

# Tiny Pascal User's Manual

Catalog Number 26-2009

**Radio Shack®**  
**TRS-80™**  
**MICRO**  
**COMPUTER**  
**SYSTEM**

Radio Shack's Tiny Pascal is a cassette-based system for creating, editing, compiling and executing Pascal programs. Tiny Pascal is a subset of the standard Pascal language.

This manual summarizes Tiny Pascal and explains how to use it on the Model I Level II TRS-80.

## Contents

1 / Overview of the System .....	1
2 / The Monitor .....	2
3 / The Editor .....	3
4 / The Compiler .....	5
5 / Starting Instructions .....	10
6 / Using the 32K Tiny Pascal .....	13
7 / Error Codes .....	14
8 / Useful Calls and Addresses .....	17
9 / The Rules to BLOCKADE .....	18
A / Memory Maps .....	19
B / Sample Programs .....	20
C / Syntax Diagrams .....	23
Index .....	27
Customer Information .....	28



---

# 1/ Overview of the System

Tiny Pascal is a complete, self-contained operating system for creating, compiling, running, saving and loading Pascal programs for the TRS-80. Once you have loaded Tiny Pascal, you can use all the "subsystems":

**Monitor:** Provides run-time support, checks for errors, and provides the necessary utilities to save and load programs to and from cassette tape.

**Compiler:** Compiles your PASCAL source program into P-code, ready to be executed. The compiler also checks for syntax errors.

**Editor:** Lets you create or modify Tiny Pascal source programs.

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

**The minimal system requirements are: LEVEL II, 16 K RAM. You will need at least 32K to examine the Compiler source. (See Chapter 6.)**

## Overview of this Manual

Chapters 2 through 4 of this manual discuss in detail the three sub-systems, what they do, and how to use them. Chapter 5 deals with the specific aspects, limitations and enhancements to Tiny Pascal. Then follows a chapter on Getting Started to help you get through the first time you bring Tiny Pascal up. Finally, you will find the error codes, syntax diagrams, listings of sample programs, and other useful information.

## 2/ The Monitor

All operations make at least some use of the Monitor, so we will begin our description of the Tiny Pascal system with it. The Monitor provides run-time support to the entire system, as well as providing you with a means of saving/loading your source programs and P-code (compiled) programs via cassette tape. From the Monitor you also give the command to compile a program, or to run that program once it has been compiled. You also invoke the Editor from the Monitor.

### Monitor Commands *Q returns to monitor.*

E	Edit old source file or create a new one.
C	Compile source program into P-code, ready to be executed. P-code is stored after source in RAM.
C/-P	<u>Compile source, but do NOT generate P-code</u> (useful for checking for syntax errors)
C/-S	Compile source, and overwrite the source program with P-code (used when you have very large programs). You will have to reload or retype the source program if you want to edit it later.
R	Run the compiled program.
R/-C	Run the compiled program and overwrite the Editor and the Compiler. You will have to reload the Pascal system if you want to edit or compile a program later.
LS <i>filename</i>	Load source program from cassette.
LP <i>filename</i>	Load P-code program from cassette.
WS <i>filename</i>	Save source program to cassette.
WP <i>filename</i>	Save P-code program to cassette.

Note that you may choose to overwrite sections of the Tiny Pascal system if you need the space for a large program. However, you must remember that the overwritten sections are gone and you must re-load the entire system if you are to use them again.

A file name can be at most six characters long. When loading a program, either in source or P-code format, the file name must be entered **exactly** as it was saved on tape. That is, when loading the Tiny Pascal system or when reloading a program which you have saved, you must be sure to use the correct and complete name.

If you accidentally type in the wrong file name when requesting a load, the Tiny Pascal system may never return control to the keyboard and you will have to reset and reload the entire system again. Also, there is no way to find out the names of files on tape, so you must **remember exactly what you called the file** when you recorded it.

### 3/ The Editor

The Editor provided with your Tiny Pascal package enables you to create and modify source programs.

The Pascal Editor is line oriented, but, unlike BASIC, does not use line numbers since they are not used in the Pascal language. 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. In our discussion of the Editor, *xx* refers to integer numbers (1-999), and *string* refers to a string. Each command ends with a carriage return (**ENTER** on your TRS-80 keyboard). Invalid commands are flagged with the message 'ILLEGAL'.

The line pointer always points to the line most recently displayed, modified or inserted. After a Delete command, the line pointer is moved up one line. When you first load the Tiny Pascal system, a sample program is loaded in, too. Thus, when you type "E" to enter edit mode, a file is already there. You will see a ">". This is the prompt from the editor that lets you know that it is waiting for a command (not text). The commands are listed below.

To erase a source program, use the "D\*" command. After a "D\*", the editor will automatically put you into Insert mode and wait for text.

#### Editor Commands




**Note:** '\*' means entirely or 'all the way':

<b>ENTER</b>	<u>A carriage return on an empty line will exit from Insert mode</u>
P	Prints the current line <i>on the screen</i>
Pxx	Prints xx lines starting from current line
P*	Prints entire file
U	Moves up one line
Uxx	Moves up xx lines
U*	Moves up to top or first line of file

---

### *Editor commands, continued*

N	Moves line pointer to next line (down)
Nxx	Moves line pointer down xx lines
N*	Moves line pointer to last line of file
D	Deletes current line
Dxx	Deletes xx lines starting at current line
D*	Deletes entire file. This will automatically put you into Insert mode and wait for text.
I	Enters Insert mode. Begins <u>inserting</u> lines <u>after</u> current line pointer. A '?' is displayed to prompt you. Press <b>(ENTER)</b> at the beginning of a line to exit from Insert mode.
Rstring	Replaces the current line by <i>String</i>
X	Extends line. The current line is displayed and the cursor is positioned to the end of the line, allowing characters to be appended. Press <b>(ENTER)</b> to save changes and return to the Editor.
S	Displays Status: Number of lines, file location, position of line pointer.
Q	Quits and returns to the Tiny Pascal 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.  over a tab moves the cursor three spaces back on the display and erases the tab.

**To illustrate:** 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 press **(ENTER)** on an empty line.

If a "MEMORY FULL" error occurs while editing or inserting, the source file is too big. You may be able to pack in the program by eliminating extra spaces and tabs.

You should experiment with the editor for awhile to make sure that you completely understand its operation.

## 4/ The Compiler

A compiler is a program that translates the statements of a high-level language into an equivalent program of machine-readable form. Tiny Pascal translates the high-level source program into an intermediate file called P-code. The P-code is then interpreted, using the run-time Monitor for support. The result is a program which executes at least four times faster, and up to eight times faster than BASIC!

Tiny Pascal is a subset of standard Pascal. The syntax is essentially identical to the full language. Syntax diagrams have been included in Appendix C for those who are just now learning the language.

This manual is not an instructional text on Pascal programming, but rather an explanation of the limits and special features of Tiny Pascal. However, we will review some essential points in the next section.

For those who need a more thorough introduction, we recommend the following:

*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

*Pascal, An Introduction to Methodical Programming*; W. Findlay and D.A. Watt. Computer Science Press, 1978

## Compiler Specifics

**Note:** See the Index for a complete list of Pascal keywords.

1. **Maximum number of procedure or function parameters** is 15; maximum number of procedure nests is seven levels; the symbol table is restricted to 75 entries (200 for big version).
2. “:=” is used for **assignment** and “=” is used for **equality**. They are **not** interchangeable!
3. “;” 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 a CASE statement.

4. **Expressions** may be either arithmetic or logical (Boolean). Thus, the following are legal:

A := B > C;

IF A + B THEN...

**Note:** The Boolean operator OR has the same precedence as the arithmetic operators “+” and “-”. AND has the same precedence 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 (analyzed) as:

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

Thus producing an unintended result.

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

5. For the TRS-80, “(”and“)” are used instead of “[” and “]”.
6. **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.
7. **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.
8. **Parameters** are passed to procedures or functions by value, i.e. a copy of the value of the parameter as defined before the call.
9. **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 use a variable defined in an outer procedure.
10. **The only data types** Tiny Pascal supports are integers and one-dimensional integer arrays. The integers are 16-bit signed. The arrays start at 0. Arrays are **not** checked for subscript-out-of-range at runtime.



## 11. Built-in functions:

Function	Meaning
A DIV B	Truncated Integer division: $27 \text{ DIV } 5 = 5$
A MOD B	$A - (A \text{ DIV } B) * B$ : $27 \text{ MOD } 5 = 2$
A SHL B	<u>Binary</u> Left Shift A by B: $27 \text{ SHL } 2 = 54$ <span style="margin-left: 20px;"><math>27 \text{ SHL } 2 = 108</math></span>
A SHR B	<u>Binary</u> Right Shift A by B: $27 \text{ SHR } 2 = 13$ <span style="margin-left: 20px;"><math>27 \text{ SHR } 2 = 6</math></span>

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) a specified memory location, such as:

A := MEM (24467) + 3; (\* READ FROM MEMORY\*)  
 MEM (T) := 0; (\* WRITE TO MEMORY\*)

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.

12. **Hex constants** are prefixed by `''%'`, e.g., `%2A00`

13. **Strings** are enclosed by single quotes (`'`). **not** double quotes (`''`). When a string appears in an expression or as a CASE label, 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 output. For example:

X := 'ABCD'

X would equal 65 (ASCII `'A'`).

14. **The READ and WRITE statements** are character-oriented, not line-oriented. More than one character can be input or output with a single 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 characters, a decimal number, and a hex number.

A := 65

WRITE ('HELLO? ', A, ' ', A#, ' ', A%)

would print:

HELLO?    A    65    0041

**Note:** `' '` represents a blank space. It is used only where necessary for illustration.

15. Since the **READ** is **character-oriented**, it is necessary to terminate an integer input by a non-integer character (such as **ENTER** or **SPACEBAR**). To input a hex number, four digits must be typed.

16. To write on a new line, it is necessary to output the ASCII code for carriage return/line-feed to the output device. That is, you must manually insert a carriage return/line feed. For the TRS-80 this can be accomplished by outputting the carriage return alone. (The TRS-80 software does the rest.) For example:

```
WRITE ('THIS IS A TEST', 13)
```

Note: 13 is an ASCII "carriage return".

17. An expression in the IF, WHILE, and REPEAT statements is said to fulfill the condition if the least significant bit is one. This is equivalent to testing whether the expression is odd. Thus after:

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

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

18. The **relational operators** ("=", ">", etc.) always produce a value of zero or one. Thus after:

```
A := X = 5;
```

A equals one if X equals five; otherwise A equals zero.

19. Comments are opened by "(\*)" and closed by "(\*)".

20. Here is a list of built-in functions and procedures:

ABS(exp)

Returns the absolute value of *exp*

CALL(exp)

A user-defined machine language subroutine where *exp* is an address to the routine.

Subroutines must conform to the following:

- 1). Save all registers upon entry.
- 2). Restore all registers before exiting.
- 3). Return from the subroutine in the following fashion:

INC DE

INC DE

RET

INKEY

Inputs from the keyboard, used like this: A := INKEY

INP(exp)

Inputs port *exp*, used like this: A = INP(*exp*)

MEMW(*exp*)  
MOVE(*b,a,n*)

a two-byte peek or poke.  
Move a block of memory of *n* bytes from address *a* to address *b*.

OUTP(*a,x*)

Outputs *a* to port *x*

MEM(*exp*)  
PLOT(*x,y,a*)

Like PEEK on right side of eq., and like POKE on left side of assignment.  
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 one if the point is set, zero if not set.

SQR(*exp*)

Returns square root of *exp*.

21. The screen control characters are the same as TRS-80 BASIC. For example, use WRITE (28,31) to clear the screen.

## 5/ Starting Instructions

In this section, 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 32/48K compiler and source to Tiny Pascal, PAS32K and COMPS1 respectively.

( L 5 ) SPACE NAME

### Start-Up

1. Turn on your machine. When asked for MEMORY SIZE, respond by pressing **(ENTER)**.
2. Type "SYSTEM" **(ENTER)**, to reach system level, You will see "\*?"
3. Make sure that your Tiny Pascal tape is at the start, and type PASCAL, then **(ENTER)** and put the recorder in the Play mode.  
*or PAS32K for 32K/48K version*
4. The tape will begin to load, and the asterisk will blink every four seconds. The entire load will take about three minutes.
5. Once the tape has loaded, press "/" **(ENTER)**. At this point you should receive the opening message:  
TINY PASCAL V-X.Y  
*1.0*  
where X is the version and Y the release number.  
*or 122592*
6. At this point you have successfully loaded the entire Tiny Pascal operating system, and can proceed to the next section below. If you did not get this far try loading the tape again. Try various volume settings.

## Creating a Program

1. From the Monitor, type "E". This will place you in the Editor. You will see a set of statistics on the current file. A sample program is loaded with Tiny Pascal. If this is your very first try, then skip ahead to step 5, otherwise proceed.
2. To delete the sample program which is always loaded with the system, you simply use the editor command: "D\*". Remember, 'D\*' will delete all lines and put you in the Insert mode.
3. At this point you may enter a program.
4. Once your program is entered, you may exit the Insert mode by pressing **ENTER** at the start of a line. This puts you back in the Editor command mode.
5. To return to the Monitor, to compile, etc., you press **Q** for "Quit".

## Compiling, Running, Saving/Loading a Program

1. Normally, to compile a source program, you press **C** **ENTER** 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 in Chapter 7 will tell you what they are. You should then go back and edit the existing source file to correct the syntax errors before re-compiling.

**Note:** Other options for compilation are discussed on the next page and on page 2.

2. Once you have successfully compiled the program, you may run it from the Monitor by typing: **R** **ENTER** from the Monitor.
3. To save the program, or the P-code, you may use the appropriate Monitor commands. That is, "WS filename" to save the source file (program text), or "WP filename" to save the P-code file. Or, at this point, you could start a new program, or load a previously stored program from tape.
4. **Remember, you must re-compile a program if you make a change in it!**
5. To load a previously stored program, you would use either the "LS filename" to load the source (text) file, or the "LP filename" to load the P-code (object code) file. If you forget and accidentally try to load an object file as a source file, or vice versa, errors will result and you may have to reload the Tiny Pascal system.

---

## Special Notes

The **(BREAK)** key causes a pause in program execution; press any other key to resume. If you press **(BREAK)** twice in a row, you will terminate the run, and return to the Tiny Pascal monitor.

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.

If an error "1001" occurs during compilation, there is not enough memory. You should try using "C/-S". Be sure to save the source first!

When a "MEMORY FULL" error occurs while running the program, either cut down your array size or try using the "R/-C" option. Be sure to save the source first!

We know that you will enjoy using Tiny Pascal, and recommend that you play with it a while just to get the hang of it and to become familiar with all its features.

## 6/ Using the 32K/48K Tiny Pascal

On Side Two of your tape is an expanded Tiny Pascal compiler. That is, it can handle larger programs. You will need **at least** 32K RAM to use it. It is exactly the same as the 16K version, except that it will use all the memory that you have in your machine.

To use it, simply follow the directions in Chapter 5, for starting Tiny Pascal, except substitute "PAS32K" for "PASCAL". The source to the compiler is immediately after "PAS32K" on Side Two. It is called: "COMPS1". To load this file simply type: "LS COMPS1". You can then examine the source to the compiler. You do not **need** to do this to run the 32K version; it is for your interest only. You can use it to study the design and exact implementation of Tiny Pascal. The source to Tiny Pascal is written in Tiny Pascal and should provide hours of enjoyment.

**Note:** Programs are not interchangeable between the two compilers. That is, a program created using the 32/48K compiler cannot be used with the normal compiler, and vice-versa.

---


## 7/ 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
- 103: Identifier Is Not Of Appr. Class
- 104: Identifier Not Declared
- 105: 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 Comparision 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 Be Neither 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 Proc/Func 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 Real 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 Or Procedure Entries  
253: Procedure Too Long  
254: Too Many Long Constants In This Procedure  
255: Too Many Errors In 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

---

## 8/ Useful Calls and Addresses

HEX ADDRESS	FUNCTION
4180	Starting address of user source program
4182	Ending address of user source program
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 (again)
418E	Address of runtime stack
4190	Ending address of runtime 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
41A0	Call address: console in (waits for key press)
41A2	Call address: console out
41A4	Call address: Inkey (input from keyboard, does not wait for key press, i.e. returns at once.)

---

## 9/ The Rules to BLOCKADE

The sample program BLOCKADE (called BLOCK) is loaded with the Tiny Pascal system. The rules are the same as the amusement hall versions. Each player tries to box in the other.

The game accepts commands from two players simultaneously. Each player moves his/her man using the keys illustrated below:

Left-Side Player

          (W) — up  
left — (A)     (D) — right  
          (X) — down

Right-Side Player

          (O) — up  
left — (K)     (I) — right  
          (.) — down

The speed is user selected between one and ten, with one being the fastest and ten the slowest. Three to four is about right for beginners.

## Appendix A/ Memory Maps

16K Version	
4060	Reserved Ram For Interpreter & Monitor
4100	Entry Points Table
4180	System Control Block
41A0	I/O Routines
41E0	Interpreter; Runtime Routines
473A	Monitor
498E	User Memory For Source & P-Code (4.5)
5BC0	Runtime Stack For Editor or Compiler (1.75)
62A0	Editor P-Code
6AB0	Compiler Table
6BD0	Compiler P-Code
7FFF	Assumed end of memory

32K/48K Version	
4060	Reserved Ram For Interpreter & Monitor
4100	Entry Points Table
4180	System Control Block
41A0	I/O Routines
41E0	Interpreter; Runtime Routines
473A	Monitor
4990	Runtime Stack For Editor or Compiler (3.25)
5690	Editor P-Code
5EA0	Compiler Table
5FC0	Compiler P-Code
73F0	User Memory For Source & P-Code

## Appendix B / Sample Programs

```
(* 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(24020):=%3FFE; (*SET CURSOR*)
        WRITE(SCORE(1)#) END;

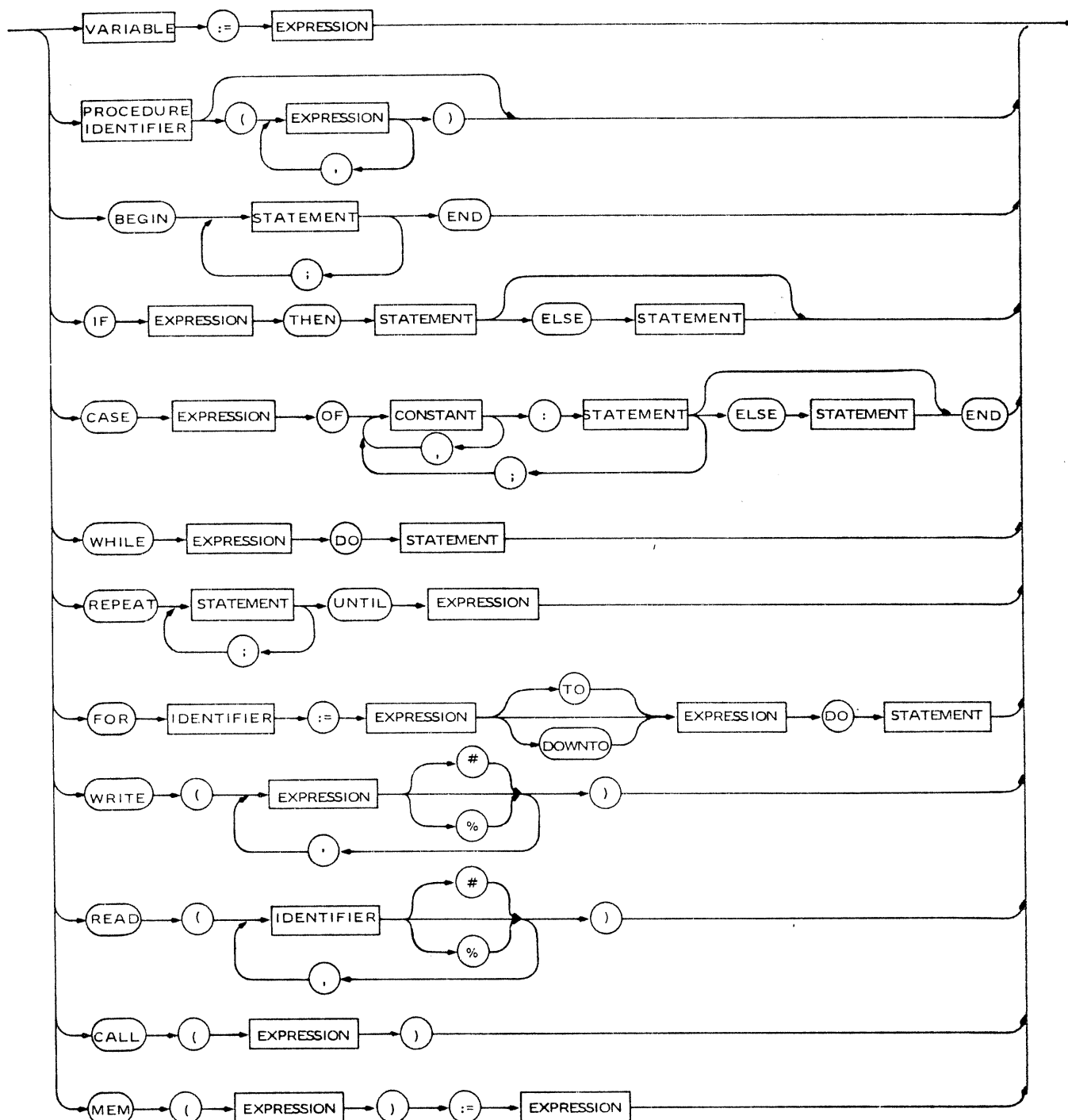
PROC BLINK;
VAR T, K, DELAY: INTEGER;
    BEGIN T:=CURSOR(I)-MOVE(I);
        FOR K:=1 TO 30 DO BEGIN
            FOR DELAY:=1 TO 100 DO;
                IF MEMW(T)=BLNK THEN MEMW(T):=MARK(I)
                ELSE MEMW(T):=BLNK
            END
        END;

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; 'L':MOVE(1):=64;
                    'J':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.

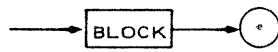
```



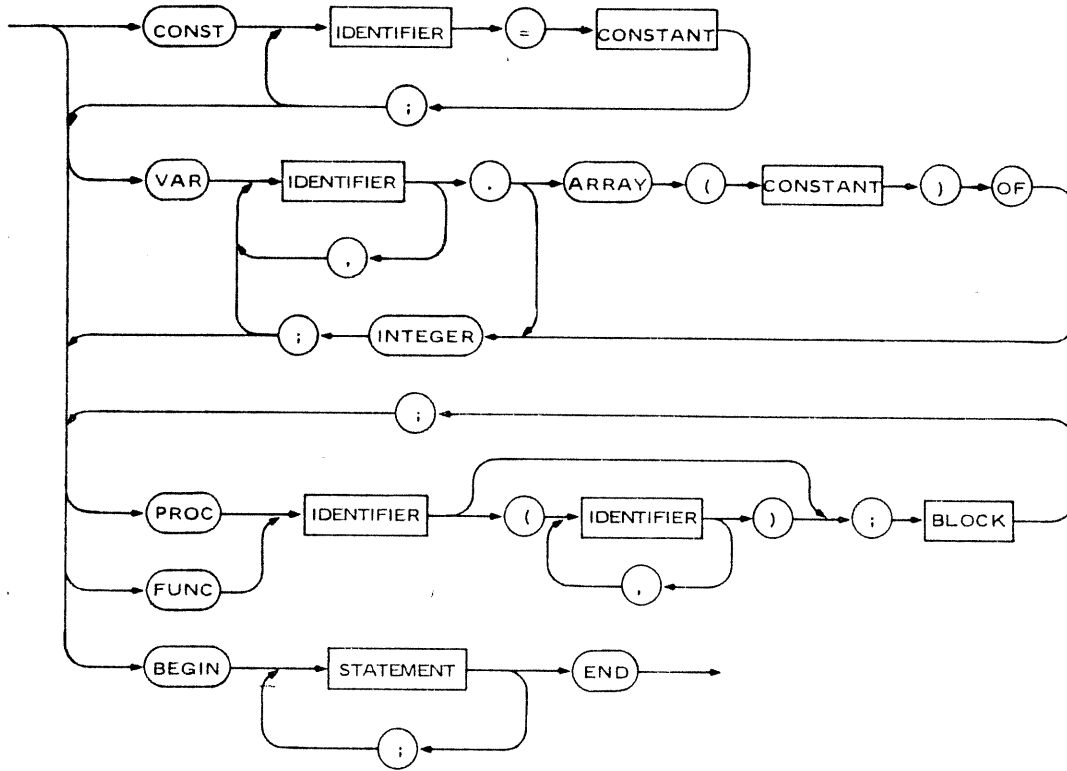
# Appendix C/ Syntax Diagrams



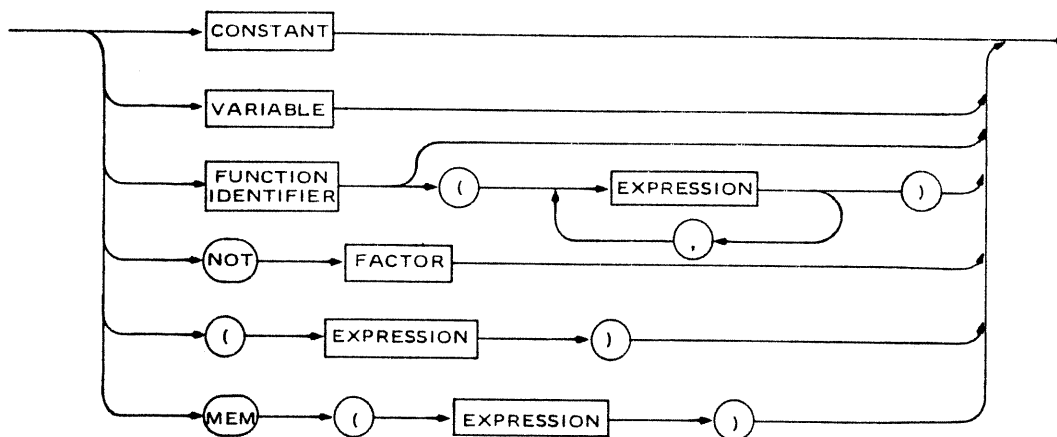
# PROGRAM

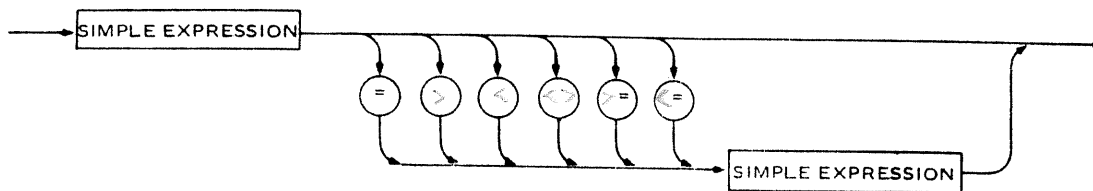


# BLOCK

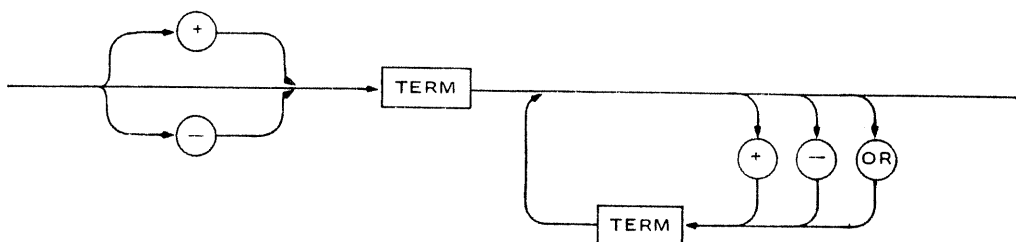


# FACTOR

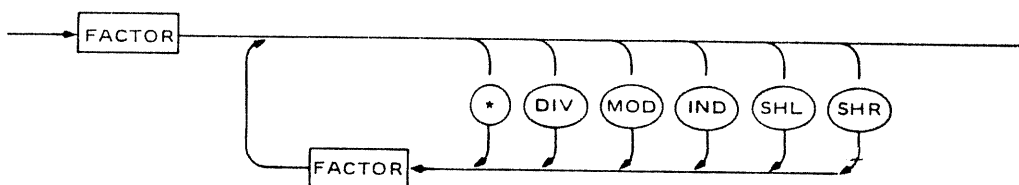




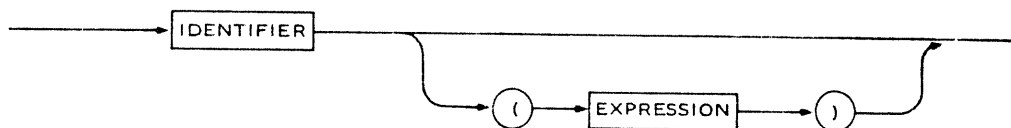
*SIMPLE EXPRESSION*



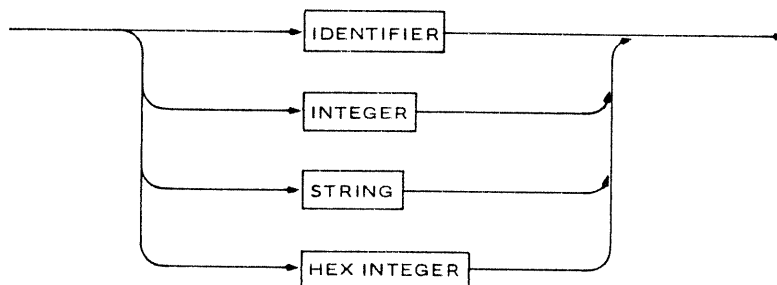
*TERM*



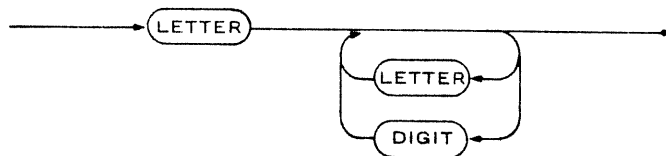
*VARIABLE*



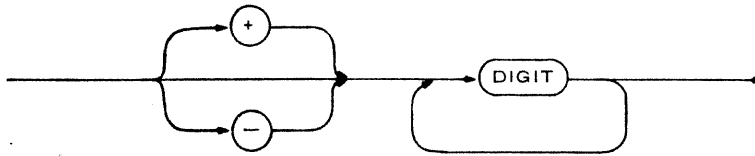
*CONSTANT*



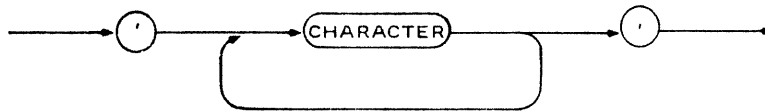
*IDENTIFIER*



### INTEGER



### STRING



### HEXINTEGER

