HISOFT

4T

PASCAL

CONTENTS PAGE

SECTION 0	PRELIMINARIES	1
0.0	Getting Started	1
0.1	Scope of this Manual	4
0.2	Compiling and Running	5
SECTION 1	SYNTAX and SEMANTICS	7
1.1	IDENTIFIER	7
1.2	UNSIGNED INTEGER	7
1.3	UNSIGNED NUMBER	7
1.4	UNSIGNED CONSTANT	8
1.5	CONSTANT	9
1.6	SIMPLE TYPE	9
1.7	TYPE	10
1.7.1	ARRAYs and SETs	10
1.7.2	POINTERS	11
1.7.4	RECORDs	11
1.8	FIELD LIST	13
1.9	VARIABLE	13
1.11	TERM	14
1.12	SIMPLE EXPRESSION	14
1.13	EXPRESSION	15
1.14	PARAMETER LIST	15
1.15	STATEMENT	15
1.16	BLOCK	18
1.17	PROGRAM	19
1.18	Strong TYPEing	19
SECTION 2	PREDEFINED IDENTIFIERS	21
2.1	CONSTANTS	21
2.2	TYPES	21
2.3	PROCEDURES and FUNCTIONS	21
2.3.1	Input and Output Procedures	21
2.3.1.1	WRITE	21
2.3.1.2	WRITELN	24
2.3.1.3	PAGE	24
2.3.1.4	READ	24
2.3.1.5	READLN	26
2.3.2	Input Functions	26
2.3.2.1	EOLN	26
2.3.2.2	INCH	26

2.3.3	Transfer Functions	26
2.3.3.1	TRUNC(X)	26
2.3.3.2	ROUND(X)	27
2.3.3.3	ENTIER(X)	27
2.3.3.4	ORD(X)	27
2.3.3.5	CHR(X)	27
2.3.4	Arithmetic Functions	28
2.3.4.1	ABS(X)	28
2.3.4.2	SQR(X)	28
2.3.4.3	SQRT(X)	28
2.3.4.4	FRAC(X)	28
2.3.4.5	SIN(X)	28
2.3.4.6	COS(X)	28
2.3.4.7	TAN(X)	28
2.3.4.8	ARCTAN(X)	29
2.3.4.9	EXP(X)	29
2.3.4.10	LN(X)	29
2.3.5	Further Predefined Procedures	29
2.3.5.1	NEW(p)	29
2.3.5.2	MARK(vl)	29
2.3.5.3	RELEASE (vl)	30
2.3.5.4	INLINE (C1,C2,C3,........)	30
2.3.5.5	USER (V)	30
2.3.5.6	HALT	30
2.3.5.7	POKE(X,V)	30
2.3.5.8	TOUT (NAME,START,SIZE)	31
2.3.5.9	TIN (NAME,START)	31
2.3.5.10	OUT(P,C)	31
2.3.6	Further Predefined Functions	32
2.3.6.1	RANDOM	32
2.3.6.2	SUCC(X)	32
2.3.6.3	PRED(X)	32
2.3.6.4	ODD(X)	32
2.3.6.6	ADDR(V)	32
2.3.6.7	PEEK(X,T)	32
	SIZE(V)	33
2.3.6.8	INP(P)	33
SECTION 3	COMMENTS and COMPILER OPTIONS	35
3.1	Comments	35
3.2	Compiler Options	35

SECTION 4		THE INTEGRAL EDITOR	39
	4.1	Introduction to the Editor	39
	4.2	The Editor Commands	40
	4.2.1	Text Insertion	40
	4.2.2	Text Listing	40
	4.2.3	Text Editing	41
	4.2.4	Tape Commands	43
	4.2.5	Compiling and Running from the Editor	44
	4.2.6	Other Commands	44
	4.3	An Example of the Use of the Editor	45
APPENDIX 1		ERRORS	47
	A.1.1	Error numbers generated by the compiler	47
	A.1.2	Runtime Error Messages	48
APPENDIX 2		RESERVED WORDS and PREDEFINED IDENTIFIERS	49
	A.2.1	Reserved Words	49
	A.2.2	Special Symbols	49
	A.2.3	Predefined Identifiers	49
APPENDIX 3		DATA REPRESENTATION and STORAGE	51
	A.3.1	Data Representation	51
	A.3.1.1	Integers	51
	A.3.1.2	Characters,Booleans and other Scalars	51
	A.3.1.3	Reals	51
	A.3.1.4	Records and Arrays	53
	A.3.1.5	Sets	53
	A.3.1.6	Pointers	53
	A.3.2	Variable Storage at Runtime	54
APPENDIX 4		SOME EXAMPLE HISOFT PASCAL PROGRAMS	57
IMPLEMENTATION NOTES ETC.			61
BIBLIOGRAPHY			81

SECTION 0 PRELIMINARIES.

0.0 Getting Started.

Congratulations! You are now the owner of an almost complete implementation of the Standard Pascal programming language. Hisoft Pascal is a very powerful tool that enables you to build highly structured and easy-to-understand programs. However, as with any computer language, it is going to take you some time to get used to using Hisoft Pascal. We strongly recommend that you adopt the following procedure when using this package for the first time:

- read the rest of this Section very carefully, trying out the example programs until you understand how they are created, compiled and executed.

- read the Implementation Note for your computer (these are found at the back of the manual and are colour-coded) and then read it again!

- now read the editor section of this manual (Section 4) and try out the example at the end of the Section.

- go through the above steps until you feel comfortable using the editor and compiling/running a Pascal program.

- if you come to a complete halt in understanding what is happening, leave the computer and do something else for a while and then return and start from scratch - it is easy to get confused when operating in a new environment.

- if you are convinced that your problems are not of your own making then please do not hesitate to contact Hisoft, our staff are always willing to answer any genuine questions regarding our products.

Now read on........

Hisoft Pascal (HP) is a fast, easy-to-use and powerful version of the Pascal language as specified in the Pascal User Manual and Report (Jensen/Wirth Second Edition). Omissions from this specification are as follows:

FILEs are not implemented although variables may be stored on tape.
A RECORD type may not have a VARIANT part.
PROCEDUREs and FUNCTIONs are not valid as parameters.

Many extra functions and procedures are included to reflect the changing environment in which compilers are used; among these are POKE, PEEK, TIN, TOUT and ADDR.

The compiler occupies approximately 12K of storage while the runtimes take up roughly 4K and the editor uses 2k. Thus, typically, the total size of the package is roughly 19K, leaving the rest of your computer's memory for the Pascal source and object programs.

Hisoft Pascal recognises various control codes entered from the keyboard, mostly within the editor. Of course, different systems can have very different keyboard designs and thus wil have different ways of reaching control codes. In this manual the control characters used will be referred to as ENTER, CC, CH, CI, CP, CS and CX. The Implementation Note for your system, which may be found at the back of this manual, will tell you the corresponding keys for your system.

Whenever HP is waiting for a <u>line</u> of input, the control characters can be used as follows:

ENTER	is used to terminate the line.
CC	returns to the editor.
CH	deletes the last character typed.
CI	move to the next TAB position.
CP	directs output to the printer (if available) or if output was going to the printer then it returns to the screen.
CX	deletes the whole line typed so far.

The method of loading Hisoft Pascal into your computer varies depending on your machine - your Implementation Note gives details - in general typing LOAD "" will suffice but please check your Implementation Note first.

When the compiler has been successfully loaded it will execute automatically and produce the message:

Top of RAM?

You should respond to this by either entering a positive decimal number up to 65536 (followed by ENTER) or by hitting ENTER. Note that we refer to the ENTER key here, on some systems this is the CR or NEWLINE or RETURN key.

If you enter a number then this is taken to represent the highest RAM location + 1 otherwise the first non-RAM location is automatically computed. The compiler's stack is set to this value and thus you can reserve high memory locations (perhaps for extensions to the compiler) by deliberately giving a value less than the true top of RAM.

In the ZX Spectrum version of Hisoft Pascal the 'true' top of RAM is taken to be the start of the user-defined graphics area (UDG in the Sinclair manual).

You will now be prompted with:

Top of RAM for 'T'

Here you can enter a decimal number or default to the 'Top of RAM' value previously specified. What you enter will be taken as the stack when the resultant object code is executed after using the editor 'T' command (See Section 4 for details). You will need to define a runtime stack different from the top of RAM if, for example, you have written extensions to the runtimes and wish to place them safely in high memory locations.

Finally you will be asked:

Table size?

What you enter here specifies the amount of memory to be allocated to the compiler's symbol table.

Again, either enter a positive decimal number followed by ENTER or simply ENTER by itself in which case a default value of (available RAM divided by 16) will be taken as the symbol table size. In nearly all cases the default value provides more than enough space for symbols. The symbol table may not extend above machine address #8000 (32768 decimal). If you specify so large a value that that this happens then you will be prompted again for 'Top of RAM' etc.

At this point the compiler and integral editor will be relocated at the end of the symbol table and execution transferred to the editor.

To whet your appetite, and to give you a feel for the package, we now give an example of using the editor and compiler to create, compile and run a small Pascal program.

Firstly, let's load the Pascal into the computer: take the tape supplied, put it into your tape recorder with the side labelled 'Hisoft Pascal' uppermost ad rewind the tape to the beginning. Now refer to the Implementation Note at the back of the manual to see how to load the Pascal into your machine – on the ZX SPECTRUM you just type LOAD "" and press ENTER.

Once the Pascal has been loaded into the computer, it will automatically run itself and produce the 'Top of RAM?' message as described above. For the examples given below all you have to do here is press the ENTER (CR, NEWLINE etc) key for each of the 3 questions (TOP OF RAM, TOP OF RAM FOR 'T' and TABLE SIZE). Now the Pascal's editor will be in charge, letting you know that it's there by the '>' prompt. Let's type the following:

```
I10,10 <ENTER>
```

you should now be prompted with the number 10 ; this is a line number and what you subsequently type will be entered into the Pascal textfile at line 10, typing <ENTER> will terminate the line and then you will be prompted with line number 20 and so on. You can continue to enter text like this (with the line numbers being generated automatically for you) until you enter the special CC code (see your Implementation Note). On the ZX Spectrum CC is CAPS SHIFT 1. So, type the following program, remembering that you do not need to type in the line numbers:

```
10 PROGRAM HELLO;
20 BEGIN
30  WRITELN('HELLO WORLD!');
40 END.
50 CC
```

Remember, CC is a special abort code (CAPS SHIFT 1 on the Spectrum).
Right, to compile this program type:

```
C <ENTER>
```

You should see a listing of you program appear with some extra numbers at the front – this is a compiler listing. If the program compiles correctly, the message Run? will appear; answer Y to this question and the program will run and print out:

```
HELLO WORLD!
```

and return to the editor. You can now run the program again by:

```
R <ENTER>
```

If your program did not compile correctly and produced an *ERROR* message then press E followed by <ENTER> to get back to the editor, then press:

```
L <ENTER>
```

to list your program and compare it with the one given. If you see a mistake in any line then simply type that line number, then a space and then the correct text followed by <ENTER>.

Now, let's try another program:

```
I10,10 <ENTER>

10  PROGRAM CHARTOASC;
20  VAR CH : CHAR;
30  BEGIN
50   REPEAT
40    WRITE('ENTER A CHARACTER  );
60    READLN;
70    READ(CH);
80    WRITELN(CH,' IS ',ORD(CH),' IN ASCII.');
90   UNTIL CH=' ';
100 END.
110 CC
```

Now compile (C) and run this program; it will prompt you to enter a character, followed by <ENTER> and then print out the ASCII equivalent of that character. This will repeat until you enter a space as the character.

For those who know Pascal well, we would encourage you to study the use of READLN and READ in the above example of reading characters – also study the relevant sub-sections (2.3.1.4 and 2.3.1.5).

We hope that the above examples have given you some idea of how to use Hisoft Pascal; please now read the rest of this section, then the Implementation Note for your system and then Section 4. Good Luck!

0.1 Scope of this manual.

This manual is not intended to teach you Pascal; you are referred to the excellent books given in the Bibliography if you are a newcomer to programming in Pascal.

This manual is a reference document, detailing the particular features of Hisoft Pascal. Section 1 gives the syntax and the semantics expected by the compiler.

Section 2 details the various predefined identifiers that are available within Hisoft Pascal, from CONSTants to FUNCTIONs.

Section 3 contains information on the various compiler options available and also on the format of comments.

Section 4 shows how to use the line editor which is an integral part of Hisoft Pascal.

The above Sections should be read carefully by all users.

Appendix 1 details the error messages generated both by the compiler and the runtimes.

Appendix 2 lists the predefined identifiers and reserved words.

Appendix 3 gives details on the internal representation of data within Hisoft Pascal – useful for programmers who wish to get their hands dirty.

Appendix 4 gives some example Pascal programs – study this if you experience any problems in writing Hisoft Pascal programs.

0.2 Compiling and Running.

For details of how to create, amend, compile and run an HP program using the integral line editor see Section 4 of this manual.

Once it has been invoked the compiler generates a listing of the form:

```
xxxx nnnn text of source line
```

where: xxxx is the address where the code generated by this line begins.
nnnn is the line number with leading zeroes suppressed.

If a line contains more than 80 characters then the compiler inserts new-line characters so that the length of a line is never more than 80 characters.

The listing may be directed to a printer, if required, by the use of option P if supported (see Section 3).

You may pause the listing at any stage by pressing CS; subsequently use CC to return to the editor or any other key to restart the listing.

If an error is detected during the compilation then the message '*ERROR*' will be displayed, followed by an up-arrow ('^'), which points after the symbol which generated the error, and an error number (see Appendix 1). The listing will pause; hit 'E' to return to the editor to edit the line displayed, 'P' to return to the editor and edit the previous line (if it exists) or any other key to continue the compilation.

If the program terminates incorrectly (e.g. without 'END.') then the message 'No more text' will be displayed and control returned to the editor.

If the compiler runs out of table space then the message 'No Table Space' will be displayed and control returned to the editor. Normally the programmer will then save the program on tape, re-load the compiler and specify a larger 'Table size' (see Section 0.0).

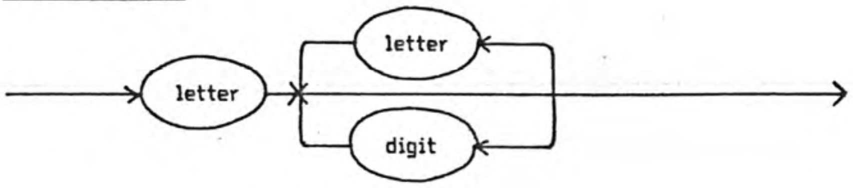
If the compilation terminates correctly but contained errors then the number of errors detected will be displayed and the object code deleted. If the compilation is successful then the message 'Run?' will be displayed; if you desire an immediate run of the program then respond with 'Y', otherwise control will be returned to the editor.

During a run of the object code various runtime error messages may be generated (see Appendix 1). You may suspend a run by using CS; subsequently use CC to abort the run or any other key to resume the run.

SECTION 1 SYNTAX AND SEMANTICS.

This section details the syntax and the semantics of Hisoft Pascal - unless otherwise stated the implementation is as specified in the Pascal User Manual and Report Second Edition (Jensen/Wirth).

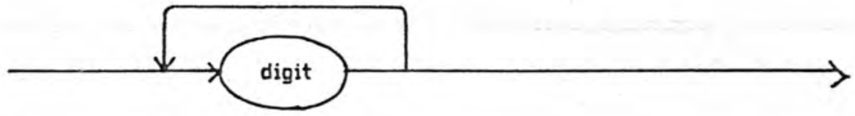
1.1 IDENTIFIER.



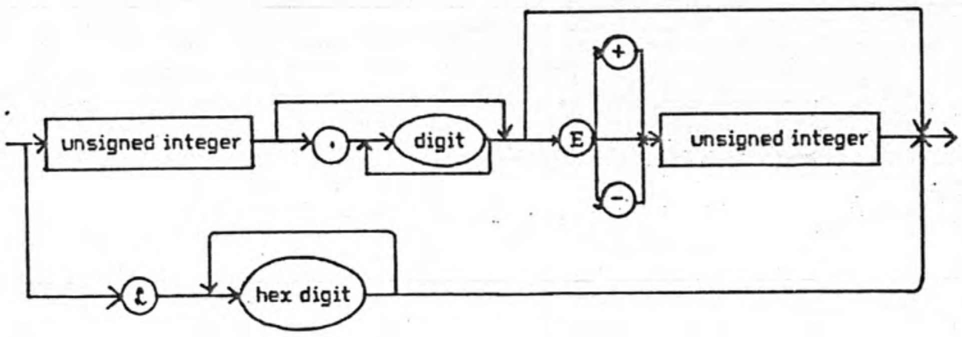
Only the first 10 characters of an identifier are treated as significant.

Identifiers may contain lower or upper case letters. Lower case is not converted to upper case so that the identifiers HELLO, HELlo and hello are all different. Reserved words and predefined identifiers may only be entered in upper case.

1.2 UNSIGNED INTEGER.



1.3 UNSIGNED NUMBER.



Integers have an absolute value less than or equal to 32767 in Hisoft Pascal. Larger whole numbers are treated as reals.

The mantissa of reals is 23 bits in length. The accuracy attained using reals is therefore about 7 significant figures. Note that accuracy is lost if the result of a calculation is much less than the absolute values of its arguments e.g. 2.00002 - 2 does not yield 0.00002. This is due to the inaccuracy involved in representing decimal fractions as binary fractions. It does not occur when integers of moderate size are represented as reals e.g. 200002 - 200000 = 2 exactly.

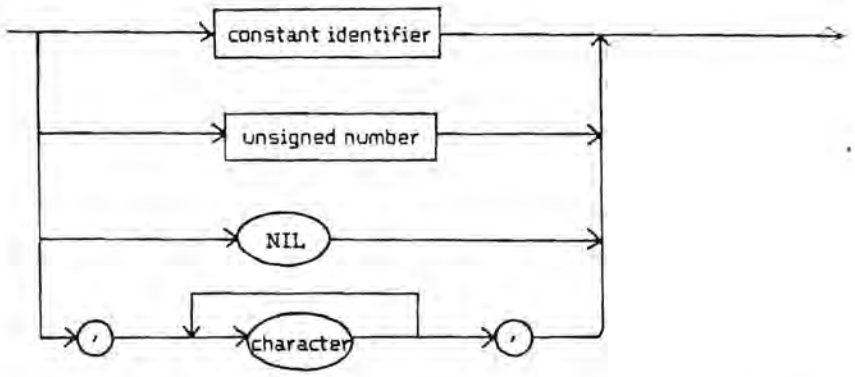
The largest real available is 3.4E38 while the smallest is 5.9E-39.

There is no point in using more than 7 digits in the mantissa when specifying reals since extra digits are ignored except for their place value.

When accuracy is important avoid leading zeroes since these count as one of the digits. Thus 0.000123456 is represented less accurately than 1.23456E-4.

Hexadecimal numbers are available for programmers to specify memory addresses for assembly language linkage inter alia. Note that there must be at least one hexadecimal digit present after the '#', otherwise an error (*ERROR* 51) will be generated.

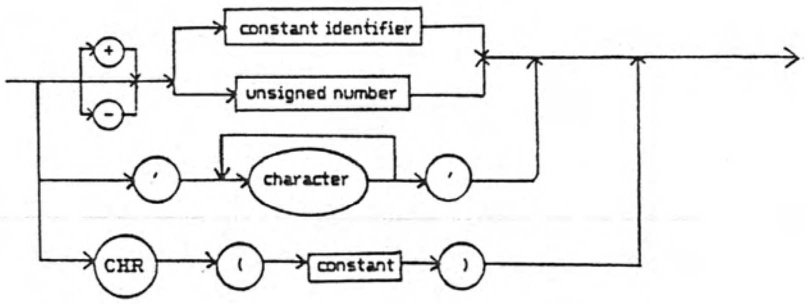
1.4 UNSIGNED CONSTANT.



Note that strings may not contain more than 255 characters. String types are ARRAY [1..N] OF CHAR where N is an integer between 1 and 255 inclusive. Literal strings should not contain end-of-line characters (CHR(13)) - if they do then an '*ERROR* 68' is generated.

The characters available are the full expanded set of ASCII values with 256 elements. To maintain compatibility with Standard Pascal the null character is not represented as ''; instead CHR(0) should be used.

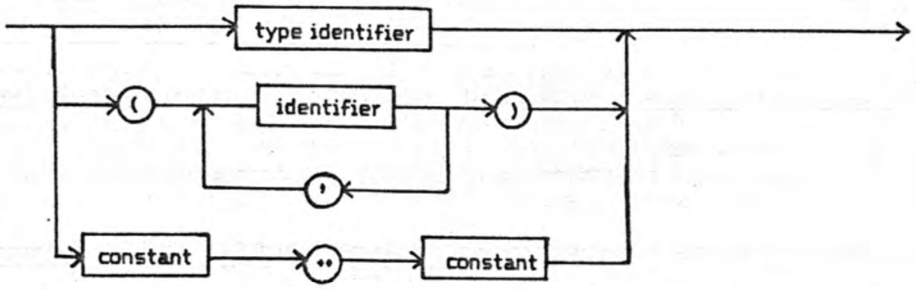
1.5 CONSTANT.



The non-standard CHR construct is provided here so that constants may be used for control characters. In this case the constant in parentheses must be of type integer.

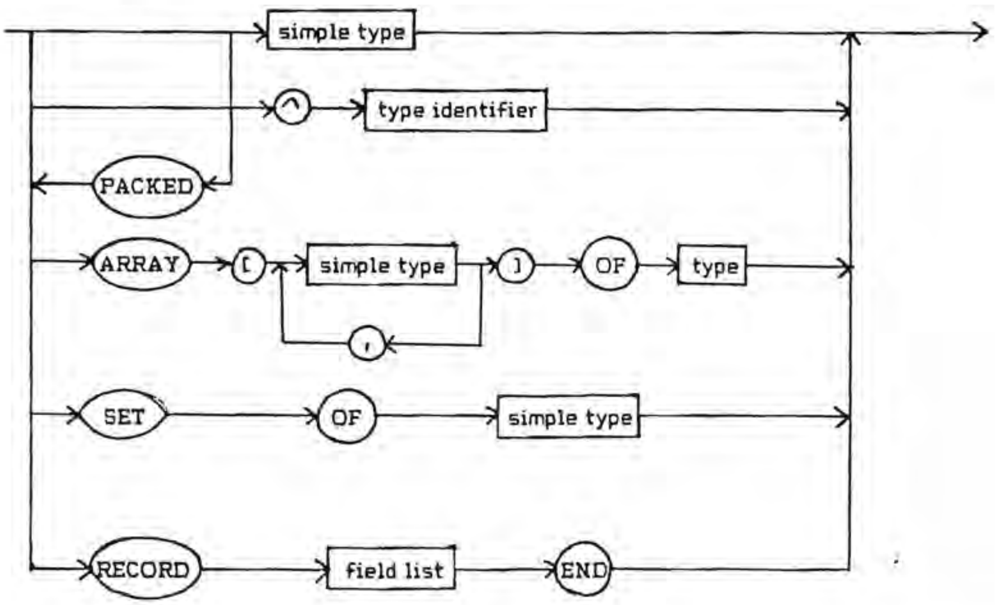
```
E.g.  CONST bs=CHR(8);
            cr=CHR(13);
```

1.6 SIMPLE TYPE.



Scalar enumerated types (identifier, identifier, ...) may not have more than 256 elements.

1.7 TYPE.



The reserved word PACKED is accepted but ignored since packing already takes place for arrays of characters etc. The only case in which the packing of arrays would be advantageous is with an array of Booleans - but this is more naturally expressed as a set when packing is required.

1.7.1 ARRAYs and SETs.

The base type of a set may have up to 256 elements. This enables SETs of CHAR to be declared together with SETs of any user enumerated type. Note, however, that only subranges of integers can be used as base types. All subsets of integers are treated as sets of 0..255.

Full arrays of arrays, arrays of sets, records of sets etc. are supported.

Two ARRAY types are only treated as equivalent if their definition stems from the same use of the reserved word ARRAY. Thus the following types are not equivalent:

```
TYPE
        tablea = ARRAY[1..100] OF  INTEGER;
        tableb = ARRAY[1..100] OF  INTEGER;
```

So a variable of type tablea may not be assigned to a variable of type tableb. This enables mistakes to be detected such as assigning two tables representing different data. The above restriction does not hold for the special case of arrays of a string type, since arrays of this type are always used to represent similar data. See Section 1.18 for a further discussion of this 'strong TYPEing'.

1.7.2 Pointers.

Hisoft Pascal allows the creation of dynamic variables through the use of the Standard Procedure NEW (see Section 2). A dynamic variable, unlike a static variable which has memory space allocated for it throughout the block in which it is declared, cannot be referenced directly through an identifier since it does not have an identifier; instead a pointer variable is used. This pointer variable, which is a static variable, contains the address of the dynamic variable and the dynamic variable itself is accessed by including a '^' after the pointer variable. Examples of the use of pointer types can be studied in Appendix 7.

There are some restrictions on the use of pointers within Hisoft Pascal. These are as follows:

Pointers to types that have not been declared are not allowed. This does not prevent the construction of linked list structures since type definitions may contain pointers to themselves e.g.

```
TYPE
    item = RECORD
           value : INTEGER;
           next  : ^item
          END;

    link = ^item;
```

Pointers to pointers are not allowed.

Pointers to the same type are regarded as equivalent e.g.

```
VAR
    first : link;
    current : ^item;
```

The variables first and current are equivalent (i.e. structural equivalence is used) and may be assigned to each other or compared.

The predefined constant NIL is supported and when this is assigned to a pointer variable then the pointer variable is deemed to contain no address.

1.7.4 RECORDs.

The implementation of RECORDs, structured variables composed of a fixed number of constituents called fields, within Hisoft Pascal is as Standard Pascal except that the variant part of the field list is not supported.

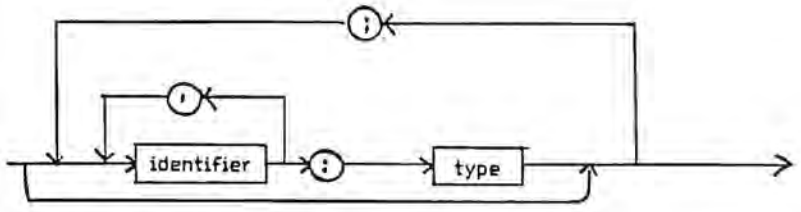
Two RECORD types are only treated as equivalent if their declaration stems from the same occurrence of the reserved word RECORD see Section 1.7.1 above.

The WITH statement may be used to access the different fields within a record in a more compact form.

RECORD declarations and WITH statements do not open a new scope. This means that you should not use the same field identifiers in two different record declarations or use the same name as a variable and a field identifier.

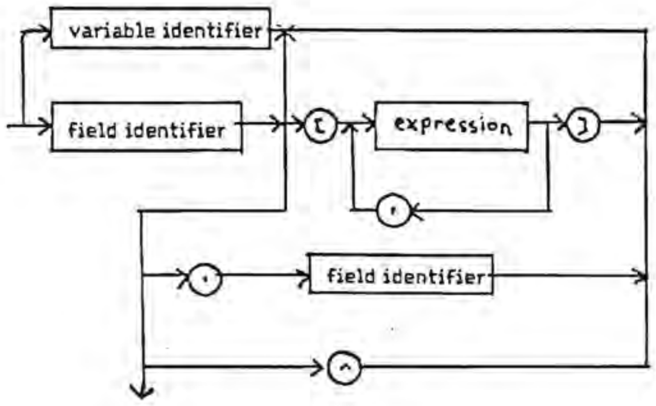
See Appendix 7 for an example of the use of WITH and RECORDs in general.

1.8 FIELD LIST.



Used in conjunction with RECORDs see Section 1.7.4 above and Appendix 7 for an example.

1.9 VARIABLE.



Two kinds of variables are supported within Hisoft Pascal; static and dynamic variables. Static variables are explicitly declared through VAR and memory is allocated for them during the entire execution of the block in which they were declared.

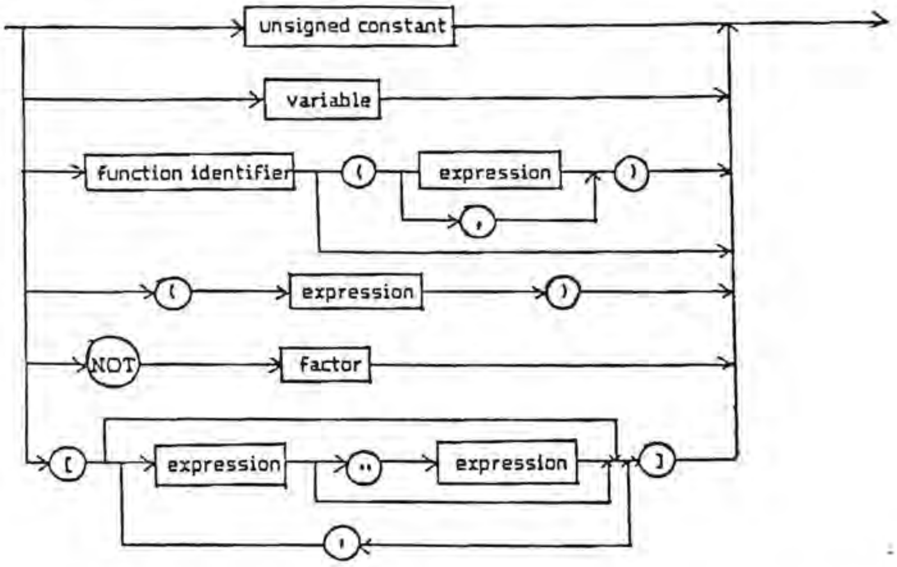
Dynamic variables, however, are created dynamically during program execution by the procedure NEW. They are not declared explicitly and cannot be referenced by an identifier. They are referenced indirectly by a static variable of type pointer, which contains the address of the dynamic variable.

See Section 1.7.2 and Section 2 for more details of the use of dynamic variables and Appendix 7 for an example.

When specifying elements of multi-dimensional arrays the programmer is not forced to use the same form of index specification in the reference as was used in the declaration.

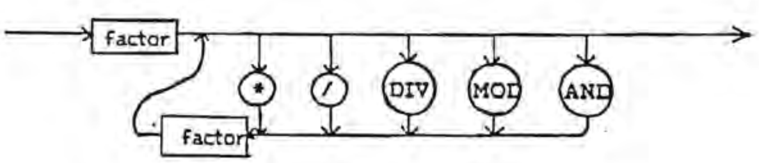
e.g. if variable a is declared as ARRAY[1..10] OF ARRAY[1..10] OF INTEGER then either a[1][1] or a[1,1] may be used to access element (1,1) of the array.

FACTOR.



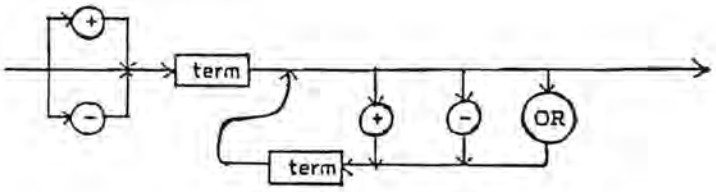
See EXPRESSION in Section 1.13 and FUNCTIONs in Section 3 for more details.

1.11 TERM.



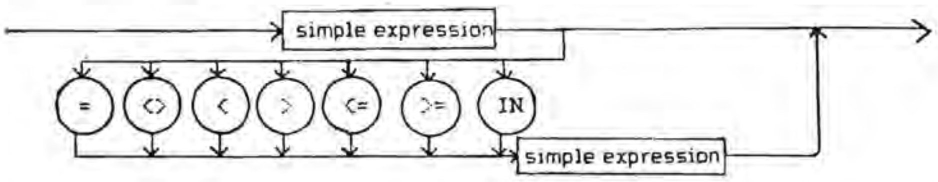
The lowerbound of a set is always zero and the set size is always the maximum of the base type of the set. Thus a SET OF CHAR always occupies 32 bytes (a possible 256 elements - one bit for each element). Similarly a SET OF 0..10 is equivalent to SET OF 0..255.

1.12 SIMPLE EXPRESSION.



The same comments made in Section 1.11 concerning sets apply to simple expressions.

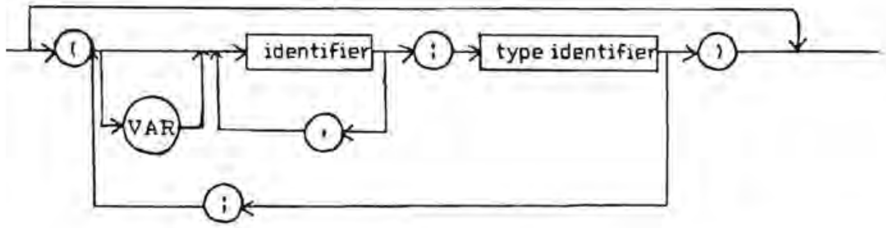
1.13 EXPRESSION.



When using IN, the set attributes are the full range of the type of the simple expression with the exception of integer arguments for which the attributes are taken as if [0..255] had been encountered.

The above syntax applies when comparing strings of the same length, pointers and all scalar types. Sets may be compared using >=, <=, <> or =. Pointers may only be compared using = and <>.

1.14 PARAMETER LIST.



A type identifier must be used following the colon - otherwise *ERROR* 44 will result.

Variable parameters as well as value parameters are fully supported.

Procedures and functions are not valid as parameters.

1.15 STATEMENT.

Refer to the syntax diagram on page 17.

Assignment statements:

See Section 1.7 for information on which assignment statements are illegal.

When assigning to subrange variables the value is not checked for being within the subrange for efficiency reasons.

CASE statements:

An entirely null case list is not allowed i.e. CASE OF END; will generate an error (*ERROR* 13).

The ELSE clause, which is an alternative to END, is executed if the selector ('expression' overleaf) is not found in one of the case lists ('constant' overleaf).

If the END terminator is used and the selector is not found then control is passed to the statement following the END.

FOR statements:

The control variable of a FOR statement may only be an unstructured variable, not a parameter. This is half way between the Jensen/Wirth and draft ISO standard definitions.

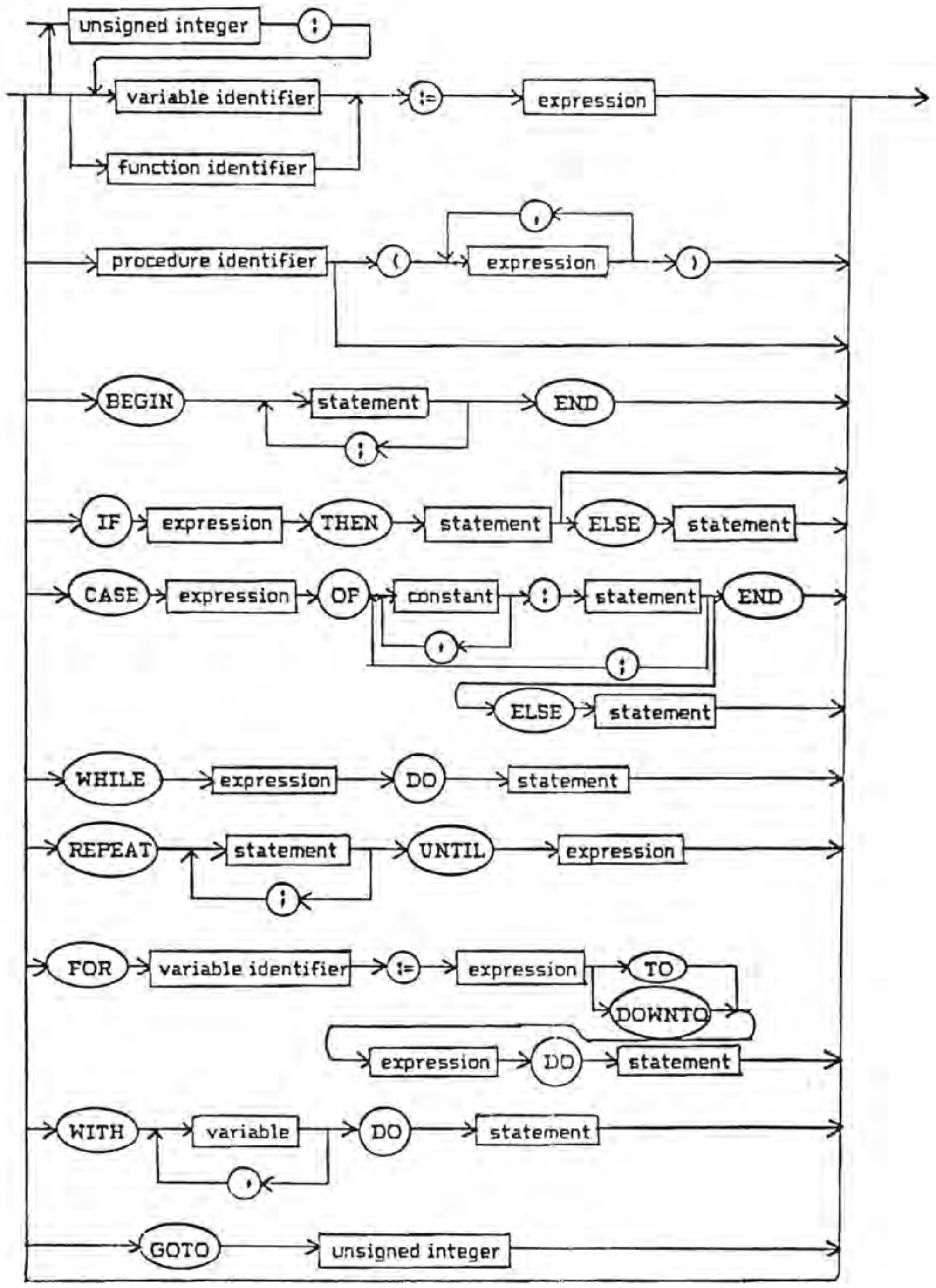
GOTO statements:

It is only possible to GOTO a label which is present in the same block as the GOTO statement and at the same level.

You may not use GOTO to jump out of a FOR statement.

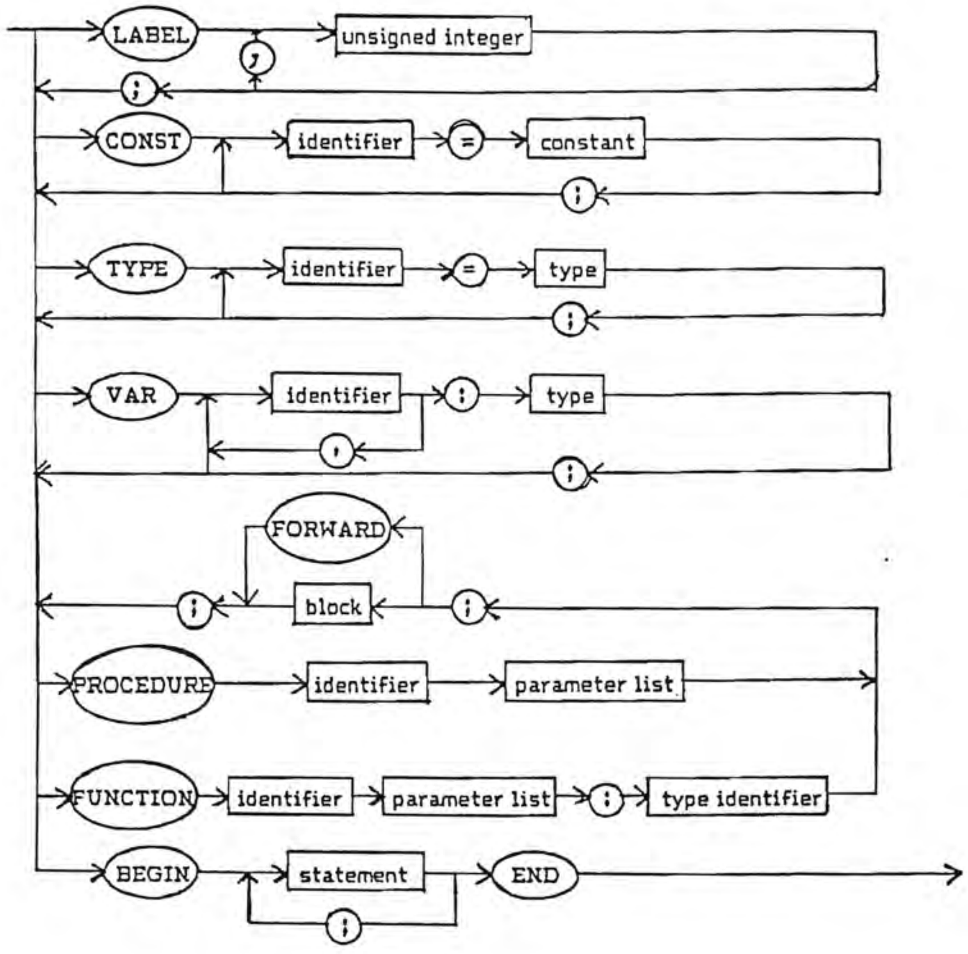
Labels must be declared (using the Reserved Word LABEL) in the block in which they are used; a label consists of at least one and up to four digits. When a label is used to mark a statement it must appear at the beginning of the statement and be followed by a colon - ':'.

STATEMENT.



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

1.16 BLOCK.


LABEL
,
unsigned integer
;
CONST
identifier
=
constant
;
TYPE
identifier
=
type
;
VAR
identifier
,
:
type
;
FORWARD
;
block
;
PROCEDURE
identifier
parameter list
FUNCTION
identifier
parameter list
:
type identifier
BEGIN
statement
;
END

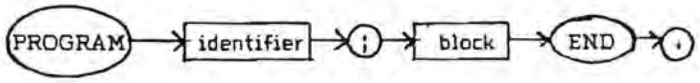
Forward References.

As in the Pascal User Manual and Report (Section 11.C.1) procedures and functions may be referenced before they declared through use of the Reserved Word FORWARD e.g.

```
PROCEDURE a(y:t) ; FORWARD;          {procedure a declared to be}
PROCEDURE b(x:t);                     {forward of this statement}
 BEGIN
 ....
 a(p);                                 {procedure a referenced.}
 ....
 END;
PROCEDURE a;                         {actual declaration of procedure a.}
 BEGIN
 ....
 b(q);
 ....
 END;
```

Note that the parameters and result type of the procedure a are declared along with FORWARD and are not repeated in the main declaration of the procedure. Remember, FORWARD is a Reserved Word.

1.17 PROGRAM.



Since files are not implemented there are no formal parameters of the program.

1.18 Strong TYPEing.

Different languages have different ways of ensuring that the user does not use an element of data in a manner which is inconsistent with its definition.

At one end of the scale there is machine code where no checks whatever are made on the TYPE of variable being referenced. Next we have a language like the Byte 'Tiny Pascal' in which character, integer and Boolean data may be freely mixed without generating errors. Further up the scale comes BASIC which distinguishes between numbers and strings and, sometimes, between integers and reals (perhaps using the '%' sign to denote integers). Then comes Pascal which goes as far as allowing distinct user-enumerated types. At the top of the scale (at present) is a language like ADA in which one can define different, incompatible numeric types.

There are basically two approaches used by Pascal implementations to strength of typing; structural equivalence or name equivalence. Hisoft Pascal uses name equivalence for RECORDs and ARRAYs. The consequences of this are clarified in Section 1.7 et al -- let it suffice to give an example here; say two variables are defined as follows:

```
VAR A : ARRAY['A'..'C'] OF INTEGER;
    B : ARRAY['A'..'C'] OF INTEGER;
```

then one might be tempted to think that one could write A:=B; but this would generate an error (*ERROR* 10) under Hisoft Pascal since two separate 'TYPE records' have been created by the above definitions. In other words, the user has not taken the decision that A and B should represent the same type of data. She/He could do this by:

```
VAR A,B : ARRAY['A'..'C'] OF INTEGER;
```

and now the user can freely assign A to B and vice versa since only one 'TYPE record' has been created.

Although on the surface this name equivalence approach may seem a little complicated, in general it leads to fewer programming errors since it requires more initial thought from the programmer.

SECTION 2 PREDEFINED IDENTIFIERS.

2.1 CONSTANTS.

MAXINT	The largest integer available i.e. 32767.
TRUE, FALSE	The constants of type Boolean.

2.2 TYPES.

INTEGER	See Section 1.3.
REAL	See Section 1.3.
CHAR	The full extended ASCII character set of 256 elements.
BOOLEAN	(TRUE,FALSE). This type is used in logical operations including the results of comparisons.

2.3 PROCEDURES AND FUNCTIONS.

2.3.1 Input and Output Procedures.

2.3.1.1 WRITE

The procedure WRITE is used to output data to the screen or printer.
When the expression to be written is simply of type character then WRITE(e) passes the 8 bit value represented by the value of the expression e to the screen or printer as appropriate.

Note:

CHR(8) (CTRL H) gives a destructive backspace on the screen.
CHR(12) (CTRL L) clears the screen or gives a new page on the printer.
CHR(13) (CTRL M) performs a carriage return <u>and</u> line feed.
CHR(16) (CTRL P) will normally direct output to the printer if the screen is in use or vice versa.

Generally though:

```
WRITE(P1,P2,.......Pn); is equivalent to:

BEGIN  WRITE(P1); WRITE(P2); .......; WRITE(Pn) END;
```

The write parameters P1,P2,......Pn can have one of the following forms:

```
<e> or <e:m> or <e:m:n> or <e:m:H>
```

where e, m and n are expressions and H is a literal constant.

We have 5 cases to examine:

1] e is of type integer: and either <e> or <e:m> is used.

The value of the integer expression e is converted to a character string with a trailing space. The length of the string can be increased (with leading spaces) by the use of m which specifies the total number of characters to be output. If m is not sufficient for e to be written or m is not present then e is written out in full, with a trailing space, and m is ignored. Note that, if m is specified to be the length of e <u>without</u> the trailing space then <u>no</u> trailing space will be output.

2] e is of type integer and the form <e:m:H> is used.

In this case e is output in hexadecimal. If m=1 or m=2 then the value (e MOD 16^m) is output in a width of exactly m characters. If m=3 or m=4 then the full value of e is output in hexadecimal in a width of 4 characters. If m>4 then leading spaces are inserted before the full hexadecimal value of e as necessary. Leading zeroes will be inserted where applicable. Examples:

```
WRITE(1025:m:H);

m=1   outputs: 1
m=2   outputs: 01
m=3   outputs: 0401
m=4   outputs: 0401
m=5   outputs: _0401
```

3] e is of type real. The forms <e>, <e:m> or <e:m:n> may be used.

The value of e is converted to a character string representing a real number. The format of the representation is determined by n.

If n is not present then the number is output in scientific notation, with a mantissa and an exponent. If the number is negative then a minus sign is output prior to the mantissa, otherwise a space is output. The number is always output to at least one decimal place up to a maximum of 5 decimal places and the exponent is always signed (either with a plus or minus sign). This means that the minimum width of the scientific representation is 8 characters; if the field width m is less than 8 then the full width of 12 characters will always be output. If m>=8 then one or more decimal places will be output up to a maximum of 5 decimal places (m=12). For m>12 leading spaces are inserted before the number. Examples:

```
WRITE(-1.23E 10:m);

m=7  gives: -1.23000E+10
m=8  gives: -1.2E+10
m=9  gives: -1.23E+10
m=10 gives: -1.230E+10
m=11 gives: -1.2300E+10
m=12 gives: -1.23000E+10
m=13 gives: _-1.23000E+10
```

If the form <e:m:n> is used then a fixed-point representation of the number e will be written with n specifying the number of decimal places to be output. No leading spaces will be output unless the field width m is sufficiently large. If n is zero then e is output as an integer. If e is too large to be output in the specified field width then it is output in scientific format with a field width of m (see above). Examples:

```
WRITE(1E2:6:2)     gives:  100.00
WRITE(1E2:8:2)     gives:  _100.00
WRITE(23.455:6:1)  gives:  _23.5
WRITE(23.455:4:2)  gives:  _2.34550E+01
WRITE(23.455:4:0)  gives:  _23
```

4] e is of type character or type string.

Either <e> or <e:m> may be used and the character or string of characters will be output in a minimum field width of 1 (for characters) or the length of the string (for string types). Leading spaces are inserted if m is sufficiently large.

5] e is of type Boolean.

Either <e> or <e:m> may be used and 'TRUE' or 'FALSE' will be output depending on the Boolean value of e , using a minimum field width of 4 or 5 respectively.

2.3.1.2 WRITELN

WRITELN outputs gives a newline. This is equivalent to WRITE(CHR(13)). Note that a linefeed is included.

WRITELN(P1,P2,........P3); is equivalent to:

BEGIN WRITE(P1,P2,.......P3); WRITELN END;

2.3.1.3 PAGE

The procedure PAGE is equivalent to WRITE(CHR(12)); and causes the video screen to be cleared or the printer to advance to the top of a new page.

2.3.1.4 READ

The procedure READ is used to access data from the keyboard. It does this through a buffer held within the runtimes - this buffer is initially empty (except for an end-of-line marker). We can consider that any accesses to this buffer take place through a text window over the buffer through which we can see one character at a time. If this text window is positioned over an end-of-line marker then before the read operation is terminated a new line of text will be read into the buffer from the keyboard. While reading in this line all the various control codes detailed in Section 0.0 will be recognised. Now:

READ(V1,V2,........Vn); is equivalent to:

BEGIN READ(V1); READ(V2); ....; READ(Vn) END;

where V1, V2 etc. may be of type character, string, integer or real.

The statement READ(V); has different effects depending on the type of V. There are 4 cases to consider:

i] V is of type character.

In this case READ(V) simply reads a character from the input buffer and assigns it to V. If the text window on the buffer is positioned on a line marker (a CHR(13) character) then the function EOLN will return the value TRUE and a new line of text is read in from the keyboard. When a read operation is subsequently performed then the text window will be positioned at the start of the new line.

Important note: Note that EOLN is TRUE at the start of the program. This means that if the first READ is of type character then a CHR(13) value will be returned followed by the reading in of a new line from the keyboard; a subsequent read of type character will return the first character from this new line, assuming it is not blank. See also the procedure READLN below.

2] V is of type string.

A string of characters may be read using READ and in this case a series of characters will be read until the number of characters defined by the string has been read or EOLN = TRUE. If the string is not filled by the read (i.e. if end-of-line is reached before the whole string has been assigned) then the end of the string is filled with null (CHR(0)) characters - this enables the programmer to evaluate the length of the string that was read.

The note concerning in 1] above also applies here.

3] V is of type integer.

In this case a series of characters which represent an integer as defined in Section 1.3 is read. All preceding blanks and end-of-line markers are skipped (this means that integers may be read immediately cf. the note in 1] above).

If the integer read has an absolute value greater than MAXINT (32767) then the runtime error 'Number too large' will be issued and execution terminated.

If the first character read, after spaces and end-of-line characters have been skipped, is not a digit or a sign ('+' or '-') then the runtime error 'Number expected' will be reported and the program aborted.

4] V is of type real.

Here, a series of characters representing a real number according to the syntax of Section 1.3 will be read.

All leading spaces and end-of-line markers are skipped and, as for integers above, the first character afterwards must be a digit or a sign. If the number read is too large or too small (see Section 1.3) then an 'Overflow' error will be reported, if 'E' is present without a following sign or digit then 'Exponent expected' error will be generated and if a decimal point is present without a subsequent digit then a 'Number expected' error will be given.

Reals, like integers, may be read immediately; see 1] and 3] above.

2.3.1.5 READLN

READLN(V1,V2,.......Vn); is equivalent to: BEGIN READ(V1,V2,.......Vn); READLN END;

READLN simply reads in a new buffer from the keyboard; while typing in the buffer you may use the various control functions detailed in Section 0.0. Thus EOLN becomes FALSE after the execution of READLN unless the next line is blank.

READLN may be used to skip the blank line which is present at the beginning of the execution of the object code i.e. it has the effect of reading in a new buffer. This will be useful if you wish to read a component of type character at the beginning of a program but it is not necessary if you are reading an integer or a real (since end-of-line markers are skipped) or if you are reading characters from subsequent lines.

2.3.2 Input Functions.

2.3.2.1 EOLN

The function EOLN is a Boolean function which returns the value TRUE if the next char to be read would be an end-of-line character (CHR(13)). Otherwise the function returns the value FALSE.

2.3.2.2 INCH

The function INCH causes the keyboard of the computer to be scanned and, if a key has been pressed, returns the character represented by the key pressed. If no key has been pressed then CHR(0) is returned. The function therefore returns a result of type character. Note that you should always disable keyboard checks when using INCH i.e. always specify compiler option $C-.

2.3.3 Transfer Functions.

2.3.3.1 TRUNC(X)

The parameter X must be of type real or integer and the value returned by TRUNC is the greatest integer less than or equal to X if X is positive or the least integer greater than or equal to X if X is negative. Examples:

TRUNC(-1.5) returns -1   TRUNC(1.9) returns 1

2.3.3.2 ROUND(X)

X must be of type real or integer and the function returns the 'nearest' integer to X (according to standard rounding rules). Examples:

```
ROUND(-6.5) returns -6     ROUND(11.7) returns 12
ROUND(-6.51) returns -7    ROUND(23.5) returns 24
```

2.3.3.3 ENTIER(X)

X must be of type real or integer - ENTIER returns the greatest integer less than or equal to X, for all X. Examples:

```
ENTIER(-6.5) returns -7    ENTIER(11.7) returns 11
```

Note: ENTIER is not a Standard Pascal function but is the equivalent of BASIC's INT. It is useful when writing fast routines for many mathematical applications.

2.3.3.4 ORD(X)

X may be of any scalar type except real. The value returned is an integer representing the ordinal number of the value of X within the set defining the type of X.

If X is of type integer then ORD(X) = X ; this should normally be avoided.

Examples:

```
ORD('a') returns 97     ORD('@') returns 64
```

2.3.3.5 CHR(X)

X must be of type integer. CHR returns a character value corresponding to the ASCII value of X. Examples:

```
CHR(49) returns '1'     CHR(91) returns '['
```

2.3.4 Arithmetic Functions.

In all the functions within this sub-section the parameter X must be of type real or integer.

2.3.4.1 ABS(X)

Returns the absolute value of X (e.g. ABS(-4.5) gives 4.5). The result is of the same type as X.

2.3.4.2 SQR(X)

Returns the value X*X i.e. the square of X. The result is of the same type as X.

2.3.4.3 SQRT(X)

Returns the square root of X – the returned value is always of type real. A 'Maths Call Error' is generated if the argument X is negative.

2.3.4.4 FRAC(X)

Returns the fractional part of X: FRAC(X) = X – ENTIER(X).

As with ENTIER this function is useful for writing many fast mathematical routines. Examples:

FRAC(1.5) returns 0.5 FRAC(-12.56) returns 0.44

2.3.4.5 SIN(X)

Returns the sine of X where X is in radians. The result is always of type real.

2.3.4.6 COS(X)

Returns the cosine of X where X is in radians. The result is of type real.

2.3.4.7 TAN(X)

Returns the tangent of X where X is in radians. The result is always of type real.

2.3.4.8 ARCTAN(X)

Returns the angle, in radians, whose tangent is equal to the number X. The result is of type real.

2.3.4.9 EXP(X)

Returns the value e^X where e = 2.71828. The result is always of type real.

2.3.4.10 LN(X)

Returns the natural logarithm (i.e. to the base e) of X. The result is of type real. If X <= 0 then a 'Maths Call Error' will be generated.

2.3.5 Further Predefined Procedures.

2.3.5.1 NEW(p)

The procedure NEW(p) allocates space for a dynamic variable. The variable p is a pointer variable and after NEW(p) has been executed p contains the address of the newly allocated dynamic variable. The type of the dynamic variable is the same as the type of the pointer variable p and this can be of any type.

To access the dynamic variable p^ is used - see Appendix 4 for an example of the use of pointers to reference dynamic variables.

To re-allocate space used for dynamic variables use the procedures MARK and RELEASE (see below).

2.3.5.2 MARK(v1)

This procedure saves the state of the dynamic variable heap to be saved in the pointer variable v1. The state of the heap may be restored to that when the procedure MARK was executed by using the procedure RELEASE (see below).

The type of variable to which v1 points is irrelevant, since v1 should only be used with MARK and RELEASE never NEW.

For an example program using MARK and RELEASE see Appendix 4.

2.3.5.3. RELEASE(v1)

This procedure frees space on the heap for use of dynamic variables. The state of the heap is restored to its state when MARK(v1) was executed - thus effectively destroying all dynamic variables created since the execution of the MARK procedure and as such it should be used with great care.

See above and Appendix 4 for more details.

2.3.5.4 INLINE(C1,C2,C3,.........)

This procedure allows Z80 machine code to be inserted within the Pascal program; the values (C1 MOD 256, C2 MOD 256, C3 MOD 256, ...) are inserted in the object program at the current location counter address held by the compiler. C1, C2, C3 etc. are integer constants of which there can be any number. Refer to Appendix 4 for an example of the use of INLINE.

2.3.5.5 USER(V)

USER is a procedure with one integer argument V. The procedure causes a call to be made to the memory address given by V. Since Hisoft Pascal holds integers in two's complement form (see Appendix 3) then in order to refer to addresses greater than #7FFF (32767) negative values of V must be used. For example #C000 is -16384 and so USER(-16384); would invoke a a call to the memory address #C000. However, when using a constant to refer to a memory address, it is more convenient to use hexadecimal.

The routine called should finish with a Z80 RET instruction (#C9) and must preserve the IX register.

2.3.5.6 HALT

This procedure causes program execution to stop with the message 'Halt at PC=XXXX' where XXXX is the hexadecimal memory address of the location where the HALT was issued. Together with a compilation listing, HALT may be used to determine which of two or more paths through a program are taken. This will normally be used during de-bugging.

2.3.5.7 POKE(X,V)

POKE stores the expression V in the comouter's memory starting from the memory address X. X is of tvpe integer and V can be of any type except SET. See Section 2.3.5.5 above for a discussion of the use of integers to represent memory addresses. Examples:

```
        POKE(#6000,'A') places #41 at location #6000.
POKE(-16384,3.6E3) places 00 0B 80 70 (hex) at location #C000.
```

2.3.5.8 TOUT (NAME,START,SIZE)

TOUT is the procedure which is used to save variables on tape. The first parameter is of type ARRAY[1..8] OF CHAR and is the name of the file to be saved. SIZE bytes of memory are dumped starting at the address START. Both these parameters are of type INTEGER.

E.g. to save the variable V to tape under the name 'VAR   ' use:

```
TOUT('VAR     ',ADDR(V),SIZE(V))
```

The use of actual memory addresses gives the user far more flexiblity than just the ability to save arrays. For example if a system has a memory mapped screen, entire screenfuls may be saved directly. See Appendix 4 for an example of the use of TOUT.

2.3.5.9 TIN (NAME,START)

This procedure is used to load, from tape, variables etc. that have been saved using TOUT. NAME is of type ARRAY[1..8] of CHAR and START is of type INTEGER. The tape is searched for a file called NAME which is then loaded at memory address START. The number of bytes to load is taken from the tape (saved on the tape by TOUT).

E.g. to load the variable saved in the example in Section 2.3.5.8 above use:

```
TIN('VAR     ',ADDR(V))
```

See Appendix 4 for an example of the use of TIN.

2.3.5.10 OUT(P,C)

This procedure is used to directly access the Z80's output ports without using the procedure INLINE. The value of the integer parameter P is loaded in to the BC register, the character parameter C is loaded in to the A register and the assembly instruction OUT (C),A is executed.

E.g. OUT(1,'A') outputs the character 'A' to the Z80 port 1.

2.3.6 Further Predefined Functions.

2.3.6.1 RANDOM

This returns a pseudo-random number between 0 and 255 inclusive. Although this routine is very fast it gives poor results when used repeatedly within loops that do not contain I/O operations.

If the user requires better results than this function yields then he/she should write a routine (either in Pascal or machine code) tailored to the particular application.

2.3.6.2 SUCC(X)

X may be of any scalar type except real and SUCC(X) returns the successor of X. Examples:

SUCC('A') returns 'B'   SUCC('5') returns '6'

2.3.6.3 PRED(X)

X may be of any scalar type except real; the result of the function is the predecessor of X. Examples:

PRED('j') returns 'i'   PRED(TRUE) returns FALSE

2.3.6.4 ODD(X)

X must be of type integer. ODD returns a Boolean result which is TRUE if X is odd and FALSE if X is even.

2.3.6.6 ADDR(V)

This function takes a variable identifier of any type as a parameter and returns an integer result which is the memory address of the variable identifier V. For information on how variables are held, at runtime, within Hisoft Pascal see Appendix 3. For an example of the use of ADDR see Appendix 4.

2.3.6.7 PEEK(X,T)

The first parameter of this function is of type integer and is used to specify a memory address (see Section 2.3.5.5). The second argument is a type; this is the result type of the function.

PEEK is used to retrieve data from the memory of the computer and the result may be of any type.

In all PEEK and POKE (the opposite of PEEK) operations data is moved in Hisoft Pascal's own internal representation detailed in Appendix 3. For example: if the memory from #5000 onwards contains the values: 50 61 73 63 61 6C (in hexadecimal) then:

```
WRITE(PEEK(#5000,ARRAY[1..6] OF CHAR)) gives 'Pascal'
WRITE(PEEK(#5000,CHAR)) gives 'P'
WRITE(PEEK(#5000,INTEGER)) gives 24912
WRITE(PEEK(#5000,REAL)) gives 2.46227E-29
```

see Appendix 3 for more details on the representation of types within Hisoft Pascal.

2.3.6.7 SIZE(V)

The parameter of this function is a variable. The integer result is the amount of storage taken up by that variable, in bytes.

2.3.6.8 INP(P)

INP is used to access the Z80's ports directly without using the procedure INLINE. The value of the integer parameter P is loaded into be BC register and the character result of the function is obtained by executing the assembly language instruction IN A,(C).

SECTION 3 COMMENTS AND COMPILER OPTIONS.

3.1 Comments.

A comment may occur between any two reserved words, numbers, identifiers or special symbols - see Appendix 2. A comment starts with a '{' character or the '(* ' character pair. Unless the next character is a '$' all characters are ignored until the next '}' character or '*)' character pair. If a '$' was found then the compiler looks for a series of compiler options (see below) after which characters are skipped until a '}' or '*)' is found.

3.2 Compiler Options.

Compiler options are included in the program between comment brackets and are the first option in the list is prefaced by a dollar symbol '$'.

Example:

(*$C-,A-*)   to turn keyboard and array checks OFF.

The following options are available:

Option L:

Controls the listing of the program text and object code address by the compiler.

If L+ then a full listing is given.
If L- then lines are only listed when an error is detected.

DEFAULT: L+

Option O:

Controls whether certain overflow checks are made. Integer multiply and divide and all real arithmetic operations are always checked for overflow.

If O+ then checks are made on integer addition and subtraction.

If O- then the above checks are not made.

DEFAULT: O+

Option C:

Controls whether or not keyboard checks are made during object code program execution. If C+ then if CC is pressed then execution will return to with a HALT message - see Section 2.3.5.6.

This check is made at the beginning of all loops, procedures and functions. Thus the user may use this facility to detect which loop etc. is not terminating correctly during the debugging process. It should certainly be disabled if you wish the object program to run quickly.

If C- then the above check is not made.

DEFAULT: C+

Option S:

Controls whether or not stack shecks are made.

If S+ then, at the beginning of each procedure and function call, a check is made to see if the stack will probably overflow in this block. If the runtime stack overflows the dynamic variable heap or the program then the message 'Out of RAM at PC=XXXX' is displayed and execution aborted. Naturally this is not foolproof; if a procedure has a large amount of stack usage within itself then the program may 'crash'. Alternatively, if a function contains very little stack usage while utilising recursion then it is possible for the function to be halted unnecessarily.

If S- then no stack checks are performed.

DEFAULT: S+

Option A:

Controls whether checks are made to ensure that array indices are within the bounds specified in the array's declaration.

If A+ and an array index is too high or too low then the message 'Index too high' or 'Index too low' will be displayed and the program execution halted.

If A- then no such checks are made.

DEFAULT: A+

Option I:

When using 16 bit 2's complement integer arithmetic, overflow occurs when performing a >, <, >=, or <= operation if the arguments differ by more than MAXINT (32767). If this occurs then the result of the comparison will be incorrect. This will not normally present any difficulties; however, should the user wish to compare such numbers, the use of I+ ensures that the results of the comparison will be correct. The equivalent situation may arise with real arithmetic in which case an overflow error will be issued if the arguments differ by more than approximately 3.4E38 ; this cannot be avoided.

If I- then no check for the result of the above comparisons is made.

DEFAULT: I-

Option P:

If the P option is used the device to which the compilation listing is sent is changed i.e. if the video screen was being used the printer is used and vice versa. Note that this option is not followed by a '+' or '-'.

DEFAULT: The video screen is used.

Option F:

This option letter must be followed by a space and then an eight character filename. If the filename has less than eight characters it should be padded with spaces.

The presence of this option causes inclusion of Pascal source text from the specified file from the end of the current line - useful if the programmer wishes to build up a 'library' of much-used procedures and functions on tape and then include them in particular programs.

The program should be saved using the built-in editor's 'P' command. On most systems the list option L- should be used - otherwise the compiler will not compile fast enough.

Example: {$F MATRIX   include the text from a tape file MATRIX};

When writing very large programs there may not be enough room in the computer's memory for the source and object code to be present at the same time. It is however possible to compile such programs by saving them to tape and using the 'F' option - then only 128 bytes of the source are in RAM at any one time, leaving much more room for the object code.

This option may not be nested.

The compiler options may be used selectively. Thus debugged sections of code may be speeded up and compacted by turning the relevant checks off whilst retaining checks on untested pieces of code.

SECTION 4 THE INTEGRAL EDITOR.

4.1 Introduction to the Editor.

The editor supplied with all versions of Hisoft Pascal is a simple, line-based editor designed to work with all Z80 operating systems while maintaining ease of use and the ability to edit programs quickly and efficiently.
Text is held in memory in a compacted form; the number of leading spaces in a line is held as one character at the beginning of the line and all Hisoft Pascal Reserved Words are tokenised into one character. This leads to a typical reduction in text size of 25%.

NOTE: throughout this section the DELETE key is referred to instead of the control code CH. It appears more natural to do this.

The editor is entered automatically when Hisoft Pascal is loaded from tape and displays the message:

```
Copyright Hisoft 1982,3,4
All rights reserved
```

followed by the editor prompt '>'.

In response to the prompt you may enter a command line of the following format:

```
C N1, N2, S1, S2
```

followed by <ENTER> where:

C	is the command to be executed (see Section 4.2 below).
N1	is a number in the range 1 – 32767 inclusive.
N2	is a number in the range 1 – 32767 inclusive.
S1	is a string of characters with a maximum length of 20.
S2	is a string of characters with a maximum length of 20.

The comma is used to separate the various arguments (although this can be changed – see the 'S' command) and spaces are ignored, except within the strings. None of the arguments are mandatory although some of the commands (e.g. the 'D'elete command) will not proceed without N1 and N2 being specified. The editor remembers the previous numbers and strings that you entered and uses these former values, where applicable, if you do not specify a particular argument within the command line. The values of N1 and N2 are initially set to 10 and the strings are initially empty. If you enter an illegal command line such as F-1,100,HELLO then the line will be ignored and the message 'Pardon?' displayed – you should then retype the line correctly e.g. F1,100,HELLO. This error message will also be displayed if the length of S2 exceeds 20; if the length of S1 is greater than 20 then any excess characters are ignored.

Commands may be entered in upper or lower case.

While entering a command line, all the relevant control functions described in Section 0.0 may be used e.g. CX to delete to the beginning of the line.

The following sub-section details the various commands available within the editor – note that wherever an argument is enclosed by the symbols '< >' then that argument must be present for the command to proceed.

4.2 The Editor Commands.

4.2.1 Text Insertion.

Text may be inserted into the textfile either by typing a line number, a space and then the required text or by use of the 'I' command. Note that if you type a line number followed by <ENTER> (i.e. without any text) then that line will be deleted from the text if it exists. Whenever text is being entered then the control functions CX (delete to the beginning of the line), CI (go to the next tab position), CC (return to the command loop) and CP (toggle the printer) may be employed. The DELETE (or BACKSPACE) key will produce a destructive backspace (but not beyond the beginning of the text line). Text is entered into an internal buffer within Hisoft Pascal and if this buffer should become full then you will be prevented from entering any more text - you must then use DELETE or CX to free space in the buffer.

Command: I n,m

Use of this command gains entry to the automatic insert mode: you are prompted with line numbers starting at n and incrementing in steps of m. You enter the required text after the displayed line number, using the various control codes if desired and terminating the text line with <ENTER>. To exit from this mode use the control function CC (see Section 0.0 and the relevant Implementation Note).

If you enter a line with a line number that already exists in the text then the existing line will be deleted and replaced with the new line, after you have pressed <ENTER>. If the automatic incrementing of the line number produces a line number greater than 32767 then the Insert mode will exit automatically.

If, when typing in text, you get to the end of a screen line without having entered 128 characters (the buffer size) then the screen will be scrolled up and you may continue typing on the next line - an automatic indentation will be given to the text so that the line numbers are effectively separated from the text.

4.2.2 Text Listing.

Text may be inspected by use of the 'L' command; the number of lines displayed at any one time during the execution of this command is fixed initially (see your Implementation Note) but may be changed through use of the 'K' command.

Command: L n,m

This lists the current text to the display device from line number n to line number m inclusive. The default value for n is always 1 and the default value for m is always 32767 i.e. default values are not taken from previously entered arguments. To list the entire textfile simply use 'L' without any arguments. Screen lines are formatted with a left hand margin so that the line number is clearly displayed. The number of screen lines listed on the display device may be controlled through use of the 'K' command - after listing a certain number of lines the list will pause (if not yet at line number m), hit control function CC to return to the main editor loop or any other key to continue the listing.

Command: K n

'K' sets the number of screen lines to be listed to the display device before the display is paused as described in 'L' above. The value (n MOD 256) is computed and stored. For example use K5 if you wish a subsequent 'L'ist to produce five screen lines at a time.

4.2.3 Text Editing.

Once some text has been created there will inevitably be a need to edit some lines. Various commands are provided to enable lines to be amended, deleted, moved and renumbered:

Command: D <n,m>

All lines from n to m inclusive are deleted from the textfile. If m<n or less than two arguments are specified then no action will be taken; this is to help prevent careless mistakes. A single line may be deleted by making m=n ; this can also be accomplished by simply typing the line number followed by <ENTER>.

Command: M n,m

This causes the text at line n to be entered at line m deleting any text that already exists there. Note that line n is left alone. So this command allows you to 'M'ove a line of text to another position within the textfile. If line number n does not exist then no action is taken.

Command: N <n,m>

Use of the 'N' command causes the textfile to be renumbered with a first line number of n and in line number steps of m. Both n and m must be present and if the renumbering would cause any line number to exceed 32767 then the original numbering is retained.

Command: F n,m,f,s

The text existing within the line range $n < x < m$ is searched for an occurrence of the string f - the 'find' string. If such an occurrence is found then the relevant text line is displayed and the Edit mode is entered - see below. You may then use commands within the Edit mode to search for subsequent occurrences of the string f within the defined line range or to substitute the string s (the 'substitute' string) for the current occurrence of f and then search for the next occurrence of f; see below for more details.

Note that the line range and the two strings may have been set up previously by any other command so that it may only be necessary to enter 'F' to initiate the search - see the example in Section 4.3 for clarification.

Command: E n

Edit the line with line number n. If n does not exist then no action is taken; otherwise the line is copied into a buffer and displayed on the screen (with the line number), the line number is displayed again underneath the line and the Edit mode is entered. All subsequent editing takes place within the buffer and not in the text itself; thus the original line can be recovered at any time.

In this mode a pointer is imagined moving through the line (starting at the first character) and various sub-commands are supported which allow you to edit the line. The sub-commands are:

' ' (space) - increment the text pointer by one i.e. point to the next character in the line. You cannot step beyond the end of the line.

DELETE (or BACKSPACE) - decrement the text pointer by one to point at the previous character in the line. You cannot step backwards beyond the first character in the line.

CI (control function) - step the text pointer forwards to the next tab position but not beyond the end of the line.

<ENTER> - end the edit of this line keeping all the changes made.

Q - quit the edit of this line i.e. leave the edit ignoring all the changes made and leaving the line as it was before the edit was initiated.

R - reload the edit buffer from the text i.e. forget all changes made on this line and restore the line as it was originally.

L - list the rest of the line being edited i.e. the remainder of the line beyond the current pointer position. You remain in the Edit mode with the pointer re-positioned at the start of the line.

K - kill (delete) the character at the current pointer position.

Z - delete all the characters from (and including) the current pointer position to the end of the line.

F - find the next occurrence of the 'find' string previously defined within a command line (see the 'F' command above). This sub-command will automatically exit the edit on the current line (keeping the changes) if it does not find another occurrence of the 'find' string in the current line. If an occurrence of the 'find' string is detected in a subsequent line (within the previously specified line range) then the Edit mode will be entered for the line in which the string is found. Note that the text pointer is always positioned at the start of the found string after a successful search.

S - substitute the previously defined 'substitute' string for the currently found occurrence of the 'find' string and then perform the sub-command 'F' i.e. search for the next occurrence of the 'find' string. This, together with the above 'F' sub-command, is used to step through the textfile optionally replacing occurrences of the 'find' string with the 'substitute' string - see Section 4.3 for an example.

I – insert characters at the current pointer position. You will remain in this sub-mode until you press <ENTER> – this will return you to the main Edit mode with the pointer positioned after the last character that you inserted. Using DELETE (or BACKSPACE) within this sub-mode will cause the character to the left of the pointer to be deleted from the buffer while the use of CI (control function) will advance the pointer to the next tab position, inserting spaces.

X – this advances the pointer to the end of the line and automatically enters the insert sub-mode detailed above.

C – change sub-mode. This allows you to overwrite the character at the current pointer position and then advances the pointer by one. You remain in the change sub-mode until you press <ENTER> whence you are taken back to the Edit mode with the pointer positioned after the last character you changed. DELETE (or BACKSPACE) within this sub-mode simply decrements the pointer by one i.e. moves it left while CI has no effect.

4.2.4 Tape Commands.

Text may be saved to tape or loaded from tape using the commands 'P' and 'G':

Command: P n,m,s

The line range defined by n < x < m is saved to tape in Hisoft Pascal format under the filename specified by the string s. Remember that these arguments may have been set by a previous command. Before entering this command make sure that your tape recorder is switched on and in RECORD mode. While the text is being saved the message 'Busy..' is displayed.

Command: G,,s

The tape is searched for a file in Hisoft Pascal tape format and with a filename of s. While the search is taking place the message 'Searching..' will be displayed. If a valid Hisoft Pascal tape file is found but has the wrong filename then the message 'Found' followed by the filename that was found on the tape is displayed and the search continued. Once the correct filename is found then 'Found' will appear on the list device and the file will be loaded into memory. If an error is detected during the load then an error message is displayed and the load aborted. If this happens you must rewind the tape, press PLAY and type 'G' again.

If the string s is empty then the first Hisoft Pascal file on the tape will be loaded, regardless of its filename.

While searching of the tape is going on you may abort the load by holding CC down; this will interrupt the load and return to the main editor loop.

Note that if any textfile is already present then the textfile that is loaded from tape will be appended to the existing file and the whole file will be renumbered starting with line 1 in steps of 1.

ZX Spectrum owners should consult their Implementation Note for details of using tape and Microdrives with Hisoft Pascal.

4.2.5 Compiling and Running from the Editor.

Command: C n

This causes the text starting at line number n to be compiled. If you do not specify a line number then the text will be compiled from the first existing line. For further details see Section 0.2.

Command: R

The previously compiled object code will be executed, but only if the source has not been expanded in the meantime - see Section 0.2 for more detail.

Command: T n

This is the 'T'ranslate command. The current source is compiled from line n (or from the start if n is omitted) and, if the compilation is successful, you will be prompted with 'Ok?': if you answer 'Y' to this prompt then the object code produced by the compilation will be moved to the end of the runtimes (destroying the compiler) and then the runtimes and the object code will be dumped out to tape with a filename equal to that previously defined for the 'find' string. You may then, at a later stage, load this file into memory (see your Implementation Note), whereupon it will automatically execute the object program. Note that the object code is located at and moved to the end of the runtimes so that, after a 'T'ranslate you will need to reload the compiler - however this should present no problems since you are only likely to 'T'ranslate a program when it is fully working.i

If you decide not to continue with the dump to tape then simply type any character other than 'Y' to the 'Ok?' prompt; control is returned to the editor which will still function perfectly since the object code was not moved.

4.2.6 Other Commands.

Command: B

This simply returns control to the operating system. For details of how to re-enter the package see your Implementation Note.

Command: O n,m

Remember that text is held in memory in a tokenised form with leading spaces shortened into a one character count and all Hisoft Pascal Reserved Words reduced to a one character token. However if you have somehow got some text in memory, perhaps from another editor, which is not tokenised then you can use the 'O' command to tokenise it for you. Text is read into a buffer in an expanded form and then put back into the file in a tokenised form - this may of course take a little time to perform. A line range must be specified, or the previously entered values will be assumed.

Command: S,,d

This command allows you to change the delimiter which is taken as separating the arguments in the command line. On entry to the editor the comma ',' is taken as the delimiter; this may be changed by the use of the 'S' command to the first character of the specified string d. Remember that once you have defined a new delimiter it must be used (even within the 'S' command) until another one is specified.

Note that the separator may not be a space.

Command: V

The 'V' command takes no arguments and displays the current default values of the line range and the two strings. The line range is shown first followed by the two strings which may be empty. Remember that certain editor command (e.g. D and N) do not use these defaults but must have values specified on the command line.

Command: X

This simply displays the end address of the compiler, in hexadecimal. This information will be useful when making a back-up copy of the compiler since to make a back-up copy you should save code from the start of the compiler up to the end of the compiler. See your Implementation Note for more details.

4.3 An Example of the use of the Editor.

Let us assume that you have typed in the following program (using I10,10):

```
 10 PROGRAM BUBBLESORT
 20 CONST
 30   Size = 2000;
 40 VAR
 50   Numbers : ARRAY [1..Size] OF INTEGER;
 60   I, Temp : INTEGER;
 70 BEGIN
 80   FOR I:=1 TO Size DO Number[I] := RANDOM;
 90   REPEAT
100     FOR I:=1 TO Size DO
110     Noswaps := TRUE;
120     IF Number[I] > Number[I+1] THEN
130       BEGIN
140         Temp := Number[I];
150         Number[I] := Number[I+1];
160         Number[I+1] := Temp;
170         Noswaps := FALSE
180       END
190   UNTIL Noswapss;
200   FOR I:=1 TO Size DO  WRITE(Number[I]:4);
210 END.
```

This program has a number of errors which are as follows:

Line 10	Missing semi-colon.
Line 30	Not really an error but say we want a size of 100.
Line 100	Size should be Size-1.
Line 110	This should be at line 95 instead.
Line 190	Noswapss should be Noswaps.

Also the variable Numbers has been declared but all references are to Number. Finally the BOOLEAN variable Noswaps has not been declared.

To put all this right we can proceed as follows:

F60,200,Number,Numbers	and then use the sub-command 'S' repeatedly.
E10	then the sequence X ; <ENTER> <ENTER>
E30	then __________ K C 1 <ENTER> <ENTER>
F100,100,Size,Size-1	followed by the sub-command 'S'.
M110,95 110	followed by <ENTER>.
E190	then X DELETE <ENTER> <ENTER>
65 Noswaps : BOOLEAN;	
N10,10	to renumber in steps of 10.

You are strongly recommended to work through the above example actually using the editor.

APPENDIX 1 ERRORS.

A.1.1 Error numbers generated by the compiler.

1. Number too large.
2. Semi-colon or 'END' expected.
3. Undeclared identifier.
4. Identifier expected.
5. Use '=' not ':=' in a constant declaration.
6. '=' expected.
7. This identifier cannot begin a statement.
8. ':=' expected.
9. ')' expected.
10. Wrong type.
11. '.' expected.
12. Factor expected.
13. Constant expected.
14. This identifier is not a constant.
15. 'THEN' expected.
16. 'DO' expected.
17. 'TO' or 'DOWNTO' expected.
18. '(' expected.
19. Cannot write this type of expression.
20. 'OF' expected.
21. ',' expected.
22. ':' expected.
23. 'PROGRAM' expected.
24. Variable expected since parameter is a variable parameter.
25. 'BEGIN' expected.
26. Variable expected in call to READ.
27. Cannot compare expressions of this type.
28. Should be either type INTEGER or type REAL.
29. Cannot read this type of variable.
30. This identifier is not a type.
31. Exponent expected in real number.
32. Scalar expression (not numeric) expected.
33. Null strings not allowed (use CHR(0)).
34. '[' expected.
35. ']' expected.
36. Array index type must be scalar.
37. '..' expected.
38. ']' or ',' expected in ARRAY declaration.
39. Lowerbound greater than upperbound.
40. Set too large (more than 256 possible elements).
41. Function result must be type identifier.
42. ',' or ']' expected in set.
43. '..' or ',' or ']' expected in set.
44. Type of parameter must be a type identifier.
45. Null set cannot be the first factor in a non-assignment statement.
46. Scalar (including real) expected.
47. Scalar (not including real) expected.
48. Sets incompatible.
49. '<' and '>' cannot be used to compare sets.
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' or 'BEGIN' expected.

51. Hexadecimal digit expected.
52. Cannot POKE sets.
53. Array too large (> 64K).
54. 'END' or ';' expected in RECORD definition.
55. Field identifier expected.
56. Variable expected after 'WITH'.
57. Variable in WITH must be of RECORD type.
58. Field identifier has not had asociated WITH statement.
59. Unsigned integer expected after 'LABEL'.
60. Unsigned integer expected after 'GOTO'.
61. This label is at the wrong level.
62. Undeclared label.
63. The parameter of SIZE should be a variable.
64. Can only use equality tests for pointers.
67. The only write parameter for integers with two ':'s is e:m:H.
68. Strings may not contain end of line characters.
69. The parameter of NEW, MARK or RELEASE should be a variable of pointer type.
70. The parameter of ADDR should be a variable.

A.1.2 Runtime Error Messages.

When a runtime error is detected then one of the following messages will be displayed, followed by ' at PC=XXXX' where XXXX is the memory location at which the error occurred. Often the source of the error will be obvious; if not, consult the compilation listing to see where in the program the error occurred, using XXXX to cross reference. Occasionally this does not give the correct result.

1. Halt	
2. Overflow	
3. Out of RAM	
4. / by zero	also generated by DIV.
5. Index too low	
6. Index too high	
7. Maths Call Error	
8. Number too large	
9. Number expected	
10. Line too long	
11. Exponent expected	

Runtime errors result in the program execution being halted.

APPENDIX 2 RESERVED WORDS AND PREDEFINED IDENTIFIERS.

A 2.1 Reserved Words.

AND	ARRAY	BEGIN	CASE	CONST	DIV	DO
DOWNTO	ELSE	END	FORWARD	FUNCTION	GOTO	IF
IN	LABEL	MOD	NIL	NOT	OF	OR
PACKED	PROCEDURE	PROGRAM	RECORD	REPEAT	SET	THEN
TO	TYPE	UNTIL	VAR	WHILE	WITH	

A 2.2 Special Symbols.

The following symbols are used by Hisoft Pascal and have a reserved meaning:

+	–	*	/		
=	<>	<	<=	>=	>
(	)	[	]		
{	}	(*	*)		
^	:=	.	,	;	:
'	..				

A 2.3 Predefined Identifiers.

The following entities may be thought of a declared in a block surrounding the whole program and they are therefore available throughout the program unless re-defined by the programmer within an inner block.
For further information see Section 2.

CONST	MAXINT = 32767;
TYPE	BOOLEAN = (FALSE, TRUE);
.	CHAR {The expanded ASCII character set};
.	INTEGER = –MAXINT..MAXINT;
.	REAL {A subset of the real numbers. See Section 1.3.}
PROCEDURE	WRITE; WRITELN; READ; READLN; PAGE; HALT; USER; POKE; INLINE;
.	OUT; NEW; MARK; RELEASE; TIN; TOUT;
FUNCTION	ABS; SQR; ODD; RANDOM; ORD; SUCC; PRED; INCH; EOLN;
.	PEEK; CHR; SQRT; ENTIER; ROUND; TRUNC; FRAC; SIN; COS;
.	TAN; ARCTAN; EXP; LN; ADDR; SIZE; INP;

APPENDIX 3 DATA REPRESENTATION AND STORAGE.

A 3.1 Data Representation.

The following discussion details how data is represented internally by Hisoft Pascal.

The information on the amount of storage required in each case should be of use to most programmers (the SIZE function may be used see Section 2.3.6.7); other details may be needed by those attempting to merge Pascal and machine code programs.

A 3.1.1 Integers.

Integers occupy 2 bytes of storage each, in 2's complement form.
Examples:

1	$\equiv$	#0001
256	$\equiv$	#0100
-256	$\equiv$	#FF00

The standard Z80 register used by the compiler to hold integers is HL.

A 3.1.2 Characters, Booleans and other Scalars.

These occupy 1 byte of storage each, in pure, unsigned binary.

Characters: 8 bit, extended ASCII is used.

'E' $\equiv$ #45
'[' $\equiv$ #5B

Booleans:
ORD(TRUE) = 1 so TRUE is represented by 1.
ORD(FALSE) = 0 so FALSE is representd by 0.

The standard Z80 register used by the compiler for the above is A.

A 3.1.3 Reals.

The (mantissa, exponent) form is used similar to that used in standard scientific notation – only using binary instead of denary. Examples:

$$2 \equiv 2*10^0 \quad \text{or} \quad 1.0_2 * 2^1$$

$$1 \equiv 1*10^0 \quad \text{or} \quad 1.0_2 * 2^0$$

$-12.5 \equiv -1.25*10^1$ or $-25*2^{-1}$

-11001_2*2^{-1}

-1.1001_2*2^3 when normalised.

$0.1 \equiv 1.0*10^{-1}$ or $\frac{1}{10} \equiv \frac{1}{1010_2} \equiv \frac{0.1_2}{101_2}$

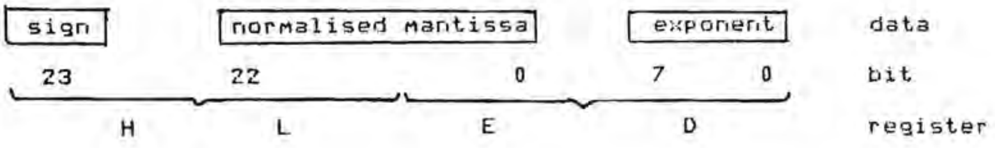
so now we need to do some binary long division..

```
        0.0001100
101 ⌈ 0.10000000000000
           101
            110
            101
             1000
              101     at this point
                      we see that the
                      fraction recurs
```

$$= \quad \frac{0.1_2}{101_2} = 0.000\dot{1}10\dot{0}_2\,.$$

$$\underline{1.1\dot{0}01\dot{1}00 * 2^{-4}} \quad \underline{\text{answer.}}$$

So how do we use the above results to represent these numbers in the computer? Well, firstly we reserve 4 bytes of storage for each real in the following format:



sign:	the sign of the mantissa; 1 = negative, 0 = positive.
normalised mantissa:	the mantissa normalised to the form 1.xxxxxx with the top bit (bit 22) always 1 except when representing zero (HL=0, DE=0).
exponent:	the exponent in binary 2's complement form.

Thus:

```
  2   ≡ 0  1000000  00000000  00000000  00000001   (#40,#00,#00,#01)
  1   ≡ 0  1000000  00000000  00000000  00000000   (#40,#00,#00,#00)
-12.5 ≡ 1  1100100  00000000  00000000  00000011   (#E4.#00.#00.#03)
  0.1 ≡ 0  1100110  01100110  01100110  11111100   (#66.#66,#66.#FC)
```

So, remembering that HL and DE are used to hold real numbers, then we would have to load the registers in the following way to represent each of the above numbers:

```
    2   ≡   LD    HL,#4000
            LD    DE,#0100

    1   ≡   LD    HL,#4000
            LD    DE,#0000

-12.5   ≡   LD    HL,#E400
            LD    DE,#0300

  0.1   ≡   LD    HL,#6666
            LD    DE,#FC66
```

The last example shows why calculations involving binary fractions can be inaccurate; 0.1 cannot be accurately represented as a binary fraction, to a finite number of decimal places.

N.B. Reals are stored in memory in the order ED LH.

A 3.1.4 Records and Arrays.

Records use the same amount of storage as the total of their components.

Arrays: if n=number of elements in the array and
s=size of each element then

the number of bytes occupied by the array is n*s.

e.g. an ARRAY[1..10] OF INTEGER requires 10*2 = 20 bytes
an ARRAY[2..12,1..10] OF CHAR has 11*10=110 elements and so requires 110 bytes.

A 3.1.5 Sets.

Sets are stored as bit strings and so if the base type has n elements then the number of bytes used is: (n-1) DIV 8 + 1. Examples:

a SET OF CHAR requires (256-1) DIV 8 + 1 = 32 bytes.
a SET OF (blue, green, yellow) requires (3-1) DIV 8 + 1 = 1 byte.

A 3.1.6 Pointers.

Pointers occupy 2 bytes which contain the address (in Intel format i.e. low byte first) of the variable to which they point.

A 3.2 Variable Storage at Runtime.

There are 3 cases where the user needs information on how variables are stored at runtime:

a. Global variables	– declared in the main program block.
b. Local variables	– declared in an inner block.
c. Parameters and returned values.	– passed to and from procedures and functions.

These individual cases are discussed below and an example of how to use this information may be found in Appendix 4.

Global variables

Global variables are allocated from the top of the runtime stack downwards e.g. if the runtime stack is at #B000 and the main program variables are:

```
VAR      i : INTEGER;
         ch : CHAR;
         x  : REAL;
```

then:

i (which occupies 2 bytes – see the previous section) will be stored at locations #B000-2 and #B000-1 i.e. at #AFFE and #AFFF.

ch (1 byte) will be stored at location #AFFE-1 i.e. at #AFFD.

x (4 bytes) will be placed at #AFF9, #AFFA, #AFFB and #AFFC.

Local variables

Local variables cannot be accessed via the stack very easily so, instead, the IX register is set up at the beginning of each inner block so that (IX-4) points to the start of the block's local variables e.g.

```
PROCEDURE     test;
VAR           i,j : INTEGER;
```

then:

i (integer – so 2 bytes) will be placed at IX-4-2 and IX-4-1 i.e. IX-6 and IX-5.
j will be placed at IX-8 and IX-7.

Parameters and returned values

Value parameters are treated like local variables and, like these variables, the earlier a parameter is declared the higher address it has in memory. However, unlike variables, the lowest (not the highest) address is fixed and this is fixed at (IX+2) e.g.

```
PROCEDURE test(i : REAL; j : INTEGER);
```

then:

j (allocated first) is at IX+2 and IX+3.
i is at IX+4, IX+5, IX+6, and IX+7.

Variable parameters are treated just like value parameters except that they are always allocated 2 bytes and these 2 bytes contain the address of the variable e.g.

```
PROCEDURE test(i : INTEGER; VAR x : REAL);
```

then:

the reference to x is placed at IX+2 and IX+3; these locations contain the address where x is stored. The value of i is at IX+4 and IX+5.

Returned values of functions are placed above the first parameter in memory e.g.

```
FUNCTION test(i : INTEGER) : REAL;
```

then i is at IX+2 and IX+3 and space is reserved for the returned value at IX+4, IX+5, IX+6 and IX+7.

APPENDIX 4 SOME EXAMPLE HISOFT PASCAL PROGRAMS.

The following programs should be studied carefully if you are in any doubt as to how to program in Hisoft Pascal.

```
{Program to illustrate the use of TIN and TOUT.
 The program constructs a very simple telephone
 directory on tape and then reads it back. You
 should write any searching required.}

PROGRAM TAPE;

CONST
  Size=10;

TYPE
  Entry = RECORD
              Name : ARRAY [1..10] OF CHAR;
              Number : ARRAY [1..10] OF CHAR
           END;

VAR
  Directory : ARRAY [1..Size] OF Entry;
  I : INTEGER;

BEGIN

{Set up the directory..}

  FOR I:= 1 TO Size DO
  BEGIN
    WITH Directory[I] DO
    BEGIN
      WRITE('Name please');
      READLN;
      READ(Name);
      WRITELN;
      WRITE('Number please');
      READLN;
      READ(Number);
      WRITELN
    END
  END;

{To dump the directory to tape use..}

  TOUT('Director',ADDR(Directory),SIZE(Directory))

{Now to read the array back do the following..}

  TIN('Director',ADDR(Directory))

{And now you can process the directory as you wish.....}

END.
```

```
 10 {Program to list lines of a file in reverse order.
 20  Shows use of pointers, records, MARK and RELEASE.}
 30
 40 PROGRAM ReverseLine;
 50
 60 TYPE elem=RECORD                               {Create linked-list structure}
 70             next: ^elem;
 80             ch: CHAR
 90            END;
100       link=^elem;
110
120 VAR prev,cur,heap: link;                       {all pointers to 'elem'}
130
140 BEGIN
150  REPEAT                                        {do this many times}
160    MARK(heap);                                 {assign top of heap to 'heap'.}
170    prev:=NIL;                                  {points to no varaible yet.}
180    WHILE NOT EOLN DO
190       BEGIN
200         NEW(cur);                              {create a new dynamic record}
210         READ(cur^.ch);                         {and assign its field to one
220                                                 character from file.}
230         cur^.next:=prev;                       {this field's pointer adresses}
240         prev:=cur                              {previous record.}
250       END;
260
270 {Write out the line backwards by scanning the records
280  set up backwards.}
290
300    cur:=prev;
310    WHILE cur <> NIL DO                          {NIL is first}
320      BEGIN
330         WRITE(cur^.ch);                         {WRITE this field i.e. character}
340         cur:=cur^.next                          {Address previous field.}
350      END;
360    WRITELN;
370    RELEASE(heap);                               {Release dynamic variable space.}
380    READLN                                       {Wait for another line}
390  UNTIL FALSE                                    {Use CC to exit}
400 END.
```

```
 10 {Program to show the use of recursion}
 20
 30 PROGRAM FACTOR;
 40
 50 {This program calculates the factorial of a number input from the
 60  keyboard  1) using recursion  and  2) using an iterative method.}
 70
 80 TYPE
 90   POSINT = 0..MAXINT;
100
110 VAR
120   METHOD : CHAR;
130   NUMBER : POSINT;
140
150 {Recursive algorithm.}
160
170 FUNCTION RFAC(N : POSINT) : INTEGER;
180
190  VAR F : POSINT;
200
210  BEGIN
220    IF N>1 THEN F:= N * RFAC(N-1)                      {RFAC invoked N times}
230           ELSE F:= 1;
240    RFAC := F
250  END;
260
270 {Iterative solution}
280
290 FUNCTION IFAC(N : POSINT) : INTEGER;
300
310   VAR I,F: POSINT;
320   BEGIN
330    F := 1;
340    FOR I := 2 TO N DO F := F*I;                        {Simple Loop}
350    IFAC:=F
360   END;
370
380 BEGIN
390   REPEAT
400     WRITE('Give method (I or R) and number    ');
410     READLN;
420     READ(METHOD,NUMBER);
430     IF METHOD = 'R'
440          THEN WRITELN(NUMBER,'! = ',RFAC(NUMBER))
450          ELSE WRITELN(NUMBER,'! = ',IFAC(NUMBER))
460   UNTIL NUMBER=0
470 END.
```

HISOFT PASCAL IMPLEMENTATION NOTE

NEWBRAIN

Loading from Tape

Turn on your NewBrain, connect your tape recorder, load the supplied Hisoft Pascal cassette into your tape recorder, press PLAY on the recorder and type 'LOAD' on the computer. After a while the filename 'HP4T NewBrain Loader' should appear on the screen. If this does not happen, or if you should subsequently experience difficulty in loading from the tape, then stop the tape, rewind it to the beginning, press 'STOP' on the NewBrain, followed by 'NEW' and 'LOAD' again. If you repeatedly experience problems then please return the tape to Hisoft for replacement.

After the Loader has been loaded into the computer, the tape will stop and the cursor will appear on the screen; now type 'RUN' and the message 'HP4T Loader. (c) Hisoft 1983.' will appear on the screen and the tape will start again - the main code of the compiler package is now being loaded.

Owing to the buffering of the tape data and the relative slowness of the tape interface, the Pascal will take a while to load - roughly 5 minutes. Once the Pascal has been loaded it will be executed automatically, displaying the 'Top of RAM?' message - now refer to Section 0 of this manual for further details.

Implementation on the NewBrain

The implementation of Hisoft Pascal on the NewBrain is a fairly standard one with one or two exceptions. We shall consider the keyboard, video, tape and printer separately.

Keyboard

Stream 1 is opened as a KBWIO (device 5) stream while stream 2 is assigned as a KBIIO (device 6) stream by Hisoft Pascal. These streams are used for all input within Hisoft Pascal - this means that the NewBrain screen editor is not supported for input.

The standard control codes given in Section 0 of this manual are reached as follows on the NewBrain keyboard:

RETURN	is NEW LINE on the NewBrain.
CC	is CTRL/C (hold CONTROL and C down together).
CH	is the cursor left character '←'.
CI	is CTRL/I.
CP	used within a program as CHR(16) - see below.
CX	is CTRL/X.
CS	reached via the 'STOP' key.

CTRL/A is used to flip CAPS LOCK on and off; initially pressing the keys 'A' .. 'Z' give you the capital letters on the keys as does SHIFT 'A' .. 'Z'. Use CTRL/A once and you are put in a mode where hitting 'A' .. 'Z' without SHIFT gives you lower case 'a' .. 'z' while using SHIFT as well gives the capital letters. Use CTRL/A to return to capitals only mode etc.

Note that the use of CTRL/0 to CTRL/9 to change the character set generated by the keyboard is not supported from within the Hisoft Pascal editor - in practice this is not a restriction since the case in which it is most useful is to generate characters whose ASCII value is greater than 127 - these characters should never be included in a Pascal program since the editor would try to treat them as Reserved Words! Instead you should use CHR to WRITE these characters e.g. WRITE(CHR(236)) to generate ' ' in character set 2.

'(' and ')' are reached via GRAPHICS/, and GRAPHICS/. respectively within character sets 1 and 2.

Video

Stream 0 is opened as a TVIO (device 0) stream but is only used for output.

Initially the video is configured as 40 characters by 24 lines - this may be changed to 80 characters by 24 lines through use of an extra editor command, the 'W' command. Using 'W' flips between a screen width of 40 and 80 characters.

From within a Pascal program, most characters are passed straight through to the video driver thus enabling the range of control sequences supported by the TVIO device to be maintained. However, 3 character values are trapped and interpreted by Hisoft Pascal, these are; CHR(8) which is changed internally to CHR(24) for backspace, CHR(12) which is converted to CHR(31) for clear screen and CHR(16) which is used to flip the printer stream (see below) on and off. If you wish to output any of these character values then they must be output directly through the TVIO device and not through a Pascal WRITE(LN) statement. We give below a Pascal procedure that takes an integer as a parameter and outputs its CHR value directly to stream 0:

```
PROCEDURE CHAROUT ( I : INTEGER);
BEGIN
 INLINE(#DD, #7E, #02, #F5, #CD, #75, #19, #3E, #1B,
        #1E, #00, #E7, #30, #F1, #E7, #30, #CD, #26, #19)
END;
```

Note that the routine at address #1975 must be called before any attempt is made to access the NewBrain operating system and the routine at #1926 must be subsequently called before returning to Hisoft Pascal.

Tape

Stream 3 is opened as a TAPE1 (device 1) stream by Hisoft Pascal 4T. The buffer size allocated is #600 (i.e. 6 units); this may be changed by POKEing location #19FA with the required size in 256 byte units. Note that you should not allocate such a large buffer size that would corrupt the compiler which starts at #1800.

Printer

Owing to the range of different printers that may be attached to the NewBrain, Hisoft Pascal makes no assumptions as to the type of printer available. Instead it is simply assumed that the user has already opened the relevant printer driver on stream 4 before entering the package. Thus stream 4 is used by the Pascal as the printer stream; this stream is not opened or closed by Hisoft Pascal.

Compiler listings may be sent to the printer by using the $P option - see Section 3.2 of this manual. Runtime output may be directed to the printer by using WRITE(CHR(16)); - all subsequent output will go to the printer until another WRITE(CHR(16)); is received when output will revert to the screen.

Making a working copy

1. Load the Pascal from the master tape and answer the questions 'Top of RAM?' etc. in the normal way.

2. Now use the 'B' editor command (press the character B and then NEWLINE) to return to BASIC.

3. Enter the following BASIC program, overtyping the existing loader program:

```
10 OPEN OUT#1,1,"%6:HP4TG"
20 FOR I=6144 TO 24082
30 X=PEEK(I)
40 PUT #1,X
50 NEXT I
60 CLOSE #1: END
```

4. Run the above program and the configured version of HP4TG will be saved to tape - remember to press RECORD and PLAY on your tape recorder.
5. To load back this configured version of the Pascal use the following BASIC program:

```
10 OPEN IN#1,1,"%6:HP4TG"
20 FOR I=6144 TO 24082
30 GET #1,X
40 POKE(I,X)
50 NEXT I
60 CLOSE #1 :  END
```

To re-enter the Pascal use CALL 6208 for a cold start, destroying any test test or CALL 6213 for a warm start, preserving the state of the compiler/editor as you left it.

Loading Object Code dumped using 'T'ranslate.
If you have dumped the object code of your Pascal program to tape, via the Editor's T command, then the code may be re-loaded and executed by using the Hisoft HP4T Loader program supplied on the master tape. Simply load this from the tape (via LOAD ""), RUN it and then put the tape with the object code on it into your tape recorder and press PLAY, the code will be loaded and executed automatically.

HISOFT PASCAL IMPLEMENTATION NOTE

SHARP MZ80K, MZ80A and MZ80B

Loading Hisoft Pascal from tape

Unpack the supplied cassette tape from its case and load it into your machine with Side A (the side with the label) uppermost. Now, ensuring that you are in monitor mode on your SHARP computer, type the relevant command to load a file from tape ('L' on the MZ80A and MZ80B and 'LOAD' on the MZ80K) followed by CR or ENT. On the MZ80B you will be prompted to enter a filename - simply press CR or ENT.

After a while the message 'Loading HP4Txxx' should appear - if this does not happen or if you subsequently experience any problems in loading the tape, then stop the tape, rewind it to the beginning, press RESET (or BREAK) and try to load the tape again. If you continue to have problems loading the Hisoft Pascal tape then please return it to Hisoft for replacement - however we trust this will never happen.

Once the package has been loaded it will execute automatically and the message 'Top of RAM? ' will be displayed - consult Section 0.0 of the Hisoft Pascal Programmer's Manual for details of how to proceed from here.

Implementation on the SHARP MZ80K, MZ80A and MZ80B.

The implementation of Hisoft Pascal on the SHARP cpmputers is fairly straightforward with one or two exceptions which are discussed below.
The various control codes mentioned in the Programmer's Manual are reached as follows on the MZ80K, MZ80A and MZ80B.

RETURN	via the CR or ENT key.
CC	via the BREAK key (SHIFT and CTRL on the MZ80K and MZ80A).
CH	via the DEL key (giving a destructive backspace).
CI	via the cursor right key.
CP	see below for printer handling.
CX	via the cursor left key.
CS	identical to CC i.e. the BREAK key.

The MZ80K and MZ80A computers do not support the true ASCII character set in that the lowercase letters are not reached via codes 97 (#61) to 122 (#7A) inclusive. Hisoft Pascal assumes a true ASCII character set and thus must incorporate a conversion routine for the lowercase characters on these computers. The result of this is that, while using Hisoft Pascal, the lowercase letters are reached via the standard ASCII codes (#61 to #7A) and the SHARP graphics characters that normally exist at these codes are reached via the SHARP codes of the lowercase letter which is replacing the particular graphic. The MZ80B uses a true ASCII character set and the above does not apply to it.

Note that, on all SHARP computers, you must not use any characters with a code greater than #7F directly within the Pascal text since such a character would be interpreted as a compacted Reserved Word. You may, of course, WRITE codes greater than #7F using CHR e.g. WRITE(CHR(132)). One result of the above restriction is that the { } form of comment is not supported - you should use (* and *) instead.

The repeat key facility of the MZ80A and MZ80B is not available under Hisoft Pascal since it is handled within the line input routine of the monitor and Hisoft Pascal handles characters, not lines, at a time.

The SFTLOCK and GRPH key of the MZ80B are supported.

Any return to the SHARP monitor from within the compiler or runtimes requires you first to hit any key on the keyboard - this is so that runtime error messages etc. may be inspected (a direct return to the monitor clears the screen).

Note that all data recorded on tape is in SHARP tape format and not in Hisoft Pascal format- this is because of the software tape interface used by the SHARP computers. This means that there is no loader for the compiler or object code files, you simply use the relevant 'LOAD' command of the monitor to reload object code dumped to tape via 'Translate'.

On the MZ80K and MZ80A the Hisoft Pascal package resides from location #1200 while on the MZ80B it starts at #1220. The end address of the package may be ascertained through use of the editor's 'X' command. To make a copy of the package on tape you could use a Pascal program such as this:

```
PROGRAM save;
CONST
  start=#1200;          (*#1220 for the MZ80B*)
  finish=#5E00;         (*ascertain this from the 'X' command*)
  IBUFE=#10F0;          (*#10C0 on the MZ80B*)
  execute=#121E;        (*#123E on the MZ80B*)
  name='HP4TZ';         (*or whatever you wish*)
BEGIN
  POKE(IBUFE,CHR(80)); (*set up the header buffer*)
  PIKE(IBUFE+1,name);
  POKE(IBUFE+6,CHR(13));  (*depends on the length of 'name'*)
  POKE(IBUFE+18,finish-start);
  POKE(IBUFE+20,start);
  POKE(IBUFE+22,execute);
  USER(#21);            (*#251 on the MZ80B - write the header*)
  USER(#24)             (*#282 on the MZ80B - write the data*)
END.
```

Executing the above program will dump the compiler, runtimes and editor to tape - use the SHARP monitor 'LOAD' command to reload the package which will autostart with a cold start to the editor.

To re-enter Hisoft Pascal from within the SHARP monitor, execute #121E (or #123E on an MZ80B) for a cold start and #1221 (#1241 on an MZ80B) for a warm start, preserving the text.

The standard SHARP printer interface routine is included in Hisoft Pascal for the MZ80K and MZ80A - this is accessed via the 'P' option as detailed in Section 3 of this manual or via WRITE(CHR(16)) which toggles the printer on and off.

For the MZ80B a parallel printer routine is included within Hisoft Pascal - this is suitable for driving an EPSON printer and is accessed in the standard way i.e. via the 'P' option or through WRITE(CHR(16)).

An extra command is included within the editor for the SHARP computers; this is the 'W' command which works like the 'P' command except that it dumps a block of text to tape in a form which is suitable for inclusion (via the compiler option 'F') at a later stage. Note that you cannot 'include' text which has been written to tape using the 'P' command - if you want to 'include' text then the text must have been saved on tape via the 'W' command.

We hope that the above information will enable you to make the most of Hisoft Pascal on your SHARP computer. Please do not hesitate to contact us if you experience any difficulties; we can only put your problems right if you tell us about them!

PARALLEL PRINTER DRIVER FOR HISOFT PASCAL UNDER SHARP MZ80K AND MZ80A

The following gives details of incorporating a parallel printer driver routine into Hisoft Pascal on the SHARP MZ80K and MZ80A computers. The driver routine assumes that a standard SHARP extension card (using ports #FE and #FF for the printer) has been fitted and that the printer being driven is compatible with the EPSON MX80 series of printers.

To incorporate the routine proceed as follows:

1. Load Hisoft Pascal into your machine and enter the editor.

2. Enter the following Pascal program (using the 'I' command):

```
PROGRAM Enterhex;

CONST  Max = 50;
VAR I : INTEGER;  A : ARRAY[1..Max] OF CHAR;

FUNCTION Hex : CHAR;
VAR
 CH  : CHAR;
 I,J : INTEGER;

BEGIN
 J:=0;
 WRITE(':');
 READLN;
 WHILE NOT EOLN DO
 BEGIN
  READ(CH);
  I:=ORD(CH);
  IF CH IN ['0'..'9'] THEN I:=I-48 ELSE
     IF CH IN ['A'..'F'] THEN I:=I-55 ELSE
        IF CH IN ['a'..'f'] THEN I:=I-87 ELSE I:=0;
  J:=16*J+I
 END;
 Hex:=CHR(J)
END;
```

```
BEGIN
 FOR I:=1 TO Max DO  A[I]:=Hex;
 FOR I:=1 TO Max DO  POKE(#1293+I,A[I]);
 POKE(#1255,0);
 POKE(#1257,CHR(0));
 POKE(#1284,CHR(12))
END.
```

Remember to use the hash symbol (SHIFT 3) and not the '#' sign when specifying hexadecimal numbers within Hisoft Pascal programs on the SHARP computers.

Having typed in the above program, compile and run it; you will be prompted with ':' – enter two-digit hexadecimal numbers as given below, two digits/letters at a time with each number terminated by CR or ENT. When you have entered the last hexadecimal number (and terminated it with CR or ENT) control will automatically return to the editor. The hexadecimal numbers to enter are:

```
20 06 CD 45 14 C3 12 00 F5 CD A6 12 F1 FE 0D C0 3E 0A F5 CD BD 12 B7 28 FA

F1 D3 FF 3E 80 D3 FE DB FE 0F 30 FB AF D3 FE C9 DB FE FE F2 20 F6 3E FF C9
```

This code assumes that your printer does not generate an automatic line feed on receipt of a carriage return character. If your printer does generate an automatic line feed then you should not enter #FE as the 14th hexadecimal character but instead you should enter #C9.

You have now modified the I/O Primitives within the Hisoft Pascal package to interface with a parallel printer. Now you must save the package to tape. Do this as instructed above.

The above modification allows you to use a parallel printer (such as the EPSON FX80 series) with Hisoft Pascal through use of the compiler 'P' option or by WRITE(CHR(16)) from within a Pascal program.

Note that the above program is not necessarily the most efficient way to implement this modification; it does, however, have the advantage of giving you a free 'read a hex number' function!

If you have any queries regarding this note, please do not hesitate to contact us.

IMPLEMENTATION NOTE

Loading the Compiler from Tape

Unpack the supplied cassette and load it into your tape recorder with the label marked 'Verison 1.5 MZ-7 64K' facing upwards. Now, from within the monitor program of the MZ700, press the 'L' key and then the 'CR' key and then press PLAY on your tape recorder. After a while, the message 'LOADING HP4T15MZ-7' should appear on the screen - if this does not happen or if you experience any further tape loading problems then please return the tape to your dealer for replacement.

After roughly two and a half minutes the package will have loaded into the computer and it will then auto-execute and produce the message 'Top of RAM? '; now refer to Section 0 of the Hisoft Pascal Programmer's manual for details of how to answer this and subsequent configuration questions. Note that, in most cases, simply pressing CR in answer to these questions will suffice.

Implementation on the SHARP MZ700 series

The implementation of Hisoft Pascal on the SHARP MZ700 series is fairly straightforward and more powerful than most other implementations. The full 64K RAM of the computer is utilised by paging the extra memory in and out as necessary - this gives a total of roughly 40K for user programs.
The various control codes discussed and used in this manual are reached as follows on the MZ700:

RETURN	via the CR key on the MZ700
CC	via CTRL/C i.e. hold the CTRL key and the letter 'C' key down together
CH	via the DEL key
CI	via CTRL/I
CP	via CTRL/P
CX	via CTRL/X
CS	via the SHIFT and BREAK key held down together

Entry from the keyboard is as normal, keys producing upper case letters unless the SHIFT key is held down. However, if CTRL/K is pressed while in editor mode, then this action will be reversed; hitting keys will produce the SHIFTed character on that key while holding SHIFT and the key down together will produce the unSHIFTed character - note that this applies to <u>all</u> keys. Hit CTRL/K again to return to the normal mode of entry.

In addition to the comprehensive line editing functions described in Section 4 of the Programmer's Manual, user's of Hisoft Pascal on the SHARP MZ700 have access to a simple screen-editing scheme - this works as follows. If while in the main editor command loop, i.e. with a '>' sign at the left-hand edge, you press any of the cursor movement keys then the cursor will move as directed and you will be placed in screen-editing mode. In this mode you may overtype any text on the screen, using INST and DEL to open up or close down a screen line as required. Then, when you press the CR key, the line with the cursor in it will be transmitted to the Hisoft Pascal editor and control returned from the screen editor to the Hisoft Pascal line editor. You should note that you must ensure that all screen-editing takes place within a single line on the screen and that, since the line is returned to the Hisoft Pascal editor, the screen line transmitted must be as it would have been if you had typed it in directly from the line editor; specifically, any line that begins with a '>' sign will not be accepted by Hisoft Pascal line editor although it will ignore leading spaces.

Some extra procedures are included in the MZ700 version of Hisoft Pascal 4T and these are as follows:

PAPER(I)

this procedure takes an INTEGER parameter in the range 0..7 and it sets the paper colour of all subsequent output to the colour corresponding to this number.

INK(I)

takes an INTEGER parameter in the range 0..7 and sets the ink colour of all subsequent output to this colour - see page 80 of the MZ700 Owner's Manual for number-to-colour correspondence.

MUSIC(S)

takes a string of characters as a parameter and then passes this string to the MUSIC procedure within the MZ700 operating system. See page 65 of the MZ700 Owner's Manual for details of the specification of the music string. Note that Hisoft Pascal insists that any string must not run over a line boundary.

TEMPO(I)

takes an INTEGER parameter in the range 1..7 and sets the tempo of any subsequent MUSICal output according to the value of this parameter.

The MZ700 Colour Plotter-Printer is supported by using control codes as given on pages 198-200 of the MZ700 Owner's Manual. For example, to draw a green line on the plotter from within a Pascal program, you could use the following:

```
PROGRAM A;
BEGIN
 WRITE(CHR(16));      (to turn the printer ON)
 WRITELN(CHR(2),'C2,I,D0,0,100,100');
 WRITE(CHR(16))     (turn the printer OFF)
END.
```

An extra command has been added to the Hisoft Pascal editor for the SHARP MZ700; this is the 'W' command which works like the 'P' command except that it dumps a block of text to tape in a form which is suitable for later inclusion (via the compiler option '$F'). Note that you cannot 'include' text that has been written to tape using 'P' - if you want to 'include' text then the text must have been written to tape using the 'W' command.

Using the 'T' command from the editor will compile the text and then save the object code to tape as described in Section 4.2.5 of this manual. This will overwrite the compiler and should only be done when the program is fully de-bugged and ready for use. To load this object code into a 'clean' MZ700 then you should use the MZ700 monitor's L command - type L <CR> and press PLAY on the tape deck.

The object code will be loaded and executed - hit any key at the end of the execution to return to the monitor. Use J1200 followed by CR to execute the object code again.

If, at any time, you wish to return to the MZ700 monitor from within the Pascal then use the 'B' command and then hit any key. To re-enter the Pascal from the monitor use J121E for a cold start (destroying any text) or J1221 for a warm start (preserving text).

HISOFT PASCAL 4TM 1.6 IMPLEMENTATION NOTE
48K ZX SPECTRUM

Loading HP4TM From Tape.

Unpack the cassette tape from its case and load it into your cassette recorder with Side A uppermost. On your SPECTRUM make sure that you are in Keyword Entry Mode and then enter:

LOAD "" (press J and then " twice)

Now press PLAY on the tape recorder: first a small BASIC loader will be loaded, this will execute automatically and proceed to load the HP4TM code. If a tape error is detected then stop the tape, rewind to the start, press NEW (on the 'A' key) on the SPECTRUM and then enter LOAD "" again. If you still get a tape loading error then try adjusting the volume on your tape recorder; if errors persist please return the tape to Hisoft and we will replace it.

Once the HP4TM code has been loaded it will execute automatically and the message 'Top of RAM?' will be displayed - now consult Section 0.0 of this manual for details of how to proceed.

Implementation on the SPECTRUM.

The ZX SPECTRUM is a rather unusual computer and, to a certain extent, the implementation of Hisoft Pascal reflects this. The various control codes discussed in the Programmer's Manual are reached as follows on the SPECTRUM:

RETURN	via the 'ENTER' key.
CC	via CAPS SHIFT and 1.
CH	DELETE i.e. CAPS SHIFT and 0.
CI	via CAPS SHIFT and 8.
CP	via CAPS SHIFT and 3 enabling 'L'isting of text to printer.
CX	via CAPS SHIFT and 5.
CS	via CAPS SHIFT and SPACE.

The ZX SPECTRUM keyword entry scheme is <u>not</u> supported (we see this as a positive advantage), instead all text must be inserted using the normal alphanumeric keys. Using SYMBOL SHIFT and any key (except I) will always reach the ASCII symbol associated with that key and not the keyword e.g. SYMBOL SHIFT T gives '>' and SYMBOL SHIFT G gives '>'. You <u>must not</u> use the single symbols <=, <> and >= ; instead these should be entered as a combination of the symbols <, > and =.

The editor comes up in upper case mode, this may be toggled in the normal way using CAPS SHIFT and 2.

You have control over the temporary attributes of the various character positions on the screen through the use of the standard control codes (e.g. WRITE(CHR(17),CHR(4)) will make the 'paper' green) but you cannot change the permanent attributes. If, while using these control codes, an invalid sequence is detected then the message 'System Call error' will be displayed and the execution aborted. Note that certain CHR codes are interpreted by Hisoft Pascal (e.g. CHR(8) is taken as DELETE) and thus these codes cannot be passed directly to the Spectrum ROM - use the SPOUT procedure (see below) if you want to write CHR codes without interpretation by Hisoft Pascal.

When dumping text or object code to tape you must be careful to have the tape recorder in RECORD mode before beginning the dump.

If you have used the 'T'ranslate command to save the object code and runtimes on tape then to load the program simply enter LOAD "" CODE from within BASIC. To execute the program enter RANDOMIZE USR 24608 from within BASIC.

From within ZX BASIC, you can re-enter the HP4TM editor in one of two ways: enter RANDOMIZE USR 24603 to perform a warm start i.e. preserving the Pascal program or RANDOMIZE USR 24598 to do a cold start, re-initialising the Pascal and clearing any existing Pascal text.

The ZX Printer is supported via the use of the compiler 'P' option (see Section 3.2 of the Programmer's Manual) and via CHR(16) in a WRITE or WRITELN statement. Note that, as a result, you cannot use CHR(16) within a WRITE(LN) statement to specify the INK colour - instead you can use CHR(15) to set the INK.

Most parallel printer interfaces are supported by Hisoft Pascal since printer output is always directed through stream 3; specifically the Kempston and Morex interfaces work well.

Any return to BASIC from within Pascal will cause execution to pause - hit any key to accomplish the return to BASIC. This pause has been added to allow runtime error messages etc. to remain on the screen - the return to BASIC clears the screen.

An extra command has been added to the Hisoft Pascal editor for the ZX Spectrum; this is the 'W' command which works like the 'P' command except that it dumps a block of text to tape in a form which is suitable for later inclusion (via the compiler option '$F'). Note that you cannot 'include' text that has been written to tape using 'P' - if you want to 'include' text then the text must have been written to tape using the 'W' command.

To make a back-up copy of HP4TM16 proceed as follows:

1. Load HP4TM16 from tape and answer the 'Top OF RAM?' etc. messages normally.

2. Return to BASIC using the editor's B command - hit ENTER twice.

3. Use SAVE "HP4TM16" CODE 24598,19736 to save the Pascal to tape.

4. You can subsequently use LOAD "" CODE to reload the compiler into the Spectrum but note that you must then enter it only via RANDOMIZE USR 24598 (for a cold start) or RANDOMIZE USR 24603 (for a warm start).

Note that you are authorised by Hisoft to make only one working copy.

Please do not hesitate to contact us if you experience any difficulty with Hisoft Pascal - we can only solve the problems if we know what they are!

PRODUCT APPLICATION NOTE PS1.1

ZX SPECTRUM SOUND AND GRAPHICS WITH HISOFT PASCAL

This note gives details on controlling the sound and graphics capabilities of the ZX SPECTRUM using Pascal procedures from within Hisoft Pascal.

1. Sound.

The following two procedures (defined in the order given below) are required to produce sound with Hisoft Pascal.

```
{This procedure uses machine code to pick up its parameters and
 then passes them to the BEEP routine within the SPECTRUM ROM.}

PROCEDURE  BEEPER (A, B : INTEGER);

 BEGIN
  INLINE(#DD, #6E, 2, #DD, #66, 3,            { LD  L,(IX+2) : LD  H,(IX+3) }
         #DD, #5E, 4, #DD, #56, 5,            { LD  E,(IX+4) : LD  D,(IX+5) }
         #CD, #B5, 3, #F3)                    { CALL #3B5     : DI }
 END;

{This procedure traps a frequency of zero which it converts into a period of silence.
 For non-zero frequencies the frequency and length of the note are approximately converted
 to the values required by the SPECTRUM ROM routine and this is then called via BEEPER.}

PROCEDURE  BEEP (Frequency : INTEGER; Length : REAL);

VAR  I : INTEGER;

 BEGIN
  IF Frequency=0 THEN  FOR I:=1 TO ENTIER(12000*Length)  DO

  ELSE  BEEPER( ENTIER(Frequency*Length), ENTIER(437500/Frequency - 30.125));

  FOR I:= 1 TO 100 DO          {short delay between notes}
 END;

Example of the use of BEEP:

BEEP ( 262, 0.5 );
BEEP ( 0, 1 );   {sounds middle C for 0.5 seconds followed by a one second silence.}
```

2. Graphics.

Three graphics procedures are given: the first plots a given (X,Y) co-ordinate whilst the second and third are used to draw lines from the current plotting position to a new position which is defined relative to the current plot position and which then becomes the current plot position.

Both PLOT and LINE take a BOOLEAN variable, ON, which, if TRUE, will cause any point to be plotted regardless of the state of the pixel in that plot position or, if FALSE, will cause any pixel already present at the plot position to be flipped i.e. if on it becomes off and vice versa. This effect is identical to that caused by the SPECTRUM OVER command.

```
{A procedure that mirrors the BASIC PLOT command. Simply plots the point X,Y  ON or OFF depending on
 whether the first parameter is TRUE or FALSE.}

PROCEDURE PLOT( ON : BOOLEAN; X, Y : INTEGER);

 BEGIN
  IF ON THEN WRITE( CHR(21), CHR(0))
        ELSE WRITE( CHR(21), CHR(1));

  INLINE(#FD, #21, #3A, #5C,                    { LD   IY,#5C3A }
         #DD, #46, 2, #DD, #4E, 4               { LD   B,(IX+2) : LD  C,(IX+4) }
         #CD, #E5, #22);                        { CALL #22E5  ;ROM PLOT routine}
 END;


{Called by the LINE procedure,  LINE1  is used to  pass the correct arguments to the DRAW routine in the
 SPECTRUM ROM.}

PROCEDURE LINE1( X,Y,SX,SY :INTEGER);

 BEGIN
  INLINE(#FD, #21, #3A, #5C,                    { LD   IY,#5C3A }
         #DD, #56, 2, #DD, #5E, 4,              { LD   D,(IX+2) : LD  E,(IX+4) }
         #DD, #46, 6, #DD, #4E, 8,              { LD   B,(IX+6) : LD  C,(IX+8) }
         #CD, #BA, #24)                         { CALL #24BA    ;ROM DRAW routine.}
 END;


{LINE draws a line from the current plot position (x,y) to (X+x,Y+y). The line
 may be 'on' or 'off' depending on the value of the BOOLEAN parameter ON.}

PROCEDURE LINE( ON : BOOLEAN; X, Y :INTEGER);

VAR
   SGNX, SGNY : INTEGER;

 BEGIN
  IF ON THEN WRITE( CHR(21), CHR(0))
        ELSE WRITE( CHR(21), CHR(1));

  IF X<0 THEN SGNX:=-1 ELSE SGNX:=1;           {The DRAW routine that is to be called}
  IF Y<0 THEN SGNY:=-1 ELSE SGNY:=1;           {within LINE1 needs the absolute values}
                                               {of X and Y and their signs.}
  LINE1( ABS(X), ABS(Y), SGNX, SGNY)           {Plot the line.}
 END;
```

```
Example of the use of PLOT and LINE:

PLOT( ON, 50, 50 );
LINE( ON, 100, -50 );     {draws a line from (50, 50) to (150, 0).}
```

3. Output through the ROM.

There are occasions where it is useful to output directly through the SPECTRUM ROM RST #10 routine rather than use WRITE(LN). For example, when using the PRINT AT control code - this code should be followed by two 8 bit values giving the (X,Y) co-ordinate to which the print position is to be changed. However, if this is done using a Pascal WRITE statement then certain values of X and Y (e.g. 8 which is interpreted by Hisoft Pascal as BACKSPACE) will not be passed to the ROM and thus the print position will not be correctly modified.

You can overcome this problem by using the following procedure:

```
{SPOUT outputs the character passed as a parameter directly through the SPECTRUM
 ROM  RST #10  routine and thus avoids any trapping by Hisoft Pascal of the value output.}

PROCEDURE SPOUT ( C : CHAR );

 BEGIN
  INLINE(#FD, #21, #3A, #5C,                { LD   IY,#5C3A }
         #DD, #7E, 2                        { LD   A,(IX+2) }
         #D7 )                              { RST  #10 }
 END;
```

```
Example of the use of SPOUT:

SPOUT ( CHR(22) ); SPOUT ( CHR(8) ); SPOUT ( CHR(13) ); {sets the print position to line 8, column 13.}
```

Hisoft hope that you will find the above routines useful and that they will enhance the way in which you use Hisoft Pascal.

HISOFT PASCAL TURTLE GRAPHICS FOR THE ZX SPECTRUM

Hisoft Pascal for the ZX SPECTRUM now (15 August 1983) comes complete with a Logo-style Turtle Graphics package on the 'B' side of the program cassette.

The package is written in Pascal and may be loaded from within the Hisoft Pascal editor by using the command 'G,,TURTLE'. This will load the turtle graphics program segment and append it to any existing program; note that, in order for it to function correctly, the Turtle Graphics must be preceded by a normal PROGRAM heading and a VAR declaration - TYPE, CONST and LABEL declarations are optional and there must be no Procedures or Functions declared previous to the inclusion of the Turtle Graphics package.

As in the majority of Turtle Graphics implementations, Hisoft Pascal's TURTLE creates an imaginary creature on the screen which the user can move around via some very simple commands. This 'turtle' can be made to leave a trail (in varying colours) or can be made invisible. The turtle's heading and position are held in global variables which are updated when the creature is moved or turned; obviously these variables can be inspected or changed at any time.

The facilities available are as follows:

Global Variables

HEADING

this is used to hold the angular value of the direction in which the turtle is currently facing. It takes any REAL value, in degrees, and may be initialised to 0 with the procedure TURTLE (see below). The value 0 corresponds to an EASTerly direction so that after a call to the procedure TURTLE the turtle is facing left to right. As the heading increases from zero then the turtle turns in an anti-clockwise direction.

XCOR, YCOR

these are the current (x,y) REAL co-ordinates of the turtle on the screen. The SPECTRUM graphics screen has a size of 256*176 pixels and the turtle may be positioned on any point within this area; if an attempt is made to move the turtle out of this matrix (using LINE - see below) then the message 'Out of Limits' will be displayed and the program will be aborted with a 'HALT' message. Initially XCOR and YCOR are undefined, use of the procedure TURTLE initialises them to 127 and 87 respectively, thus placing the turtle in the middle of his 'pool'.

PENSTATUS

an INTEGER variable holding the current status of the 'pen' (i.e. the trail left by the turtle). 0 means the pen is down, 1 means the pen is up.

PROCEDURES

The procedures PLOT, LINE and SPOUT given in the Product Application Note PS1.1 are included as standard in this package; refer to PS1.1 for further details. The only essential difference in their implementation here is that LINE calls a procedure CHECK which ensures that XCOR and YCOR cannot go outside the screen boundaries. The other procedures available are:

INK (C:INTEGER)

this takes an integer between 0 and 8 inclusive and sets the ink colour of the turtle's pen accordingly.

PAPER (C:INTEGER)

sets the background (paper) colour of the screen to the colour associated with the parameter C which is an integer in the range 0..8 inclusive.

COPY

downloads the current screen to the ZX Printer; useful for sending a completed graphics page to the ZX Printer.

PENDOWN (C:INTEGER)

sets the turtle state so that it will leave a trail in the ink colour associated with the parameter C which may be an integer between 0 and 8 inclusive. This procedure assigns 0 to PENSTATUS.

PENUP

subsequent to a call to this procedure the turtle will not leave a trail. Useful for moving from one graphic section to another. PENUP assigns the value 1 to PENSTATUS.

SETHD (A:REAL)

takes a REAL parameter which is assigned to the global variable HEADING thus setting the direction in which the turtle is pointing. Remember that a heading of 0 corresponds to EAST, 90 to NORTH, 180 to WEST and 270 to SOUTH.

SETXY (X,Y:REAL)

sets the absolute position of the turtle within the graphics area to the value (X,Y). No check is made within this procedure to ascertain if (X,Y) is out of bounds; procedure LINE does this check.

FWD (L:REAL)

moves the turtle forward L units in the direction of its current heading. A unit corresponds to a graphics pixel, rounded up or down where necessary.

BACK (L:REAL)

moves the turtle L units in the directly opposite direction to that in which it is currently heading (i.e. -180) - the heading is left unchanged.

TURN (A:REAL)

changes the turtle's heading by A degrees without moving it. The heading is increased in the anti-clockwise direction.

VECTOR (A,L:REAL)

displaces the turtle's position by L units at a heading of A - the turtle's heading remains A after the displacement.

RIGHT (A:REAL)

an alternative to TURN - RIGHT changes the turtle's heading in the clockwise direction by A degrees.

LEFT (A:REAL)

this is identical to TURN and is provided simply for convenience and compatibility with RIGHT.

ARCR (R:REAL, A:INTEGER)

the turtle moves through an arc of a circle whose size is set by R. The length of the arc is determined by A, the angle turned through (subtended at the centre of the circle) in a clockwise direction. Typically R may be set to 0.5.

TURTLE

this procedure simply sets the initial state of the turtle; it is placed in the middle of the screen, facing EAST (heading of 0), on a blue backgound paper and leaving a yellow trail. Remember that the state of the turtle is not initially defined so that this procedure is often used at the beginning of a program.

This concludes the list of facilities available with Hisoft Pascal TURTLE; although simple in implementation and use you will find that Turtle Graphics are capable of prducing very complex designs at high speed. To give you a taste of this we present some example programs below. Remember that you must have Hisoft Pascal loaded before entering the programs.

Example Programs

In all the example programs given below we assume that you have already loaded Hisoft Pascal and used 'G,,TURTLE' to load the Turtle Graphics package which starts at line 10 and finishes at line 1350. Now proceed with the examples:

1. CIRCLES

```
    1 PROGRAM CIRCLES;
    2 VAR I:INTEGER;

1360 BEGIN
```

```
1370  TURTLE;
1380  FOR I:=1 TO 9 DO
1390  BEGIN
1400   ARCR(0.5,360);
1410   RIGHT(40)
1420  END
1430 END.
```

2. SPIRALS

```
   1 PROGRAM SPIRALS;
   2 VAR

1360 PROCEDURE SPIRALS ( L,A:REAL );
1370 BEGIN
1380  FWD(L);
1390  RIGHT(A);
1400  SPIRALS(L+1,A)
1410 END;
1420 BEGIN
1430  TURTLE;
1440  SPIRALS(9,95)        (or (9,90)  or  (9,121)  ... )
1450 END.
```

3. FLOWER

```
   1 PROGRAM FLOWER;
   2 VAR

1360 PROCEDURE PETAL ( S:REAL );
1370 BEGIN
1380  ARCR(S,60);
1390  LEFT(120);
1400  ARCR(S,60);
1410  LEFT(120)
1420 END;
1430 PROCEDURE FLOWER ( S:REAL );
1440 VAR I:INTEGER;
1450 BEGIN
1460  FOR I:=1 TO 6 DO
1470  BEGIN
1480   PETAL(S);
1490   RIGHT(60)
1500  END
1510 END;
1520 BEGIN  TURTLE;
1530  SETXY(127,60);
1540  LEFT(90); FWD(10);
1550  RIGHT(60); PETAL(0.2);
1560  LEFT(60);  PETAL(0.2);
1570  SETHD(90); FWD(40);
1580  FLOWER(0.4)
1590 END.
```

For further, extended study of Turtle Graphics we highly recommend the excellent (if expensive) book 'Turtle Geometry' by Harold Abelson and Andrea di Sessa, published by MIT Press.

HISOFT - PASCAL

QD - Befehle

Druckerausgang auf Centronics schalten

```
POKE ( # 13FC , # 1400 )
POKE ( # 13FB , CHR (#C3))
```

Dateibefehle

```
TOUT ( 'Q : VAR ', ADDR ( ), SIZE, ( ))
TIN  ( 'Q : VAR ',
```

```
CONST  pma = # 1446
```

Direktbefehle

```
P 10, 100, Q : Name
G,, Q : Name

H alte Programme von Kassette
A,, Q : Pascal  =>  Arbeitskopie auf QD
```