= 'T DECL3'
150  109                      'T DECL5' ,
151  110   'T DECL5'      ..= 'T DECL4' 'T NUMBER'
152  111                      'T DECL4' 'STRING'
153  113   'T DECL7'      ..= 'T DECL1'
154                           'T DECL1' = 'T NUMBER'
155                           'T DECL1' = 'STRING'
156  116                      'T DECL5' )
157        'LP'           ..= (
158  117   'FUNC DC1'     ..= FUNCTION
159  118                      'FUNC DC7' ,
160  119   'FUNC DC2'     ..= 'FUNC DC1' 'ID'
161  120   'FUNC DC3'     ..= 'FUNC DC2' (
162  121   'FUNC DC4'     ..= 'FUNC DC3' 'T NUMBER'
163  122   'FUNC DC5'     ..= 'FUNC DC4' ,
164  123   'FUNC DC6'     ..= 'FUNC DC5' 'T NUMBER'
165  124   'FUNC DC7'     ..= 'FUNC DC6' )
166  125   'SYN DC1'      ..= 'T TYPE' 'ID' SYN
167  126                      'SI T TYPE' REGISTER 'ID' SYN
168                           'SI T TYPE' ADCON 'ID' SYN
169                           'SI T TYPE' SLCON 'ID' SYN
170  127                      'SYN DC3' 'ID' SYN
171  128   'SYN DC2'      ..= 'SYN DC1' 'T CELL'
172  129                      'SYN DC1' 'T NUMBER'
173                           'SYN DC1' 'X REG'
174  131   'SYN DC3'      ..= 'SYN DC2' ,
175  132   'SEG HEAD'     ..= SEGMENT
176  133   'PROC HD1'     ..= PROCEDURE
177  134                      'SEG HEAD' PROCEDURE
178  135   'PROC HD2'     ..= 'PROC HD1' 'ID'
179  136   'PROC HD3'     ..= 'PROC HD2' (
180                           'PROC HD4' ,
181        'PROC HD4'     ..= 'PROC HD3' 'DPARAM ID'
182  138   'PROC HD5'     ..= 'PROC HD4' )
183  139   'PROC HD6'     ..= 'PROC HD5' ;
184                           'PROC HD2D' ;
185        'PROC HD2D'    ..= 'PROC HD2'
186  140   'DECL'         ..= 'T DECL7'
187  141                      'FUNC DC7'
188  142                      'SYN DC2'
189  143                      'PROC HD6' 'STATEMENT*'
190                           'SEG HEAD' BASE 'B REG'
```

fig. 11 (4)

```
191                                 'SEG HEAD'  BASE  'EXTERN ID'
192                                 'SEG HEAD'  BASE  LOCAL
193      145     'LABEL DEF'   ..=  'ID'  ..
194      146     'BLOCKHEAD'   ..=  BEGIN
195      147                        'BLOCKHEAD'  'DECL'  •
196      148     'BLOCKBODY'   ..=  'BLOCKHEAD'
197      149                        'BLOCKBODY'  'STATEMENT'  •
198      150                        'BLOCKBODY'  'LABEL DEF'
199      151     'PROGRAM'     ..=  . 'STATEMENT' .

                                    199  RULES
```

fig. 11 (5)

*** PRINT OF PRECEDENCE FUNCTIONS ***

I	F(I)	G(I)	
1	13	ID	6
2	9	T NUMBER	5
3	6	STRING	5
4	9	X REG	5
5	12	FUNC ID	3
6	5	DPARAM ID	1
7	12	PROC ID	5
8	11	B REG	3
9	5	LABEL ID	5
10	2	EXTERN ID	3
11	12	T CELL ID	6
12	9	T CELL	5
13	4	T CELL1	6
14	4	T CELL2	6
15	5	T CELL3	6
16	5	ARITH OP	4
17	5	REL OP	9
18	5	LOG OP	4
19	5	SHIFT OP	4
20	5	UNARY OP	5
21	3	EQU	9
22	4	X REG EXP1	10
23	4	X REG EXPR	9
24	3	X REG ASS	3
25	6	T CELL EXP1	10
26	6	T CELL EXP2	10
27	4	T CELL EXPR	10
28	4	T CELL EXP*	9
29	3	T CELL ASS	3
30	4	FUNC1	3
31	5	FUNC2	3
32	4	PROC1	3
33	5	PROC2	3
34	5	MULT ASS1	6
35	5	MULT ASS2	6
36	4	MULT ASS3	6
37	10	MULT ASS4	5
38	3	MULT ASS	3
39	3	VECTOR EXPR	9
40	1	CASE SEQ	3
41	2	SIMPLE ST	3
42	5	NOT	3
43	6	CONDITION	2
44	5	COMP COND	2
45	2	COMP AOR	2
46	2	COND THEN	1
47	2	TRUE PART	2
48	1	WHILE	3
49	2	COND DO	1
50	1	ASS STEP	1

fig. 12 順位関数 (1)

51	1	LIMIT	1
52	2	DO	1
53	2	STATEMENT*	2
54	1	STATEMENT	1
55	8	SI T TYPE	9
56	6	T TYPE	9
57	11	T DECL1	9
58	6	T DECL2	9
59	7	T DECL3	9
60	5	T DECL4	9
61	4	T DECL5	9
62	4	T DECL7	9
63	7	LP	5
64	6	FUNC DC1	9
65	12	FUNC DC2	9
66	5	FUNC DC3	9
67	4	FUNC DC4	9
68	5	FUNC DC5	9
69	4	FUNC DC6	9
70	4	FUNC DC7	9
71	5	SYN DC1	9
72	4	SYN DC2	9
73	6	SYN DC3	9
74	9	SEG HEAD	9
75	6	PROC HD1	9
76	12	PROC HD2	9
77	1	PROC HD3	9
78	4	PROC HD4	9
79	1	PROC HD5	9
80	2	PROC HD6	9
81	1	PROC HD2D	9
82	1	DECL	8
83	8	LABEL DEF	1
84	8	BLOCKHEAD	3
85	1	BLOCKBODY	3
86	1	PROGRAM	1
87	14	IDI	7
88	11	IDT	6
89	6	IDS	6
90	12	IDX	6
91	13	IDF	3
92	5	IDD	2
93	13	IDP	6
94	12	IDB	4
95	5	IDL	6
96	2	EXTERNAL-BASE-ID	4
97	12	)	4
98	8	(	12
99	8	+	5
100	8	-	5
101	8	*	5
102	8	/	5
103	8	++	5
104	8	--	5

fig. 12 (2)

105	8	.LT.	10
106	8	.EQ.	10
107	8	.GT.	10
108	8	.LE.	10
109	8	.GE.	10
110	8	.NE.	10
111	8	AND	5
112	8	OR	5
113	8	XOR	5
114	7	SHLA	5
115	7	SHRA	5
116	7	SHLL	5
117	7	SHRL	5
118	8	ABS	6
119	8	NEG	6
120	8	NEGABS	6
121	5	=	11
122	5	A	5
123	8	,	4
124	5	((	6
125	12	))	4
126	5	CASE	3
127	3	OF	9
128	11	BEGIN	3
129	11	*	1
130	3	NULL	3
131	6	GOTO	3
132	3	END	1
133	8	.NOT.	3
134	6	OVERFLOW	3
135	8	THEN	5
136	8	ELSE	2
137	8	WHILE	3
138	8	DO	5
139	5	STEP	3
140	5	UNTIL	2
141	1	IF	3
142	1	FOR	3
143	10	SHORT	10
144	9	INTEGER	10
145	9	LOGICAL	10
146	9	REAL	10
147	10	LONG	10
148	9	BYTE	10
149	9	CHARACTER	10
150	9	LABEL	10
151	5	ARRAY	9
152	8	FUNCTION	9
153	8	SYN	13
154	6	REGISTER	8
155	6	ADCON	8
156	6	SLCON	8
157	10	SEGMENT	9
158	8	PROCEDURE	9

fig. 12 (3)

159	3	BASE	9
160	2	LOCAl	3
161	8	..	13
162	1	.	1

fig. 12 (4)