P E G A S U S P A S C A L ver 1 . 0

PASCAL USERS NOTE

While this version of Pascal is designed to run with a standard 4K RAM Pegasus, the memory size limits the number of lines of program that may be successfully compiled.

In normal programs this allows in excess of 50 lines of source which is adequate for small programs and most teaching applications.

However, for large programs it is recommended that the 64K RAM expansion be fitted to the Pegasus.

PEGASUS PASCAL VERSION 1 USER MANUAL

1. The PASCAL Language

INTRODUCTION

Any algorithm for a computer consists of two complementary parts - a description of the actions to be performed, and a description of the data on which the actions are to be performed. The aim of the language Pascal, designed by Professor Niklaus Wirth in 1968, is to provide a set of statements (action) and data types which are at the same time comprehensive (to cover all eventualities) and succinct (to convey the meaning without getting too complicated). Other languages such as BASIC allow for actions and data types too, so any comparison is not so much about what they can do but more how elegantly and efficiently that they do it.

DATA TYPES

Pegasus Pascal has three standard data types:

BOOLEAN	can be TRUE or FALSE
INTEGER	16 bit integer range -32768 to 32767
CHAR	a character from the ASCII character set

(REAL numbers are not available with version 1 of Pegasus Pascal.)

Pascal allows you to define a new data type to suit your own application. For instance,

```
TYPE DAYS=(SUN,MON,TUES,WED,THUR,FRI,SAT);
TYPE SUIT=(HEARTS,CLUBS,DIAMONDS,SPADES);
```

Types DAYS and SUIT are known as scalar types, or enumeration types, which means that all the elements of that type are explicitly defined by the user when the type definition is compiled.

CONSTANTS AND VARIABLES

The difference between constants and variables is important. A constant is a 16-bit integer value that never changes during program execution. The value assigned to the constant

identifier must be either an integer, one or two characters enclosed in single quote marks ('), or a previously defined constant. Note that Pegasus Pascal supports automatic type conversion, so that a character assigned to a constant identifier will always be converted to its integer equivalent, and must be assigned to a character variable before it can be output as a character.

A variable is a value that can be change - internally what a variable does is point to the storage locations in memory that are used whenever the variable is invoked. Special variables such as structures simply have a slightly more complex way of pointing to the appropriate storage, for instance an indexed array like A[I] means that the beginning of the array is pointed to by A, while I provides the index into the exact element required.

TYPED VARIABLES

Standard types or user defined types may be applied to variable or functions. When a variable is typed, it means that the range of possible values able to be stored in that variable can be absolutely known by the user and the computer. For instance :

```
TYPE SUIT = (HEARTS,CLUBS,DIAMONDS,SPADES);
VAR  CARD :  SUIT;
```

defines a variable of type SUIT with an identifier of CARD, which can only contain the four values of HEARTS, CLUBS, DIAMONDS, SPADES.

Another way of defining a variable's values is to use a subrange types with two elements of a previously defined type separated by the '..' symbol.

```
TYPE DAY = 1..31;
TYPF WEEKDAY = MON..FRI;
```

Note that the days making up WEEKDAY must have already been defined. The standard type BOOLEAN results from the evaluation of relational expressions, and may take the values of TRUE or FALSE. This type is predeclared as:

```
TYPE BOOLEAN = (FALSE,TRUE);
```

Type definitions must always appear before the variable definitions of program body in which they are used, and are usually declared after the constants have been defined.

STRUCTURED TYPES

Standard Pascal offers a number of structured data types, however, Pegasus Pascal version 1 implements only two of these, which are the array and the pointer type. An ARRAY contains a fixed number of components which are all of the same component type. The component type itself can be any simple type (e.g. CHAR, INTEGER, subrange etc). The index type must be either a scalar or subrange type - that is INTEGER and CHAR are specifically excluded. The cardinality of the index type defines the size of the array, or the number of elements.

```
VAR STRING:   ARRAY [1..10] OF CHAR;
    WEEK:     ARRAY [DAYS]  OF INTEGER;
    MONTH:    ARRAY [DAY ]  OF DAYS;
```

The above declaration defines three arrays. The first is an array of ten characters, indexed from 1 to 10. The second is an array of seven integers, indexed by DAYS, while the third is an array of DAYS, indexed by the subrange DAY (1..31). ARRAYs in Pegasus Pascal allow only one index, however, multiple array indices may be simulated by using a procedure or function to take the multiple indices and calculate the element required, e.g. for a 10 x 10 array, element (I,J) can be (I*10+J). Note that Pascal uses the square brackets to enclose array index indentifiers.

The pointer type requires special explanation. In Pascal there is a class of data known as dynamic variables that are not created automatically on entry to a block like ordinary static variables, but are created instead by a standard procedure NEW. They cannot be referenced directly by name since they do not appear in any explicit variable declaration. Instead they are referred to by a pointer variable, which is initialized by procedure NEW when they are allocated. The pointer itself is declared explicitly like any other variable an is said to be bound to the type of the referenced dynamic variable. A pointer variable PTR is bound to a type TYP by the declaration.

```
VAR PTR:^TYP;
```

The statement NEW(PTR) creates a new variable of type TYP, and sets the pointer PTR to point to this variable, which may then be referred to as variable PTR^. Dynamic variables are allocated in an area known as the heap, which starts at the high end of read write memory (RAM) and grown downwards to the Pascal stack (which starts at the end of the workspace and grows upward.) It is possible to create many variables of the same type in the heap, so long as one is careful to somehow retain pointers to all of them. When creating a list of identical variables on the heap, it is convenient to make each variable an ARRAY and set aside one element to store a pointer, or link to the previously allocated one. The backward link in the first item should be assigned the special pointer value NIL, which points to nothing at all. There are two additional statements connected with the management of space on the heap, being MARK(P), which sets the pointer variable P equal to the current heap pointer and RELEASE(P), which sets the heap pointer equal to pointer P. Since the type rules of this Pascal have been substantially relaxed, this means that a pointer variable may be forced to point to a given area in memory simply by assigning it the address required, e.g. if P is a pointer variable, then assigning

P:=256; will force it to point to the word at memory address $0100, which can then be accessed as P^.

IDENTIFIERS

Identifiers in Pegasus Pascal are user defined names which denote constants, variables, types, procedures, functions, of which only the first four characters have to be unique for distinction.

COMMENTS

Pascal programs may contain comments, which must be enclosed with the symbols (* and *). Comments may appear anywhere that a blank may appear, except embedded in a character string literal. The compiler listing option is controlled by a special form of comment, where (*$L-*) stops the listing, and (*$L+*) starts it.

CHARACTER STRINGS

String literals may be used for output in the WRITE statement enclosed in single quotes, (') and may be up to 80 characters in length. They may also be used to assign their values to constants or variables, in which case only the first two characters are used. Note that an array of characters may not be initialized by assigning it a character string, such as in BASIC.

PASCAL PROGRAMS

A Pascal program consists of a single program block which may contain one or more procedure or function blocks (which may themselves contain other nested blocks). The maximum depth of nesting blocks is seven levels. The program block is preceded by a program heading consisting of the keyword PROGRAM followed by an arbitrary program identifier, a semi-colon, then comes the program block and lastly the period or full stop, which appears at the very end of all Pascal programs.

```
PROGRAM FRED;
(*Program block goes here*)
(*Program ends with a period*)
```

Each block consists of a declaration part and a statement part. The declaration part associates identifiers with various entities, and these identifiers may only be referenced within the block in which they are declared. The portion of program text contained within the block, and within which that block's identifiers are known, is called the scope of those identifiers. This scope includes any procedure or function blocks which are imbedded, or nested in that block, but does not include the text of any of the blocks that surround it. Identifiers are said to be local to a block in which they are declared. Identifiers declared in the outermost block (i.e. the program block) are known throughout the entire program, and are known as global identifiers. It is possible to redeclare identifiers in inner blocks, in which case the new declarations overide or take precedence over the original outer declarations, but only within the scope of the new declarations.

The declaration part consists of four optional sub-sections which declare all symbolic constants, types, variables, procedures and functions which will be used later in the statement part. The declaration parts, if used, must appear in the order that they are given here.

CONSTANTS

The constant definition part declares identifiers which may be used as synonyms for constants. The constant values to be associated with each identifier may be integers, strings of one or two characters, or other previously declared constants.

E.g.
```
CONST SIZE=230;
      LENGTH=-21;
      ZERO=0
```

TYPE DECLARATIONS

New user defined scalar types may be enumerated and associated with a type identifier, as well as subrange types, pointer types and ARRAY types.

```
TYPE COLOUR=(BLACK,WHITE,RED,BLUE,YELLOW,GREEN);
     ALPHABET='A'..'Z';
     FLAGCOLOURS=WHITE..BLUE; (*Includes RED*)
     FAHRENHEIT = 32..212;
     MAGIC= ARRAY 1..31  OF ALPHABET;
     PTRTYPE =  ITEM;
```

VARIABLES

Variable declarations define the actual data entities that will be used in a block. Each variable is given an identifier (or name) and a type. Note that variables are not initialized when the program is interpreted.

E.g. VAR I,J,K:INTEGER; A,B:CHAR;

PROCEDURE AND FUNCTION DECLARATION

This section introduces subroutines and associates identifiers with them so that they may be invoked by procedure statements and function references. PROCEDURES are invoked by the procedure statement and functions, which always return a result value, are activated by function references in expressions.

As in standard Pascal, parameter lists in procedure declarations must indicate the type of all parameters, and all functions must be given a type. The compiler however, treats all parameters and functions as integers. This means that you cannot pass an array or other data structure as a parameter, however it does mean that you can pass a pointer to the array or structure as a parameter. Parameters are always passed by value, which means that the parameter passed cannot have its value changed in the scope of the procedure or function being invoked.

External routines may be invoked by substituting the keyword

EXTERN for the block which normally follows each procedure and function header, followed by the decimal address of the machine code routine enclosed in parentheses. (See section F on machine language routines). Procedures may also be declared as being forward referenced, by substituting the keyword FORWARD instead of the procedure block following the header, and then repeating the procedure declaration later in the program, including the actual block the second time. This mechanism exists to allow for recursive programming techniques, as well as mutual recursion. The procedure headings of the forward declaration and the actual declaration should be identical, and they should be nested in the same parent block.

THE STATEMENT BLOCK

The statement block, which must follow all the declarations, contains the statements describing the actions to occur with the data described by the parent declaration parts. This must always begin with the keyword.

```
BEGIN
    (* and end with the keyword *)
END
```

and contain any number of individual statements separated by semicolons (;). Note that the semicolon is not a statement terminator, but a statement separator. There are several kinds of statements which are detailed below.

The assignment statement is the most fundamental of all statements. It evaluates an expression consisting of variables, constants and operators, and assigns the resulting data value to a variable on the left of the assignment operators (:=), which is pronounced as "is assigned the value of", or "becomes". The destination variable may be a simple variable, an array element, or a function identifier (functions return a value to the calling expression by assigning a value to a variable with the same name as the function, within the

function scope.)
An expression is a combination of sub-expression forms known as simple expressions, terms, and factors. Expressions consist of operators and operands, where the operators are separated into four precedence classes. (Note that a single variable or constant by itself forms an expression with a nul operator.) The highest precedence is assigned to the operator NOT, which complements all 16 bits of the operand following it. (If the operand is boolean, it negates it logically, i.e. true becomes false and false becomes true.) The second highest precedence is assigned to the "multiplying operators", * (multiply), DIV (divide), MOD (remainder) and AND (bit by bit).
The third level is the adding operators, + (addition), - (subtraction), and OR (bit by bit). The lowest precedence is assigned to the relational operators, = (equality), <> (inequality), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to), all of which yield a boolean result.
All expressions are considered to be of type INTEGER, but their operands may be any mixture of the types INTEGER, CHAR, BOOLEAN, scalar, subrange, pointer, function or array elements, and they may be assigned to any of these types. (Note that a value may be assigned to a function only within the function block.) In other words, the compiler does no type checking except in READ or WRITE statements. When writing a value of type INTEGER, a number is produced, but when writing type CHAR, a character string will be produced.

The procedure statement causes the named procedure to be executed, and may contain a list of arguments (or parameters) consisting of expressions to be passed to the procedure. Full recursion is permitted, limited only by the available stack space in memory.

The compound statement is simply a group of statements bracketed by the words BEGIN and END , where semicolons are used to seperate each statement in the group, and where the final semicolon before the END is not required.

Hereis an example of a block statement, as used in an IF statement:

```
IF (I=3)  OR (K 7) THEN
BEGIN
  I:=21;
  START(I,K);  (* Use of a procedure *)
  WRITELN(I:4)
END ELSE
BEGIN
  WRITELN('No match')
END;
```

Note from this example that spreading the statement over a number of lines makes no difference to its meaning - it only makes the statement easier to read. Note also that since this is a statement, it requires a semicolon after the last END, (unless of course there is another END following). If an END is the last one in the program, then it is usual to write it with the necessary period immediately following, e.g.

```
  END.
```

The IF statement illustrated above evaluates the boolean expression after the word IF, and then executes either the statement after the word THEN if TRUE, of the statement after the word ELSE if FALSE. Nested conditionals are allowed but care must be taken.

The CASE statementconsists of an expression known as the selector, and a list of statements, each labelled by a constant. After the expression is evaluated, the statement whose label is equal to the expression is executed, and then control passes to the statement following the terminating END.

If no label matches the expression, and if the OTHERWISE clause is included, then the statements between the OTHERWISE and its END are executed, otherwise control passes directly to the statement after the END which corresponds with the CASE keyword.

```
CASE I OF
1: WRITELN('1 is the number');
2: BEGIN WRITE('2 '); WRITELN('is it.) END;
3: WRITELN('3 takes the cake)
OTHERWISE WRITELN('Its not 1,2,3,')
END;
```

The REPEAT statement executes the statements between the word REPEAT and UNTIL at least once, and then repeats them until the boolean expression after the UNTIL becomes true.

The WHILE statement evaluates the boolean expression following the word WHILE, and if it is true, it repeatedly executes the statement following the word DO until the expression becomes false. If the expression is false to begin with, the statement is not executed at all.

The FOR statement provides for repeated execution of a statement a fixed number of times while a progression of values is assigned to the variable known as the control variable. It contains two expressions: the first defines the initial value to be assigned to the control variable, the second defines the final value. For each iteration of the statement the control variable is either incremented (if the middle keyword is TO) or decremented (if the middle keyword is DOWNTO) by 1. The sequence is iterated until the control variable is equal to the final value (on the last iteration). If the initial value is greater than (TO) or less than (DOWNTO) the final value, then the statement is not executed at all. Note that any scalar type is allowed as a control variable, not just INTEGER.

The READ statement is followed by a list of parentheses of input elements separated by commas. Each input element can be:

1) A CHAR variable of element of a CHAR array, in which case a single character is read from the keyboard, or
2) An INTEGER, subrange or scalar variable or array element, in which case a decimal number is read and converted to 16 bit binary.

Leading blanks are not permitted in the input data, and the number is terminated by the first character which is not a digit or minus sign (the terminating character would normally be a comma, space, or RETURN key). A number consisting only of a terminator would be read as zero. The range for acceptable integer input is -32768..32767. It is not possible to READ a function or pointer value directly.

The WRITE or WRITELN statement is followed by a list of output elements, each of which may be:

1) A string of up to 80 characters enclosed in quotes ('), or
2) A CHAR variable or element of a CHAR array, or
3) An integer expression (which must not being with a CHAR variable), followed optionally by a colon and a second expression which specifies the minimum field width. The integer expression is computed and written out as a signed decimal number in the range of -32768 to 32767. The number will occupy a field at lease as wide as the specified field width, with leading blanks provided if necessary. If the field width is not specified, a field width of 6 is used.

WRITELN is identical to WRITE, except that a carriage return and line feed are appended to the end of the output.

2 Syntax Diagrams

The structure of a program in Pascal in general form is illustrated using a tool called a 'syntax diagram'. These syntax diagrams that follow show what a program looks like, from the point of view of the compiler, and as such are worth taking a lot of trouble over to minimize syntax or compile-time errors. A knowledge of how the compiler will view a program can be a useful tool in producing correct code, provided the ideas behind the program have been properly thought out beforehand.

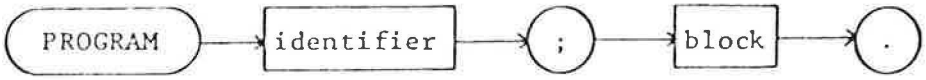
Here are some rules for interpreting syntax diagrams:

1) Symbols in circles are Pascal punctuation marks -- i.e. separators, delimiters and terminators etc.
2) Sausages (or ovals) contain either the reserved words (in capitals) or one of the 'letter', 'digit' or 'character' codes, which includes anything on the keyboard.
3) Rectangles enclose names of elements which are defined in other diagrams (e.g. 'identifier' in the first diagram is defined in the second diagram). They can be considered therefore as symbols for other complete diagrams.

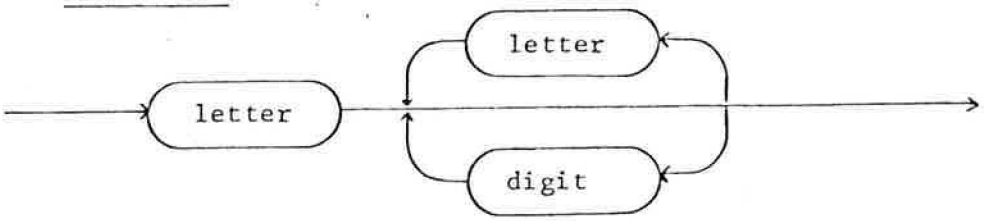
Using these diagrams, and always following the arrows, it should be possible to check the syntax of every program that will compile successfully (but just because it checks out does not necessarily mean that it will compile at all!). Please distinguish between the single quote (') and the (,), and note that the up-arrow appears as a caret (^) on the Pegasus. Another point to note is that lower case should never be used, except inside string quote marks.

Full syntax diagrams for standard Pascal appear in Niklaus Wirth's book 'Pascal User Manual and Report', but these will not apply in full to Pegasus Pascal, which is a subset of standard Pascal.

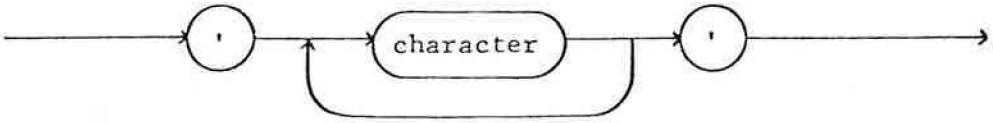
program



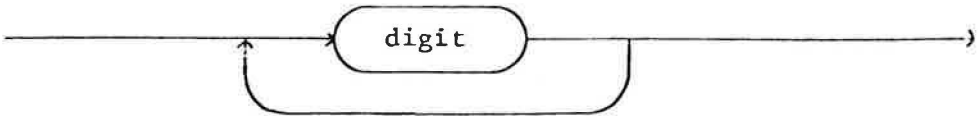
identifier



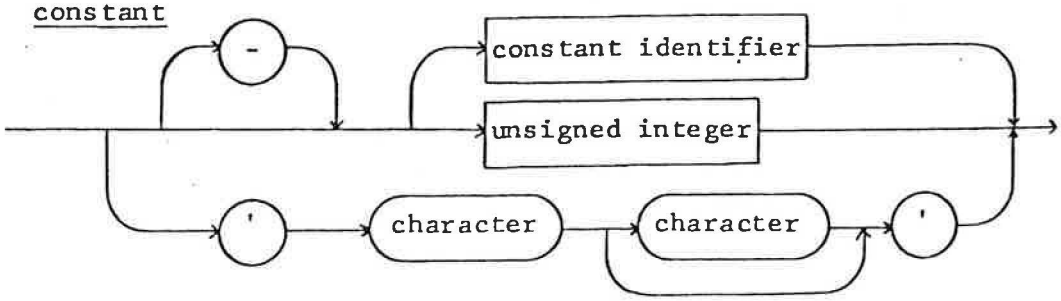
string



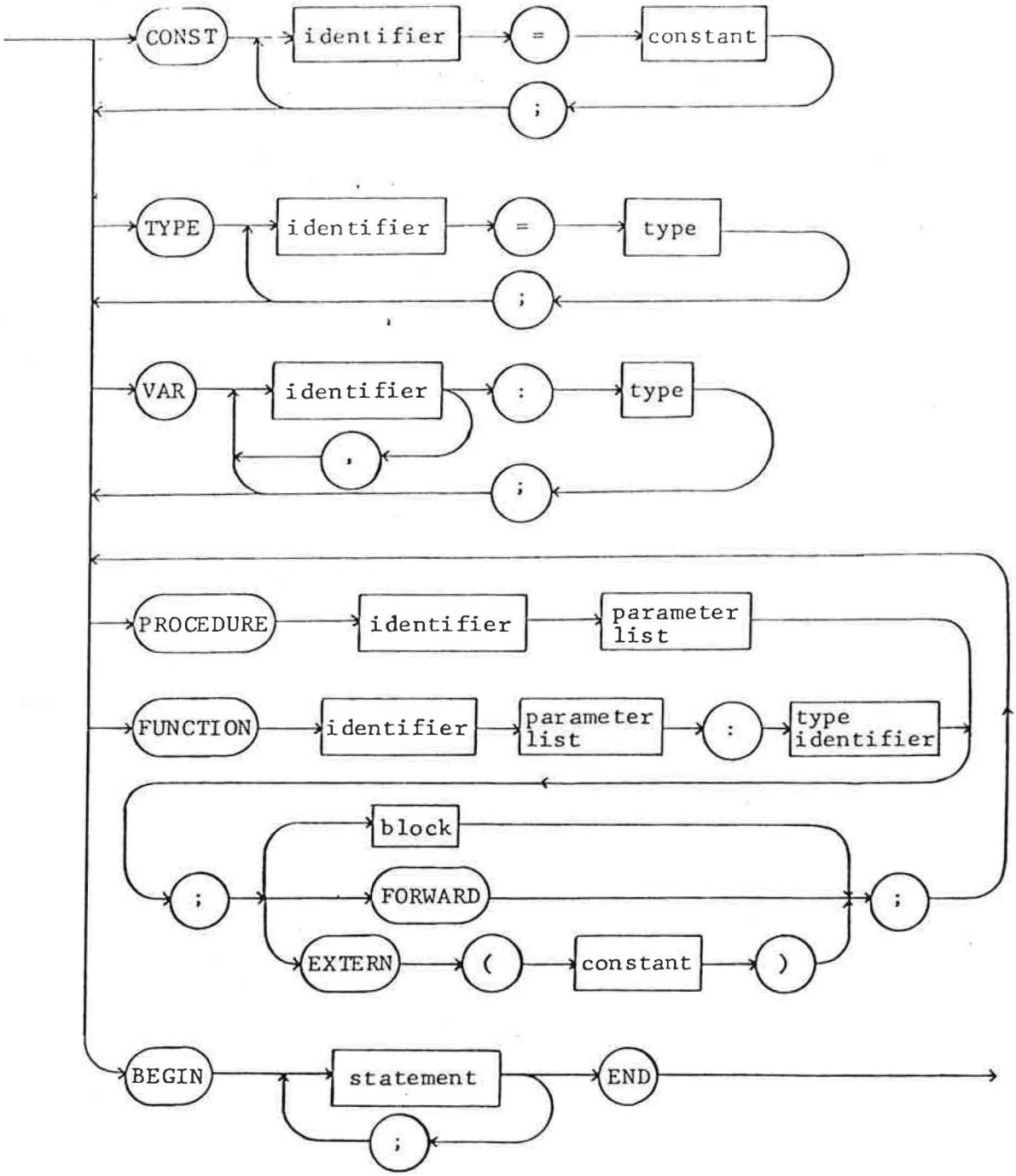
unsigned integer



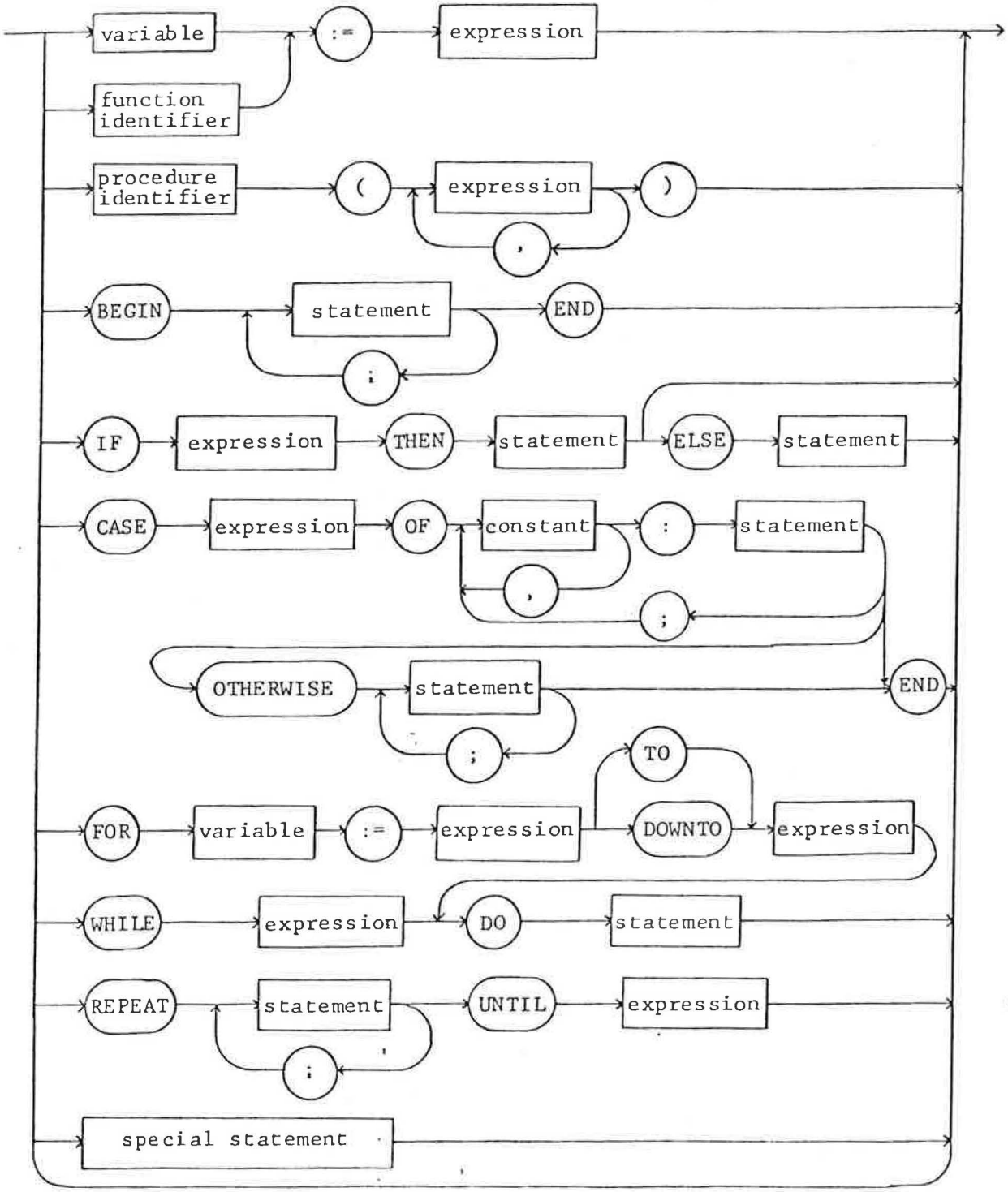
constant



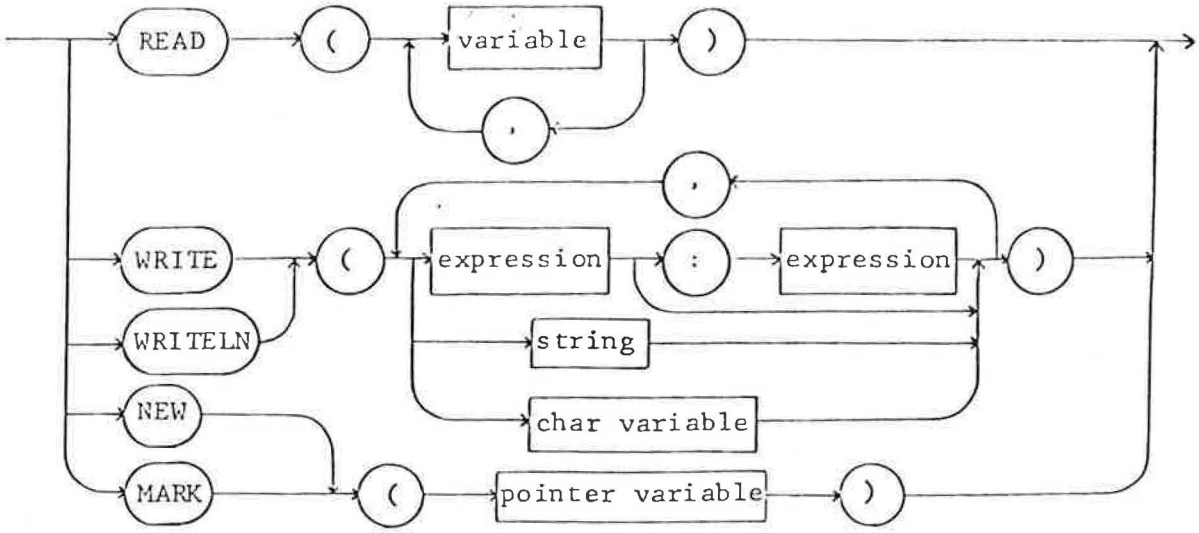
block


CONST
identifier
=
constant
;
TYPE
identifier
=
type
;
VAR
identifier
,
:
type
;
PROCEDURE
identifier
parameter list
FUNCTION
identifier
parameter list
:
type identifier
;
block
FORWARD
EXTERN
(
constant
)
;
BEGIN
statement
;
END

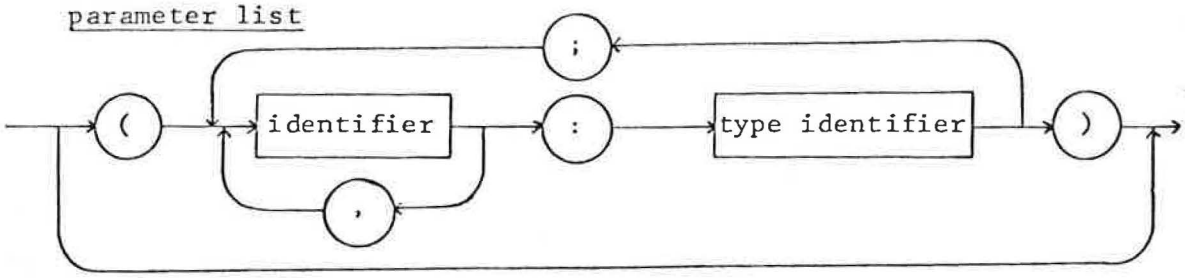
statement


variable
:=
expression
function identifier
procedure identifier
(
expression
)
,
BEGIN
statement
END
;
IF
expression
THEN
statement
ELSE
statement
CASE
expression
OF
constant
:
statement
,
;
OTHERWISE
statement
END
;
TO
FOR
variable
:=
expression
DOWNTO
expression
WHILE
expression
DO
statement
REPEAT
statement
UNTIL
expression
;
special statement

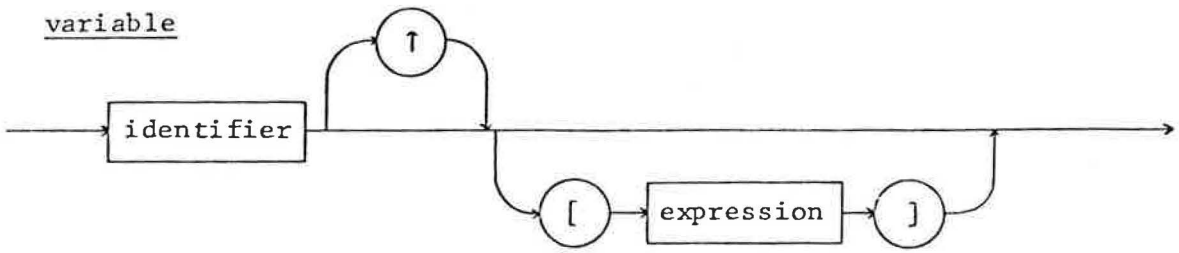
special statement



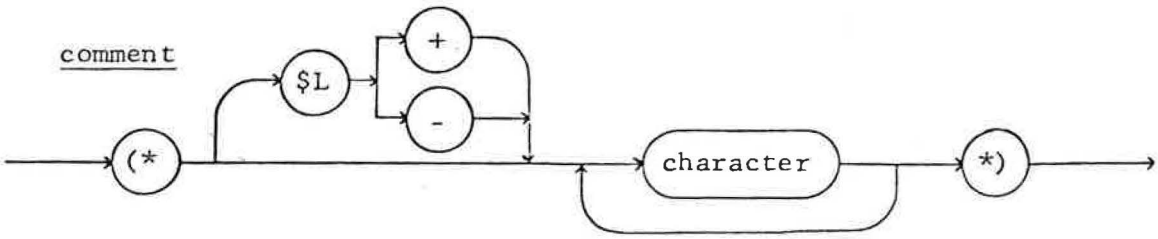
parameter list



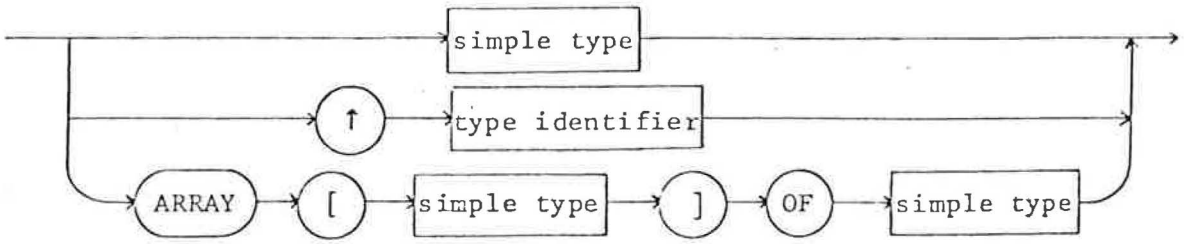
variable



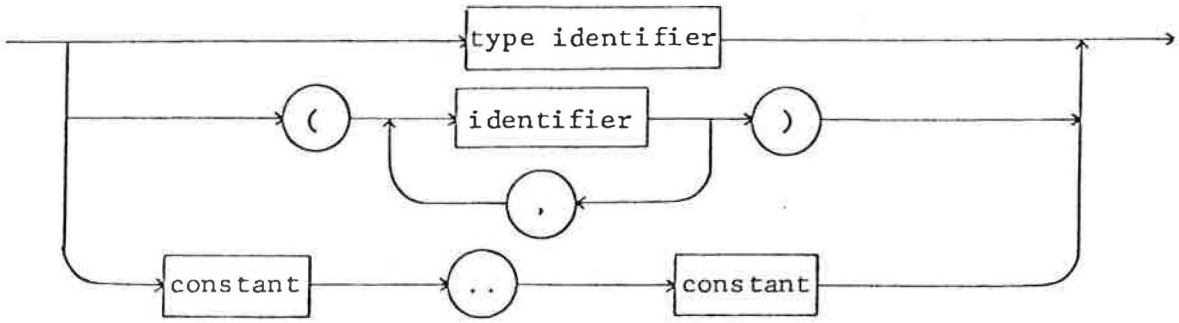
comment



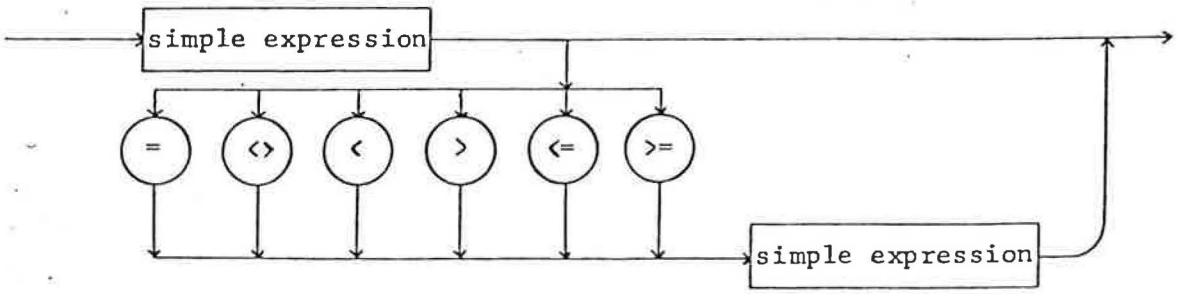
type



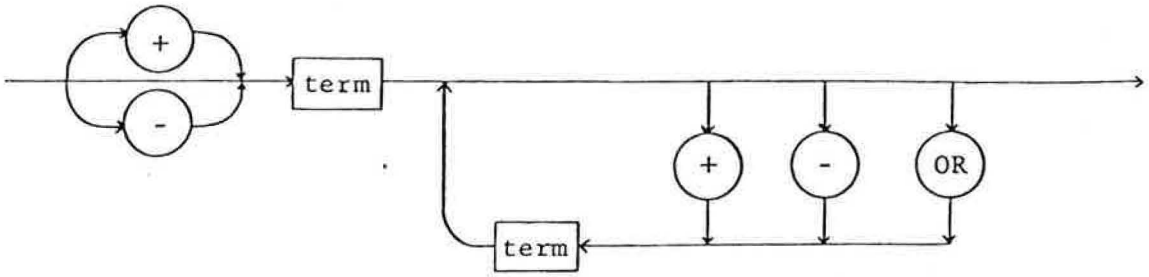
simple type



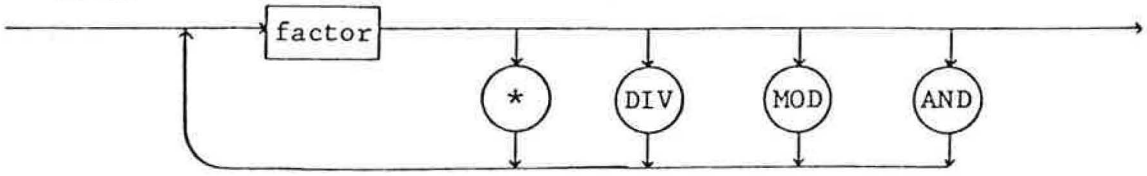
expression



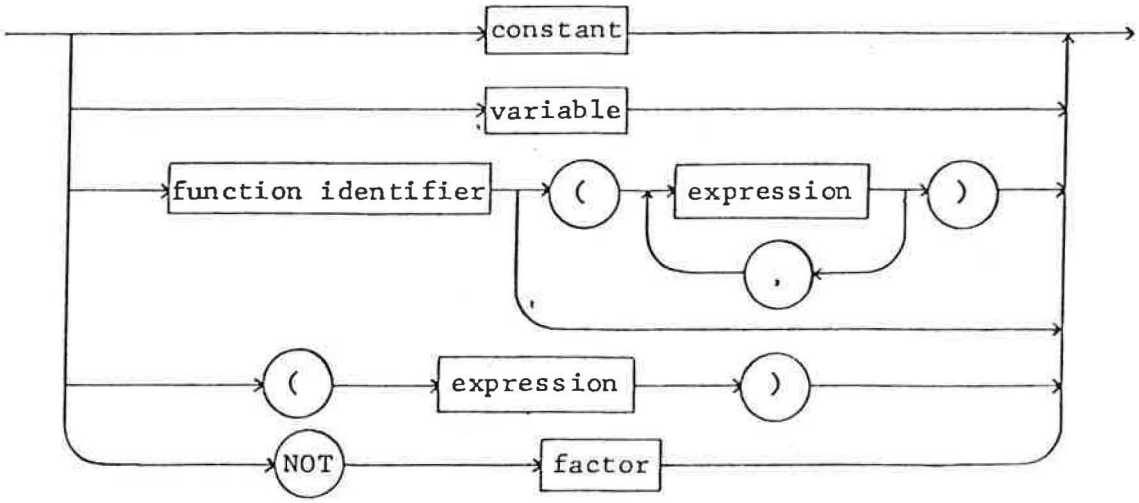
simple expression



term



factor



3. Pascal on the Pegasus

Pegasus Pascal is a ROM based, 8K package which consists of a p-code interpreter, the command module, the line editor, and the compiler, as well as the printer I/O drivers and a number of external ROM based subroutines. Here is a memory map:

0000..04BB	P-code interpreter and I/O drivers
04C0..04ED	Printer drivers INIT and PRINT
0500..05BD	Command module
0600..093B	Line editor
0940..0983	ROM based subroutines
0989..1FFF	Pascal Compiler module

This Pascal is a subset of the standard language as defined by Wirth, which discludes floating point numbers, records, sets, and some of the more complex data structuring abilities of standard Pascal such as arrays with more than one subscript. The compiler, editor and command module are all written in Pascal, which has been compiled to p-code and placed into the ROM. All programs will compile to p-code, which may then be interpreted by the p-code interpreter, which is written in 6809 microprocessor machine langugage.

The system uses a RAM workspace, into which the source of the Pascal programs is placed during the edit phase. When a satisfactory program has been created, it may then be saved on cassette tape. It is recommended that filenames used for saving the source form has '.S' on the end. The user will then start a compiler, which will compile the program line by line, producing a listing (which may be disabled with (*$L-*)), and produce p-codes which write over the source program in the workspace. The p-codes in the workspace may then be interpreted by the p-code interpreter, which will cause the program to run. The p-code form may also be saved on cassette tape, and it is recommended that the file-name have '.P' appended to it. The reason for appending '.S' and '.P' is so that the user may distinguish between

the two forms of his program when he reloads it into the workspace from cassette tape, which avoids such nasty situations as trying to compile the p-code again!

Pascal will appear on the Pegasus menu when the two EPROMs, Pascal A and Pascal B are installed correctly. The Pascal A EPROM should be installed in the socket 0000..0FFF, which is the socket furthest from the system monitor on the Pegasus board.
Typing 'P' from the menu will give the Pascal prompt
PAS>
which means that you are successfully in the comman module. Commands consist of a single letter, in one case preceded with a single digit in the range of 0 to 7. It is not necessary to type the return key, so take care as to which key your are typing.
The commands on Pegasus Pascal are:

L (Load) - load a program from the cassette. The file may be a source file, '.S' or p-code '.P'.

S (Save) - saves the contents of the workspace onto cassette. You must provide the filename, and should ensure that the correct ending is used.

E (Edit) - this causesyou to invoke the editor module

C (Compile)- a source program in the workspace will be compiled

G (Go) - a p-code program in workspace will be executed

F (Free) - a number is printed of free remaining memory bytes

nP (Print) - the Pascal source program in the workspace is printed on a line printer connected to the parallel port of the Pegasus. The default for this command is port 2, which requires the I/0 expansion interface. Port 1 is on the board.

Q (Quit) - quits from Pascal back to the Pegasus monitor

Any p-code program that is currently being interpreted may be halted by typing the BREAK key. (If you are in the input statement then you will need to type another key as well.) Printing to the screen may be halted and restarted by using the ESC key as usual. If the BREAK key is used, then an error message will indicate program termination. Pascal programs must consist entirely of upper case identifiers and reserved words, except for inside string literals, which are enclosed by single quote marks (').

PASCAL Sample Programs

```
PROGRAM TIMESTABLE;          Name of program
VAR I,J:INTEGER;             Declare two variables
BEGIN                        Start main block
  FOR I:=1 TO 10 DO          Set up outer loop
  BEGIN                      Compound statement starts here
   FOR J:=1 TO 10 DO         Set up inner loop
    WRITE(I*J:3);            This line is repeated
   WRITELN                   Move cursor to new line
  END                        End of compound statement
 END.                        End of program

PROGRAM TEMPCONV;            Name
CONST FREEZING=32;           Set a constant value
VAR CENT,FAHR:INTEGER;       Declare two variables
BEGIN  (* This is a comment *)
 WRITE('Temperature in Centigrade: ');
 READ(CENT);                 Get number from keyboard
 WRITELN;                    New Line
 FAHR := CENT * 9 DIV 5 +    Do math
 FREEZING;
 WRITELN('Equivalent Fahrenheit is ',FAHR)
END.

PROGRAM WALKING;             Name
PROCEDURE LEFTFOOT;          Declare a procedure
BEGIN
 WRITELN('TRAMP');           Write some text
 WRITELN
END;
PROCEDURE RIGHTFOOT:         Declare another procedure
VAR C:CHAR;                  Define a character variable
BEGIN  C:=9;                 Use the tab control code
 WRITELN(C,'TRAMP');         Tab then print
 WRITELN
END;
BEGIN
 LEFTROOT; RIGHTFOOT;        Start walking....
 LEFTFOOT; RIGHTFOOT
END.
```

4. Using the Line Editor

The line editor is started by typing the letter 'E' while in the command module. The editor can be used to create new Pascal programs in the workspace, or to edit any text loaded from cassette tape while under the command module.

The Pascal editor uses a line marker concept, which means that since Pascal programs do not have line numbers, like in BASIC, we must have some way to tell the editor which line we are working with. The line marker always points to the first character of the line that we are currently working with in the workspace. When the editor is first entered, the line marker points to the first character in the workspace, which is always a carriage return code. The line marker can be moved around by the commands Top, Bottom, Up, Down, and Find.

The editor always gives one of two prompts, depending upon which mode it is in. The command prompt,

E>

means that the editor is ready for a command, while the insert

>

prompt means that the editor is waiting for a line of text to be entered from the keyboard, and terminated with the return key.

All editor commands are a single letter, followed optionally by a decimal number (which must be positive.) commands should be terminated with the return key. Note that commands, once started, cannot be aborted unless you hit the BREAK key, which will take you back to the command module. You must be especially careful with the 'N' command.
The editor commands are summarised below:

N (New) - clear all text from the workspace, point to top
T (Top) - move line marker to top of workspace
B (Bottom) - move line marker to bottom of workspace
U (Up) - move line marker up by one line

Unn	- move line marker up by nn lines (nn is a number)
D (Down)	- move line marker down one line
Dnn	- move line marker down by nn lines
Ftext (Find)	- move the line marker down to the next line which contains the given text, up to 40 characters and terminated with a return key
P (Print)	- print the current line at the marker
Pnn	- print the next nn lines
I (Insert)	- Insert one line at the current line market, i.e. insert aline before the current line.
Inn	- insert a series of lines. Typically a large number e.g. 99 may be typed here to allow the insertion of a lot of lines To leave insert mode, type control-C. The line pointer is left pointing to the line it was pointing to first.
K (Kill)	- delete the current line from the workspace
Knn	- kill the next nn lines
R (Replace)	- replace the line currently marked, with one to next be typed by the user
Rnn	- replace the next nn lines
Q (Quit)	- leave the editor and re-enter command mode in the command module

Note that entering the editor for the first time will not clear the workspace, this must be done manually by the user with the 'N' command. This means that on the Pegasus you may hit the panic button, re-enter Pascal from the menu, and continue editing the text in the workspace without losing it.

A good way of printing the entire source is simply to use the T command, followed by P999. orrections may be made to lines using the backspace key, but the control-U will not work.

The editor does not have to be used only for Pascal programs - you can create text files and save them on tape just as easily as with Pascal programs. This editor cannot however, be used for writing BASIC of Forth programs, since the way they are stored internally is quite different.

5. Built in Routines

The Pascal for Pegasus has a number of predefined procedures and functions which are a standard part of the language. These are summarised below:

PROCEDURES

HALT

Terminates program control, returns to command module

LINK(PRG,STK)

Transfers control to a p-code program which starts at memory address PRG, with a runtime stack beginning at STK. If STK=0 then new program uses the same stack as the calling one.

MOVL(SRC,END,DST)

Moves the block of memory from memory address SRC through to address END to a block beginning at address DST. The bytes are moved starting with the low address.

MOVR(SRC,END,DST)

Similar to MOVL, except that DST is the END of the destination block, and the high address is moved first.

NEW(P)

Allocates a new variable in heap of the type bound to pointer P, and sets P to point to this variable.

MARK(P)

Sets the pointer variable P equal to the current heap pointer, thus marking the heap current position

RELEASE(P)

(P is a pointer which has previously been used in a MARK) This sets the heap pointer equal to P, effectively discarding all dynamic variables created prior to the last time that MARK(P) was invoked.

A number of extra procedures are listing in section F.

FUNCTIONS

ODD(I)

Returns boolean true if I is odd, else returns false.

SHL(I,N)

Returns integer whose value is I shifted left by N bits

SHR(I,N)

Returns integer whose value is I shifted right by N bits

FIND(P,C)

P is a pointer to a byte or character in memory, and C is a character (8 bit quantity). This function returns a pointer to the first byte in memory after P which contains the byte C. If none is found, the value of -1 is returned.

SYSCOM

This function returns a pointer to the system communication area in RAM. The syscom area may be thought of as an ARRAY of 5 integers of pointers consisting of the following p-machine registers:

1) WS pointer to first byte in workspace
2) EOW pointer to last byte of workspace
3) PORG pointer to start of currently executing p-code program
4) STACK pointer to base of current Pascal stack
5) HEAP pointer to most recently allocated variable on heap

An example of using SYSCOM is given below:

```
TYPE REGS=ARRAY 0..4 OF INTEGER;
VAR WS,EOW: CHAR;
    SCOM: REGS;
BEGIN
    SCOM:=SYSCOM;
    WS:=SCOM  0 ;
    EOW:=SCOM  1
END;
```

The above code segment creates two pointers, WS and

and EOW, that point to the first and last bytes in the workspace respectively.

```
PROCEDURE POKE( ADDRESS,DATA : INTEGER);
   EXTERN(2384);

*  This procedure will poke the data byte into the given
   address
*  Care must be taken so as not to poke into an area
   required
*  by the system, as unpredictable results may occur.

POKE            LDD     ,--U            Fetch data
                LDX     ,--U            and address
                STB     ,X              Do poke
                RTS

FUNCTION PEEK( ADDRESS,DATA : INTEGER);
   EXTERN(2391);

*  Data is read from the byte at the given address, and is
*  returned as the value of the function

PEEK            LDX     ,--U            Fetch address
                CLRA
                LDB     ,X              Get data
                STD     -2,U            Return value
                RTS

PROCEDURE OUTHEX(  DATA : INTEGER);
   EXTERN(2399);

*  Data in the range of 0..255 will be converted to two hex
*  digits and output using the echo routine

FUNCTION GETHEX : INTEGER;
   EXTERN(2408);

*  This function will read and echo two hex digits from
*  the keyboard.  If a non-hex character is typed, the
*  function will return a value of -1.
```

```
FUNCTION INKEY : INTEGER;
   EXTERN(2426);

*  This function will scan the keyboard for a key.  If
*  one has been depressed, then its ASCII value will be
*  returned, otherwise a value of zero will be the function
*  value.

PROCEDURE INIT( PORT : INTEGER );
   EXTERN(8128);

*  This procedure is used for initializing the parallel
*  port B that is to be used for the printer output.  The
*  port number given specifies the I/O port to be used.
*  Port 1 is the port on the standard Pegasus board.

PROCEDURE PRINT( CH : CHAR );
   EXTERN(8174);

*  This procedure will print one character to the printer.
*  The procedure INIT must be called at least once before-
*  hand
```

7. Glossary of Pascal and Related Terms

Actual Parameter -- An actual parameter is a variable or expression contained in a procedure or function call that replaces the formal parameter which is part of the procedure or function declaration.

Address -- The address of a storage location in computer memory is the number needed to access its contents.

Algorithm -- A statement that expresses a task in terms of a series of unambiguous subtasks.

Array -- A data structure which contains elements all of the same type. Each element in an array can be accessed by means of a unique set of indices and hence directly.

ASCII -- The American Standard Code for Information Interchange. Each of 96 displayable characters and 32 control codes has a unique numeric value. This gives a total of 128 different characters which can be represented in 7 bits.

Assembly Language, Assembler -- An assembly language is a language similar to machine code with the operation codes represented as mnemonics and the addresses represented as identifiers. An assembler is a program that translates a program written in assembly language into machine code.

Assignment statement -- An assignment statement contains the assignment operator ":=". On the left side of the assignment operator is a variable whose value is replaced by the value of the expression on the right side of the assignment operator.

Batch Mode -- A program is executed in batch mode when it is submitted together with all required input date (on punch cards, paper tape, disc/tape file) to the operating system and when it has completed execution, the results output (either on line printer paper or disc/tape file).

Binary -- Two-valued number system. Compare with ten-valued "decimal" number system.

Bit -- A binary digit, that is 0 or 1.

Bit Pattern -- A sequence of bits that can be interpreted in several ways such as a number, character or as a machine code instruction.

Block -- A block consists of a set of declarations (labels, constants, types, variables and or procedures) followed by a compound statement. Any Pascal program consits of a program heading, a block and END.,whereas any procedure consists of a procedure heading followed by a block.

Body of a Loop -- The simple or compound statement that is executed each time the condition which controls the loop is met.

Boolean -- 1. Term used to described anything related to the two-valued (binary) logic system developed by 19th Century English mathematician George Boole.

-- 2. A two-valued variable or expression, the "values" being TRUE or FALSE.

Branch -- A programming construct consisting of a condition which can take two or more values and a list of corresponding options. Depending on the value of the condition on of these options is executed. Pascal has two forms of branch constructs: IF..THEN..ELSE.. and CASE..OF

Buffer Variable -- See File Window

Bug -- An error in a program. Removing bugs is called "debugging".

Byte -- The number of bits used to store a character within the computer. Typically 8 bits.

Call -- The initiation of a procedure or function

Call-By-Reference -- When an actual parameter is substituted for a formal parameter within a procedure or function it is known as a call-by-reference-parameter, and the parameter is said to be passed "by reference".

Call-By-Value -- When a parameter is passed "by value" in a procedure call, a new local variable is created and initialised to the value of the actual parameter.

Character -- Some symbol associated with text -- e.g. letter of the alphabet, punctuation mark or decimal digit. (See also ASCII, Control Character.)

Code -- 1. A set of rules for interpreting a series of bit patterns -- so "machine code" is a set of bit patterns which represent different instructions for a particular computer, and ASCII code is a set of bit patterns which represent different characters etc.

-- 2. To code means to write an algorith in a computer language.

Comment -- A program statement ignored by the compiler, useful for giving information to human readers of the program.

Compiler -- A program that translates a program written in a high level language into its machine code equivalent.

Compile-time -- During compilation

Compound Statement -- A group of Pascal statements that are surrounded by BEGIN..END

Condition -- An expression which produces a value, generally TRUE or FALSE, (but which can take an scalar value), the state of which determines the branch to be executed in IF

and case statements and how many times a WHILE or REPEAT loop is executed.

Conditional -- See Branch

Constant -- A data element whose value is assigned during compilation and cannot be changed during execution of the program.

Construct -- See Control Structure

Control Character -- Non-printing character used to format input or output on a VDU or printer, such as CR, rubout etc.

Control Structure -- A device for controlling the order in which the statements of a program are executed. Three control structures (the sequence (the default order of execution), the loop and the branch) are said to constitute Structured Programming if they are exclusively used in a program.

Control Variable -- A scalar variable which is initialised and then incremented or decremented to control the number of times a FOR..DO loop is executed.

Data -- Information supplied to and manipulated by a program

Data Structure -- A logical framework for related variables which allows them to be dealt with as a single unit or individually. Examples from Pascal include Arrays and Records

Data Types -- Data is stored in bit patterns and must be interpreted by the program. In Pascal data can be stored as one of several types i.e. INTEGER, REAL, BOOLEAN, CHAR or scalar type.

Debug -- To correct a program that contains bugs or errors

Declaration -- The specification of all identifiers to be used in a program or procedure, together with their types, in order to give the compiler information about what storage space should be allocated.

Default Value -- A common or frequently held value assigned to a variable to save the user from the effort of manually inputting it. The program must give the user an opportunity to override this value should other circumstances obtain. In programs that require a large amount of user input, defaults can save considerable time at the keyboard.

Delimiter -- A delimiter is a symbol or a pair of symbols used to mark the beginning and end of an entity. Pascal uses single quotes as string delimiters and curly brackets or (* and *) as comment delimiters.

Direct Access - A method of obtaining individual records within a disc file in which the program provides a record number which is used to position the read/write head of the disc drive in the correct place so that the record can be accessed without having to access any other records. Standard Pascal does not support direct access (also called random access) files; however, most implementations of Pascal on microcomputers do.

Dynamic Data Structure -- A data structure which grows and shrinks during program execution. In Pascal dynamic data structures are accessed through a pointer type and held in the "heap".

Editor -- A program provided as part of an operating system that allows a user to input and alter a program

Execution -- The performance, by the computer, of the tasks described in a program.

Expression -- A sequence of constants and identifiers separated by operators that has a value (numeric, character or Boolean), which is the result of some processing at tun-time.

File -- A collection of data external to a program and held on disc, tape or cards.

File Window -- Given a typed file, the file window (also called the buffer variable) is a variable held in memory of the same type as the file elements. Every element (generally a record) read from the file into main memory or written from main memory to the file uses the file window.

Field -- One of the variables within a record.

Flag -- A variable, generally Boolean, which is set in a program to show that some event has occurred, and is tested later on to determine whether a course of action dependent on that event may proceed.

Formal Parameter -- An identifier declared in a procedure or function declaration and used throughout the procedure or function. When the procedure or funtion is called, the formal parameter is replaced by an actual parameter.

Forward reference -- Normally, procedures must be declared before they can be called, but if two procedures call each other then both cannot be declared before being called. Instead, the reserved word FORWARD is used to tell the compiler that a reference will be made to a procedure whose declaration will be supplied subsequently.

Function -- A procedure that returns a value. It is called by referencing its identifier (with appropriate parameters) wherever a variable of its type would be acceptable.

Garbage Collection -- A technique employed by the run- time system to recover unused heap space freed when a dynamic data structure contracts.

Global Variable -- A variable that is declared in the declaration part of the main program block and hence accessible to the main program and all procedures, except when a variable with the same identifier is declared within a procedure. In the latter case, the global variable goes temporarily out of scope.

Heading -- The first line of a program, procedure or function is called its heading.

Heap -- The area set aside in main memory at run-time to hold dynamic data structure. (Compare Stack).

High Level Language -- Any language such as Pascal in which most single instructions will translate into several machine code instructions.

Identifier -- A name given to data type, data item, data structure, procedure of function. Identifiers can be of any length but on most implementations only the first eight characters are significant.

Indentation -- Non-functional blank spaces inserted at the left of Pascal program code so that the structure of the program is visible.

Initialise -- To give a variable an initial value (usually zero) before using it in a program.

Integer -- Pre-declared Pascal type for (a limited range of) whole numbers.

Interpreter -- A program which takes as input a program written in either a high level language (e.g. BASIC, APL) or pseudo code (many Pascal systems) and translates and executes each statement one at a time.

Iterative -- Repetitive. Many mathematical functions are evaluated iteratively, that is, an initial solution is proposed, then tested and if not sufficiently accurate, the solution is improved upon and tested again. This process continues (using a loop construct) until the solution is deemed sufficiently accurate to stop.

Jump -- A control structure which directs control from one point in the program to another out of the normal sequence. (GOTO)

Linkage Loader -- A systems program that takes as input relocatable object code that is produced when a source program is compiled along with any standard functions from the library called in the user program, and produces as output a machine cade program that is ready to be executed.

Linked List -- A dynamic data structure where each element consists of a data item and a pointer to the next element.

Listhead -- A pointer to the first element in a linked list.

Local Variable -- A variable declared within the block where it is used.

Loop -- A sequence of program steps that is repeated a controlled number of times. Pascal provides three contructs for coding loops (FOR-DO, WHILE-DO, REPEAT-UNTIL).

Machine Code or Language -- The instructions that a computer's hardware actually executes.

Main Memory -- Immediate access storage locations. This is where the stack, heap and code that is actually being executed are located.

Master File -- In a data processing environment, the master file is the main file on which all alterations are made. (See update).

Module -- A functionally separate section of a program, generally a procedure. A modular program is one where each subtask is dealt with in its own distinct procedure.

Nesting -- When a logical construct has the same type of construct within itself, e.g. a loop within a loop, procedure inside a procedure.

Object Program -- A machine code program, the translation of some high-level language program as output by the compiler.

Operand -- A data item which is to be manipulated in the instruction within which it occurs.

Operator -- A symbol such as +, := etc which indicates that certain operations to take place.

Overlay -- In a computer system where memory is smaller than that required to hold the whole object program either the user or the system divides the program into sections such that only a few of these sections are in main memory at any one time and when a new section is needed it is written into main memory over an old section.

Packed, Packing -- The process of allocating the smallest amount of memory required for a structured type (record or array) is called packing. The structured type is called a PACKED type.

Parameter -- A variable passed between a calling and a called procedure or function.

Parameter List -- The list of variables in a procedure or function heading together with their corresponding types.

Peripheral -- A device added on to a computer (CPU plus memory) usually used for either backing store (disc and tape drives), or human communication (terminal and printer).

Pointer -- A variable, usually integer, containing information about the location of some data (usually a specific array or stack element). By changing the value the pointer contains, different elements can be located.

Pop -- When an element is removed from a stack it is said to be popped off the stack.

Predecessor -- In a list of scalar items the predecessor of a given item is the adjacent item occuring earlier in the list.

Procedure -- A procedure is a subprogram or subroutine, similar in form to a program. It must be declared in the declaration part of a program and is executed when its name is called.

Program Stub -- When developing a program or a section of a program instead of writing the code for each procedure that is called a short piece of code called a program stub is written, in order to test the logic of the code being currently written.

Push -- When an element is placed on a stack it is said to be pushed onto the stack.

Range -- The set of values that a given variable may take.

Real -- A pre-declared Pascal type for decimal numbers.

Record -- A structured type consisting of several related data items of different types, called fields. Records tend to be grouped together in files.

Recursive -- A procedure or function is said to be recursive if it contains a call to itself.

Reserved word -- A word that cannot be used as an identifier because it is already a word within the programming language.

Relational Operator -- An operator such as , or = which is used to compare two operands.

Return, Return Address -- After a procedure completes its processing, program control transfers to the first program statement following the one where the procedure was called. The address where program control is transferred to is called the return address and flow of control is said to return to the calling routine. Any result passed back to the calling procedure will have been "returned".

Rogue Value -- A value put at the end of a data input stream that is out of the normal range of input data and whose purpose is to signal to the program that the input stream is finished.

Run-time -- The period of time when a program is executing.

Scalar -- A variable or constant which can have any of an ordered set of values which is listed in its declaration.

Scope -- A set of procedures within which a variable is defined.

Set -- A reserved word for BOOLEAN array. If an element is TRUE the corresponding item is said to be a member of the SET.

Sequential Access -- A method of reading or writing to a file where the next item (usually a record) which can be accessed is the one that immediately follows the last one accessed.

Side Effect -- The alteration of the value of a variable brought about by making an assignment to that variable in a procedure in which it is not declared and to which it is not passed.

Source program -- High-level language program which becomes the input for the comiler at compile-time.

Stack -- 1. In general, a list of elements such that the last element input is the first element that can be accessed.

Stack -- 2. Area of memory set aside to hold declared variables and procedure calls at run-time (hence Run-Time stack).

Stack Frame -- Region of the stack containing all data relevant to a single procedure or function call. When the call is made, the whole frame is popped onto the stack.

Standard Function -- A function supplied with in a Pascal system. A standard function isn't declared within a program but can be used just by calling its name (e.g. x:=sin(y)).

Statement -- A single program instruction.

Statement Separator -- A symbol (;) which is used to separate two statements in the action part of a program.

Static Data Structure -- A data structure such as an ARRAY or RECORD which has its space allocated at compile time, via a declaration statement.

Stepwise Refinement -- A method of program development. First the main program is written, but whenever a problem is encountered it is given a name and passed over. When the main program has been completed each of the subproblems is tackled in the same way. This process of successively decomposing a problem into sub-problems is continued until the problems that are left can be easily coded.

String -- A collection of characters between quotes (literal string). Most versions of Pascal for microcomputers have a predeclared STRING data type which is equivalent to a PACKED ARRAY OF CHAR.

Structured Programming -- A technique of coding where all control structures are sequences, loops or conditionals. It is also known as "GOTOLESS" programming in contrast with "spaghetti" programming where the jumb construct is also allowed.

Subrange -- A variable can be declared such that it is only allowed to take on values in a subset of the full range available (hence subrange).

Subroutine -- A word used for a procedure in many programming languages.

Subscript -- An index used to access an element of an array.

Successor -- In a list of scalar items the successor of a given item is the adjacent item occuring later in the list.

Syntax, Syntax Diagram -- The set of rules describing what constitutes a grammatically correct program. A syntax diagram is a pictorial form designed by Wirth to state the rules clearly. A syntactically correct program will compile.

Tag Field -- In a record with a variant field the tag field is the variable in the CASE clause which picks out the currently valid variant.

Terminator -- A symbol used to show a given piece of code is finished. In the declaration part of a program the semicolon is the statement terminator whereas a full stop is the program terminator.

Test Harness -- A test harness is code written to "go aound" a procedure to allow it to be tested without having to write the whole program in which the procedure is to appear.

Textfile -- A file whose component items are characters is referred to by the reserved word TEXTFILE.

Top Down Design -- See Stepwise Refinement.

Type -- A data type defines the set of values a variable may take.

Update -- A file is updated when the records within the file are changed to the latest values.

Validation -- The process of checking that the input values are within a reasonable range and hence probably not erroneous.

Value Parameter -- See Call-By-Value.

Variable -- A memory location or collection of locations where a data item is tored and altered and referred to by an identifier.

Variable Parameter -- See Call-By-Reference.

Variant Field -- One of a choice of several alternative fields which can optionally be defined as the last field of a record.

Workspace -- An area in memory that holds temporary values of variables during execution (in Pascal, the Stack and the Heap).

8. Recommended Reading

Any foray into the field of Pascal should be accompanied by some sort of literature on the subject. If you are at school, learning the language for the first time, then your teacher will be able to provide the information you need, but if you are a hobbyist or other user, then some sort of text on Pascal is almost obligatory. Following is a list of books on Pascal that the author is acquainted with personally. The ones that are highly recommended are marked with an asterisk.

*Pascal User Manual and Report	Jensen, Wirth
*Pascal for Programmers	Eisenbach, Sadler
Problem Solving Using Pascal	Bowles
A Primer on Pascal	Conway, Gries, Zimmerman
Structured Programming and Problem Solving with Pascal	Kieburtz
Pascal Programming Structures	Cherry
Programming for Poets	Conway, Archer, Conway
*Programming in Pascal	Grogono
Programming Standard Pascal	Holt, Hume
*A Practical Introduction to Pascal	Wilson, Addyman
Pascal	Findlay, Watt
*Introduction to Pascal	Welsh, Elder
Programming Via Pascal	Rohl, Barret

Sources for Pascal books in New Zealand are the various University Bookshops, Whitcoulls, Technical Books, and a range of other bookshops will bring them in if requested.

The first book given in the list above is very highly recommended, since it is written by the author of Pascal, and thus constitutes the de facto standard for the language. This book has been known as the Pascal programmer's bible. It is rather terse in style, but nearly all the information on standard Pascal is there, with examples.

Another good source of information on Pascal is the "Pascal Users Group', which produces an excellent publication called 'Pascal News'. The latest subscription rates I have seen are to send AUS $10 to Pascal Users Group, c/o Arthur Sale, Dept of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001, Australia, Telephone 61-02-23 0561 x 435.

9. Errors

COMPILER ERRORS

When the source code programs are being compiled, it is very likely that the first time through you will come up with a number of errors. Since compiling a program will destroy it in the workspace, that is why it is strongly recommended that you save the source program on cassette tape before you compile it. If an error is detected during compilation, this will be indicated with some asterisks, ***, and an arrow pointing to the place in the line where the error(s) were detected. A list of the standard errors is given below.

1 "PROGRAM' expected - probably compiling p-code or garbage
2 Constant expected
3 '=' expected
4 Identifier expected
5 ';' or ':' or ',' missing
8 ' ' expected
9 '.' expected
10 ';' expected
11 identifier not declared
12 illegal identifier on left of assignment statement
13 ':=' assignment operator missing and expected
16 'THEN' expected after an 'IF'
17 ';' or 'END' expected
18 'DO' expected after a 'WHILE'
19 Incorrect symbol following statement
21 illegal identifier in expression
22 ')' expected to match parentheses
23 illegal factor
25 'BEGIN' expected
26 'OF' expected
28 'TO' or 'DOWNTO' expected
29 Symbol table full. (Limit approx 200 identifiers)

31 '(' expected
32 Index type must be scalar or subrange
33 ' ' expected
34 ' ' expected
35 Number of parameters does not agree with declaration
36 Illegal data type
38 Forward jump table overflow
99 End of source text reached prematurely

Note that a mumber of compile time errors may occur per line. This is because the computer does not know exactly what has gone wrong with your program, but it offers some suggestions as to what to look for. The majority of compile time errors are what is called syntax errors, which means that a through study of the Pascal syntax as given by the syntax diagrams which may be found in section B will tend to eliminate most of these errors.

RUN TIME ERRORS

Run time errors are picked up by the p-code interpreter, and usually occur when a Pascal program is being interpreted. The Message
Error nn at xxx indicates that a run time error has occured at p-code address xxxx, which is always taken as being an offset from location 0000 of the p-code program.

0 program executed a Halt
1 illegal operation code (program probably gone for a gurgler)
2 stack overflow (not enough memory space left to run)
3 attempted division by zero (naughty thing to do)

The hex value given by xxxx is the p-code address where the error occured, and may be found on the listing during compilation at the extreme left margin - this will give you some idea as to where the program crash occured.