Gammon & Gobbett
Computer Services Pty Ltd

# G-PASCAL

HIGH-SPEED PASCAL SUBSET COMPILER FOR APPLE II

Version 1.1 User Manual April 1982

* Compiles at over 6,000 lines per minute
* Extra memory not needed
* Available on cassette
* Can be loaded into language card
* Easy to use
* Only uses 12K of memory for compiler, interpreter and text editor (all one program)
* Implements: PROCEDURE, FUNCTION, WHILE, REPEAT, FOR, IF, CASE, CALL, BEGIN ... END, peeking, pokeing, random numbers, hi-res graphics, music generation, cursor positioning and more!

Distributed by: Gambit Games
(A division of Gammon & Gobbett Computer Services)
P.O. Box 124, Ivanhoe, Victoria 3079 (Australia)

## GETTING STARTED

1. Back up your G-Pascal onto another disk or cassette, just in case! (See 'Backing up your G-Pascal').

2. Load and run G-Pascal:

   a) Disk users type: BRUN GPASCAL

   b) Disk users with 16K RAMCARD type:
   RUN RAMCARD GPASCAL

   c) Cassette users type:
   CALL -151
   800.37FFR (when cassette starts)
   800G (when finished loading)
   (For more details see 'Loading G-Pascal from Cassette')

3. You will now see the question (if all is well):

   DISPLAY LOWER CASE? Y/N

   Type: Y if you have the 'lower case adapter' installed otherwise press any other key (such as the space bar).

4. Your G-Pascal is now running! You will see:

```
G-Pascal compiler Version 1.1 Ser# 0001
Written by Nick Gammon and Sue Gobbett
Copyright 1982 Gambit Games
P.O. Box 124 Ivanhoe 3079 Vic Australia
<E>dit, <C>ompile, <D>ebug, <F>iles,
<R>un, <S>yntax, <T>race, <Q>uit ?
```

Turn the page to see what to do now ...

## HOW TO USE G-PASCAL

After loading and running G-Pascal (see previous page), or after a compile or run, you will see the 'Main Menu' :

<E>dit, <C>ompile, <D>ebug, <F>iles,
<R>un, <S>yntax, <T>race, <Q>uit ?

It is called a menu because you have various choices to make, depending on what you want.

To choose one, press the letter corresponding to your choice (the letter in angled brackets). For example press C for Compile. Do not press the RETURN key. If you make an incorrect choice the Apple will 'beep' and re-display the Main Menu.

The choices are:

### Edit

Enters the Editor to allow you to type in or change a Pascal program. When the Editor is active it will display a colon (:) to let you know it is awaiting an Editor command. See 'Using the Editor' for more details.

### Compile

Compiles a program (that is, converts it to P-codes) that you have loaded or edited. If the compile has no errors you may then type R (for Run) if you want to run your program. (For long programs if you type 'R' while the program is compiling it will go directly to Run as soon as the compile has finished - if the program has no errors, otherwise the 'R' is ignored.) A program must be compiled before it can be run. See 'Compiling your program' for more details.

(more on next page)

## HOW TO USE G-PASCAL (continued)

### Syntax

This does a 'syntax check' of your program, letting you know if it has any errors. It does not generate P-codes however so you must do a compile as well before running the program. See 'Compiling your program' for more details.

### Run

This will run your program. It must have been successfully compiled first. G-Pascal will display:

**Running**

### Debug and Trace

These are a variation of the 'Run' command which, in addition to running your program, display debugging information as your program runs. You will not normally choose these options as the information they display clutters up the screen somewhat and slows down execution of the program considerably. See 'Debugging your program' for more details.

### Files

Enters the 'Files Menu' to allow you to load or save programs, turn on or off your printer, and use the Apple DOS to delete files, CATALOG etc. See 'File Handling' for more details.

### Quit

Leaves G-Pascal, but first checks that you wanted to quit by asking:

**Quit? Y/N**

If you do not type 'Y' the quit has no effect.

## SPECIAL KEYBOARD FUNCTIONS

The following keys have special meaning to G-Pascal. The notation CTRL/C (for example) refers to pressing 'C' whilst holding down the CTRL key.

### Space bar

Pressing 'space' will freeze a display on the screen (similar to CTRL/S in Basic). The words: '{PRESS ANY KEY TO CONTINUE}' will appear on the bottom line of your screen. Press any key (except CTRL/C) and the display will continue. CTRL/C will abort the display.

### CTRL/S

This will freeze the display at the end of the current line. No warning message appears. Press any key to continue.

### CTRL/C

This will abort a display, or if a program is running, will abort the program with a message showing which P-code was currently being executed.

### CTRL/T

If a program is running will cause a Trace to start. (See 'Debugging your program' for more details).

### CTRL/D

If a program is running will cause Debug mode to start. (See 'Debugging your program' for more details).

### CTRL/N

If a program is running will stop a Trace or Debug. (N = Normal). If the program is not Tracing or Debugging pressing CTRL/N will have no effect.

### Arrow keys

The forwards and backwards arrow keys function the same as in Basic - useful when in the Editor or when typing in a file name to Load or Save.

## SPECIAL KEYBOARD FUNCTIONS (continued)

### Reset

Pressing 'Reset' will take you back to the Main Menu. If you do not have the 'Autostart ROM' you will have to press CTRL/Y (return) after pressing RESET. Pressing RESET will not do much harm normally however do not press RESET while writing (saving) to your diskette. Also, if you press RESET while the Editor is in the middle of inserting, modifying or deleting your program might be corrupted. Other than that, RESET should be perfectly safe. Note that CTRL/C will usually have the same effect as RESET and it is safer to use.

### CTRL/Y

If you have 'Quit' from G-Pascal and are in the Apple Monitor (with an * showing) then pressing CTRL/Y will return you to the Main Menu with your program (if any) intact.

## 16K RAMCARD USERS ...

If you have loaded G-Pascal into your RAMCARD then it should stay there indefinitely (unless you load something else into the RAMCARD). As G-Pascal is write-protected it should not normally be damaged by rampant programs or other means. If you have left G-Pascal and want to go back to it, type:

```
CALL -151     (to enter the Apple Monitor)
C080          (to enable the RAMCARD)
CTRL/B        (to go to G-Pascal)
```

(Note: CTRL/B does a 'coldstart' which removes any G-Pascal program that might be in memory. Alternatively type CTRL/C instead of CTRL/B if there is a program that you want to keep).

## LEAVING G-PASCAL

This section only applies to disk users.

Once you have 'Quit' from G-Pascal and returned to Basic or the Apple Monitor you should immediately do a 'disk boot'. This is because G-Pascal modifies DOS pointers and DOS will not be fully operational. Various important pointers (in particular HIMEM) may not be set up correctly.

We repeat, after you have finished using G-Pascal do a disk boot.

## USING THE EDITOR

The Editor displays a colon (:) when it is ready for a commano, so if you see a : you know you are in the Editor. To leave the Editor enter: Q (return) and you will return to the Main Menu.

### Line numbers

The Editor is line-number oriented. This means that all Editor commands refer to the line-numbers of each line in your program. The line numbers are allocated automatically by the Editor, starting at 1 and going up by 1 each line. The line number allocation is 'dynamic', that is, if you insert a line between what is currently line 6 and line 7 then the new line becomes line 7 and the line that used to be line 7 becomes line 8, etc.

### Commands

All Editor commands consist of one letter followed by none, one or possibly two line numbers. For example:

```
L  10, 20
```

would List lines 10 to 20.

The commands are :

```
<C>ompile
<D>elete Line Number Range
<I>nsert Line Number
<L>ist   Line Number Range
<M>odify Line Number Range
<Q>uit
<S>yntax
```

(The above command summary appears if you just type 'RETURN' when in the Editor).
The commands are explained in detail on the following pages.

## EDITOR COMMANDS

### Delete

Deletes one or more lines. Any lines following those deleted are renumbered.

To delete one line:

D line-number e.g. D 5 would delete line 5.

To delete a range of lines:

D first-line, last-line e.g. D 10, 20
would delete lines 10 to 20.

If you attempt to delete 5 or more lines the Editor will ask you (for example):

Do you want to delete 200 lines ? Y/N

This is in case you meant to delete 20 lines but accidentally deleted 200 lines. If you want the delete to go ahead press 'Y' otherwise press any other key. Do not press 'RETURN'.

### Insert

Inserts new lines into your program. To insert, enter:

I line-number

where 'line-number' is the line that you want to insert after. To stop inserting just type 'RETURN' at the very start of a line. (Therefore, to enter blank lines into your program type at least one space before 'RETURN'). For example: I 10 inserts after line 10 (starting at line 11).
The Editor reminds you what line you are currently inserting by displaying its line number at the start of the line. To put a line at the very start of the program (line 1) enter: I 0 (insert after line 0). To put a line at the end of the program enter: I 9999 (or some big number).

EDITOR COMMANDS (continued)

List

Lists your program.
To list the whole program just type: L
To list one line, just type: L line-number
To list a range of lines, just type: L first-line, last-line

e. g. L 10 will list line number 10
L 40, 60 will list lines 40 to 60.

You can temporarily 'freeze' your list by pressing the space bar. You can stop the listing altogether by pressing CTRL/C (press C while holding down the CTRL key).

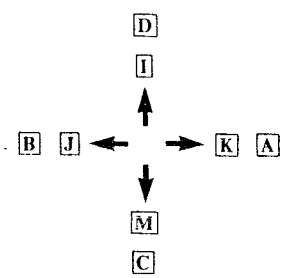
Modify

Allows you to change one or more lines in your program.
To change one line, just type: M line-number
To change a range of lines : M first-line, last-line

Modify works by:

a) Listing the line or lines requested.
b) Deleting the lines requested.
c) Entering Insert mode so that you can replace them.

Like the Delete command, Modify will warn you if you are about to change more than 4 lines.
The intention is that for making minor changes to small numbers of lines you use the 'cursor control' escape codes to move the cursor over the erroneous lines, making corrections where necessary (see below).
(These are the same codes you use to edit a Basic program. See Apple II Reference Manual for more details about them.)



## EDITOR COMMANDS (continued)

### Compile

This starts a compile of your program. It is the same as entering 'C' from the 'Main Menu'. See 'Compiling your program' for more details.

### Syntax

This does a 'syntax check' of your program (to let you know if it has errors). It is the same as entering 'S' from the 'Main Menu'. See 'Compiling your program' for more details.

### Quit

This leaves the Editor (quits) returning you to the 'Main Menu'. You will need to return to the Main Menu in order to save your program to disk or cassette etc.

## NOTES ON THE EDITOR

### Control characters

For your protection, 'control' characters (such as CTRL/C, CTRL/D etc.) are converted to a '?' by the Editor. You will not see this happening when you type the line in, but you will see the '?' characters when you List the program. This is done so that invisible control characters typed in inadvertently do not cause puzzling syntax errors.

### Lower Case

Unless you are running G-Pascal from the 16K RAMCARD you will be unable to enter lower-case characters, even if you have the lower-case adapter installed. This is because the Editor uses the Apple Monitor's 'get line' routine (GETLN) which automatically converts lower-case to upper-case. However since users who have a RAMCARD are only using a copy of the Monitor we are able to disable this 'feature' in the copy, thereby allowing you to enter lower-case.

## COMPILING YOUR PROGRAM

### Compile

If you select 'Compile' from the Main Menu or Editor you will see:

```
Do you want P-codes listed ? Y/N
```

Type 'Y' if you want to see a listing of every P-code the compiler generates (you wouldn't, normally), otherwise press any other key.

If you do not type 'Y' you will then see:

```
Do you want a listing ? Y/N
```

Type 'Y' if you want a listing of your program (with P-code addresses on the left in brackets) otherwise press any other key.

### Syntax

The 'Syntax' option is a quick way of checking for errors in your program - quick because the above two questions are not asked. The 'Syntax' option defaults to 'no listing' and also does not generate P-codes so you must do a Compile also if you have no errors and wish to run your program. In all other respects Syntax and Compile are identical.

### Asterisks

If you do not request a listing an '*' is displayed on the screen as every 32 source program lines are compiled. This is to reassure you that 'something is happening', and give a visual indication of how fast (and how far) the compilation is proceeding.

### Errors

If there is an error in your program the compilation will stop with an arrow pointing to the symbol being processed when the error was detected (this not necessarily being the actual cause of the error as such), the word 'Error', an English error message, and two 'beeps' to wake you up.

## ERROR MESSAGES

To G-Pascal all errors are fatal. In other words you only get one (or no) errors per compile. Since compilation is so fast this is not a big disadvantage but it does mean that you have to correct errors one by one. Fortunately this process is made very easy because:

a) If you get an error you are immediately returned to the Editor so you can correct it on the spot.

b) Having corrected the error just type 'S' followed by the 'RETURN' key to do another 'Syntax check'.

Here is an example of an error:

```
(Ø81F) 1: begin
(Ø822) 2:       write (FRED #);
*** Error             ^
            Undeclared Identifier

:
```

This example demonstrates a couple of important points:

a) The upward arrow (carat) points to the word or symbol being processed when the error was detected.

b) The error is not necessarily the word or symbol being pointed to - this was only the spot where it was detected. (In the example the real error was a missing VAR declaration for FRED).

In some cases the real error may be a long way from where the error condition is detected, so it is necessary to approach error messages with a broad mind. For example accidentally putting in one too many or one too few BEGINs or ENDs may result in the compiler getting to the very end of your program before noticing that there is a mismatched BEGIN/END condition. In this case you just have to look at your program carefully. This task is enormously simplified if you always indent BEGINs and ENDs and always have an END directly under its corresponding BEGIN. If you can't understand an error message refer to the syntax diagrams to see what sequence of words or symbols the compiler expected.

## MACHINE LANGUAGE SUBROUTINES

The novice user should skip this section.

Machine-language subroutines may be called by the: CALL ( address ) construct. Remember that hexadecimal constants must be preceded by a '$'. For example, to read a game paddle:

```
MEMC ( $343 ) := 0;  (* X-REGISTER = 0 FOR PADDLE 0  *)
CALL ( $FB1E );      (* READ PADDLE *)
RESULT := MEMC ( $344 );  (* RESULT IS IN Y-REGISTER  *)
```

As the example illustrates you can set up the A, X, Y registers (and condition codes) before the CALL and examine their results after the call. (See table below).

Warning:
The subroutine (if you write one yourself) should not change addresses used by G-Pascal or unpredictable results will occur. G-Pascal uses zero page registers: $50 to $65, $71 to $76, $7B to $A8 as well as those used by the Monitor ROM. G-Pascal also uses addresses $300 to $372.

Addresses of interest to machine-language programmers:

| Address | Meaning |
|---|---|
| $5C/$5D | Current runtime stack top (last used location) |
| $5E/$5F | Current stack frame base |
| $54/$55 | Current P-code address |
| $342 | A-register save/restore for CALL instruction |
| $343 | X-register save/restore for CALL instruction |
| $344 | Y-register save/restore for CALL instruction |
| $341 | Condition codes " for CALL instruction |
| $319/$31A | First free location past P-codes. (End of program + 1). Can be used for temporary data or machine code subroutines. (Note: The pair of locations $319/31A contain the address of the first free location in the normal order: low-order byte, high-order byte. Do not put your subroutine at address $319, rather put it at the address contained in $319/$31A.) |

## MEANINGS OF P-CODES

Here are what the P-codes mean. You will not normally need this information.

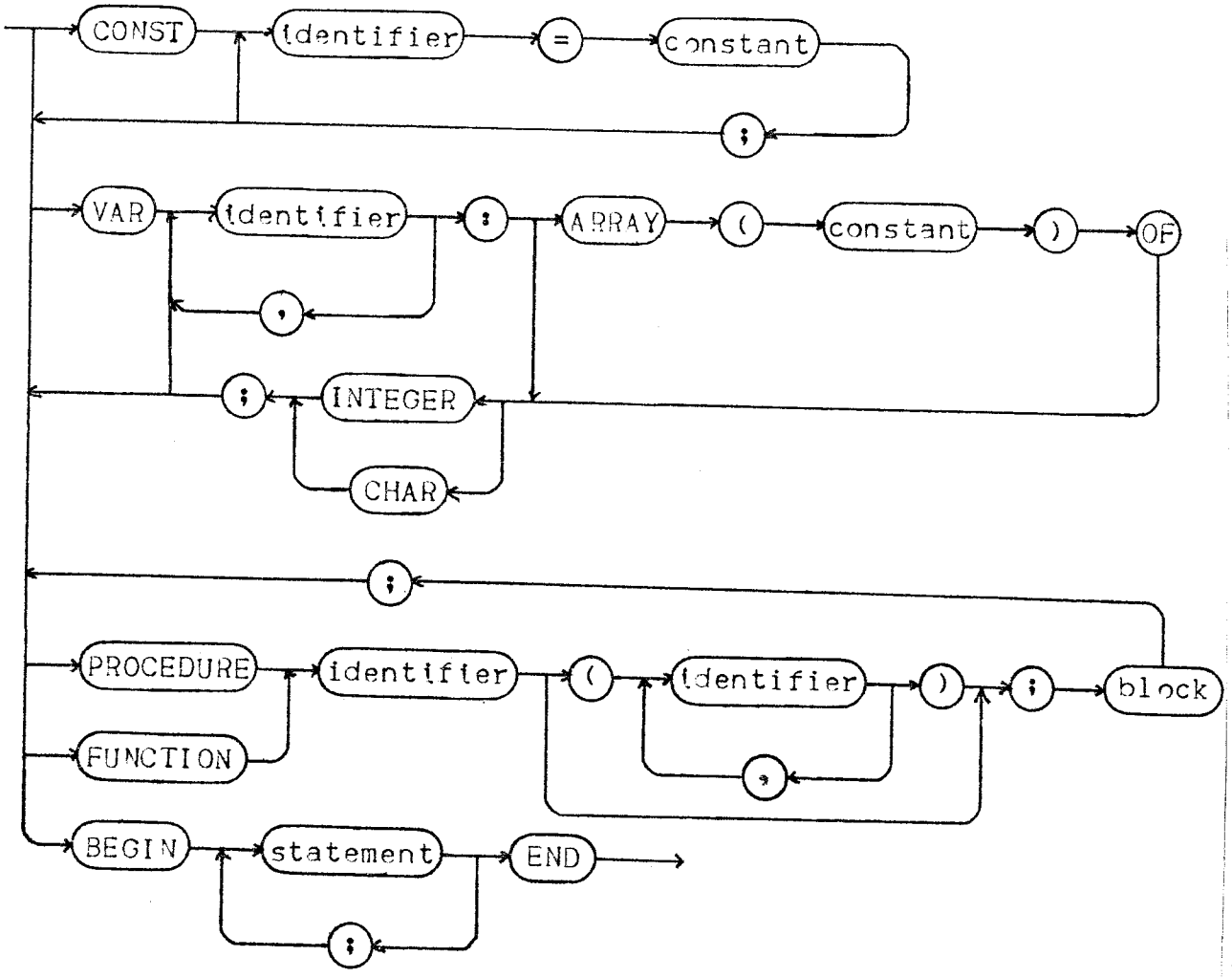
| P-code | | | |
|---|---|---|---|
| Decimal | Hex | Operand/s | Function |
| 0 | 00 | 3-byte literal | Load literal |
| 1 | 01 | none | Set hi-res graphics |
| 2 | 02 | " | Negate (sp) |
| 3 | 03 | " | HPLOT |
| 4 | 04 | " | Add (sp) to (sp - 1) |
| 5 | 05 | " | TOHPLOT (same as HPLOT) |
| 6 | 06 | " | Subtract (sp) from (sp - 1) |
| 7 | 07 | " | Music |
| 8 | 08 | " | Multiply (sp) * (sp - 1) |
| 9 | 09 | " | TEXT |
| 10 | 0A | " | Divide (sp - 1) / (sp) |
| 11 | 0B | " | MOD (sp - 1) MOD (sp) |
| 12 | 0C | level,displ | Address of integer |
| 13 | 0D | level,displ | Address of character |
| 14 | 0E | level,displ | Address of integer array |
| 15 | 0F | level,displ | Address of char array |
| 16 | 10 | none | Test (sp - 1) = (sp) |
| 17 | 11 | " | Stop run - (end of program) |
| 18 | 12 | " | Test (sp - 1) not = (sp) |
| 19 | 13 | " | Cursor position |
| 20 | 14 | " | Test (sp - 1) LSS (sp) |
| 21 | 15 | " | Random number generator |
| 22 | 16 | " | Test (sp - 1) GEQ (sp) |
| 23 | 17 | " | Input hex number |
| 24 | 18 | " | Test (sp - 1) GTR (sp) |
| 25 | 19 | " | Test (sp - 1) LEQ (sp) |
| 26 | 1A | " | Or (sp - 1) with (sp) |
| 27 | 1B | " | And (sp - 1) with (sp) |
| 28 | 1C | " | Input number |
| 29 | 1D | " | Input character |

MEANING OF P-CODES (continued)

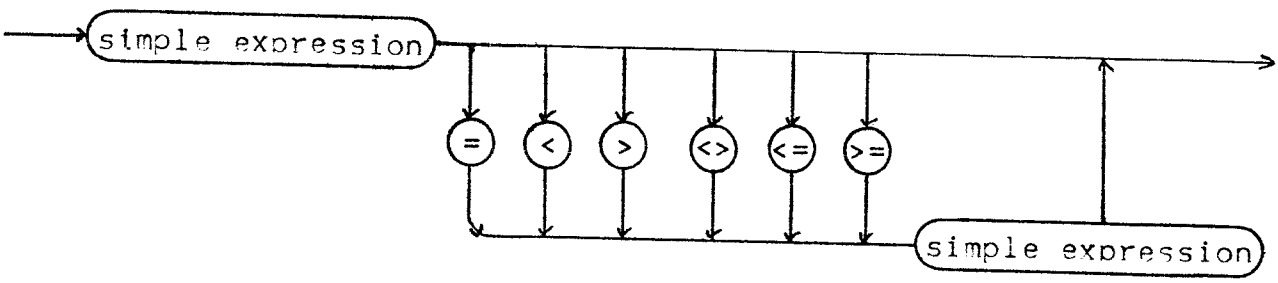
| P-code | | | |
|---|---|---|---|
| Decimal | Hex | Operand/s | Function |
| 30 | 1E | none | Output number |
| 31 | 1F | " | Output a character |
| 32 | 20 | " | Not (sp) (Reverse true/f) |
| 33 | 21 | " | Output hex number |
| 34 | 22 | " | Shift left (sp) bits |
| 35 | 23 | length,string | Output string |
| 36 | 24 | none | Shift right (sp) bits |
| 37 | 25 | max,level,displ | Input string into array |
| 38 | 26 | none | Input character |
| 39 | 27 | level,displ | Procedure/function call |
| 40 | 28 | none | Decrement (sp) by 1 |
| 41 | 29 | " | Procedure/function return |
| 42 | 2A | " | Copy (sp) to (sp + 1) |
| 43 | 2B | " | Call absolute address |
| 44 | 2C | level,displ | Load integer onto stack |
| 45 | 2D | " | Load character " |
| 46 | 2E | none | Load absolute address int. |
| 47 | 2F | " | Load absolute address char |
| 48 | 30 | level,displ | Load integer indexed |
| 49 | 31 | " | Load character indexed |
| 50 | 32 | " | Store integer |
| 51 | 33 | " | Store character |
| 52 | 34 | none | Store integer absolute |
| 53 | 35 | " | Store char. absolute |
| 54 | 36 | level,displ | Store integer indexed |
| 55 | 37 | " | Store character indexed |
| 56/57/58 | ---------- | | Not implemented |
| 59 | 3B | increment | Increment stack pointer |
| 60 | 3C | address | Jump unconditionally |
| 61 | 3D | " | Jump if (sp) zero |
| 62 | 3E | " | Jump if (sp) not zero |
| 63 to 127 | ---------- | | Not implemented |
| 128 to 255 | (Hex 80 to FF) | | Load literal: P-code - 128 |

program

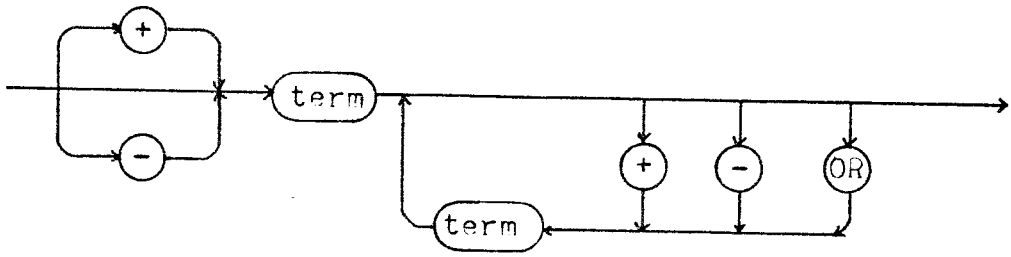

block



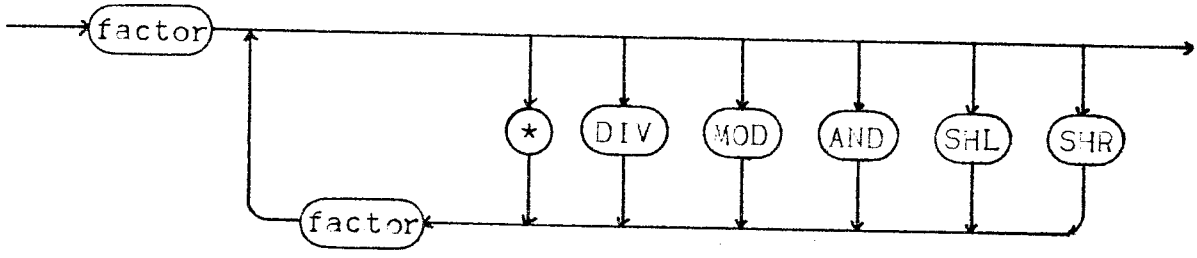

expression



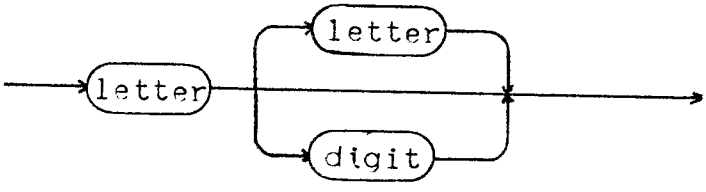
simple expression



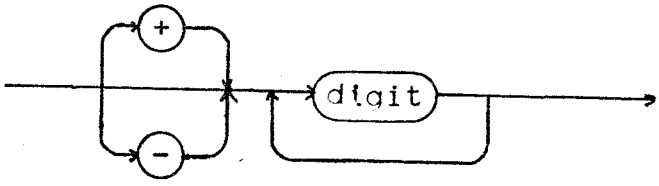

term



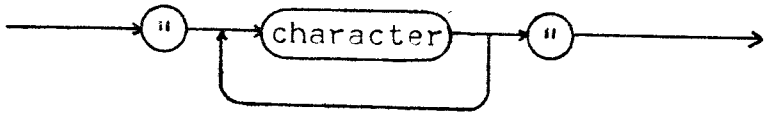

identifier



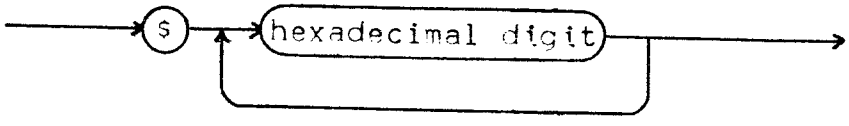

integer




string




hexinteger



statement