TRS-80™ COMPUTING

VOL. 1, No. 4 16 PAGES $1.50

PEOPLES

(NEW FORMAT: Subscribers who have been punching their copies for binder should put holes in open end of booklet, opposite to previous practice.)

PEOPLE'S PASCAL II
USER'S MANUAL, FOR TRS-80

By KIN-MAN CHUNG
And HERBERT YUEN

(The authors wrote, in the September, October and November "Byte" magazine, a three-part article titled: "A 'Tiny' Pascal Compiler", including listing for North Star Basic. Since the "Byte" series, they have re-written their tiny Pascal operating system in tiny Pascal, and compiled it into Z80-native code, for TRS-80.)

PEOPLE'S PASCAL OPERATING SYSTEM'S

Your 'Tiny' Pascal system, hereafter called People's Pascal II, is a complete, self-contained operating system for creating, compiling, running, saving and loading Pascal programs for the TRS-80. Once you have loaded People's Pascal II, you never need leave the operating system. The People's Pascal II system is composed of three inter-related sections:

Monitor: this is the sub-system which provides run-time support, checks for errors, and provides the necessary utilities to save and load programs to and from cassette tape.

Compiler: this is the program which compiles your Pascal source program into P-code, ready to be executed. The compiler also checks for syntax errors.

Editor: the editor is used to create or modify People's Pascal source programs.

All these sub-systems are loaded simultaneously, and are always present in RAM, unless you choose to overwrite portions to free memory space.

MINIMAL SYSTEM requirements are: Level II, 16K RAM!

The first sections of this users' manual will discuss in detail the three subsystems, what they do, and how to use them. The next section will deal with the specific aspects, limitations and enhancements to People's Pascal II; then follows a chapter on getting started, to help you get through the first time you bring People's Pascal up. Finally, you will find the error codes, syntax diagrams, and the sample programs.

PEOPLE'S PASCAL MONITOR:

All operations make at least some use of the monitor, hence we will begin our discussion of the People's Pascal II system with it.

The monitor provides run-time support to the entire system, as well as providing you with a means of saving or loading both source programs, and P-code programs from or to cassette tape. From the monitor one also gives the command to compile a program, or to run that program once it has been compiled. You also invoke the editor from the monitor. Below is a list of the monitor commands and what they do:

- E Edit old source file or create a new one.
- C Compile source program into P-code, ready to be executed. P-code is placed after source in RAM.
- C/-P Compile source, but do NOT generate P-code (useful to check for syntax errors).
- C/-S Compile source, and overwrite the source program (used when you have very large programs)
- R Run the compiled program.
- R/-C Run the compiled program and overwrite the editor and compiler.

- LS >filename< Load source program from cassette.
- LP >filename< Load P-code program from cassette.
- WS >filename< Save source program to cassette.
- WP >filename< Save P-code program to cassette.

It should be noted that you are given the ability to overwrite sections of the People's Pascal system if you need the space for large programs. However, you must remember that they are "gone" and you must re-load the entire system again if you are to use them further.

It should also be noted at this time that a filename can be at most six (6) characters long. Errors will result if this is not adhered to.

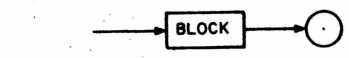
THE PEOPLE'S PASCAL EDITOR:

The text editor provided with your People's Pascal package enables you to create and modify source programs.

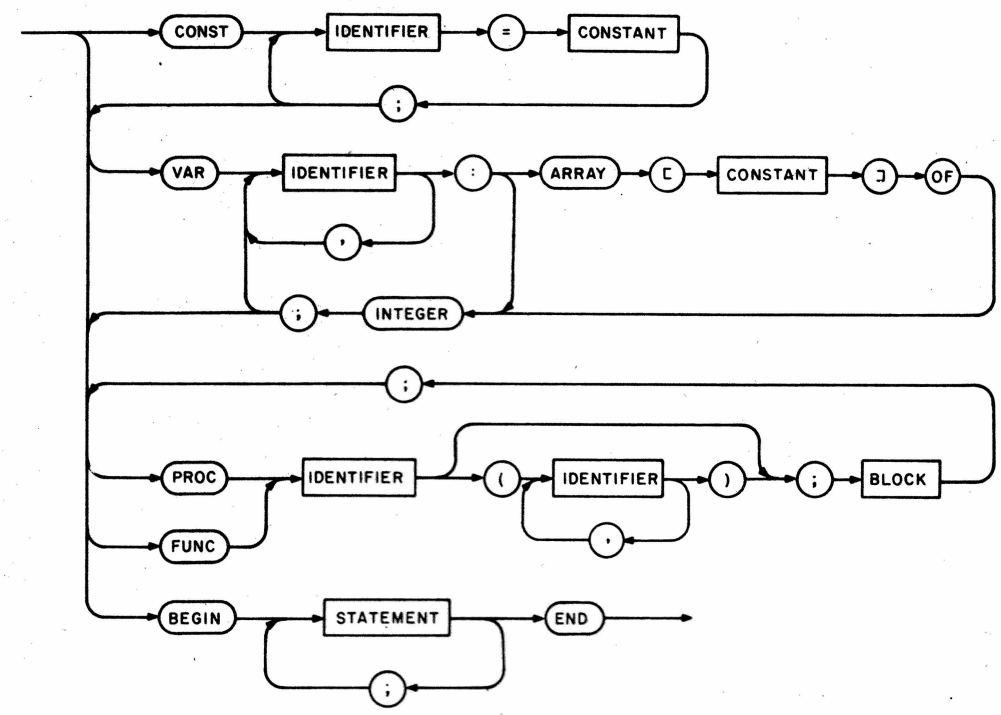
The text editor is line oriented, but, unlike Basic, does not use line numbers. The maximum number of lines of text that you can have is 600, and the maximum line length is 130 characters.

All editor commands are single characters; some may have numeric arguments following them, or a character string.

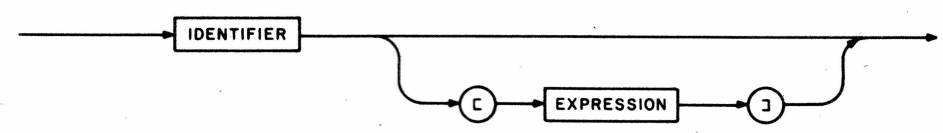
PROGRAM



BLOCK



VARIABLE



In our discussion of the editor, "xx" refers to integer numbers (1-999), and >string< refers to a string.

Each command ends with a >cr<, carriage return ("ENTER" on your TRS-80 keyboard). Invalid commands are flagged with the message "ILLEGAL".

The line pointer always points at the line most recently displayed or modified, or inserted. After a Delete command, the line pointer is moved up one line.

Below is a list of the editor commands. Note: "*" means entirely or "all the way":

>cr<		A carriage return on an empty line will exit from insert mode.
PRINT	P	Print the current line.
	Pxx	Print xx lines starting from current line
	P*	Print entire file
Up	U	Move up one line
	Uxx	Move up xx lines.
	U*	Move up to top of first line of file.
NEXT	N	Move line pointer to next line (down).
	Nxx	Move line pointer down xx lines.
	N*	Move line pointer to last line of file.
Delete	D	Delete current line.
	Dxx	Delete xx lines starting at current line.
	D*	Delete entire file (i.e., "scratch").
Insert	I	Enter insert mode (remember, you exit with a >cr< Insert lines after current line pointer. A "?" is displayed to prompt you.
Replace	R>st<	Replace the current line by >string<.
Extend	X	The current line is displayed and the cursor is at the end of the line, more characters can be appended to the end (similar to Basic).
Status	S	Status of current file displayed includes: number of lines, file location, position of line pointer.
QUIT	Q	Return to People's Pascal II monitor.

The editor also recognizes two special keys:

(—, the back arrow, for backspace, and —), the right arrow, for tab, which is three spaces.

These two keys may be used at any time for editing a command or input file.

Expanding on this: if you want to enter a program, you would type "E" from the monitor, then you would type "I" for insert. You then can enter text. To stop entering text, you type a blank carriage return on an empty line.

Hint: When "MEMORY FULL" error occurs while editing or inserting, the source is too big.

You should play with the eidtor a while to make sure that you completely understand its operation.

PEOPLE'S PASCAL COMPILER:

Roughly speaking, a compiler is a program that translates the statements of a high-level language into an equivalent program of machine-readable form.

People's Pascal II translates the high-level source program into an intermediate file called P-code.

P-code is then interpreted, using the run-time monitor for support.

The result is programs which execute at least four times faster, and up to eight times faster than Basic!

People's Pascal II is a subset of standard Pascal. The syntax is essentially identical to its larger brother. Syntax diagrams have been included for those who are just now learning the language.

It must be emphasized that this manual is not an instructional text on Pascal programming, but rather an explanation of the limits and special features of People's Pascal. However, we will review some essential points in the next section.

For those who need a more thorough introduction, we recommend the following (partial) list of books:

Programming in Pascal; Grogono Addison-Wesley, 1978

Pascal: User Manual and Report; Jensen and Wirth, Springer-Verlag, 1974

A Primer on Pascal; Conway, Gries and Zimmerman; Winthrop Publishers, 1976

COMPILER SPECIFICS:

Maximum number of procedure or function parameters is 15; maximum number of procedure nestings is seven levels; the symbol table is restricted to 75 (200 for the big version).

":=" is used for assignment and "=" is used for equality. They are not interchangeable!

";" is used to separate statements, not to end statements. Thus the last ";" in a compound statement:

```
BEGIN STATEMENT
  STATEMENT;
  IF>EXP< THEN >EXP< ELSE
  >EXP<; STATEMENT ;
END
```

is not necessary. It is, however, allowed since a Pascal statement can be a null. Note also the absence of ";" before an ELSE or an END in the "CASE" statement.

Expressions may be either arithmatic or logical (Boolean). Thus, the following are perfectly legal:

```
A := B < C;
.
.
IF A+B THEN.....
```

Note also that the Boolean operator "OR" has the same precedence as the arithematic operator "+" and "-"; "AND" the same as "*" and "DIV", etc. It is important to remember that "OR" and "AND" have precedence over "=", ">", etc, thus the need for brackets at times as shown below:

IF (A > 10) AND (A < 5) THEN...

The statement:

IF A > 10 AND (A < 5) THEN...

would be parsed as:

IF A > (10 AND (A < 5)) THEN...

thus producing the undesirable result.

There are some context-sensitive rules and meanings that cannot be inferred from the syntax diagrams, and may be particular to this implementation:

"(" and ")" are used in the TRS-80 implementation instead of "[" and "]".

Identifier names must start with a letter and may be followed with letters or digits, but only the first four characters are significant. However, reserved words must be typed in full.

Identifiers must be declared before used. Identifiers can be declared twice, but only the last one is used. Formal parameters of a procedure need not (and should not) be declared again inside the procedure.

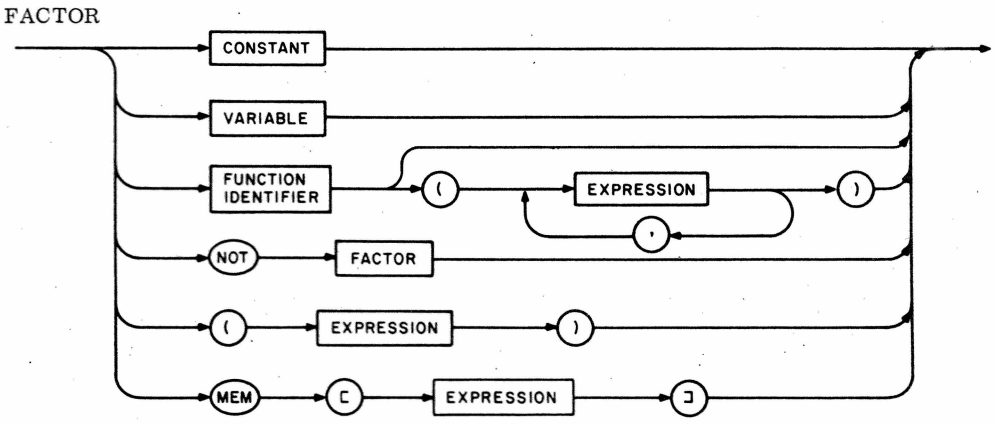
Parameters are passed to procedures or functions by value, i.e., a copy of the value of the parameter by the program before the call.

The scope rules for identifiers are the same ones used by any block-structure language. The scope of a variable is the procedure that contains it. An inner procedure can reference a variable in an outer procedure.

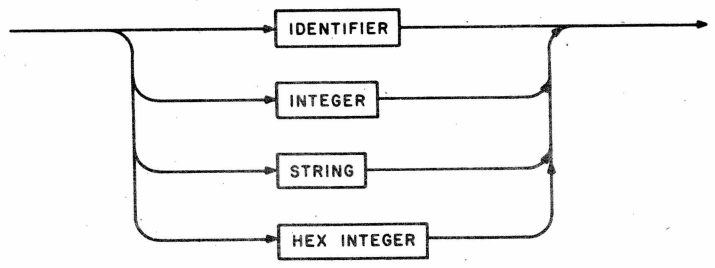
The only data types People's Pascal supports are integers are one-dimensional integer arrays. The integers are 16-bit signed, the arrays start at 0. Arrays are NOT checked

SYNTAX DIAGRAMS for People's Pascal, a Pascal subset. Cuts above and below, and on page 3, courtesy Byte magazine, from September '78 issue, the first of the three-part series, "A 'Tiny' Pascal Compiler", by Kin-Man Chung and Herbert Yuan. Copyright Byte magazine, used with permission.

FACTOR

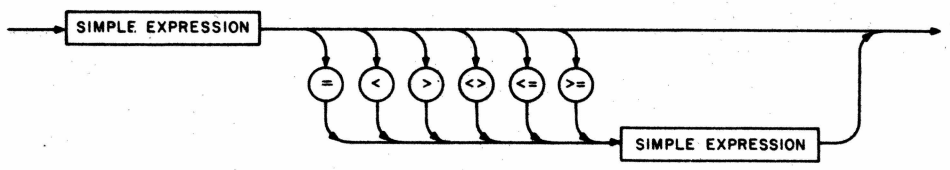


CONSTANT

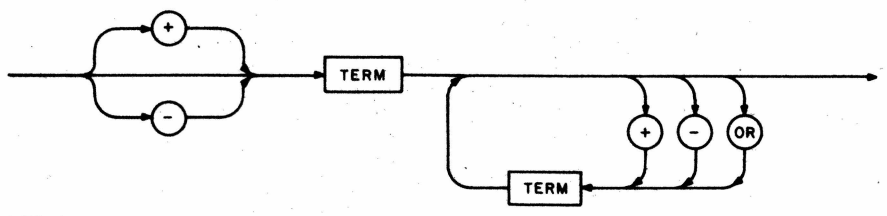


Continued: Notice that some of the diagrams, for example Factor, contain themselves in their own definitions. This is known as a recursive definition.

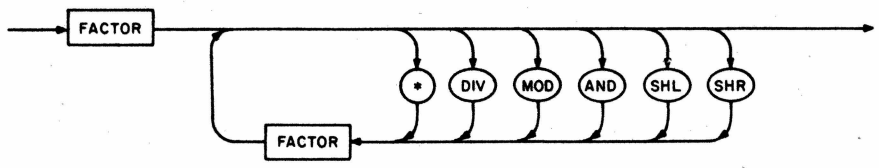
EXPRESSION



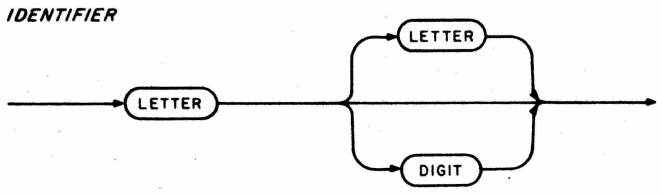
SIMPLE EXPRESSION



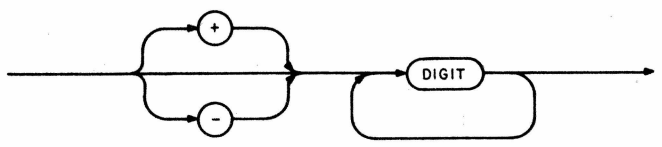
TERM



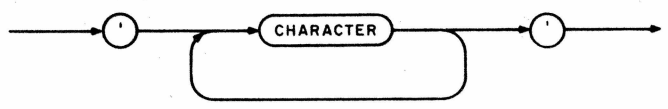
IDENTIFIER



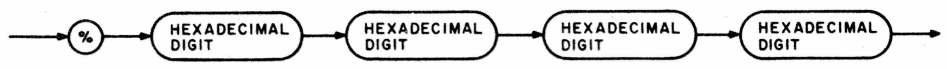
INTEGER



STRING



HEXINTEGER



for "subscript out of range" at runtime.

The meaning of certain operations is:

A DIV B —truncated integer division:
27 DIV 5 = 5

A MOD B —A - (A DIV B)*B :
27 MOD 5 = 2

A SHL B —LEFT SHIFT A BY B :
27 SHL 2 = 54

A SHR B —RIGHT SHIFT A BY B :
27 SHR 2 = 13

The built-in array MEM can be used to read to (if it appears in the left side of an assignment) or from (if it appears in an expression) to or from (if it appears in an expression) to or from a specified memory location, such as:

A := MEM (24467) +3;
MEM (T) := 0;

A second form of the MEM function is "MEMW". This enables a two-byte word to be read to or from memory using the same convention as for "MEM". Note: the low-order byte comes first, in accordance with INTEL convention.

Hex constants are prefixed by % (e.g., %2A00).

Strings are enclosed by single quote ('), not double. When a string appears in an expression or as a CASE lable, it has the value equal to the ASCII value of the first character of the string. When a string appears in the WRITE statement, the entire string would be outputed. Such as:

X := 'ABCD' X would = 'A' = 65

The READ and WRITE statements are character-oriented, not line-oriented. More than one character can be placed in the same statement. Decimal numbers or Hex numbers can be read-in from the keyboard by a "#" (decimal) or "%" (hex) after the variable in the READ statement. Similarly, a decimal integer can be printed on the output device by following the expression with the appropriate "#" or "%" for Hex.

READ (A,B,C, I#,J%)

This would READ three (3) characters, a decimal number, and a hex number.

A := 65
WRITE ('HELLO? ',A,' ',A#,'',A%)

would print:

HELLO? A 65 0041

Since the READ is character-oriented, it is necessary to terminate an integer input by a non-integer character (such as a >cr< or >sp <

To input a hex number, four (4) digits must be typed.

To write on a new line, it is also necessary to output explicitly the ASCII code for >cr< and >lf< to the output device. That is, you must manually insert carriage return : line feed. Such as:

WRITE ('THIS IS A TEST',13,10) (HERE CR = 13, LF = 10)

An expression in the IF, WHILE, and REPEAT statements are said to fulfill the condition if the least-significant bit is 1. This is equivalent to test that the expression is odd. Thus after:

IF X THEN A := 1 ELSE A := 100

A would have the value of 1 if X is odd, and 100 if X is even.

The relational operators (e.g. "=", "= " ... etc) always produce a value of 0 or 1. Thus after :

A := X = 5;

A= 1 :F X=5, OTHERWISE A=0

Comments are delimited by " (*" and " *)".

What follows is a list of built-in functions to the compiler:

ABS(X) : returns the absolute value of X
SQR(X) : returns square of X
INP(X) : inputs port X, used as: A = INP(X)
OUTP(X,A) : outputs A to port X
INKEY : inputs the keyboard, used as in: A := INKEY
PLOT(X,Y,A) : plots graphics to screen, using the X-Y coordinates. If A is odd then plot is "set", if A is even then plot is "reset".
POINT(X,Y) : just like Basic: returns a "1" if the point is filled, a "0" if blank
MOVE(B,A,N) : move a block of memory of N bytes from address A to address B.

Screen control characters are the same as TRS-80 Basic. For example, use WRITE (23,31) to clear the screen.

BRINGING UP PEOPLE'S PASCAL

In this section of the People's Pascal users' manual, we will go step by step from loading the tape the first time, to running your first program.

Side one of your tape comes with three sample programs, the first is loaded with the system, the second is "HILBER" and the third is "BLOCK".

Side two contains the big version and source to People's Pascal, "PAS32K" and "COMPS1", respectively.

STARTUP

1) Turn on your machine. When asked for MEMORY SIZE, respond by hitting the ENTER key.
2) Type SYSTEM to reach system level. TRS-80 will display the prompt: *?.
3) Make sure that your People's Pascal tape is at the start, and type PASCAL and then ENTER and turn recorder to PLAY.
4) The tape will begin to load, the star will blink every 4 seconds. The entire load will take about 3 min.
5) Once the tape has loaded, type a "/" (slash) and hit ENTER. At this point you should receive the opening message:
   "TINY PASCAL V-1.0"
6) At this point you have successfully loaded the entire People's Pascal operating system, and can proceed to the next section, below.

   If you did not get this far try loading the tape again, at various volume settings. Mark down, on the cassette label, the volume setting that was successful. If it will not load, and other commercial tapes will load, return it to CIE for replacement.

CREATING A PROGRAM:

1) From the monitor, type "E". This will place you in the editor.

   You will see one of two messages, either: EMPTY FILE...ENTER TEXT,—this is when there is no current source program, or you will see: a set of statistics on the current file.

   On the initial load, the sample program is loaded simultaneously. If this is your very first try, then skip ahead to step 5, otherwise proceed.
2) To "scratch" the sample program which is always loaded with the system, you simply use the editor command: D*.
3) At this point you may enter a program.
4) Once your program is entered, you may exit insert mode by hitting an ENTER on the next blank line. This puts you back in the editor command mode.
5) To return to the monitor, in order to compile, etc., you type Q.

COMPILING, RUNNING SAVING/LOADING A PROGRAM:

1) Normally, to compile a source program, you type C from the monitor. This creates P-code. If you have any syntax errors, they will show up here.

   If you have syntax errors, the error list on the back of this manual will tell you what they are. You should then go back and re-edit the existing source file, correcting the syntax errors, before re-compiling.
2) Once you have successfully compiled the program, you may run it by typing R from the monitor.
3) To save the program, or the P-code, You may use the appropriate monitor commands. Or you may load a previously-saved program.

Remember, you must re-compile a program if you make a change in it!

SPECIAL NOTES:

It should be noted that the BREAK key equals a temporary stop of program execution, and that any other key re-starts it. If you hit BREAK twice in a row, you will terminate the run, and return to the People's Pascal monitor (like a control-C on most other systems).

One should also note that, once a program has been compiled, only the P-code (that is, the compiled program) need be loaded for execution. In other words, it is not necessary to compile before each execution if you have saved the P-code on tape.

When error 1001 is encountered during compilation, there is not enough memory. You should try using C/-P. Be sure to save the source first!

When MEMORY FULL error occurs on running the program, either cut down array size, or try using R/-C option.

We know that you will enjoy using People's Pascal, and recommend that you "play" with it a while just to get the feel for it, and to become familiar with all of its features.

USING THE 'BIG' PASCAL ON SIDE (B):

On side two of your tape is an expanded People's Pascal compiler. That is, it can handle larger programs. You will need at least 36K RAM to use it.

To use, simply follow the directions "On Bringing Up People's Pascal", except substitute PAS32K for PASCAL.

The source to the compiler is immediately after PAS32K on side B. It is called: COMPS1. You can then "play" with the source to the compiler. Note: you will need at least 36K to compile the compiler.

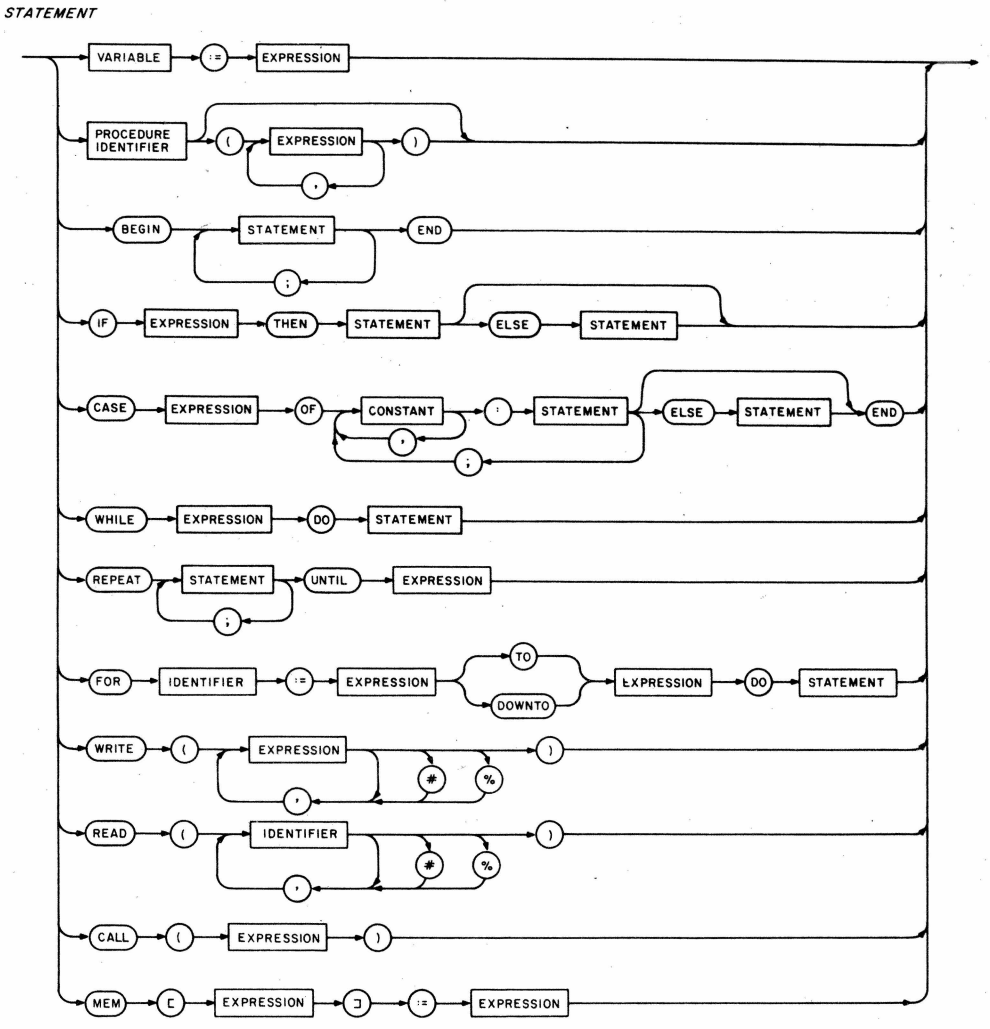
IMPORTANT: Source programs are not interchangeable between the two compilers. That is, a program created using the big compiler can NOT be used with the normal compiler, and vice versa.

ERROR CODES:

1	error in simple type
2	identifier expected
3	"program" expected
4	) expected
5	: expected
6	illegal symbol
7	error in parameter list
8	OF expected
9	(expected
10	error in type
11	(expected
12	) expected
13	END expected
14	; expected
15	integer expected
16	= expected
17	BEGIN expected
18	error in declaration part
19	error in field-list
20	, expected
21	* expected
50	error in constant
51	:= expected
52	THEN expected
53	UNTIL expected
54	DO expected
55	TO/DOWNTO expected
56	IF expected
57	FILE expected
58	error in factor
59	error in variable
101	identifier declared twice
102	low bound exceeds high bound

ELEMENTARY CONSTRUCTS for Pascal subset, cont.: Hexinteger is usually not defined in Pascal but is used here so that actual memory location can be easily manipulated. Copyright Byte magazine, used with permission.

STATEMENT



103 identifier is not of appropriate class
104 identifier not declared
!)% SIGN NOT ALLOWED
106 number expected
107 incompatible subrange types

108 file not allowed here
109 type must not be real
110 tagfield type must be scalar
111 incompatible with tagfield type
112 index type must not be real
113 index type must be scalar
114 base type must not be real
115 base type must be scalar
116 error in type of standard procedure parameter
117 unsatisfied forward reference
118 forward reference type identifier in variable declaration
119 forward declared; repetition not allowed
120 function result type must be scalar
121 file value parameter not allowed
122 forward declared function, repetition not allowed
123 missing result type in function declaration
124 F-format for real only
125 error in type of standard function parameter
126 number of parameters does not agree with declaration
127 illegal parameter substitution
128 result type of parameter function does not agree with declaration
129 type conflict of operands
130 expression is not of set type
131 tests on equality allowed only
132 strict inclusion not allowed
133 file comparison not allowed
134 illegal type of operand
135 type of operand must be Boolean
136 set element type must be scalar
137 set element types not compatible
138 type of variable is not array
139 index type is not compatible with declaration
140 type of variable is not record
141 type of variable must be file or pointer
142 illegal parameter substitution
143 illegal type of loop control variable
144 illegal type of expression
145 type conflict
146 assignment of files not allowed
147 label type incompatible with selecting expression
148 subrange bounds must be scalar
149 index type must not be integer
150 assignment to standard function is not allowed
151 assignment to formal function is not allowed
152 no such field in this record
153 type error in read
154 actual parameter must be a variable
155 control variable must neither be formal nor non-local
156 multidefined case label
157 too many cases in case statement
158 missing corresponding variant declaration
159 real or string tagfields not allowed
160 previous declaration was not forward
161 again forward declared
162 parameter size must be constant
163 missing variant in declaration
164 substitution of standard procedure/function not allowed
165 multidefined label
166 multideclared label
167 undeclared label
168 undefined label
169 error in base set
170 value parameter expected
171 standard file was redeclared
172 undeclared external file
173 (not relevant)
174 Pascal procedure or function expected
175 missing input file
176 missing output file
201 error in RREAL constant: digit expected
202 string constant must not exceed source line
203 integer constant exceeds range
204 (not relevant)
250 too many nested scopes of identifiers
251 too many nested procedures and/or functions
252 too many forward references of procedure entries
253 procedure too long
254 too many long constants in this procedure
255 too many errors on this source line
256 too many external references
257 too many externals
258 too many local files
259 expression too complicated

300 division by zero
301 no case provided for this value
302 index expression out of bounds
303 value to be assigned is out of bounds
304 element expression out of range

398 implementation restriction
399 variable dimension arrays not implemented
1000 . missing
1001 out of memory

USEFUL CALLS, ADDRESSES INSIDE THE MONITOR:

Below is a list of useful addresses for those who may wish to use them.

address	function
4180 (hex)	starting address of source
4182	ending address of source
4184	start of P-code
4186	end of P-code
4188	address of editor
418A	address of compiler
418C	start address of user source program
418E	address of run-time stack
4190	ending address of run-time stack
4192	end of memory address (7FFF for 16K)
4194	monitor entry point
4196	address of program currently executing
4198	complement of contents of 418E
419A	overflow message flag — default 0
----	I/O CALLS ----
41A0	console in
41A2	console out
41A4	INKEY (input the keyboard—CR [ENTER] not needed)

```
 (* SAMPLE TINY PASCAL PROGRAM  BY H. YUEN
VAR X0,Y0,X,Y,K,F:INTEGER;
BEGIN
   X0:=13000; Y0:=18000; F:=11;
   REPEAT X:=X0; Y:=Y0; WRITE(15,28,31);
     FOR K:=1 TO 1000 DO BEGIN
        X:=X+Y DIV 4; Y:=Y-X DIV 5;
        PLOT(X SHR 8,Y SHR 8,1) END;
     X0:=X0+X0 DIV F; Y0:=Y0+Y0 DIV F;
     F:=F+F DIV 6
   UNTIL F>70; WRITE(28,31,'THE SHOW IS OVE
END.
```

```
 (* SAMPLE TINY PASCAL PROGRAM  BY H. YUEN *
VAR X0,Y0,X,Y,K,F:INTEGER;
BEGIN
   X0:=13000; Y0:=18000; F:=11;
   REPEAT X:=X0; Y:=Y0; WRITE(15,28,31);
     FOR K:=1 TO 1000 DO BEGIN
        X:=X+Y DIV 4; Y:=Y-X DIV 5;
        PLOT(X SHR 8,Y SHR 8,1) END;
     X0:=X0+X0 DIV F; Y0:=Y0+Y0 DIV F;
     F:=F+F DIV 6
   UNTIL F>70; WRITE(28,31,'THE SHOW IS OVER'
END.
```

```
(*PLOT HILBERT CURVES OF ORDERS 1 TO N*)
CONST N=4; H0=32;
VAR I,H,X,Y,X0,Y0,U,V:INTEGER;
PROC MOVE;
VAR I,J:INTEGER;
   FUNC MIN(A,B);
      BEGIN IF A>B THEN MIN:=B ELSE MIN:=A END;
   FUNC MAX(A,B);
      BEGIN IF A<B THEN MAX:=B ELSE MAX:=A END;
   BEGIN FOR I:=MIN(X,U) TO MAX(X,U) DO
      FOR J:=MIN(Y,V) TO MAX(Y,V) DO
         PLOT(I,J,1);
      U:=X; V:=Y
   END;

PROC P(TYP,I);
   BEGIN IF I>0 THEN
      CASE TYP OF
1:       BEGIN P(4,I-1); X:=X-H; MOVE;
            P(1,I-1); Y:=Y-H; MOVE;
            P(1,I-1); X:=X+H; MOVE;
            P(2,I-1) END;
2:       BEGIN P(3,I-1); Y:=Y+H; MOVE;
            P(2,I-1); X:=X+H; MOVE;
            P(2,I-1); Y:=Y-H; MOVE;
            P(1,I-1) END;
3:       BEGIN P(2,I-1); X:=X+H; MOVE;
            P(3,I-1); Y:=Y+H; MOVE;
            P(3,I-1); X:=X-H; MOVE;
            P(4,I-1) END;
4:       BEGIN P(1,I-1); Y:=Y-H; MOVE;
            P(4,I-1); X:=X-H; MOVE;
            P(4,I-1); Y:=Y+H; MOVE;
            P(3,I-1) END
      END
   END;

BEGIN (*MAIN*)
   WRITE(15,28,31,13,' HILBERT CURVES');
   I:=0; H:=H0; X0:=H DIV 2; Y0:=X0;
   REPEAT I:=I+1; H:=H DIV 2;
      X0:=X0+H DIV 2; Y0:=Y0+H DIV 2;
      X:=X0+(I-1)*32; Y:=Y0+10; U:=X; V:=Y;
      P(1,I)
   UNTIL I=N
END.
```

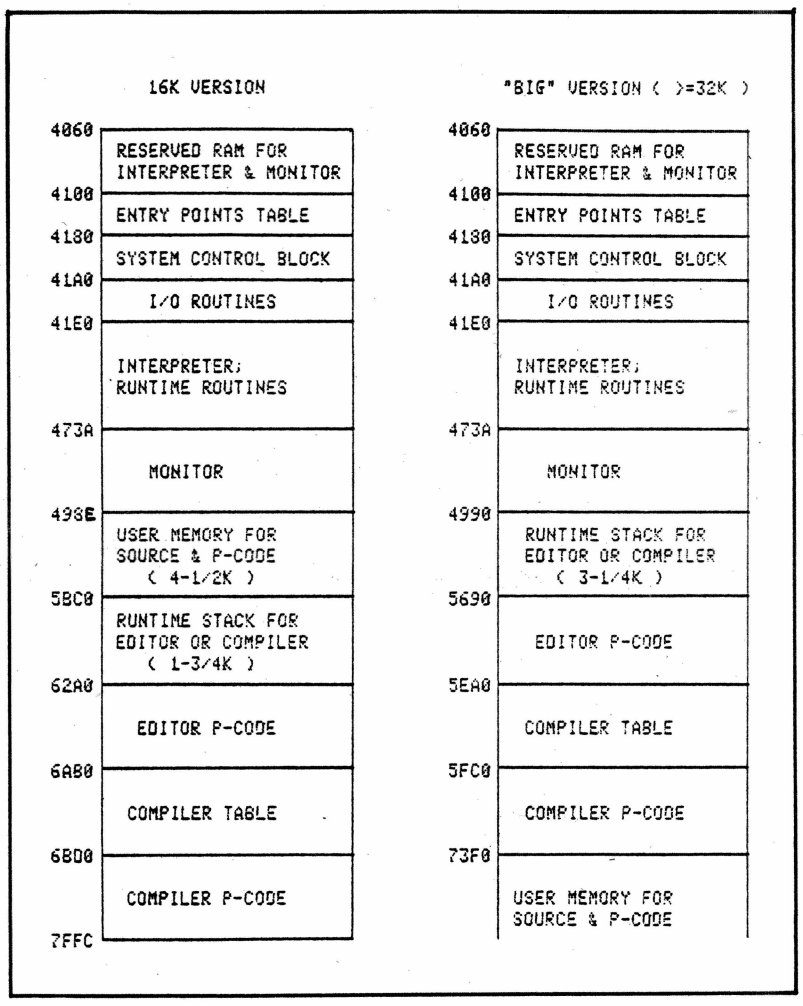
```
 (*BLOCKADE.  BY K.M.CHUNG.  4/26/79*)
VAR I,J,SPEED,ABORT,BLNK:INTEGER;
   SCORE,MARK,MOVE,CURSOR:ARRAY(1) OF INTEGER;
PROC PSCORE;
   BEGIN WRITE(SCORE(0)#);
      MEMW(%4020):=%3FFE; (*SET CURSOR*)
      WRITE(SCORE(1)#) END;

PROC BLINK;
VAR T,K,DELAY:INTEGER;
   BEGIN T:=CURSOR(I)-MOVE(I);
      FOR K:=1 TO 30 DO BEGIN
         FOR DELAY:=1 TO 100 DO;
         IF MEMW(T)=BLNK THEN MEMW(T):=MARK(I)
            ELSE MEMW(T):=BLNK
         END
   END;
```

```
BEGIN WRITE('SPEED(1-10)');
   READ(SPEED#); SPEED:=SPEED*10;
   MARK(0):='*'+'*'SHL 8; MARK(1):='('+')'SHL 8;
   BLNK:=' '+' 'SHL 8;
   SCORE(0):=0; SCORE(1):=0;
   REPEAT WRITE(15,28,31); (*TURN OFF CURSOR, CLEAR SCREEN*)
      FOR I:=9 TO 117 DO BEGIN
         PLOT(I,1,1); PLOT(I,45,1) END;
      FOR I:=1 TO 45 DO BEGIN
         PLOT(9,I,1); PLOT(10,I,1);
         PLOT(116,I,1); PLOT(117,I,1) END;
      CURSOR(0):=%3C00+64*4+12;
      CURSOR(1):=%4000-64*4-16;
      FOR J:=0 TO 1 DO MEMW(CURSOR(J)):=MARK(J);
      MOVE(0):=64; MOVE(1):=-64;
      I:=1; ABORT:=0; PSCORE;
      REPEAT UNTIL INKEY<>0; (*HIT KEY TO START*)
      REPEAT I:=1-I;
         FOR J:=1 TO SPEED DO
            CASE INKEY OF
               'W':MOVE(0):=-64; 'X':MOVE(0):=64;
               'D':MOVE(0):=2;   'A':MOVE(0):=-2;
               'O':MOVE(1):=-64; '.':MOVE(1):=64;
               ';':MOVE(1):=2;   'K':MOVE(1):=-2
            END;
         CURSOR(I):=CURSOR(I)+MOVE(I);
         IF MEMW(CURSOR(I))=BLNK THEN MEMW(CURSOR(I)):=MARK(I)
         ELSE BEGIN SCORE(1-I):=SCORE(1-I)+1;
            ABORT:=1; BLINK END
      UNTIL ABORT
   UNTIL SCORE(1-I)>=10
END.
```

PEOPLE'S PASCAL II:

MEMORY MAPS



STATEMENT OF WARRANTY & TRANSFERABILITY

People's Software, Pipe Dream and Supersoft disclaim all warranties with regard to the software contained on tape, disk or listed in manual, including all warranties or merchantability and fitness, beyond sales price paid, and reject all liability for indirect or consequential damages arising out of or in connection with the use of People's Software.

TRANSFERABILITY:
People's Software, Supersoft and Pipe Dream software and manauals are sold on an individual basis and no rights for duplication are granted. Title and ownership of the software shall at all times remain with the authors.

AN INTRODUCTION TO PROGRAMMING IN PASCAL

By CHIP WEEMS
Graduate Teaching Assistant, Dept. of Computer Science, Oregon State Univ.
Corvallis OR 97331

(Chip Weems wrote the following for the Second West Coast Comprter Fair Proceedings, and it is considered a very good explanation of Pascal. This is still available for $15 from the Faire, Box 1579, Palo Alto CA 94302. We have eliminated the author's discussion of record and file types, since it deals with features not available in People's Pascal. Since cuts really compromise the integrity of the writing, we leave in discussion of other features not present in People's Pascal, GOTO, labels, READLN, WRITELN. It should be pointed out that READ and WRITE are present in People's Pascal, but in different form. People's Pascal documentation, especially the syntax diagrams, defines exactly what features are present. ED)

ABSTRACT:

This paper will concentrate heavily on the use of the Pascal language at the beginner's level. A minimal knowledge of some other programming language such as Fortran, Basic or Algol, is assumed.

The areas which will be covered are simple and structured statements in Pascal, simple and structured (deleted—Ed) data types, plus procedures and functions. Emphasis will be placed on using Pascal statements, although some discussion of the power of user-defined data types will also be included.

PART ONE: WHAT IS PASCAL?
HISTORICAL INTRODUCTION:

Pascal is not an acronym, unlike many languages, the letters which make up its name do not stand for anything. This is perhaps a first indication that Pascal is something different and a little special.

Pascal was named after the famous mathematician Blaise Pascal (1623-1662) who invented, among other things, an eight-digit calculating machine which could perform addition and subtraction. Multiplication and division were performed by repeated addition or subtraction, respectively. He completed the first operating model at age 19, and built 50 more during the next 10 years.

The Pascal language was originally specified in 1968 by Niklaus Wirth at the Institut fur Informatik, Zurich. This makes it a relative newcomer to the world of programming languages. The first Pascal compiler became operational in 1970 and was published in 1971.

The following table shows just how new Pascal really is. Remember that most compilers are not introduced until three to five years after their initial specification. For example, APL was initially specified in 1962.

Language	Introduction Date
Fortran	1957
Algol	1960
Lisp	1961
Snobol	1962
Basic	1965
PL/1	1965
APL	1967
Pascal	1971

After two years of experience, the language was revised and re-released in 1973. This version of the language is now generally referred to as standard Pascal. The important thing to note here is that Pascal was the first major new language to be developed after the concept of structured programming was introduced.

STRUCTURED PROGRAMMING AND PASCAL:

There exists no exact definition of structured programming, although it has been termed "A collection of all good and wonderful programming practices"! One fact becomes obvious in discussing it with groups of programmers: Some people love it, and some people hate it. However, those who hate structured programming are now finding themselves more often in the minority.

Some features to be found in a structured program are that it is generally more readable and more easily shown to be correct. The design of a structured program usually involves stepwise refinement, or top-down programming. Languages designed with structured programming in mind will usually include a large group of program-flow control structures, which are entered at only one point and from which there is only one exit. Another notable point about such languages is that they often require explicit definition of all variables and data structures in the code. What does all of this mean? How does it relate to Pascal?

READABILITY:

One of the outstanding features of Pascal is that well-written Pascal code is very readable; more so than most other programming languages. Probably the greatest single factor which makes this language so easy to follow, is the construction of data names. In Pascal there is no limit to the acceptable length of names. Generally, the compiler only uses the first eight characters of a name to distinguish it from all others, with the remainder of the name simply being ignored. This lack of constraints usually leads to very meaningful names in Pascal. Note that I have specifically avoided writing "variable names". Pascal permits not only variables to be named, but also constants, files, records, complex data structures, procedures and functions; all with the same naming conventions in effect. Compare this to any other languages such as Basic or Fortran!

Pascal's readability is also enhanced by the wording of its statements. When meaningful names are used, almost always the coded statements will make sense as English phrases. This would almost seem to take the place of program comments, but even so, Pascal provides one of the most flexible commenting schemes possible. Comments may appear anywhere in a Pascal program except in the middle of words!

STEPWISE REFINEMENT:

In writing a Pascal program it becomes very easy to use top-down programming style. This is mainly due to the flexibility and ease of writing procedures and functions. It is not unusual to see incredibly complex Pascal programs, several hundred lines long, in which the main program accounts for less than one hundred lines. Such a main program will usually consist of the overall program flow-logic with dozens of calls to well-named procedures and functions.

Procedures and functions correspond roughly to subroutines and functions in Fortran, but are actually part of the Pascal program. This means that procedures and functions inherit all variables defined in the main program, similar to subroutines in Basic, but they can also include declarations of variables and constants which are only valid themselves.

It should also be noted that procedures and functions are fully recursive in Pascal, that is they may in turn call themselves.

Simply using the name of a procedure or function will invoke it; thus it becomes very easy to write code with procedure names and worry about all of the messy details at a later date. This is, of course, the basis of top-down programming.

EXPLICIT DEFINITIONS:

Another level of stepwise refinement is careful pre-planning of a program. Usually, Pascal programs are most easily planned-out by using a form of loose, English-like pidgin Algol.

One thing should be noted here: Pascal is probably best classified as a descendant of Algol. People who know Algol seldom have any difficulty in learning Pascal. In fact, Algol-60 is generally considered to be a subset of Pascal.

Careful pre-planning is encouraged by the fact that Pascal has very rigid rules requiring virtually all data structures to be defined at the start of a program. Unlike many languages, you can't just throw in an extra variable, in the code, when you discover that you need it. Because Pascal also requires such things to be defined, careless pre-planning often becomes quite self-evident just by looking at the declarations. This feature is something which Basic programmers typically have a hard time getting used to, but it often makes assembly language hackers feel right at home.

Probably the greatest single new idea to come out of Pascal is the user-defineable data type. This construct, which appears in the declarations, permits the programmer to specify new types of data beyond the standard Real, Integer, Character and Boolean types. Data types of arbitrary complexity may be constructed; in fact adding complex numbers to a Pascal program is generally considered to be a trivial case!

Users may define data types as outrageously complex as say, a five dimensional array of records of arrays, scalars, records with variant parts, pointers and complex numbers. The programming power added by this concept is almost difficult to imagine; it provides us with the ability to create structured data as well as structured processes.

SINGLE-ENTRY/SINGLE-EXIT CONTROL STRUCTURES:

One of the requisites for being able to show that a program will work correctly is that it must be possible to trace out all of the possible execution paths, through the program, for given sets of inputs. Usually, this is done by first breaking the program down into small units, showing that each unit works correctly, and then showing that combinations of units work correctly and so on.

This all sounds very simple, except for one item — the GOTO statement throws a monkey wrench into the whole thing. The problem is that it doesn't take too many GOTOs combined with conditional branches, before an almost infinite number of possible execution paths appear in a program. How can you prove that a block of code will perform correctly, when you can't even be sure where it will be entered from, or where control will exit to, once it has completed?

As an example, consider a section of a Basic program, possibly a scoring routine for a game, which is invoked by GOTOs from 20 different locations. In addition, these GOTO statements jump into the scoring routine code at six different points, depending on flags set by previous passes through the routine, and upon other outside events. Depending on the data present and the entry point, the routine may branch to several places in itself, loop in two places, or fall straight through. Also, when it completes, depending on outside conditions and also upon previous passes through it, the routine may branch to any one of eight other program sections. Stop and think about how much effort it would take to trace all possible paths through such a mess! This code might be clever and efficient, but is it worth all of the headaches which it will cause in the long run?

Not only is such convoluted logic difficult to follow and understand, but it is also a major chore to get all of the bugs out of it; and you can never be sure that all of them ARE out. As if that isn't enough, just try to make a major change to such a piece of code — it would probably be easier to discard the whole thing, rather than try to patch it.

Now that we've raked the GOTO statement over the coals, what is there which will take its place? The answer is: single-entry / single-exit control structures. Flow of control, in a program, always enters the top of such a structure, and will only exit out through the bottom. This means that, if the program unit inside of the structure is correct, we can trace an effective straight line through the whole thing. A familiar example of a single-entry / single-exit structure is the FOR-NEXT loop in Basic, but without any GOTOs which enter or leave the middle: Flow will enter at the top, looping will occur, but eventually flow will continue through the bottom of the FOR-NEXT.

As it turns out, there are only three structures required to replace the GOTO statement. They are:

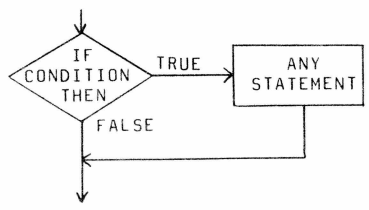
—The WHILE statement,
—The IF-THEN statement,
—And Compound statements.

In Pascal, anyplace a statement can go, may be placed a Compound statement. Compound statements consist of the word BEGIN, followed by any group of statements (which may include more Compound statements), followed by the word END.

Pascal also includes WHILE and IF-THEN statements, plus several other singe-entry / single-exit structures which add to the convenience of GOTO-less programming.

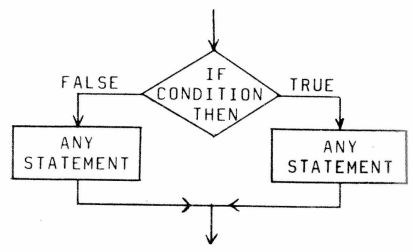
The following is a list of all of the structured statements in Pascal, along with flowchart segments to indicate how they function:

This is, of course, the well-known IF-THEN statement:

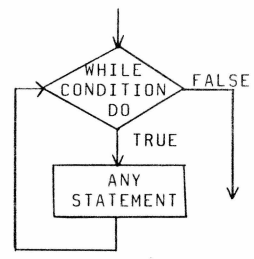


Note that the assignment statement is very free-form: Spaces may be inserted as needed, the assignment may continue onto more than one line, etc. The only restriction is that words can not be broken in the middle.

A convenient form of the IF-THEN statement is the IF-THEN-ELSE:



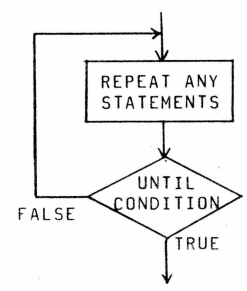
The WHILE statement has the form:



This next one is the REPEAT-UNTIL statement. There is an important difference between this and the WHILE statement which should be noticed:

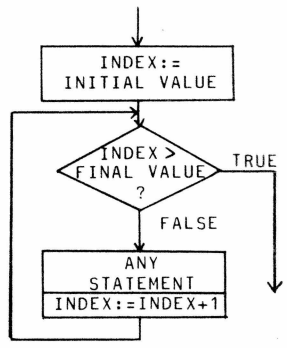
> If the condition is false, when a WHILE statement is entered, no action takes place — control skips around the ANY STATEMENT part.

In a REPEAT-UNTIL however, the ANY STATEMENT part always gets executed at least once, regardless of the conditional part.

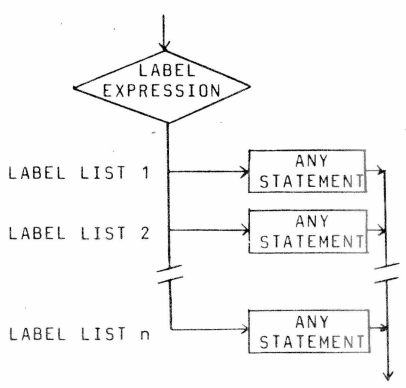


The FOR-UPTO statement is very similar to the FOR-NEXT statement in Basic, except that it is restricted to an increment of one (1). This is intended to add to the reliability of the construct, since most digital computers can not exactly represent fractional numbers. If other increments were permitted, it might be possible for the increment to not exactly match the terminator when it reached the desired value, and so perhaps the loop would continue for an extra pass. This is a very frustrating problem, because it is usually highly machine dependent, and will typically only show up in a very few specific instances. All of this is eliminated by Pascal's restrictions of the increment value to one (1). One positive side effect which results from this is that the speed of the statement is often greatly increased, since many machines have single instructions for incrementing and testing memory locations, or registers.

The FOR-DOWNTO statement is identical to this, except that the index is decremented by one, each time through the loop:



This last one is the CASE statement, which is somewhat like the ON-GOTO statement in Basic:



All of this should not be taken to imply that Pascal is a GOTO-less language (People's Pascal is GOTO-less—Ed); it does have labels and GOTO's. The important point is that the experienced Pascal programmer will almost never use them, since they are never needed and only rarely of any value.

PART TWO: SUMMARY OF PASCAL STATEMENTS, WITH EXAMPLES

CHARACTER SET:

The standard Pascal character set includes: Letters A-Z (and depending on the implementation, a-z), numbers 0-9, special characters + - * / = ⟨ ⟩ ( ) [ ] . , ; : ' ↑ and the space or blank character.

NAMES:

Names in Pascal consist of letters and/or digits, and may be any number of characters in length. The first character must be a letter, and the first eight characters must be different than the first eight characters of any other name.
Examples:

ENDOFDATA TYPES AVERAGE TOTAL SCORES PAYRATE CARDCOUNT

NUMBERS:

Numbers in Pascal are either real or integer. They may be signed or unsigned.

Integers are a string of digits. Examples:

+7 43 365 -18 8388607 4092 0

Reals have three forms:

digits . digits
digits . digits E-scale factor
digits E-scale factor

The E notation indicates multiplication by 10 raised to the scale factor power.
Examples:

3.1415 6.02E23 9.11E-31 -1E9

Note that the scale factor is always an integer.

COMMENTS:

The compiler will ignore, as being comments, anything typed between the symbols "*". Example: *comment*. On systems which have them, curly brackets { } are used instead.

OPERATIONS:

Integer operations:

* Multiplication
DIV Division (integer part only, remainder discarded)
+ Addition
- Subtraction
MOD Modulo (A MOD B = A-((A DIV B)*B))

Real operators:

* Multiplication
/ Division
+ Addition
- Subtraction

Boolean operations:

AND Logical AND
OR Logical OR
NOT Logical NOT

Relational operations (give Boolean results:

⟨ Less than
⟩ Greater than
= Equal to
⟨= Less-than or equal-to
⟩= Greater-than or equal-to
⟨⟩ Not-equal to
IN Used with tata type SET, to determine membership of an element

Examples:

A * B	A times B
X DIV Y	X divided by Y
TOP ⟨=BOTTOM	Numerical comparison
ABOVE AND BEYOND	True if both (ABOVE and BEYOND) are true Boolean variables

FUNCTIONS:

ABS	-Absolute value
SQR	-Square
TRUNC	-Truncate to integer part
ROUND	-Rounded-up integer form
SUCC	-Next highest (integer or character)
PRED	-Next lowest (integer or character)
SIN	-Trigonometric sine
COS	-Trigonometric cosine
ARCTAN	-Trigonometric arctangent
LN	-Natural (Base e) logarithm
EXP	-e raised to the power
SQRT	-Square root
ORD	-Numeric value associated with the character
CHAR	-Character associated with the numeric value
ODD	-True if the integer argument is odd
EOLN	-True when end-of-line is reached
EOF	-True when end-of-file is reached

RESULT TYPE FOR ARGUMENT OF TYPE:

Name	Integer	Real	Character
ABS	Integer	Real	
SQR	Integer	Real	
TRUNC		Integer	
ROUND		Integer	
SUCC	Integer		Character
PRED	Integer		Character
SIN	Real	Real	
COS	Real	Real	
ARCTAN	Real	Real	
LN	Real	Real	
EXP	Real	Real	
SQRT	Real	Real	
ORD			Integer
CHR	Character		
ODD	Boolean		
EOLN	Argument is always a file name, result is always boolean.		
EOF	Argument is always a file name, result is always boolean.		

STATEMENTS:

PROGRAM HANDLING:

PROGRAM programname (filename, filename,...);
Example: PROGRAM TESTSCORES (INPUT, OUTPUT);

CONSTANT DEFINITION:

CONST constname = value; constname = value;...
Example:

```
CONST ENDOFDATA =-1.0; PI=3.141592;
      MAXSCORE = 100; MINSCORE
      = 0;
```

Note that the constant definitions can continue onto more than one card, but the CONST is only typed once.

There are some predefined values which do not need to be declared as constants in Pascal programs. these are:

TRUE	Boolean true value
FALSE	Boolean false value
MAXINT	Largest integer the computer can work with
NIL	Null pointer

VARIABLE DEFINITION:

VAR varname, varname,...: type;
varname, varname,...: type;...
Example:

```
VAR SCORE,MAX,MIN,TOTAL: INTE-
    GER;     RADIUS, DIAMETER, CIR-
    CUMFERENCE: REAL;
    FOUND, DONE, FLAG, OK: BOO-
    LEAN;
```

Note that the declarations may continue on several times, but only one VAR is required.

PROCEDURE DEFINITION:

meters; VAR variable parameters);
body of procedure
Example:

```
PROCEDURE INCREMENTBY (INCREM-
ENT:REAL;
  VAR VARIABLETOBEINCREMENTED:
  REAL);
BEGIN
  VARIABLETOBEINCREMENTED :=
    VARIABLETOBEINCREMENTED +
      INCREMENT
END;
```

FUNCTION DEFINITION:

FUNCTION functname (value parameters): result-type;
body of function
Example:

```
FUNCTION RADIUS (CIRCUMFERENCE:
  REAL):
  REAL;
CONST TWOPI = 6.2831;
BEGIN
  RADIUS:=CIRCUMFERENCE / TWOPI
END;
```

ASSIGNMENT STATEMENTS:

varname := expression
Examples:

```
WEEKSPAY := PAYRATE * HOURSWORK-
  ED;
VOLTS:=AMPS*OHMS;
CONEVOLUME:= (PI*SQR(RADIUS)*
  HEIGHT)
  /3.0;
ARRAYLOCATION :=ARRAYLOCATION
  + 1;
```

Note that the assignment statement is very free-form: Spaces may be inserted as needed, the assignment may continue onto more than one line, etc. The only restriction is that words can not be broken in the middle.

THE COMPOND STATEMENT

In Pascal, any place where a statement can be used, a compound statement may also be used. A compound statement is formed by the word BEGIN, a group of any statements, followed by the word END.
Examples:

```
BEGIN
  SCORESUM:=SCORESUM+SCORE;
  SCORECOUNT:=SCORECOUNT+1
END
BEGIN
  X:=(Y+Z)/100;
  BEGIN
    T:=(Q/75)+15;
    F:=N-18
  END
END
```

PLACEMENT OF SEMICOLONS:

The simplest rule for the placement of semicolons, in a Pascal program, is:

Place a semicolon between any two Pascal statements.

Note: BEGIN and END are not Pascal statements. They are simply delimiters. A compound statement is a statement, and must be separated from other statements. Also note one exception to the rule:

The ELSE in the IF-THEN-ELSE takes the place of a semicolon in separating the two statements.

CONDITIONAL STATEMENTS THE IF-THEN STATEMENT:

IF expression THEN statement
Example:

```
IF MAXSCORE ⟨ SCORE THEN
  MAXSCORE:=SCORE
```

THE IF-THEN-ELSE STATEMENT:

If expression THEN statement ELSE statement
Example:

```
IF TIME ⟨ 0 THEN TIME:=0 ELSE
  TIME:=1
```

THE CASE STATEMENT:

```
CASE expression OF
  case-label-list:statement;
  case-label-list:statement;
  etc.
END
```

Example: (* Determine command group from a command number *):

```
CASE COMMANDNUMBER OF
  0,1,3  : GROUP:=1;
    2,4  : GROUP:=2;
  5,9,11 : GROUP:=3;
  6,7,8  : GROUP:=4;
    10   : GROUP:=5
END
```

REPETITIVE STATEMENTS: THE WHILE-DO STATEMENT:

WHILE expression DO statement
Example:

```
WHILE NOT EOF (INPUT) DO
BEGIN
  READ(SCORE);
  SCORESUM:=SCORESUM+SCORE;
  SCORECOUNT:=SCORECOUNT + 1
END
```

THE REPEAT-UNTIL STATEMENT:

REPEAT group-of-statements UNTIL expression
Example:

```
REPEAT
  X:=X-1;
  Y:=Y+1
UNTIL (X ⟨ 0) OR (Y ⟩ 0)
```

THE FOR STATEMENT (two forms):

FOR control-variable := initial-value TO final-value DO statement
FOR control-variable := initial-value DOWNTO final-value DO statement
Examples:

```
FOR INCEX := 1 TO ARRAYTOP DO
  ARRAY[INDEX] := 0
FOR INDEX := 100 DOWNTO
  ARRAYBOTTOM DO IF ARRAY[INDEX]
  ⟨ 0 THEN ARRAY[INDEX]:=0
```

TRANSFER OF CONTROL STATEMENTS:

The conditional and repetitive statements previously described are sufficient control structures to perform any required computation.

Remember that although labels and GOTOs are provided in Pascal (not People's Pascal), they are unnecessary and will often only create confusion in program logic.

Therefore it is recommended that they be avoided except in those rare extreme cases where they actually have some value.

LABEL DEFINITION (no labels in people's pascal):

The label definition is placed after the CONST declarations in the program.

LABEL integer, integer, ...;
Example:

```
LABEL 10, 20, 25, 100, 9999;
```

GOTO STATEMENT (not available in people's pascal):

GOTO label
Example:

```
GOTO 9999
```

INPUT AND OUTPUT IN PASCAL:

Pascal I/O statements are not really statements, but are actually calls to predefined procedures. Nonetheless, they are often referred to as statements.

INPUT PROCEDURES:

```
READ(variable-list)
READLN(variable-list)
READ(filename, variable-list)
READLN(filename, variable-list)
```

Examples:

```
READ (X, Y, Z, MAXVAL)
READLN (HIGHSCORE, LOWSCORE,
  AVGSCORE)
READ (WEATHERFILE± TEMP,
  HUMIDITY, PRESSURE)
READ (CUSTOMERFILE, NAME,
  NUMBER, BALANCE)
```

The filename must have been declared in the program heading.

The difference between READ and READLN is that successive READ statements will continue to input successive values from the same record, only going to a new record when all values on the current one have been exhausted.

A READLN, on the other hand, will skip any additional values on the current record, and go to the next record to begin reading values.
Example (two records):

```
0.0 1.0 2.0
3.0 4.0 5.0
```

```
READ(A,B);
READ(C,D)
```

The result of this would be A=0.0, B=1.0, C=2.0, D=3.0.

```
READLN(A,B);
READLN(C,D)
```

Would result in A=0.0, B=1.0, C=3.0, D=4.0.

OUTPUT PROCEDURES:

```
WRITE (expression-list)
WRITELN(expression-list)
WRITE(filename, expression-list)
WRITELN(filename, expression-list)
```

Examples:

```
WRITE(A,B,C)
WRITELN(X*Y/Z,MAX,SQRT(Q),'*****')
WRITE(NEWFILE,NAME,ADDRESS,
  PHONE,AMT+1.0)
WRITELN(PLOTFILE,XCOORD,YCOORD,
  PENPOS,MARK)
```

Successive WRITE statements cause the values to be written, all as one record. Each time a WRITELN is executed, however, a new record is output.
Examples:

```
WRITE ('A','B');
WRITE('C','D')
```

Would output ABCD.

```
WRITELN('A','B');
WRITELN('C','D')
```

Would output AB
CD.

Formatting numeric output is very easy in Pascal. Each expression in a WRITE or WRITELN can actually have one of the following three forms:

expression
expression:width-expression
expression:width-expression:fraction-width-expression

The expression gives the value which is to be output. The width-expression gives the minimum number of character positions to be included in the output. If the expression value doesn't require all of the positions, the extras will be filled with blanks. If the number is too big to fit in the area, the area size is expanded to accommodate the number.

The fraction-width-expression specifies how many digits will be printed to the right of the decimal point for a real number.
Examples:

```
A=100, B=1.5, C=137875.3217,
  D=128.34152
WRITE(A:5,B:5,C:5,D;9:3)  would output
  100 1.5137875.3217 128.341
WRITE (A:3,B:5:2,C:9:1) would output
  100 1.50 137875.3
```

CARRIAGE CONTROL:

Although this is machine and implementation dependent, most Pascal systems will destroy the first character of each record output to a printing device. Thus, an extra character must be provided at the start of each output line, usually a space.

In reality, this character acts as a carriage control command, which is either directly implemented in the hardware of the printer, or which is simulated by the monitor or operating system, in software.

The following are the standard carriage control command characters used in Pascal:

character	action
space	normal, single spaceing
0 (zero)	double space, skip 1 line
1	page eject

Depending upon how the carriage control is implemented, using other characters may have different effects, which may or may not be desireable.

DATA TYPES:

All data type definitions are placed between the CONST and VAR declarations at the start of the program.

SCALAR TYPES:

TYPE typename = (identifier, identifier,...);
Example:

```
TYPE MONTH = (JAN,FEB,MAR,APR,
  MAY,JUN,JUL,AUG,SEP,OCT,NOV,
  DEC);
```

SUBRANGE TYPES:

```
TYPE typename = constant... constant;
VAR varname-list : constant... con-
  stant;
```

Examples:

CONTINUED ON PAGE (12)

PEOPLE'S PASCAL I
TRS-80 PEOPLE'S PASCAL SYSTEM DOCUMENTATION

pipe dream software, berwick australia
COPYRIGHT APRIL 1979
ALL RIGHTS RESERVED

1. INTRODUCTION:

The TRS-80 People's Pascal system is a program development system for Tiny Pascal, a subset of the Pascal programming language introduced by Niklaus Wirth of the Engineering University at Zurich.

Tiny Pascal was defined in Byte magazine—"A 'Tiny' Pascal Compiler", by Kin-Man Chung and Herbert Yuen, in three parts, September, October and November

The Pascal language is defined in "Pascal: User Manual and Report" by Kathleen Jensen and Niklaus Wirth (Springer-Verlag 1974). A good introductory book on Pascal is "Microcomputer Problem Solving Using Pascal" by Kenneth L. Bowles (Springer-Verlag 1977).

The following programs are supplied with People's Pascal:

1.1 TEXT EDITOR:

The editor is line-oriented. Intra-line editing is not provided. Text files may be manipulated in the following ways: create, edit, list, print, merge, copy, read from and write to cassette.

The editor uses a 3,000-character (3K) text buffer and 240-character blocked records on cassette files to achieve its results. Source files of indefinite length may be manipulated, but short files are recommended for convenience and modularity. Editor commands are: insert, delete, replace, list, print, read and merge, write, re-number, compile, and free. Refer to the editor operating instructions and program documentation for details.

1.2 COMPILER:

Included in the same program as the editor, to save time-consuming swapping between programs during program development, the People's Pascal compiler translates the source program in the text buffer and/or included from one or more source program text files into a Pcode object program on cassette.

The compiler accepts the following Pascal subset: AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNTO, ELSE, END, FOR, FUNC, IF, INTEGER, MOD, NOT, OF, OR, PROC, READ, REPEAT, SHL, SHR, THEN, TO, UNTIL, VAR, WRITE.

Note that character arrays, records, files, reals, programmer-defined types pointers and GOTO are omitted from People's Pascal.

In addition, extensions are provided to read from and write to absolute memory addresses, and to allow the calling of assembly language subroutines at absolute memory addresses, together with limited numeric I/O formatting and the definition of hex constants.

A "$INCL" include-source-file feature is provided to allow modular program development.

The compiler is a one-pass compiler using recursive descent. Refer to language definition, compiler operation and compil-program documentation for details.

1.3 THE INTERPRETER:

The interpreter reads a P-code program object file produced by the compiler into memory and interprets the program, performing the actions required of the imaginary P-machine, which has P-code as its instruction set.

The interpreter has the following debugging routines:

SET BREAKPOINT(s), CLEAR ALL BREAKPOINTS, EXAMINE PROGRAM, GO, EXAMINE STACK CONTENT, EXAMINE NEXT PROGRAM LOCATION, QUIT, RUN, SINGLE STEP, TRACE, EXAMINE PREVIOUS PROGRAM LOCATION, DISPLAY P-MACHINE REGISTERS, DISPLAY BREAKPOINTS.

The current version of the interpreter written in Basic is slow, with a double level of interpretation.

P-code object programs of up to 8.6K bytes may be interpreted. Refer to interpreter operating instructions and interpreter program documentation for details.

1.4 TRANSLATOR:

The translator reads in a P-code object file produced by the compiler and translates P-code instructions into fast Z-80 code (machine language instructions) using the Z-80 stack pointer for the People's Pascal stack.

Translated programs run about five times faster than Level-II Basic. Graphics instructions run about eight times faster.

The translator also has the option to optimise for minimum memory useage, reducing program size to half at the cost of some speed reduction. Refer to translator operating instructions and program documentation for details.

1.5 PEOPLE'S PASCAL SOURCE LIBRARY:

The People's Pascal library uses the "$INCL" compiler option to allow an extendable set of standard routines to be incorporated into user programs. The following routines are provided:

SET(ON/OFF,X,Y):; set/reset graphics (=SET(X,Y))
RND(SEED):; pseudo random number generation
AT(cursorposition): cursor control (=PRINTE)

User-written routines may be added to the People's Pascal library. Other routines also may be provided, such as UCSD "turtle" graphics procedures MOVE(distance), TURN(angle), MOVETO(X,Y), PENCOLOR(white/black/none).

1.6 RUN-TIME SYSTEM:

This program is writeen in Z-80 assembly language and provides subroutines called by translated People's Pascal programs for multiply, divide, set, I/O, etc. It occupies about 1 K byte. Both source and object programs are supplied. Refer to run-time-system program documentation for details.

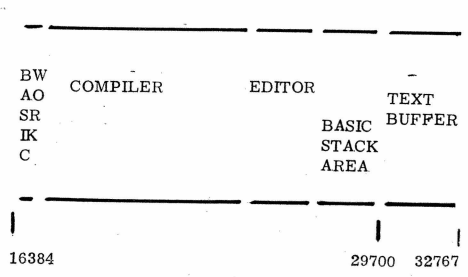
2. PEOPLE'S PASCAL LANGUAGE:

It is not the aim of this document to teach the Pascal programming language.

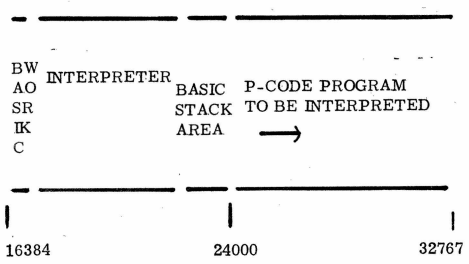
TRS-80 People's Pascal includes the full set of program structuring statements, such as IF, THEN, ELSE, BEGIN, END, WHILE, REPEAT, FOR, CASE, PROC, FUNC, but includes integer and array of integer data types only (16 bit). Refer to the language reference documentation for details.

3. MEMORY MAPS:

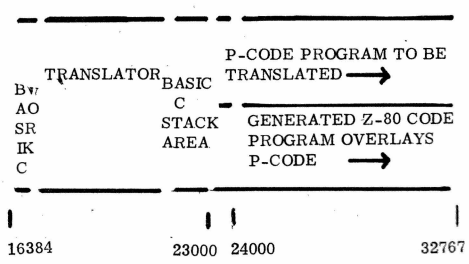
3.1 EDITOR/COMPILER (TPEC):



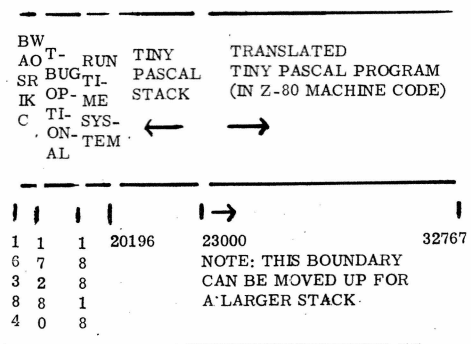
3.2 INTERPRETER (PPINT):



3.3 TRANSLATER (PPTRANS):



3.4 PEOPLE'S PASCAL TRANSLATED PROGRAM AT RUN TIME:



4. PEOPLE'S PASCAL STACK:

The Pascal pseudo machine is stack-oriented. For a complete understanding of the system, it is first necessary to understand the P-machine and its stack. The P-machine has two registers, (T) and (B). (T) is the stack pointer, which always points to the top element on the stack. (B) is the base register, which points to (i.e., holds the address of) a stack location which is the stack base for the "block" (i.e. program, procedure, or function) that is currently executing. The base is used as a reference point for variable addresses.

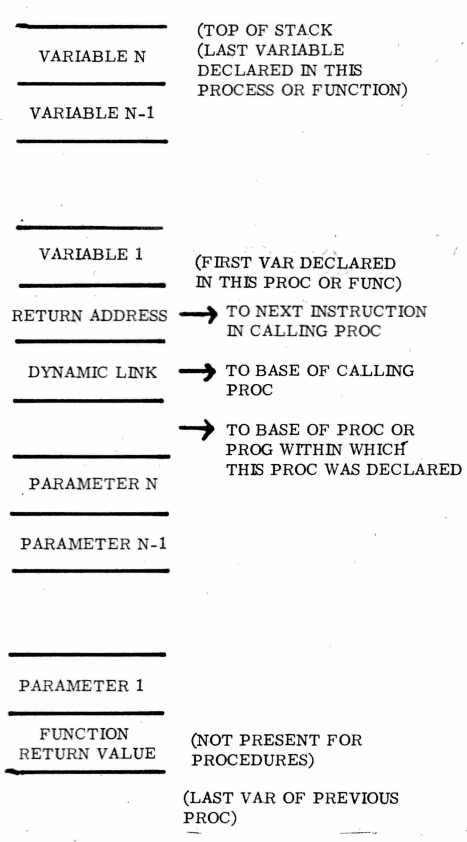
When a block is entered (i.e., when a procedure is called) space for the variables it declares is allocated on the stack in a new "stack frame", which uses the space just above the last-used stack locations.

All variables declared within a block (in a stack frame) are identified within the P-code by an offset from the base of a stack frame, rather than by an absolute address as in some other systems. Variables which were declared in the block which is currently executing can be obtained by adding their offset to the contents of the base register. This forms the absolute address of the variable. On the other hand, variables which were declared in some other block must have their offsets added to the base of the stack frame of that outer block in order to be referenced by their absolute address.

The base of the outer block can be obtained because at the base of each stack frame is a word which contains the absolute address of the previously-entered (outer) stack frame. Thus stack frame bases are linked together in a linked list which descends down the stack to the base of the stack frame of the outermost block (mainline) of the program.

In fact there are two lists: A "static" list, which links stack frames for obtaining variable addresses. This list reflects the lexical structure of the program, i.e., the static nesting of procedure declarations. The second list links stack frames in execution sequence, reflecting the sequence of active procedure calls, at program run time. The second (dynamic) list is used to regain the base of the calling procedure on exit from the called procedure.

5.1 DIAGRAM OF PEOPLE'S PASCAL STACK FRAME:



5. THE P-CODES:

P-codes are the machine language of the imaginary P-machine.

P-codes occupy four bytes each. The first byte is the operation code (op-code). There are nine basic P-code instructions, each with a different op-code.

The second byte of the P-code instruction contains either zero or a lexical level offset, or a condition code for the conditional jump instruction.

The last two bytes taken as a 16-bit integer form an operand which is a literal value, or a variable offset from a base in the stack, or a P-code instruction location, or an operation number, or a special routine number, depending on the op-code.

5.1 P-CODE DETAILS:

p-code	hex op-code	description
LIT Ø,N	00	load literal value onto stack
OPR Ø,N	Ø1	arithmetic or logical operation on top of stack
LOD L,N	Ø2	load value of variable at level offset L, base offset N in stack onto top of stack
LODX L,N	12	load indexed (array) variable as above
STO L,N	Ø3	store value on top of stack into variable location at level offset L, base offset N in stack
STOX L,N	13	store indexed variable as above
CRL L,N	Ø4	call PROC or FUNC at at P-code location N declared at level offset L
INT Ø,N	Ø5	increment stack pointer (T) by N (may be negative)
JMP Ø,N	Ø6	jump to P-code location N
JPC C,N	Ø7	jump if C=value on top of stack to P-code location N (C can = 0 or 1)
CSP Ø,N	Ø8	call standard procedure number N

LANGUAGE DESCRIPTION

1. INTRODUCTION:

People's Pascal is a Pascal subset containing all the program structuring constructs except GOTO, but without many of the data structuring facilities.

2. PASCAL FEATURES NOT PRESENT IN PEOPLE'S PASCAL:

2.1 DATA TYPES:

Integer and array of integer (16-bit) data types are the only data types provided (range -32767 to +32767).

Integer arrays may be of/single-dimension only. No Boolean, real or CHAR data types. No records, files, or pointer types. No programmer-defined types. No sets.

However, note that integers and integer arrays can be used to store character data, and that single-character literals are accepted, and that a write character string facility is provided. e.g., write ('HELLO THERE!'); note also that logical operations are allowed on integers, e.g. IF A THEN..; WHILE 1 DO..; (loops forever).

2.2 PROGRAM STRUCTURE:

No GOTO. No statement labels. Structured programming must be used exclusively—this can lead to more easily understood programs. For any function that uses GOTOs, there is another function which performs the same operation without, gotos using only sequential, conditional (IF, CASE) and iterative (WHILE, REPEAT, FOR) structures.

2.3 PARAMETERS PASSED BY VALUE:

People's Pascal only provides for procedure and function parameters passed by value (i.e. there can be no output parameters from a procedure).

Note that a function can return a value and that procedures can alter global variables (variables declared outside themselves) as alternatives. The second alternative is best avoided where possible, to minimize the dependence of the procedure on its environment, and make the program less complex.

2.4 WRITELN:

WRITELN is not provided. Use WRITE instead.

3. ADDITIONAL FEATURES OF PEOPLE'S PASCAL:

3.1 ACCESS TO MEMORY:

A special "built in" array called "MEM" is provided. This array does not need to be declared.

The MEM array is mapped onto absolute memory.

The contents of MEM(X) consists of the byte of memory at absolute address X. E.g. A:=MEM(X); or MEM(3Ø):=Ø;

This facility is equivalent to Basic PEEK, POKE.

3.2 ACCESS TO ROUTINES IN ASSEMBLY LANGUAGE:

A "CALL" facility is provided to allow the invocation of assembly language (Z-80 code) routines. E.g., CALL(32650) or CALL(RTN).

All necessary registers are saved by the People's Pascal run-time system before the user's routine is entered, and restored on return.

If the routine is called from within a procedure, then the procedure parameters can be accessed on the stack by the called routine.

3.3 FORMAT CONTROL ON READ & WRITE:

Without format control, when an integer is written, the result will be that the character whose ASCII value is that of the integer will be output. I.e., WRITE(65) will cause the character "A" to be written to the display. WRITE(13) will cause a carriage return. WRITE(23) will cause

wide characters. WRITE(28) will home the cursor. WRITE(31) will clear the screen from the current cursor position onwards. WRITE(28,31) will clear the whole screen. Refer to Level-II Basic Reference Manual, pages c/1 and c/2. Note that all special control character values can be declared as constants using "CONST". For purposes of standardization, the following names are recommended:

BS = 8 (backspace and erase)
LF =10 (linefeed/carriage return)
FF =12 (top of form — form feed)
CR =13 (linefeed/carriage return)
CON =14 (cursor on)
COFF = 15 (cursor off)
WIDE = 23 (convert to 32 chars/line — wide characters)
CBACK = 24 (backspace cursor)
CFWD = 25 (advance cursor)
CDOWN = 26 (move cursor down)
CUP = 27 (move cursor up)
HOME = 28 (home cursor — move to top LH corner of screen)
BLINE = 29 (move cursor to beginning of line)
ERASE = 30 (erase to end of line)
CLEAR = 31 (clear to end of screen)

Similarly, a READ will cause the integer being read to be assigned the ASCII value of the input character.

Read and write formatting are provided to override this facility.

A WRITE (X) will cause a number to appear on the screen equivalent to the value of X. READ (A) will cause the input digits to be converted to a 16-bit integer and stored in A. The "#" is the numeric format indication character. I.e., WRITE(65#) will cause the characters "6" and "5" to be written to the screen at the current cursor location.

3.4 HEXADECIMAL CONSTANTS:

Hexadecimal constants are provided for, and are specified by a leading percent sign. Hex constants must contain four hex digits, e.g., %ØØ3A, %FFFF.

3.5 ELSE ON CASE STATEMENTS:

An "else branch" is provided on the CASE statement, which will be taken if the CASE variable does not have a value which matches any of the other specified values. Be especially careful not to use spurious semicolons before this statement, or before the end of the case statement. Look at the syntax diagrams carefully.

```
E.G.: CASE X OF
        1: WRITE('X EQUALS ONE',CR);
        2: Write('X EQUALS TWO',CR)
        (*NO ';' HERE*)
      ELSE WRITE ('X OUT OF
        RANGE') (*OR HERE*)
      END (*CASE*) ;
```

If you had an extra case element before the 'ELSE' (e.g., 3:... in the above example), remember to put a semicolon on the end of the previous line (e.g., 2:... in the above example).

4. TIPS ON PROGRAMMING IN PEOPLE'S PASCAL:

4.1 MODULES:

Break the program up into functional components.

Write these components as procedures or functions. If the procedure or function is of general use, then it can be placed in your own library, or into the People's Pascal library.

Try to connect the procedures to the rest of the program by parameters and function return values.

Declare variables and constants only required within one procedure inside that procedure rather than outside it.

Refer to "Structured Design" (Larry Constantine and Ed Yourdon, Yourdon Inc., 1133 Avenue of the Americas, New York NY 10036) for a thorough discuss on of functional module design.

Avoid declaring procedures within other procedures.

People's Pascal optimization in the translator has been designed so that "well structured" (structured in the above manner) programs will execute fastest under "fast" optimization and occupy least memory when optimized with the "small" option. Thus there should be no conflict between good programming practice and efficiency.

With a little ingunuity, procedures and functions can be compiled and tested separately, which can speed up the development process.

To compile a procedure or program mainline which uses other lower-level procedures, "dummy" procedure declarations can be inserted before the main block. Dummy declarations consist of only the procedure name and formal parameter declaration followed by a "begin end;" (e.g. FUNC RND(SEED); BEGIN END;)

This will allow the main block to compile without having to wiat for all its procedures to be compiled first.

Of course, to test any procedure, function or mainline, it will usually be necessary to have compiled in all of the procedures that it uses.

The above process of "dummy" declarations currently only applies to the syntax (gramatical) error detection process. However, if lower-level modules are compiled and tested first, then as development proceeds, these lower level modules are always available for testing the next level.

The "$INCL" feature should be used in conjunction with the separate procedure/function concept.

A complete procedure or set of procedures can be put into its own file, and "$INCL"uded into the program.

Note that all procedures must be declared before they are referenced (used). This is a one-pass compiler.

4.2 INITIALIZATION OF VARIABLES:

People's Pascal variables are not cleared when they are allocated on the stack.

Thus their initial value is unpredictable (will be whatever happend to be in that stack location).

Therefore it is important to explicitly initialize variables (e.g., X:=Ø;).

4.3 ARITHMETIC AND STACK OVERFLOW:

No run-time checking for arithmetic occurs, so an estimate of stack-space requirement should be made and sufficient stack space allocated.

Arithmetic overflow may be checked for in the interpreter.

5. THE PEOPLE'S PASCAL LIBRARY:

The following routines are available:

SET(ONOFF,X,Y); sets the graphic point at location X,Y. If ONOFF is set on. If ONOFF=0 then point set off.

RND(seed); returns a pseudo-random number between Ø and 32767. The seed should be set to the returned value, for the next call.

AT(CURPOS); sets the cursor to position CURPOS on the screen.

6. PEOPLE'S PASCAL RESERVED WORDS:

AND -logical-and operator
ARRAY - array declaration
BEGIN - compound statement opening delimiter
CALL - invoke assembly-language routine
CASE - multiple-statement selection
CONST - constant declaration section keyword
DIV - integer divide arithmetic operator
DO -while statement component
DOWNTO - FOR statement component
ELSE - IF and CASE statement alternative branch
END - compound statement delimiter
FOR - iteritive (looping) statement component
FUNC - function declaration keyword
INTEGER - integer data type declaration keyword
MEM - memory array keyword
MOD - arithmetic operator giving division remainder
NOT - logical not operator
OF - CASE statement component
OR - logical-OR operator
PROC - procedure declaration keyword
READ - READ statement
REPEAT - iterative statement keyword
SHL - logical shift-bits-left operator
SHR - logical shift-bits-right operator
THEN - IF statement component
TO - FOR statement component
UNTIL - REPEAT statement component
VAR - variable declaration section keyword
WHILE - iterative statement keyword
WRITE - WRITE statement

7. PEOPLE'S PASCAL SPECIAL SYMBOLS:

NOTE 1. The square bracket characters used in Pascal for array index delimiters are not available on the TRS-80. The round bracket characters are used instead as in Level-II Basic, rather than the Pascal alternative "(." and ".)".

NOTE 2. The squiggly bracket characters used in Pascal for comment delimiters are not available on the TRS-80. The character combinations "(*" and "*)" are used instead.

\# read/write numeric format indicator (WRITE(A#))
$ compiler directive line indicator ($INCL FRED)
% hex constant indicator (%AØ4F)
' character string delimiter (e.g. 'A')
(arithmetic or logical expression delimiter
(array index opening delimiter (e.g. AR(3Ø):=1;)
(* comment opening delimiter
) arithmetic or logical expression delimiter
) array index closing delimiter
* multiplication operator
*) comment closing delimiter
: variable declaration component
:= assignment operator
= equal-to operator
- unary minus and binary subtraction operator
+ addition operator
; statement separator
< less-than operator
<= less-than-or-equal-to operator
, separator
> greater-than operator
>= greater-than-or-equal-to operator
<> not-equal-to operator
. end-of-program indicator

8. PEOPLE'S PASCAL OPERATORS:

\+ addition
\- subtraction and unary minus
* multiplication
DIV integer division
MOD remainder after integer division
SHL logical shift left (can be used multiplication of positive numbers by a power of two)
SHR logical shift right (can be used for fast division of positive numbers by a power of two)
NOT logical-NOT unary operator
OR logical OR
AND logical AND
= equal-to
< less than
<= less-than or equal
> greater than
>= greater than or equal
<> not equal to

OPERATING INSTRUCTIONS

COMPILER

1. INTRODUCTION:

The People's Pascal compiler accepts language statements from an edit-buffer and / or cassette files, and translates these into P-code, which is output to an object file on cassette.

The compiler also produces a screen display both of the source code lines and of the generated object code.

P-codes are machine-language instructions for a simplified stack-oriented virtual (or imaginary) machine. These P-codes can either be interpreted by a program which performs the actions expected of the virtual machine, or they can be translated into code for a different (real) machine, in this case the Z-80 microprocessor of the Tandy TRS-80 microcomputer. Both of these options are provided in the Pipe Dream People's Pascal implementation.

In addition, the compiler has the option to compile for syntax errors only (no P-code object file being produced), for increased speed and possibly less operator intervention.

The compiler also has the option to produce a listing on a lineprinter if one is attached to the system.

2. INVOKING THE COMPILER:

To compile a People's Pascal program, it is first necessary to "CLOAD" and run the "PPEC" editor/compiler program. Refer to People's Pascal editor operating instructions. A source program may be typed into the text buffer for compilation, and/or source code may be compiled from cassette using the "$INCL" include-file option. In addition, a source program may be read into the text buffer for compilation and/or editing.

After any required editing of the source program, the compiler can be started with the "C" command. The compiler starts compilation with the first line of source code in the text buffer. If it is required to compile from an existing source file on cassette, then it will be necessary to enter a line such as:

```
100$INCL ]FILENAME[
```

into the text buffer before compilation, where /FILENAME/ is the name of the People's Pascal source file which is to be compiled.

Note that currently, the "$INCL" compiler option is not nestable. It can only be used as part of a line of source code in the text buffer (i.e., as part of the program mainline).

Also, the line must appear exactly as specified, without any leading or embedded spaces in the "$INCL" statement, apart from the space before the filename.

Note also that the filename supplied is currently required as an operator aid only. No filename checking is performed in the current version.

3. OBJECT FILE OPTION (OBJ FILE?):

After initialization of the "C" command, the compiler will prompt with "OBJ FILE?". If the reply is null (just Enter) then no object file will be produced, and the compile will be for syntax error-check only. Any non-blank reply to this prompt will result in a P-code cassette object file being produced.

4. LINEPRINTER OPTION (LP?):

The compiler will then prompt with "LP?".

If the reply to this question is "Y", then the input source program lines and the compiled object code will be printed on the lineprinter, as well as being shown on the display.

If the reply to this question is null (just (enter)) or "N", the no-lineprinter output will be generated.

For systems without a lineprinter, it is suggested that this prompt be deleted from the PPEC compiler code, to avoid the annoyance of a prompt to which the answer is always the same.

5. COMPILER OPERATION:

After initialization, the compiler will proceed to compile the specified source program.

On encountering a syntax or (gramatical) error, the compiler will display a diognostic error message indicating the type of error.

The compiler will then return to the editor, to allow the error in the source code to be corrected, and possibly re-compiled.

6. MOUNTING OBJECT CASSETTES (OBJ CAS READY?):

If a P-code object file is being produced, then after it has compiled about 50 P-codes, the compiler will prompt for an output cassette for the object file to be written to.

A previously-erased cassette should be mounted and the recorder placed in record mode.

When the output cassette is ready, the Enter key may be pressed, and the compiler will write a block of object code to the cassette and proceed with the compilation.

If no input cassette is being used ($INCL) then no further operator intervention should be required until the compilation is complete. Assuming a successful compilation, the output cassette will contain a P-code object file version of the program, which may be used as input either to the People's Pascal interpreter to test the program, or to the translator to make a final system-loadable version of the program in Z-80 machine language.

If the compiler is accepting input from a cassette file ($INCL), and an object file is being generated, then it will be necessary to periodically swap cassettes and cassette recorder operating modes when the compiler prompts.

This process is made possible by the blocking of both source and object data on cassette. Data is read in or out a block at a time, and the cassette is stopped on an inter-block gap, at which time it may be safely removed and later remounted without any loss of data.

The process of swapping cassettes is somewhat tedious and human-error prone, so be careful. If an additional cassette recorder is available, then both recorders can be mounted in parallel (with extra jack plugs, etc.) with a ganged switch between them.

One can be left with the object cassette in Record mode, and the other with the source cassette in Read mode, and the switch operated between them on prompt from the compiler.

If an expansion interface is available, as well as a second cassette, then the compiler can be simply modified to write its object files to the second cassette, and no operator intervention will be required for the object cassette after initialization.

7. MOUNTING SOURCE CASSETTES (READ CASE1?):

If the "$INCL" option is used, then the compiler will, on encountering the $INCL line, prompt for the required file as follows:

```
FILE /FILENAME/ REQD - READ CAS?
```

At this point, the appropriate source cassette should be mounted (remove any object cassette first) and positioned just before the start of the file.

The cassette recorder should be placed in Read (replay) mode.

When everything is ready, the Enter key may be pressed, the compiler will read the first block of source code from the cassette and proceed with the compilation.

(Note that the line numbers in the included file bear no relation to the line numbers in the including file).

If an object file is bing generated, then it will be necessary to periodically swap cassettes on prompt from the compiler. Cassettes should not be swapped in anticipation, since it is not always possible to predict which cassette will be required first (refer to previous section).

Note that neither object cassettes nor source cassettes should be rewound or otherwise interfered with during compilation. They should be simply ejected or remounted as required by the compiler.

\# \# \#

Here's how to load People's Pascal

When receiving your People's Pascal I Program Development System, you might first enjoy seeing a compiled program run. Find the Bullseye demonstration object (machine language) program, and load it under the SYSTEM command'

when Level II prompts READY
type SYSTEM (and enter)
computer prompts: *?
user types NONAME (and enter)
after load is complete, computer prompts: *?
user types: /
play the game.

1-TO USE PEOPLE'S PASCAL I:

Load PPEC, the editor/compiler (first program on side A). But first, set memory size to 29700. This is of utmost importance. The computer will ask MEMORY SIZE? when you break out of Bullseye, and when it comes up from a cold start, and when you type, from the READY prompt:

SYSTEM (enter)

it prompts *?, and you enter /Ø, then 29700. Now on the READY prompt type and enter CLOAD. After loading, and before typing RUN, if you want to make a back-up copy, replace Tape 3 with a blank and type CSAVE. After verifying your CSAVE with CLOAD?, you are ready to proceed. Once you progress beyond the menu on the screen, you can never regain this list. The commands are written on page 13 of TRS-80 Computing 1:4. It should be noted that R (read) is similar to CLOAD, and W (write) to CSAVE. Also, L (list) is a display command ("print" in TRS-80 lingo), and P (print) is a true print command, "print" as in "printer" (LPRINT or LLIST in TRS-80ese).

A-To try compiling the Bullseye source program, locate the demonstration Pascal program, the third load on Side A. Use R1ØØ; it is easier than repetitive Rs. Make sure your Remote plug is into the tape recorder whenever using the Editor/Compiler, Translator or Interpreter. DO NOT allow the recorder to stay in Play position any length of time while the computer has the recorder stopped, because the pressure roller pushing against the capstain will cause a permanent depression that will result in a fatal dropout—if not on side A then side B. If the display says you have a FC error, read tape again and reload the source program. FC error means read error.

B-To Compile, have the file in memory (step A above, or C below) by either typing it in or reading it in by cassette. To the questions LP? and OBJ FILE?, hitting Enter means a no, and Y "yes". If you answer OBJ FILE? "YES" and are commanded by the Pascal program (source file) to use $INCL, you are going to have to switch back and forth from reading the source program of the Pascal Library to writing onto a blank tape.

When OBJ CAS READY? is displayed, remove the Pascal source-program tape from the recorder. Do not rewind or change position of the source tape. Insert the blank tape, advanced beyond the leader; put recorder into Record mode and depress Enter key. If READ CAS1 is displayed, just put the source tape back in and push Play and again hit Enter. The display will come back again with OBJ FILE READY. You must put the blank tape (same one back in, configure for Record, and hit Enter. Reminder: have your Remote plug in when working with Editor/Compiler, Translator and Interpreter.

C-To load other files just delete (D) the resident file and type or load from tape the new program to be compiled. When keyboarding, precede and end every line with a double-quote sign (") before the line number.

2-Load PPINT if you wish to debug a People's Pascal program, prior to translating it into a Z-80 object program. This should not be necessary for the sample programs. The P-code interpreter executes Pascal object code (P-code) output from the People's Pascal compiler. The P-code cassette file is read into memory and then interpreted under operator control. The program is written in People's Pascal, and both source (Pascal) and object (Z-80) versions are supplied. Load the object version of the interpreter under SYSTEM, with filename being NONAME. If you wish to copy PPINT onto another tape, to provide a back-up copy, use T-Bug, or a copying program.

To the questions "PCODE ADDRESS", "READ IN PCODES" and "MOUNT PCODE INPUT FILE ON CAS1", hitting Enter means a yes answer. Use "N" for no.

When INT>? comes up and you put in a command "E", the program comes back with a "?" (question mark), insert a line number.

3-Load PPTRANS, which translates P-code files into Z-80 machine code (object files). This program is written in Basic, and must have memory size set to 23000. To do this, type:

SYSTEM the computer prompts with:
*? you reply:
/Ø (making the line appear *? /Ø)

the computer asks:
MEMORY SIZE? reply 23000 (and Enter).

Then CLOAD the translator program. For back-up copy of PPTRANS, CSAVE before running it. If your answers to program questions are the same as the answers shown in parentheses, just hit Enter. Example:

PCODE START ADDRESS (24000)?-

Write down the last address; message will read:
last/ADDR =23988 (HEX 5DB4)

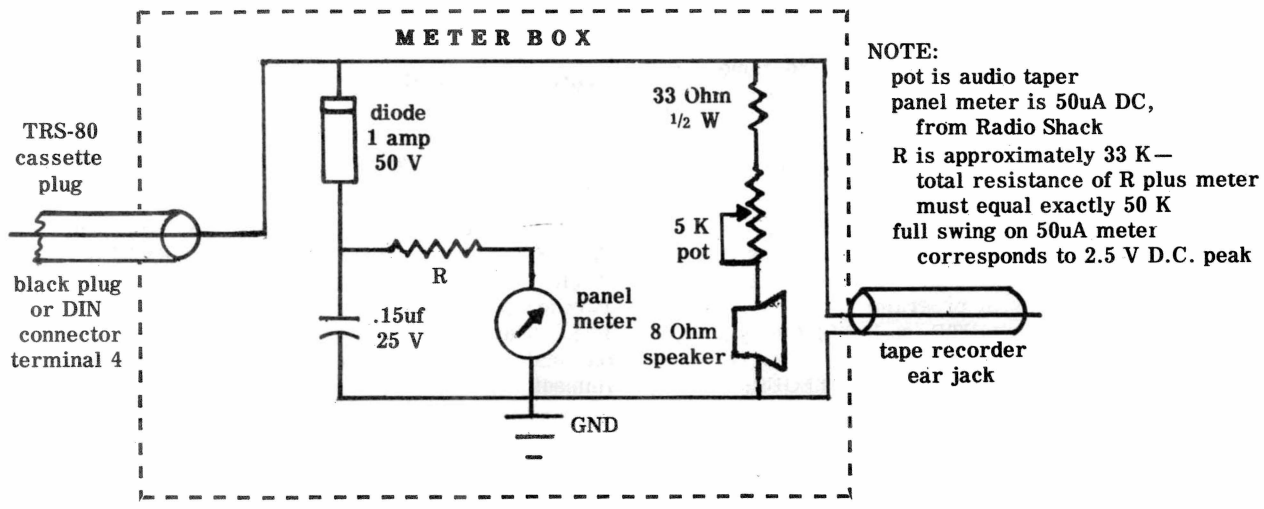
4-Load PPRUN run-time system (object file version) program. This provides subroutines which are called by translated People's Pascal programs. Refer to documentation, the 1:4 TRS-80 Computing, page 12., for details. Two source versions of the run-time system, in Editor/Assembler, also are included on Tape 3. Enter SYSTEM mode:

SYSTEM and at the prompt type filename
*? NONAME and at the next prompt mount T-Bug in recorder and enter filename
*? TBUG and at next prompt enter "/'
*? /

Now the system will write the user's object program to tape. It is wise to make two copies. These copies are made using the "P" command in T-bug.

5-You now should be able to run the People's Pascal Z-80 translated programs, in the SYSTEM mode:

SYSTEM at the prompt, enter filename
*? the name you gave the program (NONAME?)
*? / for more detail, see page 8 and 9 of the documentation, TRS-80 Computing 1:4.



INEXPENSIVE LOADING AID—This circuit is input to unmodified TRS-80 used by CIE to test People's Software release tapes. User might still find meter helpful even with free Radio Shack XRX-III modification, which everyone should get. Level II tapes load at 10uA on meter, Level I at 20uA. Circuit was designed by John Strong.

TAPE LOADING AIDS HELP

How can the TRS-80 owner ensure easy tape loading? There is no easy answer!

Back in TRS-80 year one, John Strong designed his **Strong box** meter circuit. CIE has used these ever since, currently using one on the computer we use to test People's Software release prints. See diagram for details of **Strong box**. All parts are available at Radio Shack.

The next improvement in tape-input reliability came as the free Radio Shack XRX-III modification. We have heard good reports on this mod, but have had less than spectacular results with it here at CIE. Two of our four machines have XRX-III boards installed, but the two used to test People's Software release prints are unmodified, so that our tapes are tested under the most severe conditions.

Our modified machines are less "touchy" than the unmodified ones, but they are still more stubborn than we would like them.

Next to come to our attention was the supposed-improved loading of tapes when you use Level 3 Basic from Microsoft. We didn't notice any improvement in loading.

Dave Miller and Paul Goelz sent us one of their E-Z Loader bare boards, which they sell for $6. We were very excited about this circuit, which is explained in the September **73 Magazine**. Imagine, a little digitizer circuit that restores your tape-input signal to the same shape it was when outputted to computer, for only $6.

The parts list for E-Z Loader gives Radio Shack stock numbers. We went to Radio Shack and bought all the parts except resistors, which we had in the parts box. Cost was over $24. Then, in order to get the board made, so that it wouldn't just sit around as a pile of parts, we asked a neighbor to build it for us. That cost $18. Our E-Z Loader totaled as much as a commercially-made product, but was a naked circuit board with exposed wires, two of which were 110 V AC !

Did E-Z Loader work? It made loading exceedingly easy, as its product name implies. It did not make tape-to-tape duplication possible, although the authors claimed it would. E-Z Loader reproduces only the top of the wave form. Our experience has been that both top and bottom spikes are needed.

E-Z loader is a good buy if you either have most of the parts in stock, or don't mind taking the time to order them from a parts house such as Jade, **a-n-d** you you have the time to guild the board. It is available for $6 plus self-addressed-stamped envelope from: Paul Goelz, 2228 Madison pl., Evanston IL 60202.

At this point, having been more than a year and a quarter into the project, and still not having easy loads, in a nice-looking package, **plus** tape-to-tape duplication (we are intrigued by the idea of adding such circuits to our tape-duplication amplifiers), we received a new product to test.

Data Dubber from The Peripheral People is a nice-looking plastic box that sells for $49.95. We bought one, and found it makes loading very easy. Just turn up the volume a little past the point where a red LED starts to flicker, and you're loading. We tried dubbing tape-to-tape and were delighted to find we could go three generations, but the third generation is a bit "touchy" on loading. This means the TRS-80 owner can copy good quality tapes that were CSAVed on a TRS-80 (original tapes), but should get questionable-quality dupes from duplicated tapes (commercial program tapes). Peripheral People are at Box 524, Dept. C, Mercer Island WA 98040, phone 206 232-4505.

Microsette offers a similar product, called Data Enhancer, and Alphanetics has its Tape Digitizer, for $44.94 without cassette remote on-off switch, or $49.95 with, postage paid, from Alphanetics, Box 597, Forestville CA 95436. Microsette is at 777 Palomar av., Sunnyvale CA 94086.

DISTRIBUTION INSTRUCTIONS, Cont:

1 CLOAD and RUN the Editor/Compiler, PPEC.BAS.
2 Position tape at the start of the Spirolateral source program, SPIRO.PAS (People's Pascal source files can be located on the distribution tape audibly by their numerous short blocks).
3 Read the complete file SPIRO.PAS into the text buffer with an R100 command. An FC ERROR AT 247 message indicates that the cassette file has been misread, or a file of the wrong format has been read (not a People's Pascal source file.
4 List the contents of the text buffer with the L command. You will see comments indicating where the library procedures start and finish. Use Enter—not Break—to halt the listing.

The library procedures are SET(ON, X, Y), RND(SEED) and AT(POSN). Note the line number range.

5 Mount a blank cassette. This will be your library file cassette.
6 Save the library procedures to the cassette, with a write command using the line number range containing the library procedures, i.e., W200-299. Then use the E command to write an EOF (end-of-file) mark to the cassette.
7 Repeat the save further up the tape for a back-up copy.
8 Rewind and dismout the tape, and remove the write-enable tab, to prevent accidental erasure.

—JOHN ALEXANDER

ERROR IN DOCUMENTATION

A paragraph on page 8 of the **TRS-80 Computing** Pascal documentation contains two errors which might mislead the user. About one-third of the way down column 1, it reads: "A write (X) will cause a number to appear on the screen equivalent to the value of X."

This should read "A write (X#) will cause a number to appear on the screen equivalent to the value of X." This is the essential fact that is being communicated, that the " x " sign is a formatting character which causes data items to appear as numbers.

Also, in the next sentence, i.e. "READ (A) will cause the input digits to be converted to a 16-bit integer...", should start "Read (A#) will cause . . ",

There is also a mistake on the second line of the last paragraph, but one on page 9, starting with "This normally will be hex 4A00 (decimal 18818)",actually should be, "This normally will be hex 4A00 (decimal 18944)".

People's Pascal distribution instructions:

1-TURTLE GRAPHICS PROCEDURES

The turtle graphics procedures may be copied to a separate file in the same way the library features may be separated from the spirolateral source program (see number 6 below). They could be copied to the same file as the library procedures, but the turtle procedures may not be required for all programs that use the library procedures. Refer to Ken Bowles' book "Microcomputer Problem solving using Pascal" (Springer-Verlag, 1977) for a full description of these procedures, and numerous example programs. For compatability, call the turtle file TURTLE.PAS.

Note that in the turtle system, location (0,0) corresponds to the center of the screen (63 23) in the Tandy screen graphics co-ordinate system. After calling the INITURTLE procedure, the turtle is initialized at location (0,0), screen center, at heading 0, pointing up the screen. Pencolor is initialized at white, ready for drawing.

It can be useful to create a file of dummy turtle procedures that in compiles can be used for syntax-error detection only. These compiles will be speeded up because the body of the turtle procedures will not be compiled. Once the program has been compiled free of errors, then the real procedures can be used instead.

A set of dummy turtle procedures would look like this:

```
PROC MOVE(DIST); BEGIN END;
PROC MOVETO(X,Y); BEGIN END;
PROC TURN(DEGREES); BEGIN END;
PROC TURNTO(ANGLE); BEGIN END;
PROC PENCOLOR(COLOR); BEGIN END;
PROC INITURTLE; BEGIN END;
```

You should use the same line-number range as for the real turtle procedures, so that the same program area can be used in case you want to make the turtle procedures into part of your program (the line numbers do not matter for $INCL(uded) code).

USING TURTLE GRAPHICS PROCEDURES

In order to use the turtle graphics procedures in your programs, it is necessary to have the following declarations in your program:

```
CONST BLACK = 0; WHITE = 1; NONE = 2;
(*PENCOLORS*)
VAR TURTLE: ARRAY (5) OF INTEGER;
PROC SET(ONOFF,X,Y); BEGIN CALL (19968);
END;
$INCL TURTLE.PAS (*TURTLE GRAPHICS
PROCEDURES*)
```

Note that the SET procedure is in the library, so you do not need an explicit declaration of it if you have $INCL(uded) or merged in the library.

Note also that the $INCL TURTLE.PAS line is not required if the turtle procedures are part of the program, as in the sample SPIRO.PAS.

In addition, the INITURTLE procedure should be called before any other turtle procedures. This procedure initializes the "turtle" array, which could otherwise contain unpredictable junk.

The turtle array is used as follows (this information is not required to use the turtle procedures):

TURTLE(0) = X-coord of turtle shifted left 7 bits
TURTLE(1) = Y-coord of turtle shifted left 8 bits
TURTLE(2) = current heading of turtle (0-360 degrees)
TURTLE(3) = sine of turtle(2) * 128
TURTLE(4) = cosine of turtle(2) * 128
TURTLE(5) = current pen color (0, black; 1, white; 2, none)

2-NEW P-CODE/Z80 TRANSLATOR

This version (2.2) eliminates one bug whereby consecutive literal strings in a write statement were not translated properly.

This version also has an additional translation option (not required) allowing relocation of translated Z80-code to a new address. The translated code will be stored at the Z80-code address specified in the previous question, but will be generated to execute at the relocation address.

The relocation address defaults to the same address as the Z80-code address.

This feature allows Z80-code to be generated to run anywhere in RAM memory except for the space occupied by the run-time system. Previously, the area used by the translator could not be used for translated Z80-code, since the translator would be overwriting a part of itself.

In order to use the relocation address feature, the generated code will have to be moved to its execution (i.e. the relocation) address with the RSM monitor from Small Systems Software, or its equivalent, or a small program. The new version of the People's Pascal interpreter has been relocated in this manner.

After the code has been relocated it is necessary to change the run-time system branch address to point to the new starting address rather than hex 59D8 (decimal 23000), which is the normal starting address for translated Z80 code.

The starting address of the translated Z80 code is held at address hex 4A09 in the run time system. To modify the run-time system for a relocated program, it is necessary to change the contents of the two bytes starting at this address from hex D8,59 (i.e. address 59D8) to the low and high bytes of the new starting address (i.e. the relocation address entered into the translator). The sequence of steps would be as follows:

1 Run the translator, entering the desired relocation address (in decimal), to produce the Z80-code version of the program.
2 Load RSM or the equivalent, and move the generated Z80 code from its current address (23000 decimal by default) to the relocation address.
3 Load the run-time system, TPRUN.Z80, and change the two bytes at 4A09 from D8,59 to the low and high bytes of the relocation address.
4 Save a complete image containing the modified run-time system code and the translated Z80 code onto a blank cassette. Run the relocated program by starting at the run-time-system entry point of 4A00.

The relocation address feature is not required for most programs. It is provided to allow people to edit, recompile, translate and move the People's Pascal interpreter if they wish. If you do not want to use this feature, then simply press enter in response to the RELOCATION ADDRESS? question.

3-NEW P-CODE INTERPRETER

A new faster version of the People's Pascal interpreter is supplied. This version is written in People's Pascal.

This version requires P-code to start at address 24500 or higher, rather than the 24000 allowed by the Basic version.

The source code for the interpreter comes in three files: the mainline (TPINT.MAIN.PAS); the single P-code execution procedure (TPINT.EXEC. P-code read-in procedure (TPINT.READPCODES. PAS). The latter two files are $INCL(uded) into the main program. Note that this program will immediately try to read the cassette if the response to the READ IN P-CODES (Y)? question is Y.

This program resides from hex 4980 to hex 5F44 with entry point hex 4A00.

It's name for SYSTEM loading purposes is NONAME. The T-Bug area (below 4980) is used for stack space. The program has been compiled to interpret programs with a maximum stack size of 400 16-bit entries.

4-NEW RUN-TIME SYSTEM

A new version of the run-time system is supplied. This version contains a 90-byte sine lookup table for the turtle graphics procedures, and cassette read and write subroutines for the new version of the People's Pascal interpreter.

The new source code is entirely contained within the TPSET.ASM source file.

The main run-time system source file TPRUN. ASM has not been altered. (Both objects are included in the TPRUN.Z80 object code). TPRUN. Z80 now resides from hex 4980 to hex 4F6E with entry point hex 4A00. For SYSTEM-loading purposes its name is NONAME.

5-SPIRO DEMO PROGRAM

This program is based on one in Ken Bowles' book. Note how to read a character string into an array in People's Pascal. The program draws a line for each R or L typed in. The line starts off one unit in length and is increased by one unit at each step.

The whole picture is multiplied by the size input. The turtle turns left or right depending on whether the current letter is L or R.

The angle turned is the one input. For instance, try SIZE=2, angle 92, SEQ=RRRRRRRRRR RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR

If Enter is pressed after the pattern has been drawn, the pattern will be repeated starting at the current angle.

A wide variety of interesting patterns can be produced with this program. You may give a copy of SPIRO.Z80 to your friends without infringing on the Pipe Dream Software copyright.

SPIRO.Z80 resides between hex 4980 and hex 6200, and its entry point is hex 4A00. It's name for SYSTEM loading purposes is NONAME.

6-THE PASCAL LIBRARY

The People's Pascal source library procedures are supplied on the distribution tape both as a separate recording and as part of the Spirolateral source program. If you should want to separate these procedures from the Spirolateral source, here is how you do it. This procedure is necessary if you want to have a copy of the turtle graphics procedures, and the way you do it is the same. You need a separate copy for easy inclusion ($INCL) or merging into your own programs.

Note: merging leads to easier compiles, but inclusion at compile time may be the only alternative if your porgram is large.

To make your own copy of the library:

(Continued on next page)

PEOPLE'S PASCAL I

Tape contents:

SIDE A:

PPEC.BAS (Editor/Compiler), CLOAD, set memory size to 29700.

PPINT.Z80 (P-code interpreter, in Z80 code), load under SYSTEM "NONAME", (#P4980 $F44 4A00 NONAME).
This new version of the People's Pascal Interpreter contains a complete copy of the run-time system TPRUN.Z80, except that two bytes have been changed, so that it jumps to a new starting address of the translated program on initialization. This is because the PPINT.Z80 program has been relocated lower down in memory to give the maximum availalbe user memory for P-code programs—8 Kbytes.

PPINT.MAIN.PAS, interpreter, mainline source program, written in Pascal. If you wish to re-compile it (not necessary, since the Z80 code version is above), load under PPEC.

PPINT.READ P-CODES.PAS (part of the interpreter Pascal source, to read P-codes), load at compile-time under PPEC.

PPINT.EXEC.PAS (part of the Interpreter Pascal source that executes P-codes), load at compile time under PPEC.

SPIRO.PAS, Spirolateral demonstration Pascal source program (needs compiling), load under PPEC.

SPIRO.Z80, Spirolateral demonstration object (machine language) program, compiled version of above, translated into Z80 code. Load under SYSTEM, with file name NONAME (#P 4980 6200 4A00).

SIDE B:

PPTRANS (Pascal P-code to Z80 translater), PPTRANS.BAS (Pascal P-code to Z80 translator), version 2.2. CLOAD, setting memory size to 23000.
This new version fixes a minor bug, where two consecutive literal strings in a "write" statement were not translated properly, i.e.: write ('hello', ' there'). Also, contains object code relocation facility explained in distribution instructions.

PPRUN.Z80, run-time system for translator, load under SYSTEM, NONAME. This new version contains 90-byte sine table and cassette routines, as explained in distribution instructions. For copying under T-Bug, punch commands (#P) are 4980 4F6E 4A00 NONAME.

PPRUN.ASM, run-time system source, load under Tandy TRS-80 editor/assembler (optional).

PPSET.ASM, part of the run-time system source, to set graphics, load under Tandy TRS-80 editor/assembler (optional).

BULLSEYE demonstration Pascal source program (needs compiling), load under PPEC.

BULLSEYE demonstration object (machine language) program, Z80 code, load under SYSTEM, NONAME (to make T-Bug copy, punch commands are 4982 65A8 4A00).

PCODE INTERPRETER /PPINT/

OPERATING INSTRUCTIONS

1. INTRODUCTION:

The People's Pascal P-code interpreter (PPINT) executes Pascal object code (P-code) output from the People's Pascal compiler.

The P-code cassette file output from the compiler is read into memory and then interpreted under operator control.

Various debugging commands are provided, such as the setting of breakpoints, to allow monitoring of program execution.

The interpreter currently is written in Basic. A faster version will soon be available written in People's Pascal.

PPINT is not intended for normal running of programs. Once a program has been debugged, it will be translated to Z-80 machine code with the People's Pascal translator for fast execution.

2. RUNNING the INTERPRETER:

Required operator responses are underlined. Here is what you see on the screen:

READY
/SYSTEM

*? /Ø

MEMORY SIZE? 24000

(Mount the PPINT program cassette for read).

READY
/CLOAD

(Wait until PPINT has loaded).

/RUN

TRS-80 TINY PASCAL INTERPRETER
PIPE DREAM SOFTWARE

P-CODE START ADDRESS (24000)?

(The default of 24000 is shown. If this is OK then just press]Enter[, otherwise enter the address you want to use. This is the address that the P-code program will be loaded at, and address of the first P-code instruction for the "R" (RUN) command. In the current version, this address cannot be less than 24000.)

READ IN P-CODES (Y)?

(The default of Y (yes) is shown. If [Enter] is pressed, then the interpreter will read in the P-code program from cassette. If the reply is "N", then it is assumed that the P-code program is already resident in memory and the next question will be bypassed).

MOUNT P-CODE INPUT CASSETTE ON CAS1?

(Mount the P-code object file output by the compiler on the cassette deck and press play.

Press [Enter] on the keyboard when cassette is ready.

The P-codes will be read into memory, and some "ADD AT" forward-reference fixups should appear on the screen).

INT/

(The interpreter is now ready to accept program execution and monitoring commands described below):

3. INTERPRETER COMMANDS:

Interpreter commands consist of single-letter mnemonics terminated by [Enter]. Some commands will result in further prompting.

3.1 RUN PROGRAM — R:

Initializes the program-counter and runs the program.

3.2 SINGLE STEPS:

Executes the next P-code instruction and returns to command mode.

3.3 GO ON — G:

Continues the program from the current program counter location.

The program may have stopped at a breakpoint, or after a single-step (S) command.

3.4 DISPLAY PROGRAM STATUS — X:

Displays the program counter (P), the base register (B) and the stack pointer (T) of the Pascal P-machine (which the interpreter is emulating).

The top two stack locations also are displayed.

3.5 DISPLAY PROGRAM TRACE — T:

Displays the last few P-code instructions executed by the P-code program, in time-of-execution sequence.

3.6 DISPLAY STACK LOCATIONS — K:

Prompts for a stack location (offset from start of stack) and displays six stack locations starting from this point.

3.7 SET BREAKPOINT — B:

Prompts with the breakpoint number for a breakpoint address (P-code program location). When the program counter reaches any value equal to a breakpoint value, the program will be stopped before the execution of the instruction at that location. The status of the program and its variables may be examined. The program may be continued with the "G" (GO) command or with the "S" (SINGLE STEP) command.

3.8 CLEAR BREAKPOINTS — C:

Clear all the breakpoints set by the "B" command.

3.9 DISPLAY BREAKPOINT LOCATIONS-Y:

Displays breakpoint locations previously set with the "B" command.

3.10 EXAMINE P-CODE LOCATION — E:

Prompts for a P-code location, which is loaded into the P-code location display pointer.

The P-code instruction at this location is displayed.

3.11 EXAMINE NEXT P-CODE INSTRUCTION — N:

Increments the P-code location display pointer and displays the P-code instruction at that location.

(Note: if this instruction has been used once, it is only necessary to press [Enter] to repeat it, stepping on to the next location.)

3.12 EXAMINE LAST P-CODE LOCATION — U:

Decrements the P-code location display pointer and displays the P-code instruction at that location.

3.13 QUIT — Q:

Exit from interpreter.

4. P-CODE INSTRUCTIONS:

(Note: POP X means remove the top element of the stack and load it into X (the stack is now one smaller). PUSH X means place the value of X onto the top of the stack (the stack is now one bigger).)

LIT	Ø,NN	LITERAL: PUSH NN
OPR	Ø,Ø	PROCESS AND FUNCTION return operation
OPR	Ø,1	NEGATE: POP A, PUSH - A
OPR	Ø,2	ADD: POP A, POP B, PUSH B+A
OPR	Ø,3	SUBTRACT: POP A, POP B, PUSH B-A
OPR	Ø,4	MULTIPLY: POP A, POP B, PUSH B*A
OPR	Ø,5	DIVIDE: POP A, POP B, PUSH B/A
OPR	Ø,6	LOW BIT: POP A, PUSH (A and 1)
OPR	Ø,7	MOD: POP A, POP B, PUSH (B MOD A)
OPR	Ø,8	TEST EQUAL: POP A, POP B, PUSH (B=A)
OPR	Ø,9	TEST NOT EQUAL: POP A, POP B, PUSH (B<>A)
OPR	Ø,1Ø	TEST LESS THAN: POP A, POP B, PUSH (B<A)
OPR	Ø,11	TEST GREATER-THAN or EQUAL: POP A, POP B, PUSH (B>=A)
OPR	Ø,12	TEST GREATER THAN: POP A, POP B, PUSH (B>A)
OPR	Ø,13	TEST LESS THAN or EQUAL: POP A, POP B, PUSH (B<=A)
OPR	Ø,14	OR: POP A, POP B, PUSH (B OR A) NOTE: these are the logical operators OR, AND and NOT)
OPR	Ø,15	AND: POP A, POP B, PUSH (B and A)
OPR	Ø,16	NOT: POP A, PUSH (NOT A)
OPR	Ø,17	SHIFT LEFT: POP A, POP B, PUSH (B shifted left by A bits)
OPR	Ø,18	SHIFT RIGHT: POP A, POP B, PUSH (B shifted right by A bits)
OPR	Ø,19	INCREMENT: POP A, PUSH A+1
OPR	Ø,20	DECREMENT: POP A, PUSH A-1
OPR	Ø,21	COPY: POP A, PUSH A, PUSH A
LOD	L,D	LOAD: LOAD A from (base of level offset L)+D, PUSH A
LOD	255,0	LOAD byte from memory address which is on top of stack onto top of stack: POP address, load A with byte from address, PUSH A
LODX	L,D	INDEXED LOAD: POP index, load A from (base of level offset L)+D+ Index, PUSH A
STO	L,D	STORE: POP A, store A at (base of level offset L)+D
STO	255,Ø	STORE IN MEMORY: POP A, POP address, store low byte of A at address
STOX	L,D	INDEXED STORE: pop Index pop A, store A at (Base of level offset L) + D + Index
CAL	L,A	Call procedure or function at P-code location A, with base at level offset L
CAL	255,Ø	Call procedure address in memory: POP address, PUSH return address, JUMP to address
INT	Ø,NN	ADD NN to stack pointer
JMP	Ø,A	JUMP to P-code location A
JPC	Ø,A	JUMP IF TRUE: POP A, IF (A and 1)=Ø then jump to location A
CSP	Ø,Ø	INPUT 1 CHARACTER: INPUT A, PUSH A
CSP	Ø,1	OUTPUT 1 CHARACTER: POP A, OUTPUT A
CSP	Ø,2	INPUT AN INTEGER: INPUT A#, PUSH A
CSP	Ø,3	OUTPUT AN INTEGER: POP A, OUTPUT A#
CSP	Ø,8	OUTPUT A CHARACTER STRING: POP A, FOR I:=1 TO A DO BEGIN POP B; OUTPUT B; END

NOTE: the result of a logical operation such as (A=B) is defined as 1 if the condition was met and Ø otherwise.

PCODE TO Z80CODE TRANSLATER /PPTRANS/

OPERATING INSTRUCTION

1. INTRODUCTION:

The People's Pascal translator program translates P-code object files output by the People's Pascal compiler (PPEC) into Z-80 microprocessor machine language object programs which can be saved with T-bug onto cassette, and loaded under the "system" command.

The translator has two optimization options. Z-80 code object programs can be optimized for speed, in which case the the program occupies about the same space as the P-codes.

Alternatively, object programs can be optimized for minimum memory usage, in which case the program occupies about half the memory but runs at about half the speed.

The P-code object program is read into memory. The normal starting address is 24000 (decimal).

After two passes of the P-code to generate a sorted table of P-code jump destinations and their corresponding Z-80 code addresses, Z-80 code is generated and stored in memory, normally starting at address 23000.

If the Z-80 program is large enough then it will overwrite the early portion of the P-code program, which is no longer required.

The end of the Z-80 program cannot "catch-up" with the end of the P-code program in a 16 K machine.

For a larger memory, the P-code may be started at a higher address.

Because of the size of the PPTRANS program, Z-80 code cannot be stored at addresses lower than 23000.

This leaves about 9.5 K bytes for the Z-80 program.

(Note that some of the memory below 23000 is used by T-bug and the People's Pascal run-time system (PPRUN). The rest is available for user-generated assembly-language subroutines callable from People's Pascal and/or for stack space — refer to memory maps.)

2. CHOOSING ADDRESSES:

Normally, the default addresses shown below are satisfactory for translated People's Pascal programs.

However, the translator provides the option to specify other addresses for exceptional cases, such as where a program has a very large stack requirement (i.e., greater than 3 K due to large arrays or use of recursion).

To obtain a larger stack, if the program itself is not too large, then 32500 may be used as the stack address.

If the program is large, then the program itself may be created at a higher address than 23000. Note this may involve reading-in the P-codes at a higher address also), and the now-larger space beneath the program used for stack. Note that this alternative is less desirable since the total amount of memory to be saved onto cassette is correspondingly larger as the runtime system is at a fixed location, and thus the program takes longer to load.

3. OPERATION:

Note: Operator responses are underlined.

1. READY
   >SYSTEM
2. *? /Ø
3. MEMORY SIZE? 23ØØØ

   (now mount PPTRANS program cassette)
4. READY
   >CLOAD

   (wait until load is finished)
5. READY
   >RUN
6. TRS-80 PEOPLE'S PASCAL TRANSLATOR
   DEFAULT REPLIES TO PROMPTS ARE SHOWN IN BRACKETS: (parentheses):

   P-CODE START ADDRESS (24000)?
   <ENTER> or your address
7. Z-80-CODE START ADDRESS (23000)?
   <ENTER> or your address
8. Z-80 STACK ADDRESS (GROWS DOWN) (22999)? >ENTER< or your address
9. OPTIMIZATION (F=fast, S=small) (F)?>ENTER< or S
10. DISPLAY CODES (Y)?
    <ENTER> or N

    (pptrans runs faster if codes are not desplayed)
11. PRINT CODES (N)? <ENTER> or Y
12. MOUNT P-CODE INPUT FILE ON CAS1 AND TYPE "RUN"

    (Now rewind and remove PPTRANS program cassette and mount P-code object program cassette for input, type "RUN" and "RUN". The translator will read in the P-code cassette, and perform translation. Wait until translation is complete, with the ratio between P-code and Z-80 code, etc., being displayed on the screen.)
13. (Note the last-used Z-80 address. The translated Z-80 program is now residing in memory starting at address 23000, or your chosen address, at which this code will be executed.)
14. (Note the last address used by the Z-80 program which is displayed on the screen. Rewind and remove the P-code input cassette. Mount the run-time system object cassette.)
15. READY
    >SYSTEM

    (The run-time system will be read into memory. Now memory contains your translated program and the Peoples Pascal run-time system. When loading of the run-time system is complete, rewind and remove the run-time system cassette.)
17. (At this point it is possible to run the program via the system command. However, it is wise to save two copies of the program first, using T-Bug. To do this, mount the T-Bug cassette for input).
18. *? TBUG

    (The Tandy T-Bug program will be loaded into memory. At this point, memory contains your program, the run-time system and T-Bug. Using T-Bug, it is possible to make a copy of your program and the run-time system, with or without a copy of T-Bug. This copy will be loadable under the "SYSTEM" command. Rewind and remove the T-Bug cassette. Mount a blank cassette for output.)
19. *? /
20. # P 438Ø XXXX 4AØØ YYYYYY (for program with T-Bug)
    OR
    # P 498Ø XXXX 4AØØ YYYYYY (for program withort T-Bug)

    (Where XXXX is the last-used address of the Z-80 program in hex noted after step 13, and YYYYYY is the filename to be assigned to the program on cassette).
21. (Wait until T-Bug "P" command is complete, then reposition output cassette and repeat step 20 as many times as required. Rewind and remove the blank cassette and write on it the program name and the cassette tape counter locations of each copy of the program. The date can also be useful.)
22. # J 4AØØ (To run the program if required).

4. RUNNING TRANSLATED PASCAL PROGRAMS:

Normally, all that is required to run a People's Pascal program is to load it under the "SYSTEM" command, and to run it by typing "/" after it has been loaded into memory.

This causes control to be passed to the address which was specified as the program entry point on the cassette tape file.

This normally will be hex 4AØØ (decimal 18818), which is the standard entry point of the People's Pascal run-time system.

There is an alternative entry point to the run-time system, hex 4AØE, which allows the user to initialize the People's Pascal stack pointer at other than the fixed value, and/or to run a program which does not start at the standard address (59D8 hex or 23ØØØ decimal).

When entered at this point (4AØE) the run-time system will prompt for these values.

T-Bug may be used for debugging translated programs, although it is usually easier to use the P-code interpreter to find bugs.

To use T-bug, type "/1728Ø" after loading the program, rather than just "/". This is the T-Bug entery point. Refer to Tandy T-Bug operating instructions for further help in using T-Bug.

PROGRAM DOCUMENTION

TEXT EDITOR

1. INTRODUCTION:

The PP editor is a line-oriented text editor using line numbers for text identification.

Intra-line editing is not supported in the current version.

Lines are stored in a reserve area of high-address memory called the text buffer.

The program is written in Level-II Basic with machine language subroutines to move text up and down in the text buffer for speed efficiency.

2. COMMANDS:

People's Pascal commands are:

C- COMPILE - Press control to the compiler.
D- DELETE - Delete line number range.
E- EOF - Write end-of-file mark to cassette file.
F- FREE - Free bytes left in text buffer enquiry.
L- LIST - List lines in text buffer on screen.
N- NUMBER - Renumber lines of text in the buffer.
P- PRINT - Print lines on the line printer.
R- READ - Read block(s) from a cassette file.
W- WRITE - Write line(s) of text to cassette.

3. RECORD FORMATS:

3.1 LINE IN TEXT BUFFER:

Byte Ø- - Length of line (0-255) including self and line number bytes.
Byte 1,2- - Line number of this line in binary 0-32767 (1-32766 available for user)
Byte 3-N - Text of line

3.2 TEXT BUFFER:

Line 1- Dummy line, text = " ", line-number = Ø
Lines 2 to N-1: actual lines of text
Line N: Dummy line, no text, line-number = 32767
Top byte: Buffer has been initialized flag (14 = has, any other value = has not)
Top Byte-2-1: Saved copy of FA variable. This is the only variable that needs to be saved over a run command. (FA = address of last byte used in text buffer.

3.3 LINE RECORD IN CASSETTE FILE BLOCK:

Byte Ø: Length of text of line in bytes. If length would be equivalent to certain ASCII characters such as quote ("), then one space is added to line and length is incremented by 1 to avoid trouble with Level-II Basic I/O.
Bytes 1 to M: Line number in ASCII numeric characters (Ø <M< 6)
Bytes M to N: Text of line

3.4 CASSETTE FILE BLOCK FORMAT:

Maximum Size = 240 characters

Byte Ø: Quote symbol (") to hold block together through Level-II Basic I/O

Bytes 1 to N: Lines of text

4. PROGRAM VARIABLES:

L$ = Current line
LN = Current line number
LG = Length of a string
V = Varptr of a variable
W = Varptr of a variable and temporary variable
X = 16-bit number (work variable)
P = Pointer to (holds address of) current line record in text buffer
TA = Top address (32767)
FA = Address of last byte used in text buffer
SA = Address of start of text buffer
ML = Maximum line number allowed (32767)
YY$ = Used in sneaky transfer of line from text buffer to L$
LM = Last cassette I/O mode (read/write source/object cassette)
CM = Current Cassette I/O mode
BR = Bottom of line number range
TR = Top of line number range
A$ = Temporary string variable
QL = Line number of current line in text buffer
BL$ = String to hold cassette source file I/O block
B$ = Temporary string variable
PO = Pointer to last (old) current line in buffer
Q1 = Text buffer move parameter — source address
Q2 = Text buffer move parameter — destination address
Q3 = Text buffer move parameter — byte count (HL=Q1, DE=Q2, BC=Q3, FOR LDDR, LDIR Z-80 instructions)
z8$ = String variable used to hold Z-80 code subrs executed via USR (Ø)

5. PROGRAM ROUTINES

Note: The program has been renumbered from 1 with an increment of 1 to reduce its memory size.

Because of the long lines allowed (256 bytes) in Level-II Basic, additions and changes are still possible.

If the program is renumbered, then lines 1-3Ø should still start at 1 in increments of 1 to avoid undue expansion.

It is suggested that other lines be renumbered so that the new numbers equal the old numbers times 10, so patches etc. can still be applied, yet the new version still resemble the old.

194-2Ø1 Once-only program initialization
203 Input command prompt (mainline loop)
204 Command interpreter
205 Line insertion/deletion
209 Extract line number (LN) from line
211 Position P pointer at line with line number LN in text buffer
213 Decode line number range in L$ into BR and TR
218 Display line on screen/printer
219 Interpret line number range and find first line in text buffer
220 List (L command) routine
222 Delete (D) routine
224 Write (W) routine
230 Write end-of-file mark (E) routine
231 Re-number (N) routine
234 Write block to cassette (BL$)
236 Write line to cassette (append line to BL$)
242 Read (R) routine
243 Put lines from current block (BL$) into text buffer
244 Read a block (BL$) from cassette
247 Split next line from BL$
248 Point to next line in text buffer
250 Copy current line in text buffer into L$,LN
251 Insert (replace, delete) line in text buffer
254 Delete line from buffer
256 Restore FA from reserved high memory after a run command (which wipes all variables)
257 Save value of FA in reserved high memory
258 Put low, high byte of X into Z8$
259 Set up Z-80 machine language move routine for text buffer and execute this routine by obtaining address via VARPTR for USR (Ø)
260 Execute Z-80 machine language routine in Z8$ via USR (Ø)

TINY PASCAL COMPILER

PROGRAM DOCUMENTATION

1. INTRODUCTION:

For a full discussion of the principles of operation of this compiler, refer to "Byte"magazine, October, 1978, "A Tiny Pascal Compiler — Part 2: the P-Compiler", by Kin-Man Chung and Herbert Yuen.

This program is largely based on the program listed in that article, but recoded in Level-II Basic and optimized for minimum memory usage.

The compiler is a one-pass compiler using a technique called recursive descent.

Tandy Microsoft Level-II Basic is used recursively.

The compiler has its own stacks, one for strings and the other for numeric variables. For maximum speed and memory efficiency, all numeric variables are declared to be of integer type.

In effect, to compile a program, the compiler simply follows the syntax diagrams (railroad diagrams) of the language, deciding which route to take by looking at the source program text, and emitting object code like smoke as it goes.

One disadvantage of the compiler is that it does not have the ability to recover and continue after an error in the source program. To provide this facility would increase the complexity of the compiler, and thus its memory requirement, cutting in to the size of the text buffer, or the ability to correct source program errors without having to load in a different program for editing.

2. PROGRAM VARIABLES:

T$() = Symbol Table — Identifier name string array
S() = Stack — Compiler's number stack
S$() = Stack — Compiler's string stack
T1() = Symbol Table — Absolute program lexical level at which identifier was declared
T2() = Symbol Table — Value if constant, or displacement from base if variable, or P-code location if process or function
T3() = Symbol Table — Array size for array, else number of parameters for process or function identification
S9 = Numeric Stack Pointer
P8 = String Stack Pointer
M$ = P-code Operator Mnemonics string values
WØ$ = People's Pascal Reserved Words string values
TØ = Maximum Number of Symbols (size of symbol table) checked for
FL = Nested File Level for "$INCL" (max 1 in current version)
T1 = Pointer into Symbol Table Arrays T1(), T2(), T3()
K1 = Number of Parameters in previous process, function
OF$ = Object File Flag — Non-null = >object file to be produced
OB$ = Object File Cassette output block area
BZ = Pointer into OB$ object file block area
NØ = Number of Reserved Words in WØ$
N1 = Maximum Value of an integer
N2 = Length of identifier
I$ = Constant String of value "IDENT"
Y9 =
LN = Current Program Line Number
L$ = Current Line of program text
CI = Character Pointer into L$
X$ = Current Character of Program Text (also used to hold "expected" in error section)
R = String Value of Next Token expected by the compiler
E = Error Code Number
U,V,W = P-Code Generation — Parameters to code-generation routine: U=op-code, V=relative level, W=value
O = String Variable containing next program token (also used to hold "missing" in error section)
ML = Maximum Program Line Number
BL$ = Cassette Input file block area
CM = Current Cassette I/O Mode (refer to LM)
I,J,K = Temporary Loop and work variables
A$ = Next Program Text Token, returned by scanner
T = ASCII Value of X$
B$ = Temporary String Variable
Z$ = Temporary String Variable
N3 = Value of Token for "NUM" type tokens
C$ = Value of a String Literal
K$ = Symbol Table Entry Type — C=constant, A=array, P=process, Y=function, V=variable
Y$ = Temporary String Variable to hold parameter to be pushed onto, or having been popped from the string stack S$()
TT$ = Temporary String Variable to hold symbol table entry type (refer K$)
X = Temporary Variable to hold value to be pushed onto or to be popped from number stack S()
K2 = Procedure or Function Call — Number of actual parameters
K3 = Procedure or Function Call — Index of entry in symbol table
C1 = P-Code Location Pointer
I1 = Case Statement — Number of case labels
I2 = Case Statement — Number of nested case statements
F9 = Flag 1=TO, Ø=DOWNTO; also 1= parameters, Ø=no parameters
DØ = Pascal Stack Location Holder
L1 = Absolute Static (lexical) level of procedure or function declaration
CC = Next Byte of P-Code to be output
N4 = VARPTR of W
LG = Address of OB$ (output block area)
LM = Last Cassette I/O mode (refer CM)
1= Write source file (editor)
2= Write object file (compiler)
3= Read source file (editor)
4+= Read source file (compiler)
LP = Line - Printer Output Flag — 1=print, Ø=don't print

3. PROGRAM ROUTINES:

2 - Check that current token is as required (R) and issue error message number (E) if not
3 - Get next token, check that it is as expected and issue error message if not
4 - Push X onto numeric stack
5 - Pop X from numeric stack
6 - Get next character of program text into X$ and ASCII value into T
7 - Issue error message number (E)
8 - Analyse expression
9 - Code generation — output 4-byte P-code specified by U, V, W
10 - Get next token
11 - Analyze a statement
12 - Enter symbol in A$ into symbol table at position T1
13 - Generate P-code with V (level offset)=Ø
14 - Push string in Y$ onto stack
15 - Pop string from stack into Y$
16 - Analyze array index expression
17 - Code Generation — generate variable-level reference portion of P-code
18 - Generate OPR P-code (U=1, V=Ø)
19 - Generate LIT P-code (U=Ø, V=Ø)
20 - Generate P-code with W=Ø and V=Ø
21 - Scan for start of array index expression
22 - Scan for left parenthesis
23 - Scan for right parenthesis
24 - Initialise various compiler variables
25 - Start of compiler execution — INIT
26 - Compiler mainline — compile block + "." at end, re-run program to clear all variables
28 - Check that current token is as required, and emit error message if not
34 - Input a new line of source code
35 - Initialize $INCL(ude) cassette file input
36 - Read line from $INCL file
38 - Get next token from source program into string variable O (no dollar ($) for brevity since this is so common)
69 - Search symbol table for identifier
70 - Analyze constant (CONST) declaration
71 - Obtain value of constant
76 - Analyze single VAR(iable) declaration
77 - Analyze simple expression
83 - Analyze term
88 - Analyze factor
103 - Analyze expression
111 - Analyze statement
113 - Analyze variable assignment (A:=B)
119 - Analyze write statement
124 - Analyze read statement
138 - Analyze IF statement
140 - Analyze compound statement
141 - Analyze compound statement (BEGIN... END)
143 - Analyze repeat statement
145 - Analyze WHILE statement
146 - Analyze CASE statement
155 - Analyze FOR statement
159 - Analyze block
162 - Analyze CONST declaration
164 - Analyze CONST declaration
167 - Analyze ARRAY declaration
170 - Analyze PROC declaration
171 - Analyze FUNC declaration
177 - Analyze BEGIN
181 - Code Generation — Output 4-byte P-code to object file
187 - Output "Fix up forward reference" pseudo P-code to object file and display
188 - Output 1 byte of P-code in CC to cassette output block
189 - Output block of object code in OB$ to cassette and reinitialize OB$

NOTES:

Every attempt has been made to reduce to minimum the size of the compiler.

This is the reason for the "jump table" at the front of the program. These short line numbers are used frequently and take less space.

Whenever a subroutine ends with GOSUB XXXX: RETURN, this code has been replaced with GOTO XXXX, which is functionally equivalent, takes less space, but tends to make the program messy to read. However, these occurrences are recognizable, with a bit of practice.

The construct RETURNELSERETURN has been used at the end of IF statement lines to avoid the memory overload of usinganother program line.

Some IF statements involving the comparison of quoted logical operators, etc., have caused Level-II Basic a few headaches, and will not work without embedded spaces.

The current version of the compiler is combined with the editor program, but these two programs are relatively separate, only sharing certain initialization code, and the routines for finding the next line in the text buffer and copying that line into LN, L$. The cassette source read routine is also shared together with the line unpack routine.

CONVERSION TO DISK:

The following tasks would be required/desirable:

1. - Separate editor and compiler into two separate programs.
2. - Add disk file access capability for editor, compiler, translator and interpreter, for both source and P-code object files.
3. Add capability to write translated Z-80 object code files to disk either as a translator facility, or as an extra program or as an option of the run-time system.
4. - Allow greater depth of nesting of "$INCL"(ude)s.
5. - Alter emphasis in compiler from minimum memory requirement to higher speed.

CONVERSION FOR ADDITIONAL MEMORY:

The initialization of TA (top address)

would need to be altered from 32767.

Care would be required with address calculation in integer mode when handling addresses over 32767. It might be necessary to use floating-point data types for such variables.

If the text buffer were to be significantly enlarged it would be desirable to use a machine-language routine to replace the Basic routine used to position pointer Pa at the address of the line in the text buffer with a given line number.

This simple function could be easily implemented and would eliminate any apparent delay in most commands.

P-CODE INTERPRETER TPRINT

PROGRAM DOCUMENTATION

1) INTRODUCTION:

The People's Pascal P-code interpreter (PPINT) executes Pascal object code (P-code) output from the People's Pascal Compiler. The P-code cassette file output from the compiler is read into memory and then interpreted under operator control. The program currently is written in Level-II Basic.

2) PROGRAM VARIABLES:

SZ -Size of stack array for program to be interpreted.
S1 -Size of stack at which overflow message is emitted. A little less than SZ.
S() -Stack array.
M$ -Holds P-code instruction mnemonics.
PS -P-code start address.
PP -P-code pointer—points to current P-code during read-in from cassette.
Z$ -Temporary string variable.
P1 -First byte of 4-byte P-code.
P2 -Second byte of P-code.
P3 -Third byte of P-code.
P4 -Fourth byte of P-code.
U -Size of trace array.
BL -Maximum number of breakpoints allowed.
TR() -Trace array stores last few P-codes executed.
BR() -Breakpoint array stores breakpoint locations.
BA -Copy of base for Level L.
B -Base register of current stack frame holds address of base of stack frame of current block.
L -Level offset.
A -P-code second (16-bit) operand.
Z -
T -Stack pointer.
P -Program counter (holds P-code locations).
ST -Stop execution flag, Ø=OK, 1=stop.
PØ -P-code location display pointer.
TP -Trace array pointer (circulates around TR() trace array as instructions are executed and stored in TR().
K -
X -P-code address in memory.
N! -16-bit operand.
F -P-code op code.
IX -LODX, STOX indexing flag, 0=not, 1=indexing.
SA -Top of stack 16-bit word.
SB -Top-1 of stack 16-bit word.
M! -Temporary variable.
H -Parameter for hex input/output.
PC -Parameter for hex.
PC -P-code location parameter.
N -Pointer parameter into M$.
I -Temporary variable.
J -Temporary variable.
BP -Number of breakpoints currently set.
CM$ -Command mnemonic string.
IB$ -Input data block from cassette file.
IP -Pointer to next byte in IB$ input block area.
Z9$ -String to hold Z-80 machine code routine to read in a block of data from P-code input file.
LN -Length of input block IB$ in characters.
ZZ -Temporary variable.

3) PROGRAM ROUTINES:

100 -Initialization.
1000 -Initialization — parameter input.
1013 -Cassette file read-in to memory. Forward reference fix-ups output to the P-code object file are fixed up in memory as they are encountered. Pseudo-P-codes 253 and 254 are used to label these items. Pseudo P-code 255 is used as an end-of-program indicator.
9900 -Initialization.
20040 -Routine to bind the base address corresponding to a given level offset.
20060 -P-code program initialization.
20090 -"Execute P-code instruction" routine. Ends at line 20680.
20120 -P-code or op-code branch out depending on value of op-code.
20140 -LIT — Execute literal instruction.
20150 -OPR — Execute OPR instruction.
20520 -LOD — Execute load instruction.
20530 -STO — Execute store instruction.
20540 -CAL — Execute call instruction. Note: if it is an absolute call, and the address is that of the graphics "SET" routine, then a SET/RESET will be performed instead of a call.
20550 -INT — Execute increment-stack pointer instruction.
20560 -JMP — Execute JUMP instruction.
20570 -JPC — Execute conditional jump instruction.
20580 -CSP — Perform CSP function.
20690 -Get 2nd P-code instruction operand (16-bit).
20710 -Display P-code instruction at location PC.
20760 -Check if a breakpoint has been encountered.
20820 -MAINLINE — Accept and execute operator commands.
20830 -Input command and execute it.
20840 - 20970 Command interpreter.
30010 -Get next P-code from cassette file.
30070 -Z-80 machine language routine to read-in a block of data from cassette input file. Level-II Basic I/O is bypassed to avoid records being truncated if certain values [e.g. (")] occur in data.
30080 -Routine to read Z-80 routine into Z9$.
30100 -Fix up forward reference item encountered on cassette input file.
30210 -Routine to execute machine language subroutine in Z9$ which reads a block of data into the Level-II Basic 256-character I/O buffer at address 16870, and to copy this data into block area IB$.
30300 -Routine to call an assembly-language subroutine whose address is on top of the Pascal stack, unless the address is that of the graphics "SET" routine, in which case, a Level-II Basic SET/RESET instruction is performed instead.

P-TO-Z80CODE TRANSLATER PTRANS

PROGRAM DOCUMENTATION

1) INTRODUCTION:

The P-code-to-Z80code translator program (PPTRANS) translates a P-code program into a Z-80-microprocessor machine-language program. The P-code program is input from a P-code object cassette file generated as output by the People's Pascal compiler.

A People's Pascal program, when translated to Z-80 machine language, will typically run about five times faster than an equivalent Level-II Basic program. The

The following People's Pascal statements execute in about 5 seconds:

```
FOR I:=0 TO 127 DO BEGIN
   FOR J:=0 TO 47 DO BEGIN
      SET(ON,I,J);
   END; (*FOR*)
END; (*FOR*)
```

Whearas the equivalent Level-II Basic statements:

```
FOR I=0 TO 127: FOR J=0 TO 47: SET (I,J): NEXT J: NEXT I
```

take about 42.5 seconds.

2) DESCRIPTION:

The following actions are performed:
1) Initialization — Translation parameters are prompted for and saved, then initialization code is deleted and the program run again. Note: a "CSAVE" after running the program will not produce a viable copy.
2) P-codes are read-in from cassette and stored in memory normally starting at address 24000. Forward reference fix-ups generated by the one-pass Pascal compiler are fixed up as they are encountered in the cassette file. These forward reference fixups ("add X at Y") are stored as pseudo P-codes in the P-code cassette file using op-codes 253 and 254.
3) Pass-1: establish table of P-code jump or call destination locations by looking for JMP, JPC and CAL op-codes; remove duplicates and sort table into ascending P-code location sequence (= ascending Z-80 address sequence).
4) Pass-2: generate Z-80 addresses corresponding to P-code locations in table by translating P-code to Z-80 code and obtaining the length of each Z-80 code.
5) Pass-3: generate Z-80 codes, including correct addresses from table; store in memory normally starting at address 23000, and list-out P-codes with equivalent Z-80 codes in hex and addresses in decimal and hex.

3) PROGRAM VARIABLES:

DI =Display flag: 0=don't display object code, 1=do.
LP =Print flag: 0=don't print object code, 1=do.
OP =Optimization flag: 0=optimization for speed, 1=minimum memory use.
PA =P-code address table — array.
ZA =Z-80 code address table — array.
JT =Run-time system jump-table address.
CO$ =P-code op-code mnemonics stored in string.
PS =P-code storage start address.
ZS =Z-80 code storage start address.
PP =Current P-code pointer.
ZP =Current Z-80-code pointer.
ZZ$ =Temporary string variable.
P1 =Value of first byte of current P-code.
P2 =Value of second byte of current P-code.
P3 =Value of third byte of current P-code.
P4 =Value of fourth byte of current P-code.
P5 =Value of third and fourth bytes of current P-code taken as a 16-bit integer.
Z8$ =Storage area for bytes of current Z-80 code (the Z-80 instruction's equivalent to the current P-code).
A$ =Temporary string variable.
PC =Current P-code pointer.
I =Temporary loop variable.
J =Temporary loop variable.
AN =Actual number of addresses in address table.
K =Temporary variable.
N! =Temporary variable.
CL =Current Z-80 code length in bytes.
AP =Index into address tables PA and ZA.
X =P-code indexed LOD/STO operation (LODX/STOX) flag. Also work variable in initialization.
LT$ =P-code literal string accumulation area for CSP 8.
XX =Pointer to address within jump table.
RT =Run-time system routine number.
XL =Low byte of XX (also temporary variable).
XH =High byte of XX (also temporary variable).
P6 =-2*P5.
P7 =Low byte of P6.
P8 =High byte of P6.
HX$ =Holds two-character hex string equivalent to one binary byte.
IX =Index into P-code address table.
BY =Byte to be converted to hex.
BH =Four-bit "nibble" of BY.
HB =Hex base, ="Ø" or "A".
HX =Four-bit nibble to be converted to hex character.
Z9$ =String to hold Z-80 read-cassette machine language routine required to bypass Level-II Basic input routine.
LN =Length of block read from cassette (P-code input file).
IB$ =Area to hold P-code block read from cassette.

4) PROGRAM ROUTINES:

1-25 = Initialization — input parameters.
27 =Prompt for number showing default value. Accept reply and save it.
29 =Save a value in high memory.
31-35 Read-in P-code cassette file.
37 =Append "Push HL" Z80-code onto Z8$.
39 =Restore a value from high memory after "run".
41 =Further initialization, mainline.
45 =Pass 1, mainline.
47 =Pass 2, mainline.
49 =Pass 3, mainline.
53 =Termination, mainline.
59 =Scan P-codes in memory for P-code jump destinations and store these in P-code location table.
69 =Bubble sort P-code location table.
73 =Obtain current P-code into P1, P2, P3, P4, P5.
77 =Calculate P5 from P3, P4.
79 =Display current P-code on screen.
85 =Pass-2: Calculate Z80 addresses corresponding to P-code locations and store in ZA.
97 =Pass-3: Generate, display and store Z80 codes.
103 =Display current Z80 code on screen.
105 =Generate Z80 code corresponding to current P-code.
113 =Translate LIT P-code to Z80 code.
115 = Translate OPR P-code to Z80 code.
125 =Translate LOD P-code.
135 =Translate STO P-code.
143 =Translate CAL P-code.
149 =Translate JMP P-code.
159 =Translate JPC P-code.
165 =Translate CSP P-code.
173 =Convert XX to XL and XH low and high bytes.
175 =Append A "CALL XX" Z80 code to Z8$.
177 =Put a "LD HL,[P6]" Z80 code into Z8$.
181 =Put a "LD HL, [P5]" Z80 code into Z8$.
183 =Put a "LD L, (IX + [P7])
LD H, (IX + [P7 + 1])
PUSH HL" Z80 code into Z8$
185 =Put a "POP HL
LD (IX+[P7]), L
LD (IX+[P7+1]),H" code into Z8$.
187 =Calculate P7 from P5.
189 =Append a "LD A, [P2]" Z80 code into Z8$.
191 =Append a "JP XX" Z80 code to Z8$.
193 =Find Z80 address in table ZA corresponding to P-code location held in P5 by looking up this P-code location in table PA (linear search).
195 =Look up P-code location held in P5 in table PA.
199 =Display Z80 code in Z8$ in hex plus current Z80 address in decimal and hex and store Z80 code in memory at current Z80 address.
209 =Convert binary byte in BY to two hex characters in HX$.
211 =Convert 4-bit nibble in HX to hex character and append to HX$.
215 =Store a 4-byte P-code at the current P-code location.
217 =Display the contents of the PA and ZA tables.
219 =Get next P-code from cassette in P1, P2, P3, P4.
231 =Z80 machine language routine to read a block of P-codes from the cassette input file into the Level-II Basic I/O buffer area.
233 =Routine to read Z80 machine language routine into Z9$.
237 =Routine to apply forward reference fixup "pseudo-P-codes" (op-codes 253 & 254) to P-code in memory as these are encountered on the P-code cassette input file.
239 =Routine to execute the Z80 machine language cassette-read routine held in Z9$ with the USR(Ø) function, and transfer the data read into the string area IB$.

5) Z80 CODES GENERATED FOR EACH P-CODE:

(Note: IX register is used for Pascal current stack (B) base register, SP is used for stack pointer (T), HL is used for argument register, A is used to hold level offset).

5.1) OPTIMIZING FOR SPEED:

MNEMONIC	OPERATION	Z80-code
LIT Ø,NN	load literal onto stack	LD HL,NN PUSH HL
OPR 2	add operation	POP DE POP HL ADD H,DE PUSH HL
OPR 19	increment operation	POP HL INC HL PUSH HL
OPR 20	decrement operation	POP HL DEC HL PUSH HL
OPR 21	copy top of stack	POP HL PUSH HL PUSH HL
OPR N	arithmetic or logical operation	CALL OPRN

LOD Ø,N	load variable onto stack (-64 < N < 64)	LD L, (IX+[-N*2])
		LD H, (IX+[-N*2+1])
		PUSH HL

LOD Ø,NN	load variable onto stack	LD HL,NN CALL LOD
LOD L,M	load Level L variable onto stack	LOD HL,NN LD A,L CALL LOD1
LODX,Ø,M	load current level indexed (array) variable onto stack	LD HL,M CALL LODX
LODX L,M	load Level L indexed variable onto stack	LD A, L CALL LODX1

STO Ø,N	store current level variable from top of stack (-64 < N < 64)	POP HL
		LD (IX+[-N*2]),L
		LD (IX+[-N*2+1]),H

STO Ø,NN	store current level variable from top of stack	LD HL,NN CALL STO

STO L,M	store Level L variable from top of stack	LD HL,NN
		LD A,L
		CALL STO1

STOX Ø,M	store current level indexed (array) variable from top of stack	LD HL,M CALL STOX

STOX L,M	store Level L indexed variable from top of stack	LD HL,M
		LD A,L
		CALL STOX1

CALL Ø,M	call procedure or function at P-code loc.M	CALL CAL JP [Z80ADR]
CALL L,M	call procedure or function declared at Level L	LD A,L CALL CAL1 JP [Z80ADR]
CALL 255,Ø	call machine language subroutine	CALL CALA
JMP Ø,M	jump to P-code location M	JP [Z80ADR]
JPC Ø,M	jump if condition false to P-code location M	POP AF JNC [Z80ADR]
JPC 1,M	jump if condition true to p-code location M	POP AF JC [Z80ADR]
CSP Ø,N	call standard procedure number N	CALL CSPN
INT -1	adjust stack pointer	POP BC
INT -2		POP BC, POP BC
INT -3		3 X POP BC
INT 1		DEC SP, DEC SP
INT 2		4 X DEC SP

INT M	LD HL,[-M*2]
	ADD HL,SP
	LD SP,HL

5.2) OPTIMIZING FOR MINIMUM MEMORY USE:

The same code as above is produced except as follows:

mnemonic	operation	Z80 code
LIT Ø,N	load small positive	RST 4; RESTART 4
(Ø <=N< 256)	literal onto stack	DEFB N
LOD Ø,N	load variable	RST5;
(-64<N<64)	with small offset at this level	RESTART 5 DEFB -2*N
STO Ø,N	store variable with small off-	RST 6; RESTART 6
(-64<N<64)	small offset at this level	DEFB -2*N
LOD 1,N	store variable	RST 7;
(Ø<=N<128)	with small positive offset at one level higher	RESTART 7 DEFB-2*N
STO 1,N	store variable with small positive offset at one level higher	RST 1; RESTART 1 DEFB-2*N

NOTE: In order to make the best use of the minimum-memory option, the programmer may use the following techniques:

1) Do not declare procedures within procedures. All procedures should be declared at the outermost block level. (This rule will also make programs run slightly faster, and is quite sensible from a human point of view, as well as being compatible with the single level of the $INCL compiler option. Usually there is no need to declare procedures and functions at any other than the outermost block level.

2) Declare all single variables before declaring any array variables. This will generally ensure that all variables have an offset of less than 64 stack locations from the base and therefore allow the translator to make use of the "small" option. The size of the offset of array variables does not matter.

PROGRAM DOCUMENTATION

RUN-TIME SYSTEM

1) INTRODUCTION:

The People's Pascal run-time system provides subroutines which are called by translated People's Pascal programs.

Subroutines are provided for such functions as multiply and divide, keyboard input, etc.

Code for these functions could be inserted "in-line" into the program by the translator, but then People's Pascal programs would be very large. In general, the factor which decides whether a given function should be performed in-line or as a subroutine, is the size of the code required to perform the function. The larger the code is, the more economical it is to have only one copy of it as a subroutine, and the less the proportional overhead in execution time of the actual subroutine call and return instructions against the code executed to perform the function.

The run-time system is entirely self contained apart from two Level-II Basic routines which are used to input a character from the keyboard and to output a character to the screen. To convert to computer such as the Sorcerer, it should only be necessary to provide the equivalent of these two routines.

As well as providing subroutines, the run-time system is entered initially when a People's Pascal program is run. Certain initialization functions are performed before control is passed to the program.

2) THE JUMP TABLE:

Most subroutines within the run-time system are accessed via a jump table included in the run-time system.

This allows modification of subroutine locations within the system without modifying the addresses of the subroutine entry points.

This also allows modifications to the run-time system without modifying the translator program or previously-translated programs, providing of course that the jump table itself is not moved. Also, subroutine entry points within the jump table are at a constant offset from the starting address of the jump table. Thus if ever the jump table is moved, (reassembled with a different origin), then the only parameter to be changed in the translator is the address of the start of the jump table (JMPTAB).

3) RESTART (RST) INSTRUCTIONS:

An exception to the use of the jump table is the use of RST instructions in People's Pascal programs that have been translated with the minimum-memory usage optimization option. For certain common functions, the restart (RST) instructions (1-byte subroutine calls to fixed low-memory addresses) are used, as follows:

LIT 0,N (0<=N<256)	RST 4, DEFB N
LOD 0,N (-64<N<64)	RST 5, DEFB -2*N
STO 0,N (-64<N<64)	RSR 6, DEFB -2*N
LOD 1,N (0<=N<128)	RST 7, DEFB -2*N
STO 1,N (0<=N<128)	RST 1, DEFB -2*N

Use of the RST instructions is made possible by the flexible approach taken by Microsoft in designing Level-II Basic.

RST instructions jump to low memory (ROM) addresses, but at these locations, Microsoft has put jump instructions out into RAM locations 4000 hex onwards for RST 1 to RST 7 (RST 0 is not used in this way). These locations at 4000 hex are set to jump back into ROM, or perform other functions when the memory size question is answered.

On initialization, the People's Pascal run-time system overwrites these locations at 4000 hex with the addresses of the relevant subroutines itself. These addresses are restored when the Level-II keyboard/screen I/O routines are called. This feature is not used by programs optimized for speed.

4) PROGRAM VARIABLES AND CONSTANTS:

STK - Stack location used by PPRUN during initialization.
CR - Carriage return code.
KBUFL - Number of characters in keyboard buffer.
KBUFP - Pointer to next character in keyboard buffer.
KBUF - Keyboard buffer area (max 64 characters).
RST - Area containing restart table overwrite data. This is copied to 4000 hex on initialization and after keyboard I/O.
NORST - Area containing copy of Level-II Basic version of restart table. This is copied to 4000 hex on keyboard I/O.
K1Ø - Table of powers-of-ten for binary to decimal conversion for CSP3 (write #).

5) REGISTER USAGE:

SP - Used for People's Pascal stack pointer (T). It is also used for subroutine return linkage.
HL - Generally used as an argument register. It is used in code called by RST instructions to hold addresses of trailing arguments.
DE - General purpose.
BC - General purpose.
A - Used to hold relative level offset when not Ø. Also general purpose.
IX - Used for People's Pascal base register (B).
IY - Frequently used to save subroutine return address popped from stack at start of subroutine and jumped to at end.
Alternate register set - Used in CSP1 only.

6) PROGRAM ROUTINES:

START - Normal initialization entry point.
INAD - Alternate entry point — allows override of stack address and entry of non-standard program start address.
DORST - Overwrites Level-II Basic restart table at 4000 hex.
UNDO - Overwrites 4000 hex with original Level-II contents. Note NORST must be in HL.
LITB - Small literal — only for minimum-memory translation option.
CALA - Absolute memory address call (CALL(MEM)). Note IX register is saved. Any other register can be overwritten, so programmer does not need to worry about destroying register values in his subroutine.
CAL1 - Call procedure at non-zero level offset.
CAL - Call procedure at Ø level offset (CAL Ø,N).
OPRØ - Subroutine return.
OPR1 - Negate top of stack (TOS).
OPR2 - Add — not currently used — in line instead.
OPR3 - Subtract.
OPR4 - 16-bit signed multiply.
OPR5 - 16-Bit signed divide.
OPR6 - Test TOS for odd value.
OPR7 - MOD (uses divide, multiply and subtract).
OPR8 - Compare equal.
OPR9 - Compare not equal.
OPR10 - Compare less-than.
OPR11 - Compare greater-than or equal.
OPR12 - Compare greater than.
OPR13 Compare less-than or equal.
OPR14 - OR operation.
OPR15 - AND operation.
OPR16 - NOT operation.
OPR17 - Shift left operation.
OPR18 - Shift right operation.
OPR21 - Not used — in line instead.
KBIN - Routine to input a line of characters from the keyboard, echoing them to the screen and allowing the delete key to operate if required.
CSPØ - Input a character. Calls KBIN to get next character out of input line.
CSP1 - Output a character. Also resets KBIN input line pointer and length to zero so next call to CSPØ will cause a new read.
CSP2 - Read a number. Calls KBIN to get characters of number. Number is terminated by first non-digit character.
CSP3 - Write a number.
CSP8 - Output a string of characters. These are supplied in form of a trailing argument terminated by a null (Ø) byte. Also clears KBIN input line length and pointer, so next read will cause true input to be done.
M8 - Unsigned 16-bit-by-8-bit multiply.
NEGHL - Negate the HL register (internal subroutine only).
LODA - Load from absolute memory address (:=MEM(X)).
LOD1B - Load from small offset at previous level (small option only).
LOD1 - Load, level <> Ø.
LODB - Load from small offset at current level (small option only).
LOD - Load, Level=Ø.
LODX1 - Indexed (array) load, level <> Ø.
LODX - Indexed load, level=Ø.
BASE - Find base register value corresponding to level offset supplied in A register and return base value in BC register.
STO1B - Store to small offset, Level=1 (small option only).
STO1 - Store, level <> Ø.
STOB - Store to small offset, level =Ø.
STO - Store, Level =Ø.
STOX1 - Store indexed (array), level <> Ø.
STOX - Store indexed, level =Ø.
JMPTAB - Jump table.

7) SPECIAL SUBROUTINES:

The following two subroutines are used in the Level-II Basic ROM:

ØØ33 hex - Output character in A-register to screen.
ØØ2B hex - Try for character from keyboard. A-register will have character if there was one, otherwise A-register will be zero. This is called in a loop until A <> Ø.

These are the only external facilities used by the run-time system, and equivalent routines would need to be supplied in their stead for a different micro system. Also, some modification would probably need to be made to the restart system for minimum-memory optimization if this feature was to be retained under a different system.

PROGRAMING IN PASCAL

CONTINUED FROM PAGE (6)

```
TYPE LETTER = A... Z;
TYPE WINTERTERM = JAN... MAR;
VAR SCORE:0...100;
```

ARRAY TYPES:

TYPE typename = ARRAY [index-type] OF element-type;
VAR varname-list : ARRAY[indextype] OF element-type;

Examples:

```
TYPE COEFFICIENTS = ARRAY [0...4]
  OF REAL;
VAR SAMPLELIST = ARRAY[0...100]
  OF REAL;
```

Note: INTEGER and REAL are not permitted as index types.

Multidimensional arrays are defined by specifying multiple index-types.

TYPE typename = ARRAY[index-type, index-type,...] OF element-type;

Examples:

```
TYPE SIMLINEQS &ARRAY[0...5,0...6]
  OF REAL;
VAR FOURSPACE : ARRAY[0...10,0...
  10,0...10,0...10] OF INTEGER;
VAR NAMELIST : ARRAY]1...100,1...
  30] OF CHAR;
```

Packed arrays are almost identical to normal arrays, except that by declaring an array to be packed, it may be possible to reduce the size of the memory space used by it.

The amount of reduction depends upon the machine and the implementation, and may in fact be nil.

This may also reduce the running speed of the program.

TYPE typename = PACKED ARRAY [index-type-list] OF element-type;
VAR varname-list = PACKED ARRAY [index-type-list] OF element-type;

Example:

```
VAR FOURSPACE : PACKED ARRAY
  [0...10,0...10,0...10,0...10] OF
  INTEGER;
```

Elements in arrays are referenced by placing the index expression(s) between square brackets associated with the array name. (Since TRS-80 does not have square brackets, Pipe Dream Software utilized the /(and)/ signs. Ed)

array-name[index-expression-list]
(or in People's Pascal: array-name /(index-expression-list)/)

Examples:

```
A[1,5] FOURSPACE[X,Y,Z,T] LIST[N+1]
or in Peiple's Pascal:
A/(1,5)/ FOURSPACE/(X,Y,Z,T)/ LIST
  /(N+1)/
```

CONCLUSION:

Pascal is a relatively new and powerful general-purpose programming language. It is also one of the first languages to employ many of the principles of structured programming.

As a result of this, programs written in Pascal are usually more straightforward and considerably more readable than those written in most other contemporary languages.

Since its introduction, Pascal has seen an amazing rise in popularity throughout the world. This fact is well evidenced by the number of colleges and universities whose computer science departments in the past few years have switched their emphasis from Fortran or Basic to Pascal.

Educators are discovering that Pascal is an excellent introductory language, since it is not only easy to learn, but also teaches good programming habits right from the beginning.

Pascal is certainly not the utopia of programming languages — it is far from perfect. However, it provides a significant improvement, in general purpose computing, over most of those older languages listed earlier, thus it would seem to be the next logical rung on an endless ladder reaching towards a perfect language.

REFERENCES:

Kathleen Jensen and Niklaus Wirth, "Pascal User Manual and Report", New York: G. Michael Schneider, Steven W. Weingart, and David M. Perlman, "An Introduction to Programming and Problem Solving with Pascal", New York: John Wiley + Sons, 1977.

Anthony Ralston (ed.), and Chester L. Meek (asst. ed.), "Encyclopedia of Computer Science," New York: Petrocelli/Charter, 1976.

Andy Mikel (ed.), "Pascal News" #9/10, Minneapolis, MN: Pascal User's Group, 1977.

Richard Conway, David Gries, and E. C. Zimmerman, "A Primer on Pascal", Cambridge, MA: Winthrop, 1976.

Kenneth L. Bowles, "Problem Solving Using Pascal", New York: Springer-Verlag, 1977.

APPENDIX: PASCAL USER'S GROUP

The Pascal User's Group is in its third year (4th now--ED), and already boasts a world-wide membership with branch offices in Europe and Australia.

The group is based at the University of Minnesota, under the direction of Andy Mickel.

The main function of the User's Group is to promote the use of Pascal, by providing an open forum for members, in the form of the quarterly-published "Pascal News". The content of Pascal News is determined by the motto "All the news that fits, we print."

Membership/subscription dues are $12 per academic year. To join, or get more information, it's best to just join, and then send in a letter for publication) write to:

Pascal User's Group, c/o Andy Mickel
University Computer Center: 227 EX
208 SE Union st.
University of Minnesota
Minneapolis MN 55455

When joining, send along your $12, your name, address, phone number, type(s) of computers you are using (especially if one or more has a Pascal implementation), and be sure to date your letter.

Or, if you know someone who already gets Pascal News, just copy the all-purpose coupon from one of the issues, and send that in.

SYSTEM RESALE NOT ALLOWED

Sale of CIE People's Pascal I or II does not include permission to resell the run-time systems. Only Pipe Dream Software and SuperSoft can license commercial users to distribute the run-time systems with their People's Pascal object programs.

Writing programs in People's Pascal I or II is a good idea, since the object programs require less memory and run five to eight-times faster than Basic programs.

People wishing to sell programs written on the People's Pascal development systems, however, should enquire about run-time-system licensing from Pipe Dream Software (Tape 3), or SuperSoft (Tape 6).

Addresses are:

PIPE DREAM SOFTWARE
28 Palmerston st.
Berwick, Victoria 3806
Australia

SUPERSOFT
Box 1628
Champaign IL 61820

EDITOR/COMPILER

OPERATING INSTRUCTIONS

1. INTRODUCTION:

The People's Pascal editor is a line-oriented editor. Edit commands operate on lines of text in a text buffer, which has room for just over 3,000 characters or 50-200 lines of text, depending on line length. Intra-line editing is not provided.

Lines of text in the text buffer may be:

- inserted from keyboard or cassette files,
- replaced from keyed-in or tape files,
- deleted,
- renumbered,
- written to a cassette file,
- listed on the screen,
- printed on the lineprinter,
- compiled.

Files of any length may be created or edited. PPEC files are not loadable via the "CLOAD" command or the "SYSTEM" command, nor are they compatible with the Tandy editor/assembler. However, they may easily be read by a Level-II Basic program. PPEC cassette files are blocked for efficiency.

2. LINE NUMBERS:

In the line-oriented PP editor, lines of text are identified by line numbers. Lines always occur in line-number sequence both in the text buffer and in cassette files. Line numbers may range from 1 to 32,766.

Many of the editor commands operate on text lines having line numbers falling within a line number range. A line number range is expressed as a starting line number followed by a single dash (i.e. "-") character, followed by a final line number (e.g. 500-1000).

The following variations of this form are allowed:

A. No final line number, (e.g. 500). In this case, the final line number will default to the value of the starting line number, and the command will operate on that line only.

B. No starting or final line number. In this case, the starting line number will default to 1, the final line number will default to the highest line number allowed (32,766), and the command will operate on all lines in the text buffer.

C. Starting line number replaced by a full stop (full point or ".") character, (e.g. -500). In this case, the starting line number will take the value of the current line number, and the command will operate from the current line to the final line number.

D. Missing final line number, but a dash character present (e.g. 100- or .-). In this case the final line number will default to the largest line number allowed, and the command will operate on lines from the starting line number to the end of the buffer.

3. COMMANDS:

People's Pascal editor commands consist of a single letter, possibly followed by a line number range or other numeric argument.

The following commands are accepted:

C — COMPILE — Compile People's Pascal program in the text buffer.

D — DELETE — Delete line(s) from the text buffer.

E — EOF — Write an end-of-file mark to the output-cassette file.

F — FREE — Enquire how many bytes "free" (available) in the text buffer.

L — LIST — List line(s) in the text buffer on the screen.

N — NUMBER — Re-number lines in the text buffer.

P — PRINT — Print lin(s) in the text buffer on the line printer.

R — READ — Read block(s) from the input cassette file, and insert or replace lines in the text buffer.

W — WRITE — Write line(s) from the text buffer to the output cassette file.

Commands must be typed precisely. Leading spaces are not allowed. Embedded spaces are not allowed between command mnemonics ("C", "D", "E", etc.) and the numeric arguments. An unrecognized command will cause a "??" message from the editor.

4. INSERTING AND REPLACING LINES:

NOTE: The current version of the editor requires that any lines containing the characters "," or ":" be preceded by the quote sign (') , otherwise the Level-II Basic I/O will truncate the line at the comma or colon, and emit this message: "EXTRA IGNORED". It is good practice to precede every line containing source code (text) by a quote sing. The quote is "thrown away" by the Level-II Basic input routine. The editor lists lines in alignment with lines typed in this fashion, for the consistant appearance if People's Pascal block-structure indentation of the on the screen. It is not necessary to precede command lines by a quote, since these never contain a comma or a colon.

A line to be inserted into the text buffer is typed preceded by its line number. If there was already a line in the buffer with this number, then the new line will replace the old line. Otherwise the new line will be inserted into the buffer in position according to its line number.

5. DELETING LINES - D:

To delete a single line, simply type its line number. This line will be deleted from the buffer. To delete several lines, the D command can be used. D alone will delete line 100 (same effect as just typing delete lene 100 (same effect as fust typing 100). D100-500 will delete lines 100 to 500 inclusive. Some delay may be noticed when many lines are deleted at once.

6. LISTING LINES-L:

The L command is used to list lines in the text buffer on the screen. Just L will list all the lines in the buffer. L100 will list line 100. L100-500 will list lines 100 to 500 inclusive. "L." (without quotes) will list the current line. L.- will list from the current line to the end of the text buffer. Note that the ENTER key will cause the operation of the list command to be terminated, with the current line being the last line listed (i.e. L.- will continue the listing again from the point where it was terminated by hitting ENTER.

7. RENUMBERING LINES - N:

The N command is used to assign new line numbers to lines in the text buffer. This can be useful if it is desired to insert text from a cassette file into a given location. Lines can also be moved by saving them on cassette, deleting them, renumbering the remaining lines, and then reading back in the original lines from cassette

Lines are renumbered starting from a base number until the end of the buffer. Just N will cause all lines in the buffer to be renumbered. N500 will cause lines 500 onwards to be renumbered. When the N command has been entered, the editor will ask for the new base and increment. Lines will be renumbered starting from this base with this increment. The new base must be greater than the line number of the line below the lines being renumbered.

8. READING TEXT FROM CASSETTE-R:

Lines on PPEC cassette files are stored as variable-length records in variable-length blocks of up to 240 characters. The R command will cause the bext block(s) on the cassette to be read into the text buffer in position according to their line numbers. If there is already a line in the text buffer with the same number as a line being read from cassette, then the old line will be replaced. Just R will read in the next block. R100 will read in the next 100 blocks, or stop at the next end-of-file mark, or until the text buffer becomes full, whichever occours first.

9. WRITING LINES TO A CASSETTE FILE - W:

The W command will cause lines to be written to cassette. Just W will cause all the lines in the buffer to be written to the cassette in blocks. W100-500 will cause lines 100 to 500 to be written to cassette. Lines are written as variable-length records in variable-length blocks of up to 240 characters. A "short" block may be written as the last block of a line number range. In addition, a line starting with a "$" sign will always be the last line in its block (Refer to "$INCL" in compiler documentation).

On completion of the write command, the editor will ask whether the lines written out to cassette should be deleted from the text buffer ("DELETE?"). A reply of "Y" will cause these lines to be deleted from the text buffer. Any other reply will leave these lines unchanged in the text buffer. This option is provided for the editing of files that are too big to all fit in the edit buffer at once.

If the write command was a write to the end of the text buffer, then TPEX will prompt with "EDF?" on completion of the write. If the reply to this prompt is "Y" then an end-of-file mark will be written to the file at this point (same effect as E command).

10. COMPILE — C:

The C command will pass control from the People's Pascal editor to the compiler, which will attempt to compile lines from the text buffer. If it is required to compile a file from cassette, then it will be necessary to insert a line such as "100$INCL FRED" into the text buffer before invoking the compiler. Before compilation commences, the compiler prompts with an "LP?". If the reply is "Y", then the listing output will be printed on the lineprinter rather than being displayed on the screen.

The compiler then prompts with an "OBJ FILE?". If the ENTER key is pressed, then no object file will be generated, and the compile will be for syntax errors detection only. Any other reply will cause an object file to be generated. In one-cassette systems, where source input is being accepted from cassette, it will be necessary to change cassettes and cassette-drive operating modes (play/record) when the compiler prompts, between the source file cassette(s) and the object file cassette.

11. PRINT ON LINEPRINTER TEXT BUFFER LINE(s) — P:

This command is used in exactly the same way as the L command, except that the output appears on the lineprinter rather than the screen.

12. CREATING, MAINTAINING LARGE FILES:

It is possible to create and maintain cassette files of indefinite length with the People's Pascal editor, even with only one cassette drive, although large files of People's Pascal source code are not recommended (refer to compiler documentation on modular programming).

To create a large file, type lines into the text buffer until it becomes nearly full, mount an output cassette and write the contents of the buffer out with the W command, deleting the text that has been written out. Type more lines into the text buffer, with higher line numbers, and repeat the process until the complete file has been written out. Write an end-of-file mark to the output cassette.

To edit a large file, a new copy of the file is made on a fresh cassette, as follows. Mount the input cassette containing the current version of the file. Read several blocks of the file into the text buffer and edit them. Remove the input cassette and mount the chosen output cassette. Write out the edited lines from the text buffer to the output cassette. Remove the output cassette and remount the input cassette. Read in some more text from the input cassette and repeat the process. Repeat until all input text has been processed (end-of-file — or "#EOF") encountered, and all edited text has been written to the output cassette. It is advisable to keep one old version of a file, in case the most up-to-date version is lost, accidentally erased, or becomes unreadable. Alternatively, a copy of the current version may be made by using the above process without any editing.

IMPORTANT NOTE:

PPEC text files are blocked on cassette. This means that the file consists of blocks of data with "gaps" in between. It is this feature which makes many of the features of the program possible. It is expecially important that cassettes that are to be used as output files from the editor should be erased before they are used. Bulk erasure is not necessary. Simply rewind the cassette, set the recorder on manual (disconnect computer remote control) and record over the whole cassette. This can be done any time the recorder is not being used by the computer.

BULLSEYE

PEOPLE'S PASCAL SAMPLE PROGRAM

```
1070 (* BULLSEYE *)
1080 (*----------*)
1090
1100 CONST CR=13; HOME=28; CLEAR=31; WIDE=23;
1110 VAR STARS,X,Y,DX,HX,DY,HY,I,J:INTEGER;
1120   MODE,ALIVE,SEED,M:INTEGER;
1130   XSTARS,YSTARS:ARRAY(100) OF INTEGER;
1140
1150 $INCL PASLIB
1160
1170 FUNC TARGET(X,Y); VAR I:INTEGER; BEGIN
1180   WRITE(HOME,CLEAR);
1190   FOR I:=-2 TO 2 DO BEGIN
1200     SET(1,X+I,Y+I); SET(1,X+I,Y-I);
1210   END;
1220   SEED:=RND(SEED);
1230   HX:=HX+(SEED SHR 1) AND 1 - (SEED SHR 2) AND 1;
1240   HY:=HY+(SEED SHR 3) AND 1 - (SEED SHR 4) AND 1;
1250   TARGET:=((X>=0)AND(X<128)AND(Y>=0)AND(Y<48));
1260 END;
1270
1280 FUNC FIRE; VAR Q,W,E:INTEGER; BEGIN
1290   WRITE(HOME,'FIRE!');
1300   Q:=42; W:=86;
1310   FOR E:=23 DOWNTO 12 DO BEGIN
1320     SET(1,Q,E+E); SET(1,W,E+E);
1330     Q:=Q+2; W:=W-2;
1340   END;
1350   FOR E:=1 TO 100 DO;
1360   FIRE:=((X<66)AND(X>62)AND(Y<26)AND(Y>22));
1370 END;
1380
1390 FUNC ACTION; VAR CHR:INTEGER; BEGIN
1400   CHR:=MEM(15359); (*KBD MEM.AREA*)
1410   IF CHR>=64 THEN BEGIN DX:=DX-2; CHR:=CHR-64 END;
1420   IF CHR>=32 THEN BEGIN DX:=DX+2; CHR:=CHR-32 END;
1430   IF CHR>=16 THEN BEGIN DY:=DY-1; CHR:=CHR-16 END;
1440   IF CHR>=8 THEN BEGIN DY:=DY+1; CHR:=CHR-8 END;
1450   ACTION:=1; IF CHR>0 THEN ACTION:=NOT (FIRE);
1460 END;
1470
1480 PROC CHANGE; VAR I,TMPX,TMPY:INTEGER; BEGIN
1490   X:=X+DX+HX;
1500   Y:=(Y SHL 1 +DY+HY) SHR 1;
1510   FOR I:=1 TO STARS DO BEGIN
1520     TMPX:=XSTARS(I); TMPY:=YSTARS(I);
1530     TMPX:=((TMPX SHL 4)+TMPX-63-DX) SHR 4;
1540     TMPY:=((TMPY SHL 4)+TMPY-23-DY) SHR 4;
1550     IF((TMPX<0)OR(TMPY<0)OR(TMPX>127)OR(TMPY>47)) THEN
1560     BEGIN
1570       SEED:=RND(SEED); TMPX:=SEED AND %007F;
1580       SEED:=RND(SEED); TMPY:=SEED AND %003F;
1590     END;
1600     YSTARS(I):=TMPY;
1610     XSTARS(I):=TMPX;
1620   END;
1630   IF((XSTARS(1)<30) OR (XSTARS(1)>110)) THEN BEGIN
1640     XSTARS(2):=XSTARS(1)+1;
1650     YSTARS(2):=YSTARS(1);
1660   END;
1670 END;
1680
1690 PROC SIGHTS; BEGIN
1700   SET(1,64,23); SET(1,64,25);
1710   SET(1,63,24); SET(1,65,24);
1720 END;
1730
1740 BEGIN
1750  WRITE(HOME,CLEAR,'MODE (A,D)? '); READ(MODE);
1760  WHILE 1 DO BEGIN (* LOOP FOREVER *)
1770   DX:=0; DY:=0; HX:=0; HY:=0;
1780   ALIVE:=1;
```

 copyright april 1979, Pipe Dream Software:

PEOPLE'S PASCAL TRANSLATOR

```
1 POKE16553,255:CLS:PRINT"TRS-80 PEOPLE'S PASCAL TRANSLATOR V2.1":
  PRINT"COPYRIGHT (C) PIPE DREAM SOFTWARE APRIL 1979":PRINT
2 IFPEEK(16598)+PEEK(16599)*256+2<>23000THENPRINT"WARNING - MEMORY
  SIZE SHOULD BE 23000 FOR NORMAL USE":STOP
3 PRINT"DEFAULT REPLIES TO PROMPTS ARE SHOWN IN BRACKETS:":PRINT
5 CLEAR(550):DEFINTA-Z:Y=32766
7 X=24000:A$="PCODE START ADDRESS":GOSUB27
9 X=23000:A$="Z80 CODE START ADDRESS":GOSUB27
11 X=22999:A$="Z80 STACK ADDRESS (GROWS DOWN)":GOSUB27
13 IN$="F":INPUT"OPTIMISATION: F=FAST,S=SMALL (F)";IN$:IFIN$="S"THEN
  OP=1
15 X=OP:GOSUB29
17 IN$="Y":INPUT"DISPLAY CODES (Y)";IN$:IFIN$="Y"THENDI=1
19 X=DI:GOSUB29
21 IN$="N":INPUT"PRINT CODES (N)";IN$:IFIN$="Y"THENLP=1
23 X=LP:GOSUB29
25 PRINT"MOUNT PCODE INPUT FILE ON CAS1 AND TYPE 'RUN'":DELETE1-29
27 PRINTA$" ("X")";:INPUTX
29 Y=Y-2:I=VARPTR(X):POKEY,PEEK(I):POKEY+1,PEEK(I+1):RETURN
31 Y=32766:GOSUB39:PS=X:GOSUB233:PP=PS
33 GOSUB219:IFP1<>253ANDP1<>254THENPOKEPP,P1:PP=PP+1:POKEPP,P2:
  PP=PP+1:POKEPP,P4:PP=PP+1:POKEPP,P3:PP=PP+1ELSEGOSUB237
35 IFP1<>255THEN33ELSE41
37 Z8$=Z8$+CHR$(229):RETURN
39 Y=Y-2:X=PEEK(Y)+PEEK(Y+1)*256:RETURN
41 CLEAR(330):DEFINTA-Z:DIMPA(190),ZA(190):Y=32766:GOSUB39:PS=X:
  GOSUB39:ZS=X:GOSUB39:SP=X:GOSUB39:OP=X:GOSUB39:DI=X:GOSUB39:LP=X
43 JT=18818:CO$="LITOPRLODSTOCALINTJMPJPCCSP":ZS=ZS+12
45 AS=1:PP=PS:ZP=ZS:PRINT"PASS 1":GOSUB59:GOSUB69
47 AS=2:PP=PS:ZP=ZS:PRINT"PASS 2":GOSUB85
49 ZS=ZS-12:LT$="":AS=3:PP=PS:ZP=ZS
51 XX=SP:GOSUB173:Z8$=CHR$(221)+CHR$(33)+CHR$(XL)+CHR$(XH)+CHR$(221)+
  CHR$(249):XX=JT+120:GOSUB173:Z8$=Z8$+CHR$(33)+CHR$(XL)+CHR$(XH)+
  CHR$(229)+CHR$(229)+CHR$(229):GOSUB199:ZP=ZP+LEN(Z8$)
53 GOSUB97:Z8$=CHR$(195)+CHR$(0)+CHR$(0):GOSUB199
55 A$="-CODE SIZE=":ZP=ZP+3:PRINT:PRINT"P"A$;PP-PS:PRINT"Z"A$;ZP-ZS:
  PRINT"RATIO="(ZP-ZS)/(PP-PS):PRINT"FIRST ADDR="ZS;:X=ZS:GOSUB57:
  PRINT"LAST ADDR="ZP;:X=ZP:GOSUB57:END
57 Y=VARPTR(X):BY=PEEK(Y+1):GOSUB209:PRINT"(HEX "HX$;:BY=PEEK(Y):
  GOSUB209:PRINTHX$")":RETURN
59 PC=PP:GOSUB73
61 IFP1=255THENRETURN
63 IFP1=6ORP1=7ORP1=4THENPA=PC:GOSUB75
65 PP=PP+4
67 GOTO59
69 FORI=1TOAN-1:FORJ=2TOAN-I+1:IFPA(J)<PA(J-1)THENK=PA(J):
  PA(J)=PA(J-1):PA(J-1)=K ELSE IFPA(J)=PA(J-1)THEN PA(J-1)=0:
71 NEXTJ:NEXTI:RETURN
73 P1=PEEK(PC):P2=PEEK(PC+1):P4=PEEK(PC+2):P3=PEEK(PC+3):GOTO77
75 AN=AN+1:PA(AN)=P5:RETURN
77 N!=256*P3+P4:IFN!>32767THENP5=N!-65536:RETURNELSEP5=N!:RETURN
79 IFP1>=14OR(LP=0ANDDI=0)THENRETURNELSEA$=STR$((PP-PS)/4)+" "+MID$
  (CO$,3*P1+1,3):IFX=1THENA$=A$+"X"
81 A$=A$+STR$(P2)+" "+STR$(P5):IFDI>0THENPRINTA$;
83 IFLP>0THENLPRINTA$;:RETURNELSERETURN
85 CL=0:AP=1
87 ZP=ZP+CL:PC=PP:GOSUB73
89 IFP1=255THENRETURNELSEGOSUB105:CL=LEN(Z8$)
91 IF(PC-PS)/4>PA(AP)THENAP=AP+1:IFAP<=ANTHENGOTO91
93 IF(PC-PS)/4=PA(AP)THENZA(AP)=ZP:AP=AP+1
95 PP=PP+4:IFAP<=ANTHEN87ELSERETURN
97 PC=PP:GOSUB73:IFP1=255THENRETURN
99 GOSUB105:GOSUB79:GOSUB103
101 PP=PP+4:GOTO97
103 GOSUB199:ZP=ZP+LEN(Z8$):RETURN
105 Z8$="":X=0:IFP1>8THENX=1:P1=P1-16
107 IFNOT(P1=0 OR (P1=8 AND P4=8))THENLT$=""
109 IFP1>=0ANDP1<=8ON P1+1GOTO113,115,125,135,143,149,157,159,165
111 PRINT"BAD PCODE":STOP:RETURN
113 LT$=LT$+CHR$(P4):IFOP=1ANDP5>=0ANDP5<256THENZ8$=CHR$(231)+
  CHR$(P4):RETURNELSEGOSUB181:GOTO37
115 IFP4<>2ANDP4<>19ANDP4<>20ANDP4<>21THENXX=JT+(20+P4)*3:GOTO175
117 IFP4=20THENZ8$=CHR$(225)+CHR$(43):GOTO37
119 IFP4=19THENZ8$=CHR$(225)+CHR$(35):GOTO37
121 IFP4=2THENZ8$=CHR$(225)+CHR$(209)+CHR$(25):GOTO37
123 Z8$=CHR$(225):GOSUB37:GOTO37
125 IFP2=255THENXX=JT+39*3:GOTO175ELSERT=X+X
127 IFP2=0ANDX=0ANDP5>-64ANDP5<64THENIFOP=1THENGOSUB187:
  Z8$=CHR$(239)+CHR$(P7):RETURNELSE183
129 IFP2=1ANDX=0ANDP5>=0ANDP5<128ANDOP=1THENZ8$=CHR$(207)+
  CHR$(256-P5-P5):RETURN
131 GOSUB177:IFP2<>0THENGOSUB189:RT=RT+1
133 XX=JT+RT*3:GOTO175
135 IFP2=255THENZ8$=CHR$(209)+CHR$(225)+CHR$(115):RETURN
137 IFP2=0ANDX=0ANDP5>-64ANDP5<64THENIFOP=1THENGOSUB187:
  Z8$=CHR$(247)+CHR$(P7):RETURNELSE185
139 IFP2=1ANDX=0ANDP5>=0ANDP5<128ANDOP=1THENZ8$=CHR$(255)+
  CHR$(256-P5-P5):RETURN
141 RT=4+X+X:GOTO131
143 IFP2=255THENXX=JT+19*3:GOSUB175:RETURN
145 RT=8:IFP2<>0THENGOSUB189:RT=RT+1
147 XX=JT+RT*3:GOSUB175:GOSUB193:GOSUB191:RETURN
149 IFP5=0THENRETURNELSEIFP5>2ORP5<-3THEN155
151 Z8$="":IFP5>0THENFORXL=1TOP5:Z8$=Z8$+CHR$(59)+CHR$(59):NEXT:RETURN
153 FORXL=1TO-P5:Z8$=Z8$+CHR$(193):NEXT:RETURN
155 XX=-P5-P5:GOSUB173:Z8$=CHR$(33)+CHR$(XL)+CHR$(XH)+CHR$(57)+
  CHR$(249):RETURN
157 IFP5<>(PP-PS)/4+1THENGOSUB193:GOSUB191:RETURNELSERETURN
159 Z8$=CHR$(241):RT=210:IFP2>0RT=218
161 GOSUB193:Z8$=Z8$+CHR$(RT)+CHR$(XL)+CHR$(XH)
163 RETURN
165 IFP4=8THENLN=ASC(RIGHT$(LT$,1))+1:IFOP=1THENZP=ZP-2*LN
  ELSEZP=ZP-4*LN
167 XX=JT+(10+P4)*3:GOSUB175
169 IFP4=8THENZ8$=Z8$+LEFT$(LT$,LN-1)+CHR$(0)
171 RETURN
173 XH=VARPTR(XX):XL=PEEK(XH):XH=PEEK(XH+1):RETURN
175 GOSUB173:Z8$=Z8$+CHR$(205)+CHR$(XL)+CHR$(XH):RETURN
177 P6=-P5*2:P8=INT(P6/256):P7=P6-P8*256:IFP8<0THENP8=256+P8
179 Z8$=CHR$(33)+CHR$(P7)+CHR$(P8):RETURN
181 Z8$=CHR$(33)+CHR$(P4)+CHR$(P3):RETURN
183 GOSUB187:Z8$=CHR$(221)+CHR$(110)+CHR$(P7)+CHR$(221)+CHR$(102)+
  CHR$(P7+1):GOTO37
185 GOSUB187:Z8$=CHR$(225)+CHR$(221)+CHR$(117)+CHR$(P7)+CHR$(221)+
  CHR$(116)+CHR$(P7+1):RETURN
187 P7=-P5*2:IFP7<0THENP7=256+P7:RETURNELSERETURN
189 Z8$=Z8$+CHR$(62)+CHR$(P2):RETURN
191 Z8$=Z8$+CHR$(195)+CHR$(XL)+CHR$(XH):RETURN
193 GOSUB195:XX=ZA(IX):GOSUB173:RETURN
195 FORIX=1TOAN:IFPA(IX)<>P5NEXT
197 IFPA(IX)<>P5THENIX=0:RETURNELSERETURN
199 A$="":X=LEN(Z8$):IFX>0THENIF(DI=1ORLP=1)THENA$=STR$(ZP)+" ":
  BY=PEEK(VARPTR(ZP)+1):GOSUB209:A$=A$+HX$:BY=PEEK(VARPTR(ZP)):
  GOSUB209:A$=A$+HX$+" ":FORI=1TOX:GOSUB207:GOSUB209:A$=A$+HX$:
  NEXT:ELSEFORI=1TOLEN(Z8$):GOSUB207:NEXT
201 IFDI>0THENPRINTTAB(20);A$
203 IFLP>0THENLPRINTTAB(20);A$
205 RETURN
207 BY=ASC(MID$(Z8$,I,1)):POKE(ZP+I-1),BY:RETURN
209 HX$="":BH=INT(BY/16):HX=BH:GOSUB211:HX=BY-BH*16:GOSUB211:RETURN
211 HB=ASC("0"):IFHX>9THENHB=ASC("A"):HX=HX-10
213 HX$=HX$+CHR$(HB+HX):RETURN
215 POKEPC,P1:POKEPC+1,P2:POKEPC+2,P3:POKEPC+3,P4:RETURN
217 RETURN
219 IFIP> LEN(IB$)THENGOSUB239
221 P1=ASC(MID$(IB$,IP,1)):IP=IP+1
223 P2=ASC(MID$(IB$,IP,1)):IP=IP+1
225 P4=ASC(MID$(IB$,IP,1)):IP=IP+1
227 P3=ASC(MID$(IB$,IP,1)):IP=IP+1
229 RETURN
231 DATA 62,0,33,230,65,205,18,2,205,150,2,205,53,2,71,119,35,205,53,
  2,119,16,249,205,248,1,201,-1
233 Z9$="":RESTORE
235 READX:IFX>=0THENZ9$=Z9$+CHR$(X):GOTO235ELSEIP=1:RETURN
237 IFP1=253THENM!=P3*256+P4:RETURNELSEPOKEM!*4+PS+2,P4:POKEM!*4+PS+3,P
  PRINT"ADD"P3*256+P4"AT"M!:RETURN
239 X=VARPTR(Z9$):POKE16526,PEEK(X+1):POKE16527,PEEK(X+2):X=USR(0):
  LN=PEEK(16870):IB$=STRING$(LN,"*"):J=VARPTR(IB$)+1:J=PEEK(J)+
  PEEK(J+1)*256:FORI=1TOLN:POKEJ+I-1,PEEK(16870+I):NEXT:IP=1:RETURN
```

PEOPLE'S PASCAL INTERPRETER

```
1 GOTO9900
100 SZ=600:S1=SZ-20:DIMS(SZ):M$="LITOPRLODSTOCALINTJMPJPCCSP"
1000 CLS:PRINT"TRS-80 PEOPLE'S PASCAL INTERPRETER (PPINT)":
     PRINT"COPYRIGHT (C) 1979, PIPE DREAM SOFTWARE":PRINT
1002 PS=24000:PRINT"PCODE ADDRESS ("PS")";:INPUTPS:PP=PS
1010 Z$="Y":PRINT"READ IN PCODES ("Z$")";:INPUTZ$:IFZ$="N"THENRETURN
     ELSEINPUT"MOUNT PCODE INPUT FILE ON CAS1";Z$
1012 GOSUB30080
1013 GOSUB30010:IFP1<>253ANDP1<>254THENPOKEPP,P1:PP=PP+1:POKEPP,P2:
     PP=PP+1:POKEPP,P4:PP=PP+1:POKEPP,P3:PP=PP+1:ELSEGOSUB30100
1014 IFP1<>255THEN1013ELSEIB$="":RETURN
9900 CLEAR(600):DEFINTA-Z:POKE16553,255
9905 U=15:BL=5:DIMTR(U),BR(BL):GOSUB100:GOTO20820
20040 BA=B
20041 IFL<=0THENRETURNELSEBA=S(BA):L=L-1:GOTO20041
20060 U=5:A=0:Z=PS:T=0:B=1:P=0:ST=0:S(1)=0:S(2)=0:S(3)=-1:P0=0:TP=U:
      K=0:FORI=0TOU:TR(I)=-1:NEXT:RETURN
20090 X=P*4+Z:GOSUB20690:A=N!:TP=TP+1:IFTP>UTHENTP=0
20100 TR(TP)=P:P=P+1:P0=P:K=K+1:F=PEEK(X):IFF<=8THENIX=0ELSEIX=1:F=F-16
20120 ON F+1 GOTO20140,20150,20520,20540,20530,20550,20560,20570,20580
20130 PRINT"BAD OPCD":ST=1:RETURN
20140 T=T+1:S(T)=A:RETURN
20150 T=T-1:SA=S(T):SB=S(T+1):ONA+1GOTO20200,20210,20220,20230,20240,
      20250,20260,20270,20280,20290,20300,20310,20320,20330,20340,
      20350,20360,20370,20380,20390,20400,20410
20160 PRINT"BAD OPR":ST=1:RETURN
20200 T=B-1:B=S(T+2):P=S(T+3):RETURN
20210 T=T+1:S(T)=-S(T):RETURN
20220 S(T)=SA+SB:RETURN
20230 S(T)=SA-SB:RETURN
20240 N!=SA*SB:IFN!<32767THENS(T)=N!ELSEM!=INT(N!/32768):S(T)=N!-M!*
      32768:PRINT"MUL OFLO"SA"*"SB"="S(T)
20241 RETURN
20250 S(T)=INT(SA/SB):RETURN
20260 T=T+1:S(T)=SAAND1:RETURN
20270 S(T)=SA-SB*INT(SA/SB):RETURN
20280 S(T)=SA=SB:RETURN
20290 S(T)=SA<>SB:RETURN
20300 S(T)=SA<SB:RETURN
20310 S(T)=SA>=SB:RETURN
20320 S(T)=SA>SB:RETURN
20330 S(T)=SA<=SB:RETURN
20340 S(T)=SA OR SB:RETURN
20350 S(T)=SA AND SB:RETURN
20360 T=T+1:S(T)=NOT S(T):RETURN
20370 S(T)=INT(SA*INT(2[SB+.5)):RETURN
20380 S(T)=INT(SA/INT(2[SB+.5)):RETURN
20390 T=T+1:S(T)=S(T)+1:RETURN
20400 T=T+1:S(T)=S(T)-1:RETURN
20410 T=T+2:S(T)=S(T-1):RETURN
20520 L=PEEK(X+1):IFL=255THENS(T)=PEEK(S(T)):RETURNELSEIFIXTHENA=A+S(T)
20524 T=T+1-IX:GOSUB20040::S(T)=S(BA+A):RETURN
20530 L=PEEK(X+1):IFL<>255THENGOSUB20040:S(T+1)=BA:S(T+2)=B:S(T+3)=P:
      B=T+1:P=A:ELSEGOSUB30300
20532 RETURN
20540 L=PEEK(X+1):IFL=255THENPOKES(T-1),S(T):T=T-2:RETURN:ELSEIFIX
      THENA=S(T-1)+A
20542 GOSUB20040:S(BA+A)=S(T):T=T-1-IX:RETURN
20550 IFT>S1-ATHENPRINT"STACK OFLO":ST=1:RETURNELSET=T+A:RETURN
20560 P=A:RETURN
20570 IFABS(S(T))=ABS(PEEK(X+1))P=A
20572 T=T-1:RETURN
20580 ONA+1GOTO20600,20610,20620,20630,20640,20650,20660,20670,20680
20590 PRINT"BAD CSP":ST=1:RETURN
20600 T=T+1:INPUTA$:S(T)=ASC(A$):RETURN
20610 PRINTCHR$(S(T));:T=T-1:RETURN
20620 T=T+1:INPUTS(T):RETURN
20630 PRINTS(T);:T=T-1:RETURN
20640 T=T+1:INPUTA$:GOSUB23000:S(T)=H:RETURN
20650 H=S(T):GOSUB24000:PRINTA$;:T=T-1:RETURN
20660 GOTO20590
20670 GOTO20590
20680 FORIX=T-S(T)TOT-1:PRINTCHR$(S(IX));:NEXT:T=T-S(T)-1:RETURN
```

```
20690 N!=PEEK(X+3)*256+PEEK(X+2):IFN!>32767THENN!=N!-65536:RETURN
      ELSERETURN
20710 X=PC*4+Z:N=PEEK(X)*3:Z$="  ":IFN>24N=N-48:Z$="X"
20720 GOSUB20690:IFN<LEN(M$)THENPRINT;PC;"  ";MID$(M$,N+1,3);Z$;
      PEEK(X+1);",";N!;:IFN=0ANDN!<256ANDN!>32THENPRINTCHR$(N!)
      ELSEPRINT
20730 RETURN
20760 IFP<0THENST=1:RETURNELSEFORI=1TOBP:IFBR(I)=PTHENPRINT" BREAK ";:
      PC=P:GOSUB20710:ST=1
20770 NEXT:RETURN
20820 GOSUB20060:GOSUB20710:BP=0
20830 PRINT"INT>";:INPUTCM$:GOSUB20840:GOTO20830
20840 IFCM$<>"G"THEN20850
20842 ST=0:GOSUB20090:GOSUB20760:IFST=0THEN20842ELSERETURN
20850 IFCM$="S"GOSUB20090:PC=P:GOSUB20710:RETURN
20860 IFCM$<>"X"THEN20870
20862 PRINT"P="P" B="B" T="T:IFT>-1PRINT" S(T)="S(T):IFT>0PRINT
      " S(T-1)="S(T-1)
20864 PRINT:RETURN
20870 IFCM$="R"GOSUB20060:GOTO20842
20880 IFCM$<>"T"THEN20890
20882 PRINT"*TRACE*":FORI=0TOU:TP=TP+1:IFTP>UTHENTP=0
20884 IFTR(TP)>=0THENPC=TR(TP):GOSUB20710
20886 NEXT:RETURN
20890 IFCM$="K"INPUTI:FORJ=TTOT-ISTEP-1:PRINTS(J);:NEXT:PRINT:RETURN
20900 IFCM$<>"B"THENGOTO20910
20902 IFBP<8THENBP=BP+1:PRINTBP":";:INPUTBR(BP):RETURNELSERETURN
20910 IFCM$="C"BR(1)=0:BP=0:PRINT:RETURN
20920 IFCM$="Y"FORI=1TOBP:PRINTBR(I);:NEXT:PRINT:RETURN
20930 IFCM$="E"THENP0=0:INPUTP0:PC=P0:GOTO20710
20940 IFCM$="U"P0=P0-1:PC=P0:GOTO20710
20950 IFCM$="N"P0=P0+1:PC=P0:GOTO20710
20960 IFCM$="Q"THENSTOP
20970 PRINT"??":GOTO20830
30010 IFIP>LEN(IB$)THENGOSUB30200
30020 P1=ASC(MID$(IB$,IP,1)):IP=IP+1
30030 P2=ASC(MID$(IB$,IP,1)):IP=IP+1
30040 P4=ASC(MID$(IB$,IP,1)):IP=IP+1
30050 P3=ASC(MID$(IB$,IP,1)):IP=IP+1:RETURN
30070 DATA 62,0,33,230,65,205,18,2,205,150,2,205,53,2,71,119,35,205,
      53,2,119,16,249,205,248,1,201,-1
30080 Z9$="":RESTORE
30082 READX:IFX>=0THENZ9$=Z9$+CHR$(X):GOTO30082
30084 IP=1:RETURN
30100 IFP1=253THENM!=P3*256+P4:ELSEPOKEM!*4+PS+2,P4:POKEM!*4+PS+3,P3:
      PRINT"ADD"P3*256+P4"AT"M!
30120 RETURN
30200 '
30210 X=VARPTR(Z9$):POKE16526,PEEK(X+1):POKE16527,PEEK(X+2):X=USR(0):
      LN=PEEK(16870):IB$="":FORI=1TOLN:IB$=IB$+CHR$(PEEK(16870+I)):
      NEXT:IP=1:RETURN
30300 IFS(T)<>19968THENPOKE16526,S(T)-INT(S(T)/256)*256:POKE16527,
      INT(S(T)/256):ZZ=USR(0):T=T-1:RETURN
30305 T=T-3
30310 IFS(T-1)<0 OR S(T-1)>47THEN30340
30320 S(T-2)=S(T-2)-INT(S(T-2)/128)*128
30325 IFS(T-2)<0THENS(T-2)=S(T-2)+128:GOTO30325
30330 IFS(T-3)=1THENSET(S(T-2),S(T-1))ELSERESET(S(T-2),S(T-1)
30340 T=T+3:RETURN
```

:SOURCE LISTING:

copyright april 1979, Pipe Dream Software:

EDITOR ASSEMBLER

```
1 GOTO194
2 IFR=0THENRETURNELSE7
3 GOSUB10:GOTO2
4 S(S9)=X:S9=S9+1:RETURN
5 S9=S9-1:X=S(S9):RETURN
6 IFC0<LEN(L$)THENC0=C0+1:X$=MID$(L$,C0,1):T=ASC(X$):RETURNELSEGOSUB3
  IFLN<ML THEN6ELSEE=9:R="."
7 GOTO28
8 GOSUB77:GOTO103
9 IFOF$<>""THEN181ELSERETURN
10 IFT=32THENGOSUB6:GOTO10ELSE38
11 GOTO111
12 T1=T1+1:T$(T1)=A$:T0$=LEFT$(T0$,T1-1)+K$:IFK$="C"THENT2(T1)=N3:RETURN
   ELSET1(T1)=L1:IFK$="V"ANDF9THENT2(T1)=D0:D0=D0+1:RETURNELSERETURN
13 V=0:GOTO9
14 S$(P8)=Y$:P8=P8+1:RETURN
15 P8=P8-1:Y$=S$(P8):RETURN
16 GOSUB21:GOSUB8:R=")":E=34:GOTO2
17 W=T2(V):V=L1-T1(V):GOTO9
18 U=1:GOTO13
19 U=0:GOTO13
20 W=0:GOTO13
21 R="(":E=33:GOSUB3:GOTO10
22 R="(":E=31:GOTO3
23 R=")":E=22:GOTO2
24 DEFINTA-Z:DEFSTRO,R:N1=32767:T0=50:DIMT$(T0),S(50),S$(20),T1(T0),
   T2(T0),T3(T0):M$="LITOPRLODSTOCALINTJMPJPCCSP":RETURN
25 W0$="AND ARRAYBEGINCALL CASE CONST DIV DO DOWNTELSE END FOR FUNC
   IF INTEGMEM MOD NOT OF OR PROC READ REPEASHL SHR THEN TO
   TYPE UNTILVAR WHILEWRITE"
26 O="":INPUT"LP";O:LP=NOT(O=""ORO="N"):OF$="":INPUT"OBJ FILE";OF$:
   OB$=STRING$(240,"*"):BZ=0:P=SA:N0=32:N2=8:I$="IDENT":L$=""
27 T=32:GOSUB10:GOSUB159:R=".":E=9:GOSUB2:U=255:GOSUB20:IFOF$=""ORBZ=0
   THENRUNELSEGOSUB191:RUN
28 PRINTUSING"#####";LN;:PRINT" "L$:PRINTTAB(C0+4)"[ ERROR: ";:
   X$=" EXPECTED":O=" MISSING":L$="ILLEGAL "
29 IFE=2THENPRINT"CONST"X$ELSEIFE=4THENPRINTI$;X$ELSEIFE=5THENPRINT"'
   ' OR ':'"OELSEIFE=10THENPRINT"';'"OELSEIFE=11THENPRINT"UNDECLARED '
   I$ELSEIFE=12THENPRINTL$;I$ELSEIFE=16THENPRINT"THEN"X$ELSEIFE=17THEN
   PRINT"';' OR END"X$ELSEIFE<18THEN3
30 IFE=18THENPRINT"DO"X$ELSEIFE=19THENPRINT"BAD SYMBOL"ELSEIFE=20THEN
   PRINT"RELATIONAL OPR."X$ELSEIFE=21THENPRINTL$"USE OF PROC."I$ELSE
   IFE=23THENPRINTL$"FACTOR"ELSEIFE=25THENPRINT"BEGIN"X$ELSEIFE=26THEN
   PRINT"OF"X$ELSEIFE<27ANDE>18THEN33
31 IFE=27THENPRINTL$"HEX CONST"ELSEIFE=28THENPRINT"TO"X$ELSEIFE=30THEN
   PRINT"NO. OUT OF RANGE"ELSEIFE=35THENPRINT"PARAMETER MISMATCH"ELSE
   IFE=36THENPRINTL$"DATA TYPE"ELSEIFE>=27THEN33
32 RUN
33 PRINT"'"R"'"X$:RUN
34 IFFL>0THEN36ELSE:GOSUB248:GOSUB250:L$=L$+" ":C0=0:GOSUB218:
   IFLEFT$(L$,1)<>"$"THENRETURN
35 L$=RIGHT$(L$,(LEN(L$)-5)):PRINT"FILE"L$"REQD-";:LM=-1:FL=FL+1:BL$='
   CM=3+FL
36 CM=3+FL:GOSUB246:IFL$="#EOF"THENPRINTL$:FL=FL-1:GOTO34
37 GOSUB209:GOSUB218:L$=L$+" ":C0=0:RETURN
38 IFT<65ORT>90THEN48
39 K=0:A$=""
40 IFK<N2THENK=K+1:A$=A$+X$
41 GOSUB6:IFT>47ANDT<58ORT>64ANDT<91THEN40ELSEIFA$="PROCEDUR"OR
   A$="FUNCTION"THENA$=LEFT$(A$,4)
42 A$=A$+STRING$(12-LEN(A$)," ")
43 I=1:J=N0*5-4:B$=LEFT$(A$,5)
44 K=INT((I+J)/10)*5+1:Z$=MID$(W0$,K,5):IFB$<=Z$J=K-5
45 IFB$>=Z$I=K+5
46 IFI<=JTHEN44
47 IFI-5>JTHENO=B$:RETURNELSE:O=I$:RETURN
48 Z$="":IFT<48ORT>57THEN51ELSEO="NUM"
49 Z$=Z$+X$:GOSUB6:IFT>=48ANDT<=57THEN49
50 N3=VAL(Z$):IFN3<=N1THENRETURNELSEE=30:GOSUB7:N3=N1:RETURN
51 IFX$=":"THENGOSUB6:IFX$="="THENO=":=":GOTO6ELSEO=":":RETURN
52 IFX$="<"THENGOSUB6:O="<"ELSE55
53 IFX$=">"THENO="<>":GOSUB6ELSEIFX$="="THENO="<=":GOSUB6
54 RETURN
55 IFX$<>">"THEN57
56 GOSUB6:O=">":IFX$="="THEN O=">=":GOTO6ELSERETURN
57 IFX$<>"'"THEN59ELSEO="STR":C$=""
58 GOSUB6:IFX$="'"THEN6ELSEC$=C$+X$:GOTO58
59 IFX$<>"("THEN63
60 GOSUB6:IFX$<>"*"THENO="(":RETURN
61 GOSUB6
62 IFX$<>"*"THEN61ELSEGOSUB6:IFX$<>")"THEN62ELSEGOSUB6:GOTO10
63 IFX$<>"%"O=X$:GOTO6
64 GOSUB6:O="NUM":N3=0:FORI=1TO4:T=ASC(X$)
65 IFT>47ANDT<58THEN67
66 IFT>64ANDT<71THENT=T-7ELSE68
67 T=T-48:N3=N3*16+T:GOSUB6:NEXT:RETURN
68 IFI>1THENE=27:GOSUB7:O="%":RETURN
69 FORI=T1TO1STEP-1:IFA$<>T$(I)THENNEXT:I=0:RETURNELSERETURN
70 R=I$:E=4:GOSUB2:R="=":E=3:GOSUB3:GOSUB10:GOSUB71:K$="C":GOSUB12:GOT
71 IFO="NUM"THENRETURNELSEIFO=I$THEN73
72 R="STR":E=2:GOSUB2:N3=ASC(C$):RETURN
73 GOSUB69:IFI=0THENE=2:GOSUB7
74 IF MID$(T0$,I,1)<>"C"THENE=2:GOSUB7
75 N3=T2(I):RETURN
76 R=I$:E=4:GOSUB2:K$="V":GOSUB12:GOTO10
77 IFO<>"+"ANDO<>"-"THENGOSUB83:GOTO79ELSEIFO="-"Y$=O:GOSUB14
78 GOSUB10:GOSUB83:GOSUB15:IFY$="-"U=1:W=1:GOSUB13
79 IFO="+"ORO="-"ORO="OR "THEN80ELSERETURN
80 Y$=O:GOSUB14:GOSUB10:GOSUB83:GOSUB15
81 W=14:IFY$="-"THENW=3ELSEIFY$="+"W=2
82 GOSUB18:GOTO79
83 GOSUB88
84 IFO="*"ORO="DIV "ORO="AND "ORO="MOD "ORO="SHL "ORO="SHR "THEN
   ELSERETURN
85 Y$=O:GOSUB14:GOSUB10:GOSUB88:GOSUB15
86 W=15:IFY$="DIV "THENW=5ELSEIFY$="MOD "THENW=7ELSEIFY$="*"THENW=4
   ELSEIFY$="SHL "THENW=17ELSEIFY$="SHR "W=18
87 GOSUB18:GOTO84
88 IFO=I$THEN91ELSEIFO="NUM"THEN97ELSEIFO="STR"THEN98
89 IFO="("THEN99ELSEIFO="MEM "THEN100ELSEIFO="NOT "THEN102
90 E=23:GOSUB7
91 GOSUB69:IFI=0THENE=11:GOSUB7
92 TT$=MID$(T0$,I,1):IFTT$="P"THENE=21:GOSUB7
93 IFTT$="Y"THENU=5:W=1:GOSUB13:I=I-1:GOTO133
94 IFTT$="A"THEN101
95 IFTT$="C"THENW=T2(I):GOSUB19:GOTO10
96 U=2:V=I:GOSUB17:GOTO10
97 U=0:W=N3:GOSUB13:GOTO10
98 W=ASC(AC$):GOSUB19:GOTO10
99 GOSUB10:GOSUB8:IFO=")"THEN10ELSEE=22:GOTO7
100 GOSUB16:GOSUB10:U=2:V=255:W=0:GOTO9
101 X=I:GOSUB4:GOSUB16:GOSUB5:U=18:V=X:GOSUB17:GOTO10
102 GOSUB10:GOSUB88:W=16:GOTO18
103 IFO="="ORO="<>"ORO="<"ORO="<="ORO=">"ORO=">="THEN104ELSERETURN
104 Y$=O:GOSUB14:GOSUB10:GOSUB77:GOSUB15
105 W=8:IFY$="<>"THENW=9
106 IFY$="<"THENW=10
107 IFY$=">="THENW=11
108 IFY$=">"THENW=12
109 IFY$="<="THENW=13
110 GOTO18
111 IFO=I$THEN113ELSEIFO="IF "THEN138ELSEIFO="FOR "THEN155ELSE
    IFO="WHILE"THEN145ELSEIFO="CASE "THEN146ELSEIFO="REPEA"THEN143ELSE
    IFO="BEGIN"THEN140ELSEIFO="READ "THEN124ELSEIFO="WRITE"THEN119ELSE
    IFO="MEM "THEN141
112 IFO="CALL "THEN132ELSERETURN
113 GOSUB69
114 IFI=0THENE=11:GOSUB7ELSETT$=MID$(T0$,I,1):IFTT$="A"THEN115ELSE
    IFTT$="V"THEN116ELSEIFTT$="Y"THEN116ELSEIFTT$="P"THEN133ELSE
    E=12:GOSUB7
115 X=I:GOSUB4:X=16:GOSUB4:GOSUB16:GOTO117
116 X=I:GOSUB4:X=0:GOSUB4
117 GOSUB10:IFO=":="GOSUB10ELSEE=13:GOSUB7
118 GOSUB8:GOSUB5:K=X:GOSUB5:U=3+K:V=X:GOTO17
119 GOSUB22
120 GOSUB10:IFO="STR"THENL=LEN(AC$):U=0:V=0:FORI=1TOL:W=ASC(MID$(AC$,I,1))
    GOSUB9:NEXT:W=L:GOSUB9:U=8:W=8:GOSUB9:GOSUB10:GOTO123
121 GOSUB8:K=1:IFO="#"K=3:GOSUB10ELSEIFO="%"K=5:GOSUB10
122 U=8:W=K:GOSUB13
123 IFO=","THEN120ELSEGOSUB23:GOTO10
124 GOSUB22
125 R=I$:E=4:GOSUB3:GOSUB69:IFI=0THENE=11:GOSUB7
126 X=I:GOSUB4:IFMID$(T0$,I,1)="A"THEN130ELSEIFMID$(T0$,I,1)="V"THEN
    L=0ELSEE=4:GOSUB7
127 GOSUB10:K=0:IFO="#"THENK=2
128 U=8:W=K:GOSUB13:IFK>0GOSUB10
129 GOSUB5:U=L+3:V=X:GOSUB17:IFO=","THEN125ELSEGOSUB23:GOTO10
130 GOSUB16
131 L=16:GOTO127
132 GOSUB22:GOSUB10:GOSUB8:GOSUB23:U=4:V=255:W=0:GOSUB9:GOTO10
133 K2=0:K3=I:IFT3(I)=0THEN136ELSEGOSUB22
134 X=K2:GOSUB4:X=K3:GOSUB4:GOSUB10:GOSUB8:GOSUB5:K3=X:GOSUB5:K2=X+1:
    IFO=","THEN134
135 IFK2<>T3(K)E=35:GOSUB7:GOSUB23
136 U=4:V=K3:GOSUB17:IFK2<>0THENU=5:W=-K2:GOSUB13
137 GOTO10
138 GOSUB10:GOSUB8:R="THEN ":E=16:GOSUB2:GOSUB10:X=C1:GOSUB4:U=7:GOSUB20
    GOSUB11
139 IFO<>"ELSE "THEN185ELSEGOSUB5:K=X:X=C1:GOSUB4:U=6:GOSUB20:X=K:
    GOSUB186:GOSUB10:GOSUB11:GOTO185
```

```
140 GOSUB10:GOSUB11:IFO=";"THEN140ELSEIFO="END  "THEN10ELSEE=17:GOTO7
141 GOSUB16
142 R=":=":E=13:GOSUB3:GOSUB10:GOSUB8:U=3:V=255:W=0:GOTO9
143 X=C1:GOSUB4
144 GOSUB10:GOSUB11:IFO=";"THEN144ELSER="UNTIL":E=10:GOSUB2:GOSUB10:
    GOSUB8:GOSUB5:U=7:W=X:GOTO13
145 GOSUB10:X=C1:GOSUB4:GOSUB8:X=C1:GOSUB4:U=7:GOSUB20:R="DO  ":E=18:
    GOSUB2:GOSUB10:GOSUB11:GOSUB5:K=X:GOSUB5:U=6:W=X:GOSUB13:X=K:GOTO186
146 GOSUB10:GOSUB8:R="OF  ":E=25:GOSUB2:I2=1
147 I1=0
148 GOSUB10:GOSUB71:W=21:GOSUB18:W=N3:GOSUB19:W=8:GOSUB18:GOSUB10:
    IFO=":"THEN150
149 R=",":E=5:GOSUB2:X=C1:GOSUB4:U=7:V=1:W=0:GOSUB9:I1=I1+1:GOTO148
150 K=C1:U=7:GOSUB20:IFI1>0THENFORI=1TOI1:GOSUB185:NEXT
151 X=K:GOSUB4:GOSUB10:X=I2:GOSUB4:GOSUB11:GOSUB5:I2=X:IFO="ELSE "THEN153
152 IFO<>";"THEN154ELSEK=C1:U=6:GOSUB20:GOSUB185:X=K:GOSUB4:I2=I2+1:
    GOTO147
153 K=C1:U=6:GOSUB20:GOSUB185:X=K:GOSUB4:GOSUB10:X=I2:GOSUB4:GOSUB11:
    GOSUB5:I2=X
154 R="END  ":E=17:GOSUB2:FORI=1TOI2:GOSUB185:NEXT:U=5:W=-1:GOSUB13:
    GOTO10
155 R=I$:E=4:GOSUB3:GOSUB113:GOSUB4:F9=1:IFO<>"TO  "R="DOWNT":E=28:F9=0
156 GOSUB10:GOSUB8:GOSUB5:K=X:X=C1:GOSUB4:W=21:GOSUB18:U=2:V=K:GOSUB17:
    W=13-F9*2:GOSUB18:X=C1:GOSUB4
157 U=7:GOSUB20:X=F9:GOSUB4:X=K:GOSUB4:R="DO  ":E=18:GOSUB2:GOSUB10:
    GOSUB11:GOSUB5:U=2:V=X:GOSUB17
158 K=X:GOSUB5:W=20-X:GOSUB18:U=3:V=K:GOSUB17:GOSUB5:K=X:GOSUB5:U=6:
    W=X:GOSUB13:X=K:GOSUB186:U=5:W=-1:GOTO13
159 D0=3:T2(T1-K1)=C1:U=6:GOSUB20:X=T1-K1:GOSUB4
160 IFO="CONST"THEN162ELSEIFO="VAR  "THEN164
161 IFO="PROC "THEN170ELSEIFO="FUNC "THEN171ELSEIFO="BEGIN"THEN177
    ELSEE=25:GOSUB7
162 GOSUB10
163 GOSUB70:R=":":E=5:GOSUB2:GOSUB10:IFO="VAR  "THEN164ELSEIFO="PROC "
    THEN170ELSEIFO="FUNC "THEN171ELSEIFO="BEGIN"THEN177ELSE163
164 L=0:F9=1
165 GOSUB10:GOSUB76
166 L=L+1:IFO=",":THEN165ELSER=":":E=5:GOSUB2:GOSUB10:IFO="ARRAY"THEN167
    ELSER="INTEG":E=36:GOSUB2:GOTO169
167 GOSUB21:GOSUB71:R=")":E=34:GOSUB3:R="OF  ":E=26:GOSUB3:R="INTEG":
    E=36:GOSUB3:D0=D0-1
168 FORI=T1-L+1TOT1:T0$=LEFT$(T0$,I-1)+"A"+RIGHT$(T0$,LEN(T0$)-I):
    T3(I)=N3+1:T2(I)=D0:D0=D0+N3+1:NEXT
169 R=";":E=5:GOSUB3:GOSUB10:IFO="PROC "THEN170ELSEIFO="FUNC "THEN171
    ELSEIFO="BEGIN"THEN177ELSEL=0:F9=1:GOSUB76:GOTO166
170 R=I$:E=4:GOSUB3:K1=0:K$="P":GOSUB12:L1=L1+1:GOTO172
171 R=I$:E=4:GOSUB3:K$="F":GOSUB12:L1=L1+1:K1=1:K$="Y":GOSUB12
172 K2=K1:GOSUB10:X=T1:GOSUB4:X=D0:GOSUB4:IFO<>"(":THEN175
173 GOSUB10:F9=0:GOSUB76:K1=K1+1:IFO=",":THEN173
174 GOSUB23:GOSUB10:T3(T1-K1)=K1-K2
175 R=";":E=5:GOSUB2:FORI=1TOK1:T2(T1-I+1)=-I:NEXT
176 GOSUB10:GOSUB159:L1=L1-1:GOSUB5:D0=X:GOSUB5:T1=X:R=";":E=5:GOSUB2:
    GOSUB10:GOTO161
177 GOSUB10:GOSUB5:K=X:X=T2(K):GOSUB186:T2(K)=C1:U=5:W=D0:GOSUB13
178 GOSUB11:IFO=";"THENGOSUB10:GOTO178
179 IFO<>"END  "E=17:GOSUB7
180 GOSUB10:W=0:GOTO18
181 CC=U:GOSUB188:CC=V:GOSUB188:N4=VARPTR(W):CC=PEEK(N4):GOSUB188:
    CC=PEEK(N4+1):GOSUB188:IFU>250THENRETURN
182 IFU>16THENB$="X":U=U-16ELSEB$=""
183 B$=MID$(M$,U*3+1,3)+B$:IFLPTHENLPRINTTAB(40)AC1" "B$" "V;W:
    ELSEPRINTTAB(40)AC1" "B$" "V;W
184 C1=C1+1:RETURN
185 GOSUB5
186 IFOF$="#"THENRETURNELSEU=253:V=0:W=X:GOSUB9:U=254:W=C1:GOSUB9:P9=P9-8
187 IFLPTHENLPRINTTAB(40)"ADD"C1"AT"X:RETURNELSEPRINTTAB(40)"ADD"C1"AT"X:
    RETURN
188 IFOF$="#"THENRETURN
189 IFBZ>=240THENGOSUB191
190 BZ=BZ+1:LG=VARPTR(OB$)+1:POKE(PEEK(LG)+PEEK(LG+1)*256+BZ-1),CC:RETURN
191 LG=BZ:OB$=LEFT$(OB$,BZ)
192 CM=2:IFLM<>CMTHENINPUT"OBJ CAS READY";B$
193 PRINT#-1,CHR$(LG)+OB$;:OB$=STRING$(240,"*"):BZ=0:LM=CM:RETURN
194 '
195 DEFINTA-Z:POKE16553,255:FORI=16480TO16492:READJ:POKEI,J:NEXT:
    FORI=16435TO16437:READJ:POKEI,J:NEXT:POKE16405,0:DATA205,227,3,183,
    200,14,23,16,254,13,32,251,201,195,96,64
196 CLS:PRINT@7,CHR$(23);"TRS-80 PEOPLE'S PASCAL";:PRINT@78,
    "EDITOR/COMPILER";:PRINT@140,"(PPEC) VERSION 2.3":PRINT@266,
    "COPYRIGHT (AC) 1979";:PRINT@330,"PIPE DREAM SOFTWARE";
197 PRINT@396,"BERWICK AUSTRALIA";:PRINT@586,"DISTRIBUTED BY C.I.E.";:
    PRINT@642,"BOX 158, SAN LUIS REY, CA 92068";:PRINT@712,
    "COMPLETE SYSTEM - $15.00";
198 FORI=1TO10STEPI:FORJ=1TOI:NEXT:PRINTCHR$(28);:FORJ=1TOI:NEXT:
    PRINTCHR$(23);:NEXTI:PRINT@64*15,;:SA=29700:I1=PEEK(16598)+
    PEEK(16599)*256+2:
199 IFI1<>SATHENPRINTCHR$(28);:PRINT@64*13,"* * * ERROR - MEMORY SIZE
    SHOULD BE"SA"NOT"I1"* * *":ENDELSEINPUT"OK";A:TA=32767:P=SA:
    POKESA,4:POKESA+1,0:POKESA+2,0:POKESA+3,32:POKESA+4,3:POKESA+5,255:
    POKESA+6,127:FA=P+6:GOSUB257:POKETA,14
200 CLS:PRINT"COMMANDS ARE:":PRINT:PRINT"C - COMPILE":PRINT"D - DELETE":
    PRINT"E - WRITE EOF":PRINT"F - FREE TEXT SPACE QUERY":PRINT"L - LIST
    LINES ON DISPLAY":PRINT"N - RENUMBER LINES":PRINT"P - PRINT LINES ON
    LINEPRINTER"
201 PRINT"R - READ BLOCK(S) C.ROM FILE":PRINT"W - WRITE LINES TO FILE":
    PRINT:PRINT"PLEASE TYPE 'RUN' AGAIN":DELETE195-201
202 CLEAR(1330):GOSUB24:SA=29700:TA=32767:LM=-1:ML=N1:YY$="*":P=SA:
    PO=SA:LN=1
203 P=PO:L$="":INPUTL$:A$=LEFT$(L$,1):GOSUB204:GOTO203
204 IFA$="R"THEN242ELSEIFA$="C"THEN25ELSEIFA$="W"THEN224ELSEIFA$="I"
    THEN203ELSEIFA$="L"THENRA$="P"THEN220ELSEIFA$="D"THEN222ELSEIFA$="E"
    THEN230ELSEIFA$="N"THEN231ELSEIFA$="C"THENGOSUB256:PRINT
    "FREE="TA-FA-3:GOTO203
205 IFL$="1"THENRETURNELSEIFL$<"0"ORASC(L$)>=58THENPRINT"??":RETURN
206 GOSUB209
207 IFLN=0ORLN>=MLTHENPRINT"??":RETURN
208 GOTO251
209 X$="":FORX=1TO5:B$=MID$(L$,X,1):IFB$<="9"ANDB$>="0"THENX$=X$+B$:NEX
210 LN=VAL(X$):L$=RIGHT$(L$,LEN(L$)+1-LEN(STR$(LN))):RETURN
211 GOSUB249:IFQL>LNTHENP=SAELSEIFQL=LNTHENRETURN
212 GOSUB248:IFQL<LNTHEN212ELSERETURN
213 L$=L$+"  ":BR=VAL(RIGHT$(L$,LEN(L$)-1)):TR=LEN(STR$(BR))+2:
    TR=VAL(MID$(L$,TR,LEN(L$)-TR)):IFLEFT$(RIGHT$(L$,4),1)="-"THENTR=ML
214 IFMID$(L$,2,1)="."THENBR=LN
215 IFBR>0ANDTR=0THENTR=BR
216 IFTR+BR=0ORTR>MLTHENTR=ML-1
217 IFBR=0THENBR=1:RETURNELSERETURN
218 IFLPTHENLPRINT"LN;L$:RETURNELSEPRINT"LN;L$:RETURN
219 GOSUB213:LN=BR:GOTO211
220 GOSUB219:LP=A$="P"
221 IFQL>TRORPEEK(15359)THENRETURNELSEGOSUB250:GOSUB218:GOSUB248:GOTO221
222 GOSUB219
223 IFQL>TRTHENRETURNELSELN=QL:GOSUB255:GOSUB249:GOTO223
224 BL$="":CM=1:GOSUB219
225 IFQL>TRTHEN227ELSEIFQL<>0THENGOSUB250:GOSUB218:GOSUB236
226 GOSUB248:GOTO225
227 IFBL$<>""THENGOSUB234
228 IFTR=ML-1THENINPUT"EOF";A$:IFA$="Y"THENGOSUB230
229 A$="":INPUT"DELETE";A$:IFA$="Y"THENP=SA:LN=BR:GOSUB211:GOTO223
    ELSERETURN
230 CM=1:L$="#EOF":LN=0:GOTO236
231 GOSUB219:IFTR=BRORTR=ML-1THENTR=100:INPUT"NEW BASE";TR
232 X=10:INPUT"INCREMENT";X
233 IFQL>=MLTHENRETURNELSEV=VARPTR(TR):POKEP+1,PEEK(V):POKEP+2,PEEK(V+1):
    GOSUB248:TR=TR+X:GOTO233
234 IFBL$=""THENRETURNELSEIFLM<>CMTHENINPUT"WRITE CAS1";A$
235 PRINT#-1,CHR$(34)+BL$:BL$="":LM=CM:RETURN
236 B$=L$:IFLN<>0THENA$=STR$(LN):L$=RIGHT$(A$,LEN(A$)-1)+L$
237 IFLEN(BL$)+LEN(L$)+2>240THENGOSUB234
238 LG=LEN(L$):IFLG=13ORLG=34THENL$=L$+"  "
239 BL$=BL$+CHR$(LEN(L$))+L$
240 IFB$="#EOF"ORLEFT$(B$,5)="$INCL"THENGOSUB234
241 RETURN
242 BL$="":GOSUB213:CM=3:FORI=1TOBR:GOSUB244:GOSUB243:IFL$="#EOF"THEN
    RETURNELSENEXT:RETURN
243 IFBL$=""THENRETURNELSEGOSUB247:IFL$="#EOF"THENPRINTL$:RETURN
    ELSE:GOSUB206:GOSUB218:GOTO243
244 IFLM<>CMTHENINPUT"READ CAS1";X$
245 INPUT#-1,BL$:IFBL$=""THEN245ELSELM=CM:RETURN
246 IFBL$=""THENGOSUB244
247 L$=MID$(BL$,2,ASC(BL$)):BL$=RIGHT$(BL$,LEN(BL$)-1-ASC(BL$)):RETURN
248 PO=P:P=P+PEEK(P)
249 QL=PEEK(P+1)+PEEK(P+2)*256:RETURN
250 X=VARPTR(YY$):POKEX,PEEK(P)-3:P=P+3:V=VARPTR(P):POKEX+1,PEEK(V):
    POKEX+2,PEEK(V+1):P=P-3:L$=YY$:LN=PEEK(P+1)+PEEK(P+2)*256:RETURN
251 GOSUB211:IFQL=LNTHENGOSUB255
252 IFL$=""THENRETURNELSEGOSUB256:IFTA-FA-7<LEN(L$)THENPRINT"WONT FIT":
    P=PO:RETURN
253 W=LEN(L$)+3:Q1=FA:Q2=FA+W:Q3=FA-P+1:XL=184:GOSUB259:POKEP,W:
    V=VARPTR(LN):POKEP+1,PEEK(V):POKEP+2,PEEK(V+1):FORJ=1TOLEN(L$):
    POKEP+J+2,ASC(MID$(L$,J,1)):NEXT:FA=FA+PEEK(P):GOTO257
254 GOSUB211
255 IFQL<>LNTHENRETURNELSEW=PEEK(P):GOSUB256:Q1=P+W:Q2=P:Q3=FA-P-N:
    XL=176:GOSUB259:FA=FA-W:GOSUB257:RETURN
256 FA=PEEK(TA-2)+PEEK(TA-1)*256:RETURN
257 POKETA-2,PEEK(VARPTR(FA)):POKETA-1,PEEK(VARPTR(FA)+1):RETURN
258 V=VARPTR(X):Z8$=Z8$+CHR$(PEEK(V))+CHR$(PEEK(V+1)):RETURN
259 Z8$=CHR$(33):X=Q1:GOSUB258:Z8$=Z8$+CHR$(17):X=Q2:GOSUB258:
    Z8$=Z8$+CHR$(1):X=Q3:GOSUB258:Z8$=Z8$+CHR$(237)+CHR$(XL)+CHR$(201):
    GOSUB260:RETURN
260 V=VARPTR(Z8$)+1:POKE16526,PEEK(V):POKE16527,PEEK(V+1):V=USR(0):RETURN
```

CONTINUED FROM PAGE (13)

SAMPLE PROGRAM

```
1790   SEED:=RND(SEED); X:=SEED MOD 80+20;
1800   SEED:=RND(SEED); Y:=SEED MOD 30+10;
1810   REPEAT BEGIN
1820     WRITE(HOME,CLEAR,WIDE);
1825     AT(10); WRITE('BULLSEYE');
1830     AT(74); WRITE('(IN TINY PASCAL)');
1840     AT(202); WRITE('HOW MANY STARS? ');
1842     IF MODE='D' THEN BEGIN
1850       STARS:=20; WRITE(STARS#,CR); FOR I:=1 TO 3000 DO;
1860     END ELSE BEGIN
1870       READ(STARS#);
1880     END;
1890   END UNTIL ((STARS>=0) AND (STARS<100));
1900   FOR I:=1 TO STARS DO BEGIN
1910     SEED:=RND(SEED); XSTARS(I):=SEED AND %007F;
1920     SEED:=RND(SEED); YSTARS(I):=SEED MOD 48;
1930   END;
1940   WHILE TARGET(X,Y) AND ALIVE DO BEGIN
1950     FOR I:=1 TO STARS DO BEGIN
1960       SET(1,XSTARS(I),YSTARS(I));
1970     END;
1980     SIGHTS;
1990     ALIVE:=ACTION;
2000     IF ALIVE THEN BEGIN
2010       CHANGE;
2020     END;
2030   END;
2040   WRITE(HOME,WIDE);
2050   IF ALIVE THEN WRITE('YOU LET IT ESCAPE! ')
2060   ELSE WRITE('BULLSEYE! ');
2070   FOR I:=1 TO 5000 DO;
2080 END;
2090 END.
```

TRS-80 Computing is published as often as monthly by Computer Information Exchange, Inc., a nonprofit educational corporation, Box 158, San Luis Rey CA 92068.

Subscription rates in the U.S. are $15 for 12 issues. To Canada and Mexico, subscriptions are $18US for 12 issues; all other countries $27US.

No advertising is accepted. Free editorial space is given any commercial product that might be of interest to TRS-80 users. One free page of dollars-off coupons will be made available to product vendors, on a space-available basis.

In keeping with TRS-80 Computing's nonprofit educational aims, any amount of material from each issue may be reproduced by any not-for-profit educational group or institution, without prior request. Tearsheet is appreciated. Required credit includes: "TRS-80 Computing, box 158, San Luis Rey CA 92068; $15 for 12 issues."

Clubs having regular newsletters may receive TRS-80 Computing in exchange, providing CIE is also granted similar reprint privileges.

Organizations or anyone having audio tapes or speakers that might be of interest to CIE readers should submit them to manager editor Bill McLaughlin. If accepted, they will be typeset free, with a copy returned with the tape. Office phone is (714) 757-4849.

Editorial contributions of all kinds are requested (software, hardware and applications articles, letters, etc.) and may be submitted either written or on cassette.

TRS-80 is a Tandy Corporation trademark licensed to Computer Information Exchange. Computer Information Exchange is solely responsible for the editorial content of this magazine and is not an agent, subsidiary, or otherwise affiliated with Tandy Corporation.

computer information exchange, inc.

Box 158, San Luis Rey CA 92068

Nonprofit Org.
U.S. POSTAGE
PAID
San Luis Rey CA
PERMIT 4

DATED MATERIAL
ADDRESS CORRECTION REQUESTED