PASCAL-80

RAMWARE™

WELCOME TO PASCAL-80

In the ten years since Niklaus Wirth designed the Pascal language, it has become one of the most popular computer languages in the world. Pascal is easy to learn and use, and it encourages well-structured and readable programming. Many students now learn Pascal rather than BASIC as their first language.

There are several different versions of Pascal: Standard Pascal (something of a misnomer, since most "Standard" versions differ in details), UCSD Pascal, and Tiny Pascal. PASCAL-80 is basically Standard Pascal, with some restrictions and extensions, especially in the disk file system.

This instruction manual tells you how to use the PASCAL-80 text editor, monitor, and compiler, and explains the special features of the PASCAL-80 language. It assumes that you have some familiarity with Pascal, or have a text which describes Standard Pascal. Recommended texts: **Programming in Pascal**, Peter Grogono, (Addison-Wesley), **Pascal User Manual and Report**, Kathleen Jensen and Niklaus Wirth, (Springer-Verlag), or **PASCAL**, W. Findlay and D.A. Watt, (Computer Science Press). Texts which describe UCSD Pascal will probably be less useful.

GETTING STARTED

To begin using the PASCAL-80 system, insert the PASCAL disk into drive 0, bootstrap the system, and type PASCAL. After loading is complete, your computer will print a list of things which PASCAL-80 can do and wait for you to choose one. To select an item from this list, just press the appropriate letter (you don't have to press [ENTER]). This display of choices is called the Monitor mode of the PASCAL-80 system.

A good way to get used to the system is to run a simple program. To do this, press E, and you will enter the Editor mode. The screen clears, and a flashing cursor appears. To put program text into the Editor, all you have to do is type a line and press [ENTER]. Note that all keys automatically repeat if you hold them down for a half-second or so.

To erase mistakes, use the backspace key as usual. To fix an error in an earlier part of the line, you can press [SHIFT] with the left-arrow to move the cursor left without erasing (SHIFT right-arrow moves the cursor right). The cursor doesn't have to be at the end of the line when you press [ENTER].

To correct a mistake after pressing [ENTER], use the up-arrow key to move the cursor back up to the line with the error, make the changes, and (very important) press [ENTER] to put the revised version of the line into your program. (For more details, see the section "How to use the Editor").

Try the following:

```
PROGRAM TEST;
VAR NUMBER : INTEGER;
BEGIN
 FOR NUMBER := 1 TO 5 DO WRITELN(1NUMBER)
END.
```

Check your typing (notice the semicolons after the first two lines, and the period at the end...and ":=", not "="). Now press [BREAK], and a line of choices will appear at the bottom of the screen. Press M, and you will find yourself back in Monitor mode. Now press R, and the program should execute. If you get an error message instead, press [ENTER], then press [E] to return to the Editor mode. The cursor will be on the line where PASCAL-80 detected the error, and this line will be at the top of the screen. Very often, the error will be at the end of the previous line (missing semicolon, etc.); to see this line, press the up-arrow. Make the correction, press [ENTER], and try again, with [BREAK]-[M]-[R].

If you're used to BASIC, notice the following differences:

(1) PASCAL is a compiled language. To run your program, the system must first translate your program text (called "source code") to a Machine Language program (called "object code"), then execute the object code. The hexadecimal number at the left of each line in the compiler listing is the number of bytes of object code which have been compiled up to that point: this can help you find the location of an error which occurs during execution (such as division by zero).

(2) The beginning of a program is always indicated by the keyword PROGRAM, with the name of the program.

(3) You must "declare" any variables which you are going to use in your program, using the keyword VAR. Variable names may be any length, and all characters are significant (VARIABLEA is different from VARIABLEB).

(4) PASCAL uses WRITELN instead of PRINT, and it must be followed by parentheses.

(5) The FOR statement has a different syntax from what you're used to in BASIC.

If you're used to other versions of PASCAL, note:

(1) The PROGRAM statement doesn't have to include (INPUT, OUTPUT). These files are automatically available.

(2) PASCAL-80 compiles programs directly into memory, without the need to create source and object disk files (though you can create them if you want to: see the section on "Monitor Mode").

(3) Scientific notation is not the default format for REAL output.

HOW TO USE THE EDITOR

The function of the Editor is to help you prepare a series of program statements. The text which you type is collected into a "text workspace" in your computer's memory, which can then be compiled or saved by the Monitor. Since the video screen is not big

enough to show the entire contents of your workspace, it functions as a movable window, displaying 15 lines of the workspace at a time.

You can correct errors before pressing [ENTER] by using the backspace key to erase characters, or you can press shifted left- or right-arrows to move the cursor in the line without erasing, and then typing over the part of the line which you want to change. (Note that the arrow keys, like all keys, repeat if you hold them down). The Editor sends the entire line to the workspace when you press [ENTER], even if the cursor is not at the end of the line.

The up- and down-arrows move the cursor vertically on the screen. When the cursor gets to the top of the screen, it will push the top of the window up, revealing earlier lines in the workspace (if any). In the same way, you can push the window down through the workspace with the down-arrow.

You can change a line of text in the workspace by moving the cursor to the line (with arrow keys), typing the corrected text, and then pressing [ENTER] to send the new version to the workspace. It's very important to press [ENTER]; if you don't, the new line doesn't get sent to the workspace. If you move the cursor away from the line with arrow keys, without pressing [ENTER], the changes which you made will be canceled (the window will still show the changed version of the line, but you can see the actual state of the workspace by pressing [BREAK] and [X]).

The unshifted right-arrow is a tab key, with tab stops at 2-space intervals in the line. For legibility, it's very important to indent your PASCAL programs so that statements at the same nesting level are indented to the same position.

Pressing [BREAK] will cause a list of choices to appear at the bottom of your screen. You can choose one by pressing the first letter of your choice. To return to normal Editor mode, press [X].

H - Homes the cursor to the beginning of the workspace.

N - Displays the next 15-line page of text (as if you had moved the cursor to the bottom of the screen and pressed down-arrow 15 times).

P - Displays the previous 15-line page of text (if any).

M - Returns to Monitor mode.

I - Inserts a blank line at the cursor position.

D - Deletes the line where the cursor is.

A - Enables the Autotab condition to help you format your PASCAL programs. The Editor will remember the position to which you most recently tabbed, and automatically indent to that point whenever you press [ENTER]. This allows you to type a series of statements at the same nesting level without repeated tabbing. Backspacing to the left of the automatic tab position will set a new automatic position for following lines. Clearing the Editor will cancel the Autotab condition.

MONITOR MODE

In addition to [E], which puts you into the Editor mode, your PASCAL-80 monitor recognizes the following commands (you don't have to press [ENTER] — just press the appropriate key).

Q - Returns you to DOS READY (to consult the directory, for example). You can return to PASCAL with the contents of the Editor intact: just type PASCAL, and hold the [ENTER] key down until the PASCAL-80 system has loaded and the menu of choices appears. Then press [E] to return to your text in the Editor.

K - Erases the contents of the Editor. To prevent accidents, the system will ask you if you really do want to clear the editor. Just press [Y] if you do. (If you don't, press [N] and nothing will happen).

R - Will run the program in the Editor (the "source" program). The program will be compiled if it hasn't already been compiled, or if (1) a previous execution of the program terminated abnormally, or (2) you have returned to the Editor since last running the program, or (3) you have loaded source text into the Editor since running the program.

C - Just compiles the program in the Editor, without trying to execute it.

S - Saves the source program in the Editor to disk — the system will ask you to type a filespec.

L and A - Load text into the Editor from a disk file which you created earlier with the S command. Use L if you want to erase the text that's in the editor, A if you want to add the disk file to the end of the text. Note, however, that if the Editor contains a complete program, the compiler will stop when it gets to the final "END." statement, even if there is more text in the Editor.

W - Will create a machine-code "object" program file which you can later execute with the X command. The advantages of saving a program as object code rather than source text are (1) it takes less disk space (2) more memory is available during execution, since the program doesn't have to share space with the compiler and source program, and (3) you save the time needed to recompile the program.

X - Executes an object program which you created earlier with the W command. To get maximum memory during execution, the system loads the program into memory on top of the compiler and monitor sections of PASCAL-80, thereby destroying them. For this reason, you will return to DOS READY rather than Monitor mode after execution is complete.

The message "BAD FORMAT" during a disk operation means that you are trying to execute a file which does not contain a PASCAL-80 object program, or to load a file into the Editor which does not contain source text.

COMPILER OPTIONS:

You can give instructions to the compiler by typing one of the following six instructions before the keyword PROGRAM. The compiler recognizes only the first letter of an option, and options may be separated by any delimiter, including spaces or carriage-returns: M/H/Z will have the same effect as MEMORY HARDCOPY ZERO.

Four options affect the way in which the compiler lists the program:

HARDCOPY causes compiler output to be routed to the line printer.

NOLIST suppresses the compiler listing (except for error messages).

MEMORY makes each line of compiler output include not only the number of bytes of code compiled so far, but also the number of byes of free stack space available to the compiler and the number of bytes of symbol table space available (in hex).

CODE makes the compiler print each byte of the object program pseudo-code as it is compiled. This is only an approximate listing of the object code, for two reasons: (1) in some situations, such as forward jumps, the compiler generates dummy place-holders, which it changes to the correct values later, and (2) the pseudo-code is self-relocating, and the relative addresses generated by the compiler are replaced by absolute RAM addresses before the execution of each block.

Two options affect the way a program will be executed:

ZERO: A program compiled with this option will set all variables to zero before execution. If you don't use this option, variables will contain whatever happens to be left in memory from the previous program: REAL variables will contain weird and wonderful values which may print as punctuation marks rather than digits.

VERIFY: The program will verify all write operations to disk files. Selecting (or not selecting) the VERIFY option overrides any choice made with the TRSDOS VERIFY command.

LIMITATIONS OF PASCAL-80

Variant records are not implemented, nor is the WITH statement. Pointer variables (along with NEW and DISPOSE), and file window (buffer) variables are not included, nor are the associated GET and PUT procedures. READ and WRITE may be used with non-text files, however: see the section on File Handling. The concept of packed and unpacked structures does not apply, since all structures are packed on byte boundaries (PACKED is not a keyword, and the procedures PACK and UNPACK do not exist).

The procedure PAGE is not included, but the statement WRITE(LP, CHR(12)) may have the same effect, depending on your printer configuration.

Structures of files (such as ARRAY OF FILE) do not exist in PASCAL-80.

PASCAL-80 does not permit the identifier of a procedure or function to be passed as a parameter to another procedure or function. The total size of an expression passed as a value parameter may not exceed 510 bytes (this limitation does not apply to VAR parameters).

Sets may have up to 256 members: if the elements of a set are numeric, they must be in the range 0..255.

EXTENSIONS TO STANDARD PASCAL

Arrays of characters may be printed with a single statement: if STRING is declared as ARRAY (.1..10.) OF CHAR, then WRITE (STRING) is equivalent to

FOR I:=1 TO 10 DO WRITE (STRING (.I.))

In assignments and comparisons of character arrays, a string constant on the right-hand side may be shorter than the item on the left (it will be padded out with blanks as necessary). In the example above, STRING:='NAME' is valid, and after this statement, STRING>'NA' will be TRUE. (The right-hand argument must have at least two characters).

PROC and FUNC are permitted as abbreviations for the keywords PROCEDURE and FUNCTION.

Predefined constants include MININT (-32768) and PI, in addition to the standard FALSE, TRUE, and MAXINT.

In addition to the type REAL (14-digit precision), variables may be declared as REAL6 (six-digit precision), to save space in large arrays (four bytes instead of eight). No significant time is saved, however, since calculations are still performed with 14-digit precision. REAL6 variables which are not members of an array or record may not be passed to a procedure or function as value parameters.

The standard files INPUT and OUTPUT need not be included in the PROGRAM statement (and the program name is also optional). In addition to these files, PASCAL-80 provides a predefined file LP for the lineprinter.

The following procedures are provided:

CLS clears the screen.

POKE (ADDRESS, VALUE) places a value (0 to 255) into a memory location (locations above 8000H are referenced with negative values, as in the BASIC POKE instruction.

CLOSE and SEEK are described in the section on File Handling.

The following functions are provided (in addition to those of Standard Pascal):

INKEY returns the value (type CHAR) of any key pressed. If no key was pressed, it returns CHR(0).

CALL (ADDRESS, VALUE) places a value (0 to 255) into the A register, and CALLs the address. It returns the contents of the A register after the call (type INTEGER).

MEM returns the number of bytes of free memory (type INTEGER).

PEEK (ADDRESS) returns the contents of the address.

FP (EXPRESSION) returns the fraction part (also called "mantissa" or "significand") of a floating point number (type REAL).

EX (EXPRESSION) returns the exponent of a floating point number (type INTEGER).

PASCAL-80 extends the CASE statement in two ways: an ELSE clause may be included, and will be executed if no other case is satisfied; if no case is satisfied and there is no ELSE clause, control simply falls through to the next statement, with no error indication.

Output format differs slightly from Standard Pascal. Both REAL and INTEGER expressions are printed with the statement WRITE (expression : fieldwidth : digits). A fieldwidth of -1 calls for scientific notation ("digits" is ignored, if present); a fieldwidth of 0 produces the default format, which is also used if you don't specify any format parameters: the number is printed with a space before it, and with as many digits after the decimal point as necessary, up to the maximum precision of the computer (14 significant figures). Fieldwidth and digits parameters are evaluated module 256.

You can freeze printed output during execution by holding down the [CLEAR] key. You can freeze the compiler output by holding down the space bar.

READING FROM THE KEYBOARD: FILE (INPUT)

Pascal was originally designed for use with stream input (such as punched cards), rather than for "interactive" input directly from the keyboard. To allow interactive programs, the procedure READLN and the functions EOF and EOLN work differently for input from the keyboard than for input from disk files (see the discussion in Grogono, p. 210). Note that if no file is specified, READ, READLN, EOF, and EOLN are assumed to refer to the keyboard: READ(INPUT,X) is equivalent to READ(X).

In PASCAL-80, EOLN(INPUT) is true if there are no unread characters remaining from the last line which you typed in response to a request for input. In particular, EOLN is true when a program begins execution, and after executing READLN without parameters. This means that if the first statement of your program reads:

```
WHILE NOT EOLN DO BEGIN READ(C); WRITE(C) END
```

nothing will get read, since EOLN is initially true. To read a line of characters, use REPEAT instead:

```
REPEAT READ(C); WRITE(C) UNTIL EOLN
```

Using READLN with parameters (READLN(X), for example), will skip any unread characters in the last line you typed, and prompt you for another line.

In PASCAL-80, EOF is true if (and only if) the next character due to be read is a special end-of-file marker produced by pressing the

[CLEAR] key: it prints as a graphics square (hex 8F), and should appear at the end of a line, following the data. Using this marker allows you to simulate the end of a punched card deck, and use programs written originally for stream-oriented input, without having to convert them for interactive input.

If you are reading from the keyboard into a numeric variable, and the first non-blank character encountered is not a number (or - or +), PASCAL-80 will print the message REDO, and prompt you to enter a valid numeric value. You will also get the REDO message if your program is reading into an integer variable and you type a non-integer (or a number outside the range -32767 to +32767).

You can interrupt a program while it is waiting for imput by pressing the [BREAK] key: this will produce the message BREAK AT 0000.

FILE HANDLING

If your program uses disk files for input or output, Pascal requires that you declare these files in the PROGRAM statement. Since the rules for Pascal identifiers differ from the rules for TRSDOS filespecs, PASCAL-80 allows you to equate a Pascal filename with a filespec, using the following notation:

PROGRAM EXAMPLE (FILEA:'DATAFILE/DAT:1');

Any reference to FILEA in your Pascal program will refer to a file on disk 1 named DATAFILE/DAT. This feature is optional, if you don't need drivespecs or file extensions:

PROGRAM SIMPLE (FILEA, FILEB);

will use FILEA and FILEB as TRSDOS filenames, on drive zero.

Don't forget that Pascal also requires file names to be declared as variables, either with the standard identifier TEXT, or as FILE OF..., for record-oriented files.

TEXT FILES

WRITE (filename, expression) writes the expression to the file. In PASCAL-80, it is not necessary to explicitly open the file: WRITE will automatically open the file, and will create it if it does not exist on the disk. A write statement always adds text at the end of a file: it will skip ahead to the end-of-file position before writing. You can use fieldwidth and digit parameters: the same characters are written to the text file as would be written to the video screen by WRITE (expression). You can write several expressions with one WRITE statement: WRITE (filename, expression1, expression2...).

WRITELN (filename, expression) works like WRITE, but adds an end-of-line marker (ASCII 13) afterwards.

READ (filename, variable) reads a number from the file into a numeric variable, or a character into a character variable. If the file does not exist on the disk, an error occurs (FILE NOT FOUND). If the file is not open, PASCAL-80 opens it, and starts reading at the

beginning. Subsequent READ statements will continue to read through the file. A MISMATCH error occurs if you try to read a non-numeric string into a numeric variable, or a non-integer into an integer variable.

READLN (filename, variable) reads a number or character from the file, then skips to the next end-of-line marker.

RESET (filename) closes a file, then reopens it at the beginning. A READ statement will now begin reading at the first character in the file (but a WRITE statement will skip ahead to the end of the file again). If the file does not exist on the disk, RESET will create it.

CLOSE (filename) closes the file — this is useful in case you want to remove a disk from the drive during the execution of a program. CLOSE without parameters will close any file which is open. All files are automatically closed when a program stops execution, or if an error occurs.

REWRITE (filename) kills the file and releases the disk space; it then reopens a new (empty) file, with the same name.

EOF (filename) is TRUE if the file is positioned at an end-of-file marker (hex 8F). If the file is not open, EOF will cause it to be opened, and will return the value FALSE, unless the file is empty.

EOLN (filename) is TRUE if the file is positioned at an end-of-line marker (hex 0D). The file will be opened if it is not open.

RECORD-ORIENTED FILES

PASCAL-80 also lets you use WRITE and READ with non-text files. The syntax is: WRITE (filename, variable, variable...) and READ (filename, variable, variable...). The variable(s) and filename must be of "identical" type (as defined in Pascal — see page 139 in Grogono).

Assume the following declarations:

TYPE BIGONE = ARRAY (. 1 .. 50 .) OF REAL;

VAR FILEA : FILE OF BIGONE; VARNAME : BIGONE;

You can now write the variable to the file with WRITE (FILEA, VARNAME). The WRITE statement will open (or create) the file, if necessary. Following WRITE statements will add additional copies of the variable after the first one.

READ (filename, variable) will read the variable from the file. The file will be opened, if necessary (but won't be created). The file pointer is advanced after each read.

SEEK (expression, filename) will position the file to the record whose number is given by the expression (the first record in each file is numbered 0, the second is record 1, etc.) If necessary, the file will be opened before the seek. It's not possible to SEEK beyond the 65535th byte of a file.

RESET (filename) may be used with non-text files: it is equivalent to SEEK (0, filename).

REWRITE (filename) and CLOSE (filename) are the same for non-text files as for text files. EOLN, EOF, WRITELN, and READLN are

not defined for non-text files. Attempting to read beyond the end of a non-text file will give undefined results.

Note that WRITE works differently in text and non-text files: in a text file, data is always added at the end, but in a non-text file, reading and writing occur at the same place, as positioned by a SEEK or RESET statement. This permits you to update the contents of a record by overwriting it.

Files may be closed and reopened with a different type: a text file may be later read as FILE OF CHAR, for example.

COMPILATION ERROR MESSAGES:

The compiler will stop when it finds an error, with an arrow pointing to the place where it discovered that something must be wrong. This will often be in the line after the error (for example, if a semicolon is missing after a statement). A missing END may not be discovered for several lines (these are pesky to find, but it helps if you indent nesting levels properly).

BAD OPTION

PASCAL-80 assumes that anything which you type before the keyword PROGRAM is an instruction to the compiler: see the section on Compiler Options.

SYNTAX ERROR

Something's wrong, but the compiler doesn't know what. This error often means that you left out a semicolon between statements or tried to begin a number with a decimal point (PASCAL requires "0.2", not ".2").

UNDECLARED

An identifier (usually a variable) or a label hasn't been declared; PASCAL requires that you declare all variables (in a VAR statement) before you use them.

DUPLICATE

You declared a name twice in the same block. Or you tried to declare a filename with the same name as one of the standard identifiers (Pascal does allow you to redefine these standard identifiers, but not as filenames.)

BAD RANGE

An array or subrange has been declared with an illogical range (such as 10..1).

REAL OVERFLOW

A real constant has a magnitude outside the permitted range (such as 1E99 or 1E-99).

BAD TYPE

An illegal type declaration. Note, for example, that PASCAL requires that parameters be of a predefined type: TYPE RAN = 1.. 10; PROC TEST (PAR:RAN) is OK, but not PROC TEST (PAR:1..10). Note also that PASCAL-80 does not support certain structures, such as FILE OF FILE.

OUT OF MEM.

Usually means that the compiler has run out of symbol table space: you can use the MEMORY compiler option to keep track of how much space is left. It can also occur if the compiler runs out of space for storing labels (maximum of 63) or disk filespecs (maximum of 12). Also, a program may not contain more than 252 different scalar types, and a scalar may not have more than 255 elements. Occasionally a short but deeply-nested program may cause the compiler to run out of stack space before it runs out of symbol table space. If this happens, it will print "OUT OF MEM", and then immediately assign more space to the stack (at the expense of the symbol table) and start the compilation over again. The additional stack space will continue to be available in future compilations until you reload the system.

MISMATCH

An attempt to perform an operation or assignment with elements of incompatible types (such as 'X' + 2). Note that PASCAL allows integers to be assigned to real variables, but (unlike BASIC) it does not allow real values to be assigned to integer variables (use the TRUNC function). Incidentally, -32768 is of type REAL under the rules of PASCAL syntax (it's the negative of 32768, which is a REAL constant), and cannot therefore be assigned to an integer variable. A mismatch can also result from incompatible file operations, such as EOF(OUTPUT) or an attempt to reference a file name which has not been declared in a VAR statement (since the compiler knows the name only as a filespec, and not as a PASCAL identifier).

UNRESOLVED GOTO

The destination of a GOTO statement doesn't exist in the program. It's always printed at the very end of compilation, since the compiler keeps hoping that the label will turn up...

STRUCTURE TOO BIG

An attempt to declare a set with more than 256 members (or with integers outside the range 0..255). Or an attempt to allocate more than 65535 bytes of storage to a block (usually in the form of arrays). Or a structure that is too deeply-nested for the compiler to handle (ARRAY OF ARRAY OF ARRAY...to a depth of about 30). Or an attempt to pass an array (or record) with more than 510 bytes as a value parameter (this restriction does not apply to variable parameters).

BREAK

You stopped the compilation by holding down the [BREAK] key.

RUN-TIME ERROR MESSAGES:

When a run-time error occurs, execution stops, and the system prints (in hex) the location where the error occurred: this location corresponds to the number printed at the left of each line during compilation, and allows you to find the approximate location of the error.

Some run-time errors don't need much explanation: OUT OF MEM., DIV. BY 0, DISK ERROR, BEYOND EOF. Less self-explanatory errors are:

BAD RANGE

A subscript of an array, or the value of a subrange-type variable is outside the range which you specified in your program.

REAL OVERFLOW

The result of a computation has a magnitude outside the range 1 E - 64 <= N < 1E + 63 (this includes both underflow and overflow conditions).

INT. OVERFLOW

The result of a integer operation is outside the range -32768 < = n < = 32767: note that PASCAL (unlike BASIC) does not automatically convert a result to REAL if it is too big.

MISMATCH

An invalid character was found while attempting to read a number from a disk file, or a non-integer was found when trying to read into an integer variable. If this happens when reading a number from the keyboard (file INPUT), the message REDO is printed.

STRUCTURE TOO BIG

An attempt to create a set at run-time with more than 256 elements: for example (.A..B.), if A = 1 and B = 300, or an attempt to assign a bigger set to a set variable than that variable was declared to have room for. Since space is allocated to sets in multiples of 16 elements, a set declared as 0..10 may actually accept elements up to 15 without an error.

ILLEGAL JUMP

It's not legal to jump (with GOTO) into a FOR loop or CASE statement or into an inactive procedure or function. In general, you can jump from a deeper nesting level to a shallower level (out of a FOR loop, for example), but you can't jump deeper. If you jump out of a function without assigning a value to the function, the value 0 will be returned. PASCAL-80 will allow you to jump from inside one FOR loop to another one at the same nesting level, but the value of the control variable will be undefined.

OPERATION UNDEFINED

The value of a function is not defined: for example SQRT(-1). Or a procedure or function was declared using a forward reference but never actually defined. Or you tried to use < or > as set operators. This message is also likely to be printed if the system gets lost after a bad disk load or memory fault.

Specifications for PASCAL-80.

Requires Model I, one disk (or more), 32K (48K recommended). Model III available soon.

Includes RECORD, SET (to 256 members), FILE (text and record-oriented), n-dimensional ARRAY (and ARRAY OF ARRAY, etc.), global GOTO, ELSE in CASE statements.

BCD arithmetic accurate to 14 digits (including log and trig functions); six digits optional.

23,600 bytes available for user programs in 48K (33,000 available at run-time). Fast compiler (written in native code) compiles over 1000 lines per minute from easy-to-use text editor. Programs may be compiled directly into memory and run without using intermediate disk files, if desired. Error messages are in plain English.

Types: boolean, integer, char, real, real6, text.

Constants: maxint, minint, true, false, pi.

Files: input, output lp.

Procedures: read, readln, write , writeln, reset, rewrite, close, seek, cls, poke.

Functions: abs, arctan, call, chr, cos, eof, eoln, exp, inkey, ln, mem, odd, ord, peek, pred, round, sin, signif, sqr, sqrt, succ, trunc.

Does not implement variant records; pointer and window variables; functions or procedures used as paramenters; all structures are packed (on byte boundaries).

Recommended text: Peter Grogono, **Programming in Pascal** Addison-Wesley, 1980).

6 South Street, Milford, NH 03055