TRS-80® MODEL III

TINY PASCAL USER'S MANUAL

CAT. NO.
26-2020

™
Radio Shack TRS-80 SOFTWARE

CUSTOM MANUFACTURED IN USA BY RADIO SHACK, A DIVISION OF TANDY CORP.

While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information contained herein.

Please refer to the Software License on the inside front cover of this manual for limitations on use and reproduction of this Software package.

10 9 8 7 6 5 4 3 2 1

Introduction

Tiny Pascal is a cassette-based program development system. It is designed for creating, compiling, and executing Pascal programs. Tiny Pascal is a subset of the standard Pascal language.

To use the Tiny Pascal System, you need:

- TRS-80 Model III with at least 16K of RAM
- Cassette Recorder (We recommend Radio Shack's CTR-80A.)

This manual is not intended to teach you Pascal programming, but rather to show you how to use Tiny Pascal on your Model III. If you need instructions on Pascal programming, we recommend the following 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.

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

1/ Overview of the System

Tiny Pascal is a self-contained system for creating, compiling, and running Pascal programs on your Model III computer. Once you have loaded Tiny Pascal from the cassette, you can use all three of the "sub-systems":

Monitor Provides run-time support, checks for errors, and provides the necessary utilities for saving programs and loading them to and from the cassette tape.

Compiler Translates your Pascal source program into "p-code" which you then can execute through the Monitor. The Compiler also checks your source code for syntax errors.

Editor Lets you create or modify Tiny Pascal source programs.

When you load the Tiny Pascal System from the tape, all three sub-systems are loaded into RAM. We also have included a sample program on the cassette tape. This, too, loads into RAM when you load the system.

If you have a disk drive on your Model III, you can transfer the Tiny Pascal program to a diskette. See Appendix E for details.

Overview of this manual

Chapter 2 shows you how to load the Tiny Pascal System and how to create, compile, and run a program. Chapters 3 through 5 discuss the three sub-systems in detail--what they do and how to use them. Chapter 6 considers the specific aspects, limitations, and enhancements of the Pascal language.

The appendices contain a list of error codes, syntax diagrams, program listings, and other useful information.

Terms and Notations

For clarity and brevity, we often use the following terms and notations in this manual:

<KEYBOARD CHARACTER>
indicates the key you must press.

lowercase underline
represents words, letters, values, or other characters you supply.

UPPER CASE and punctuation
indicate material that you must enter exactly as it appears (unless told otherwise by the text) or material that you see on your computer's video display.

2/ Starting Up

In this chapter, we show you step by step how to load the Tiny Pascal System, enter the Editor, and run a program. In later chapters, we go into detail on each aspect of the System.

Loading the System
To load the Tiny Pascal System, follow these steps. If you can't get the program to load properly, adjust the volume and try again.

1. Turn on your Model III. (If your Model III has a disk drive, override the disk startup by holding down <BREAK> and pressing the reset button at the same time.) The computer displays the prompt,

   Cass?

2. Answer the prompt by pressing <ENTER>. Basic then asks you for the memory size. Again, respond by pressing <ENTER>.

3. The computer displays the copyright and the Ready prompt. Type SYSTEM <ENTER> to reach the system level. The computer prompts you with an asterisk and question mark (*?).

4. Make sure your Tiny Pascal tape is rewound to the beginning of the tape. Then press the PLAY button on the cassette player and type PASCAL <ENTER>.

5. The tape begins to load in memory, and a blinking asterisk (*) appears in the upper right hand corner. The load takes about a minute-and-a-half.

5. Once the tape has loaded, the computer again displays the asterisk and question mark (*?). Press </> <ENTER>. Now, the computer enters the Tiny Pascal Monitor and displays the memory size and the Monitor prompt,

   Tiny Pascal:

Creating a Program

When you load your Tiny Pascal System, you also load the sample program. To look at this program, type:

```
EDIT <ENTER>
```

You are now in the Tiny Pascal Editor. The Editor prompts you with a "greater than" symbol (>). Type:

```
P* <ENTER>
```

to display the program. Now, you could delete the program by entering the command D*. Then you could enter your own program. For now, however, just return to the Monitor by typing:

```
Q <ENTER>
```

You should see the Tiny Pascal: prompt again.

Compiling the program

To run a Tiny Pascal program, you must first compile the source code into a machine readable form called "p-code." If your Model III has 32K or 48K of RAM, type:

```
COMP <ENTER>
```

or if you Model III is equipped with 16K of RAM, type:

```
COMP -S <ENTER>
```

This instruction tells the Compiler to create the p-code from the source code in the work file. The Compiler displays each line of the program as it processes the file.

Had there been an error in the source code, the compilation would have aborted, and the System would have entered the Editor. However, there are no errors in the program on your cassette, so when the Compiler is finished, you should see the Tiny Pascal: prompt again.

unning the Program

ow that you have compiled the source code and created the -code, you can run the program by typing:

```
    RUN <ENTER>
```

rocedure PGM1 creates an interesting display on your screen. rocedure PGM2 is a video game. "Appendix F" tells you how to lay the game.

3/ The Monitor

The Tiny Pascal Monitor gives run-time support to the entire system. It also lets you load and store your source and compiled programs, via the cassette tape. You invoke the Compiler and Editor from the Monitor, also.

Monitor Commands

After you load and start the system, you enter the Tiny Pascal Monitor. The Monitor prompts you with the message:

```
Tiny Pascal:
```

Now you can enter any of the Monitor commands, which are:

EDIT — Enters the edit mode. The Editor uses a "work file" in memory. If you haven't loaded a source file, the Editor creates the work file.

COMP — Compiles the source code in the work file. The compiled "p-code" locates elsewhere in user memory. Should an error occur, the compilation aborts and the System enters the Editor at or near the error line.

COMP -P — Compiles the source code in the work file but produces no p-code. This is useful for checking for syntax errors.

COMP -S — Compiles the source code in the work file and overwrites the source code with p-code. This is useful for compiling large programs.

RUN — Executes the program. Execution begins right away, if you have the p-code in the work file. The Compiler creates a new p-code before execution, if any of the following has occurred:

- You haven't compiled the source code.
- The last compilation caused an error.
- You have modified the source code since the last compilation.

SAVE filename — Saves the current work file on the cassette and names the file filename. If you have the source code in the work file, the System saves only the source code. If you have no source code, but do have a valid p-code, the System saves the p-code.

LOAD filename — Loads a source code or p-code from a cassette file named filename. This command destroys the old source program and the p-code in the work file.

CALL — Calls a machine language subroutine. The Monitor prompts you for the decimal address of the routine.

POKE — Loads a byte into memory. The Monitor prompts you for the decimal memory address and byte value.

Note that with the COMP -S command you may choose to overwrite your source code with the compiled p-code. Be sure to save the source code before you issue such a command.

The filename can have up to six characters. Remember that once you write a file to the tape, there is no way to check for its filename so you must load it with the exact name with which you stored it. If you accidentally type the wrong filename when loading a file, the Tiny Pascal System displays the name of the cassette file it read in, but may not return to the Monitor. If this happens, you must reset and reload the System.

4/ The Editor

The Editor enables you to create and modify source programs. It is line-oriented, but since Pascal doesn't use line numbers, none are stored as part of the source code, although the Editor displays the current line number in the upper right hand corner of the screen.

No line can have more than 130 characters. The total number of lines allowed is limited only by your Model III's memory, however, you cannot access lines over 999 directly by line number.

Start the Editor by typing in:

```
EDIT <ENTER>
```

from the Monitor. The Editor prompts you with a "greater than" symbol (>). Now you can enter any of the Editor commands.

You can enter each command in upper or lower case. Some commands also let you specify a number or a string of characters. The number can be any integer from 1 to 999. The string can have from 1 to 62 characters.

If you enter an invalid command, the Editor responds with the message ILLEGAL.

Editor Commands

D — Deletes the current line.

Dnumber — Deletes the number of lines specified, starting with the current line.

D* — Deletes the entire file.

Fstring — Finds the first occurrence of the string, starting with the current line. If you don't specify a string, the Editor uses the last string specified.

I	Inserts lines after the current line. The Editor prompts you with a question mark (?). To end the insert mode, press <ENTER> on a blank line.
I0	Begins the insert mode at the top of the file.
N	Moves down one line.
Nnumber	Moves down the number of lines specified.
N*	Moves down to the last line of the file.
P	Prints the current line.
Pnumber	Prints the number of lines specified, starting with the current line.
P*	Prints the entire file.
Q	Returns to the Monitor after displaying the current file status.
R	Replaces the current line. The System prompts you with the insert prompt, a question mark (?).
S	Displays the current file status, including the number of lines, number of bytes, file location in memory, and the number of free bytes remaining in user memory (rounded off to the nearest ten bytes).
U	Moves up one line.
Unumber	Moves up the number of lines specified.
U*	Moves up to the first line of the file.
X	Extends the current line. The System displays the current line and positions the cursor at the end of the line. You may add characters or backspace with the <left arrow> to make changes.

.number Moves to the line number specified.

.* Moves to the last line of the file.

<BREAK> If pressed during execution of a program, causes a pause in the program. Pressing <BREAK> twice returns you to the Monitor.

<right arrow> Tabs three spaces.

<left arrow> Backspaces once for a space or three spaces for a tab.

<up arrow> Moves up one line.

<down arrow> Moves down one line.

<ENTER> Ends the current line. If you type <ENTER> on a blank line, the Editor leaves the insert mode.

Note: If a MEMORY FULL error occurs while you are editing or inserting, the source file is too big. You might be able to solve this problem by deleting excess spaces and tabs.

5/ The Compiler

The Tiny Pascal Compiler translates your Pascal source code into an intermediate form called "p-code". The runtime Monitor translates this p-code into the actual machine commands. This compiled form runs from four to eight times faster than a similar BASIC program.

Compiling the program

After you have created a source program with the Editor, or have loaded a source program from your cassette player, you can compile it into p-code by typing:

```
COMP <ENTER>
```

If you wish, you may follow COMP with one of two options. The first option causes the compiler to generate no p-code. You can use this to check your syntax when you write programs. To use this option, type:

```
COMP -P <ENTER>
```

The second option causes the generated p-code to locate over the top of your source code in memory. You might use this option if you have a large program, because sometimes the program doesn't fit into the space normally assigned for p-code. The source code that was stored in memory is destroyed, so be sure to save your source code before you compile it. To use this option, type:

```
COMP -S <ENTER>
```

If during compilation of a program the Compiler runs out of memory to store the p-code, you may get a syntax error. If you cannot pin down an error in your code, this may be the problem. Try compiling the code with the -S option, or else try removing any unnecessary code.

The Complier Specifications

The Tiny Pascal language is a subset of standard Pascal. Essentially, the syntax of Tiny Pascal is the same as the full language. In Appendix D, you'll find syntax diagrams and notes to help you with the language.

Since we intend for this manual to be an explanation of the limits and special features of Tiny Pascal, we won't present the entire language, but rather review some of the essential points. If you need a more thorough review of Pascal, see the references in the "Introduction."

Appendix A/ Useful Addresses

Address			
Decimal	Hex	Size	Function
19200	4B00	2 bytes	Starting address of user source program
19202	4B02	2 bytes	Ending address of source program
19204	4B04	2 bytes	Number of lines of the source program
19206	4B06	2 bytes	Ending address of user p-code
19208	4B08	2 bytes	Address of Compiler p-code
19210	4B0A	2 bytes	Address of Monitor/Editor p-code
19212	4B0C	2 bytes	Address of currently running program
19214	4B0E	2 bytes	Ending address of user memory
19216	4B10	2 bytes	Address of p-code interpreter
19218	4B12	2 bytes	Address of Compiler table
19220	4B14	2 bytes	Line number where compiler error occured
19222	4B16	1 byte	Flag indicating compiler error
19223	4B17	1 byte	Flag indicating to generate p-code
19224	4B18	1 byte	reserved
19225	4B19	1 byte	Monitor state
19226	4B1A	1 byte	Flag indicating p-code is executable
19227	4B1B	1 byte	Printer on/off flag

Note: You may turn the printer flag on and off (1 and 0, respectively) and change the user memory size. Whenever you turn on the printer flag, it outputs all information to both the video display and the printer. You might want to change the memory size in order to protect your machine-language subroutines. You never should modify any of the other system controls.

Appendix B/ Memory Map

4400	Entry points table for p-code interpreter
4600	Tiny Pascal p-code interpreter
4B00	System Control Block
4B20	Keyboard and video routines
4B70	Cassette I/O routines
4CA0	Monitor/Editor p-code
5780	Compiler table
58A0	Compiler p-code
67F0	User memory for source code and p-code

Appendix C/ Sample Programs

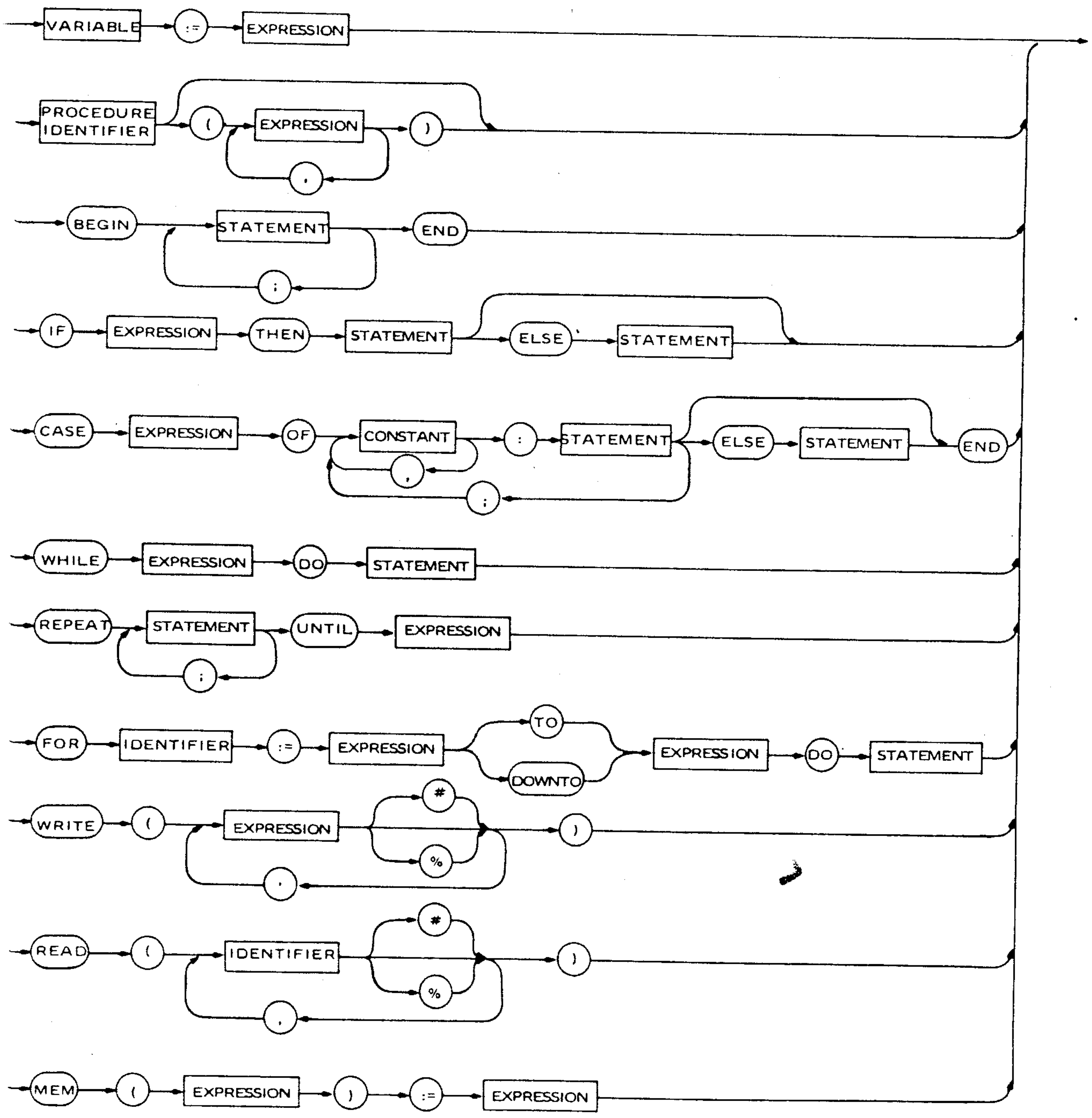
```
(* TINY PASCAL V-2.0 SAMPLE PROGRAMS *)
  VAR WHICH:INTEGER;

PROC PGM1; (*HILBERT CURVES   BY K.M. CHUNG  04/79*)
(* LAST MOD 10/17/81  H. YUEN *)
  CONST N=4; H0=32;
  VAR I,H,X,Y,X0,Y0:INTEGER;

  PROC GMOVE(DIR);
   BEGIN CASE DIR OF
    1: BEGIN FOR Y:=Y   TO   Y+H DO PLOT(X,Y,1); Y:=Y-1 END;
    2: BEGIN FOR X:=X   TO   X+H DO PLOT(X,Y,1); X:=X-1 END;
    3: BEGIN FOR Y:=Y DOWNTO Y-H DO PLOT(X,Y,1); Y:=Y+1 END;
    4: BEGIN FOR X:=X DCWNTO X-H DO PLOT(X,Y,1); X:=X+1 END END
   END;
   PROC HILBERT(R,D,L,U,I);
   BEGIN IF I>0 THEN BEGIN
     HILBERT(D,R,U,L,I-1); GMOVE(R);
     HILBERT(R,D,L,U,I-1); GMOVE(D);
     HILBERT(R,D,L,U,I-1); GMOVE(L);
     HILBERT(U,L,D,R,I-1) END
   END;

BEGIN (*PGM1*)
   WRITE(15,28,31,13,' HILBERT CURVES OF ORDERS 1 TO 4');
   FOR I:=1 TO 12 DO WRITE(13);
   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;
      HILBERT(2,3,4,1,I)
   UNTIL I=N; READ(I)
END;
```

```
ROC PGM2; (*BLOCKADE.  BY K.M.CHUNG   4/26/79*)
 * LAST MOD 10/14/81  H. YUEN *)
  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 160 DO;
         IF MEMW(T)=BLNK THEN MEMW(T):=MARK(I)
            ELSE MEMW(T):=BLNK
         END
   END;

EGIN WRITE(28,31,'SPEED(1-10)');
   READ(SPEED#); SPEED:=SPEED*10+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 ANY KEY TO START*)
      REPEAT I:=1-I;
         FOR J:=1 TO SPEED DO
            CASE INKEY OF
               'W':MOVE(0):=-64; 'O':MOVE(1):=-64;
               'D':MOVE(0):=2;   ';':MOVE(1):=2;
               'X':MOVE(0):=64;  '.':MOVE(1):=64;
               'A':MOVE(0):=-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; READ(I)
ND;
```

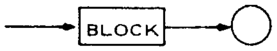
```
BEGIN (*..MAIN..*)
 REPEAT
  WRITE(28,31,14,13,13,'SAMPLE PROGRAMS',13,13);
  WRITE(9,'1: PLOT HILBERT CURVES',13);
  WRITE(9,'2: THE GAME OF BLOCKADE',13);
  WRITE(13,9,'9: QUIT',13);
  READ(WHICH);
  IF WHICH='1' THEN PGM1 ELSE IF WHICH='2' THEN PGM2
 UNTIL WHICH='9'
END.
1198 BYTES CODE. (72DF-778C)
```

Appendix D/ Syntax Diagrams and Notes

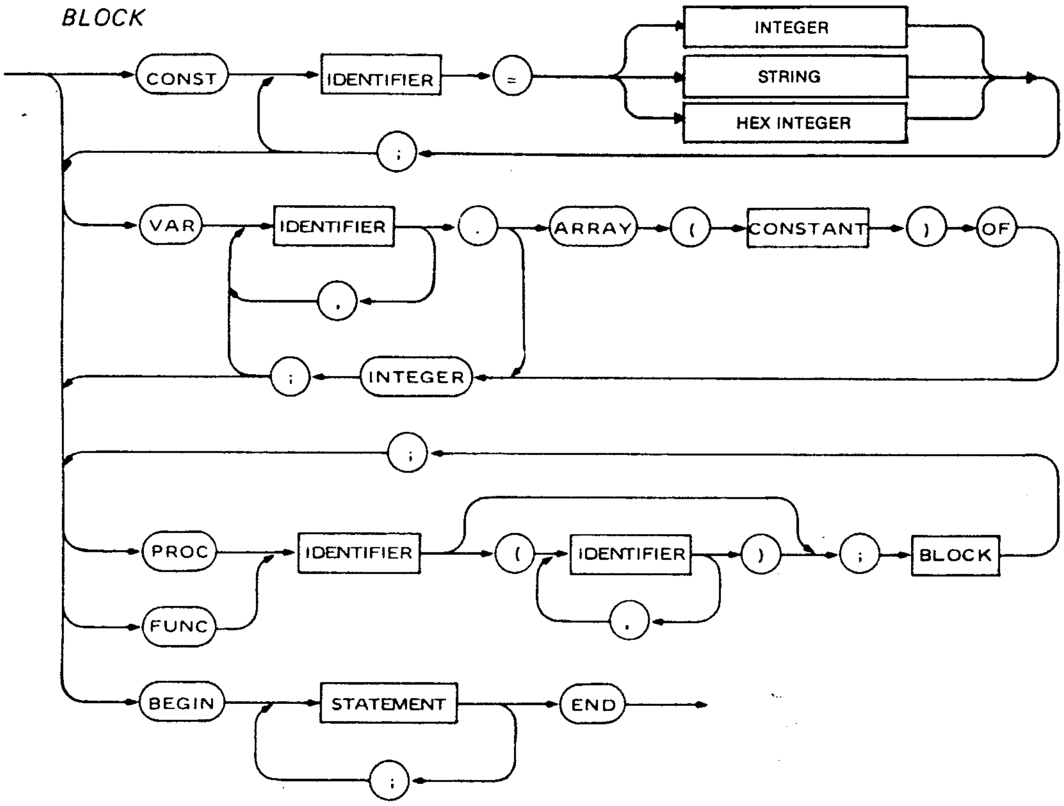
EMENT



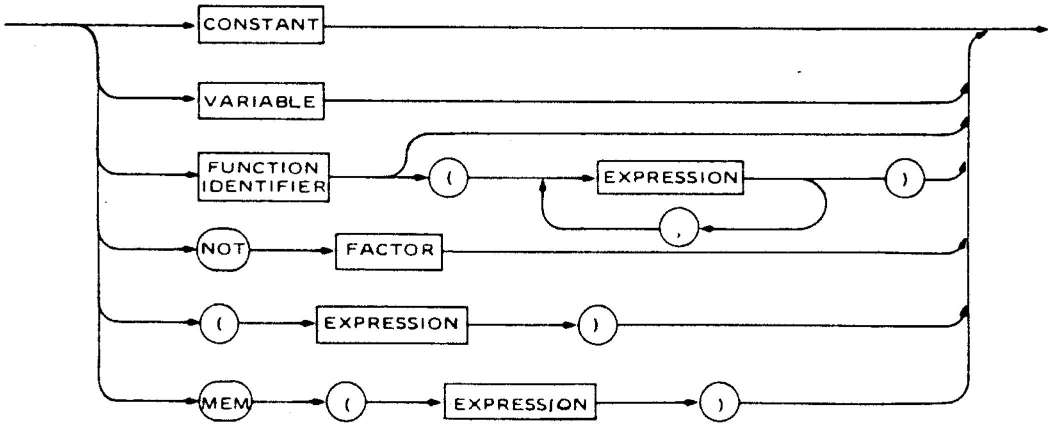
PROGRAM



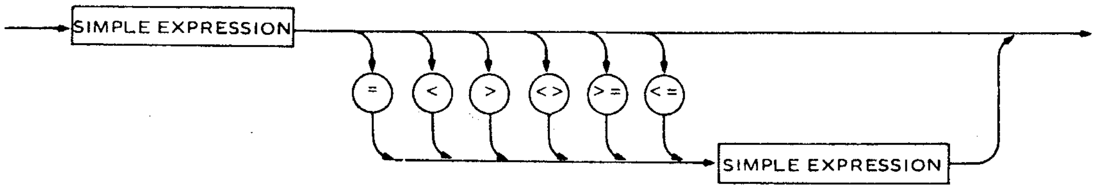
BLOCK



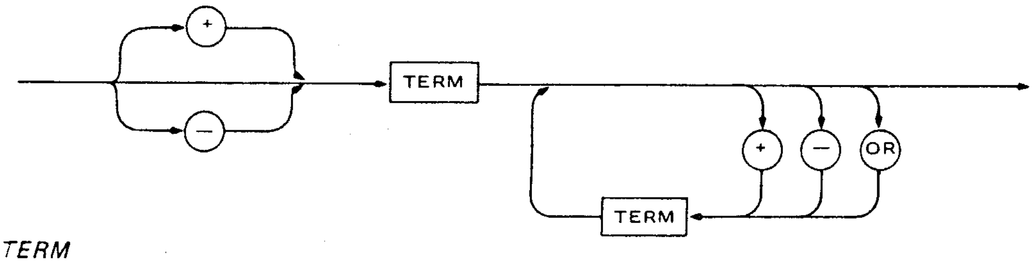
FACTOR



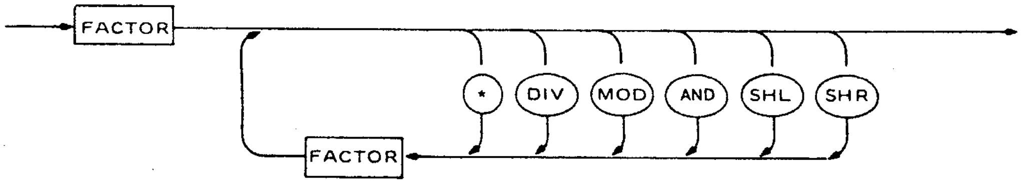
EXPRESSSION



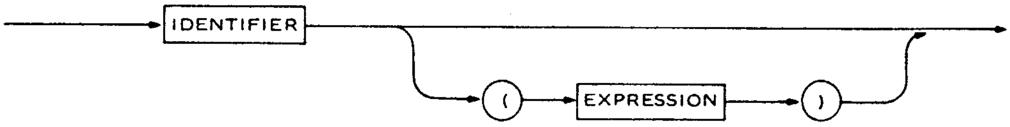
SIMPLE EXPRESSION



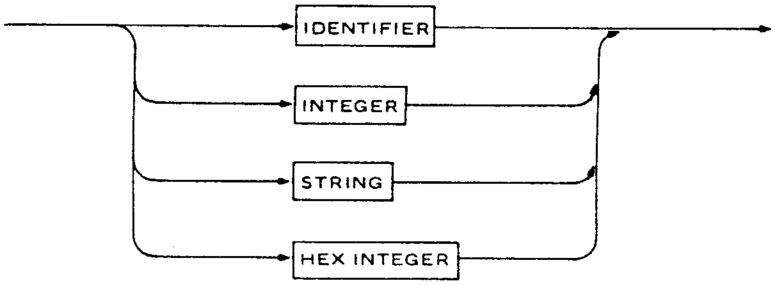
TERM



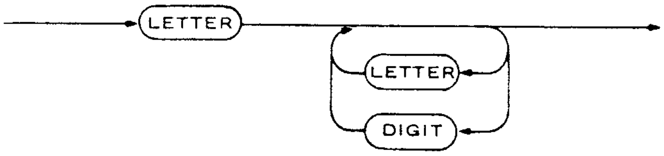
VARIABLE



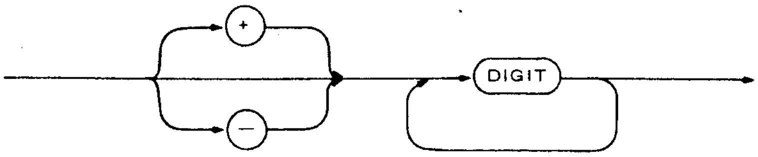
CONSTANT



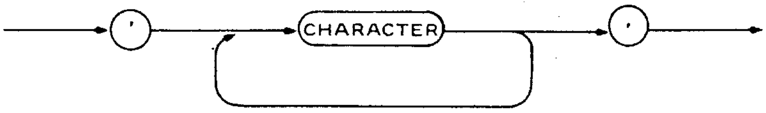
DENTIFIER



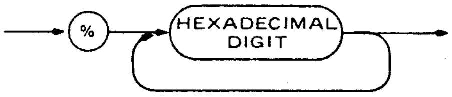
INTEGER



STRING



HEXINTEGER



Syntax Notes

Operators

Use a colon and equal sign (:=) to assign variables and an equal sign (=) for conditional statements.

Use a semi-colon (;) to separate statements, but not to end statements.

You may use both arithmetic and logical (Boolean) expressions. OR, a Boolean operator, has the same precedence as the plus sign (+) and the minus sign (-), arithmetic operators. AND has the same precedence as the asterisk (*) and DIV. Both AND and OR have precedence over the following operators; the equal sign (=), the "greater than" sign (>), and the "less than" sign (<).

You must use parentheses, the symbols (), rather than brackets, the symbols [].

Enclose comments within parentheses and asterisks, the symbols (* and *).

Identifier usage

You may use upper or lower case for your identifiers, and the Compiler recognizes the difference between upper and lower case. Each identifier must begin with a letter. You may follow this with letters or numbers, but only the first four characters are significant.

You must declare your identifiers before you use them. If you declare them more than once, the last declaration applies. Do not declare the formal parameters of a procedure inside the procedure.

The System passes parameters to procedures by value (the System makes a copy of the parameter for use by the procedure).

The scope of a variable is within the procedure that defines it.

Data types

Tiny Pascal supports 16-bit signed integers and one-dimensional integer arrays. The subscripts for arrays begin at zero and are not checked for out of range at runtime. To assign values to the elements of the array, you must assign each member individually.

Prefix hexadecimal constants with a percent sign (%). Example: %2400.

Enclose strings within single quotation marks ('). When you use a string in an expression, a CONST declaration, or CASE label, it has the value of the Ascii code of the first character of the string. When you use a string in a WRITE statement, it has the value of the entire string.

Input/Output

The READ and WRITE statements are character-oriented. This means that you can input or output more than one character with a single statement. To read or write a decimal number, follow the variable name with a number sign (#). To read or write the number as hexadecimal, follow the variable name with a per cent sign (%). Also, you must end integer or hex input by pressing <ENTER> or <SPACEBAR>.

To begin writing on a new line, you must output the ASCII code for carriage return. In decimal, this is 13. In hex, it is %0D. For example, WRITE(13) sends a carriage return to the screen. The other screen functions, such as screen clear, use the same ASCII characters as BASIC. For example, WRITE(28,31) clears the screen.

Logical Operations

In a logical expression such as IF, WHILE, or REPEAT, the condition is true if the least significant bit is one (in other words, if the expression evaluates to an odd number).

The relational operator symbols, such as < and =, always produce a value of zero or one.

unctions, Procedures, and Operators

hese are the built-in functions, procedures, and operators. Be ure to include any required punctuation when typing them.

BS(number)
eturns the absolute value of the number specified.

ALL(address)
umps to a user-defined subroutine beginning at address. The ubroutine must save all registers upon entry, restore all egisters on exit, and return from the subroutine with the ollowing instructions:

```
    INC  DE
    INC  DE
    RET
```

OMP(address1,address2,number)
ompares strings that are the specified number of bytes long, eginning at address1 and address2. If the strings are the ame, the function returns a one. If not, it returns a zero.

umber1 DIV number2
erforms truncated integer division of number1 by number2. xample: 27 DIV 5 = 5.

ILL(address,number1,number2)
ills a block of number1 bytes with the lower order byte value umber2, at the memory address specified.

NKEY
eturns the input character from the keyboard with no wait eriod. It returns a zero, unless you have typed something.
NP(number)
Returns the input value from the port named by the number.

MEM(address)
Returns the byte value at the memory address specified. It can appear on either side of the assignment sign.

MEMW(address)
Returns the 16 bit value at the memory address specified. The low order byte returns to the address, and the high order byte returns to address + 1. The value can appear on either side of the assignment sign.

number1 MOD number2
Performs modulo arithmetic of number1 on number2. Example: 27 MOD 2 = 2.

MOVE(address2,address1,number)
Moves a block of the specified number of bytes from memory address1 to memory address2.

OUTP(number 1,number2)
Outputs the byte value of number2 to the port named by the number1.

PLOT(x,y,number)
Plots a graphics block on the screen at horizontal point x and vertical point y. x can range from 0 to 127, and y can range from 0 to 47. The point is "set" if the number is odd and "reset" if it is even.

POINT(x,y)
Tests whether the graphics block at the horizontal position x and the vertical position y is set. If the point is set, the function returns a one. If you reset the point, the function returns a zero.

number1 SHL number2
Logically shifts number1 left number2 bits. Example: 27 SHL 2 = 108.

number1 SHR number2
Logically shifts number1 right number2 bits. Example: 27 SHR 2 = 6.

SQR(number)
Returns the square of number.

Appendix E/ 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

Appendix F/ How to Play Blockade

The sample program contains BLOCKADE (in procedure PGM 2) and 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 using the keys illustrated below:

```
        Left-Side Player                  Right-Side Player

              <W>--up                           <O>--up

    left--<A>      <D>--right        left--<K>      <;>--right

              <X>--down                         <.>--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 G/ Converting Tiny Pascal to Diskette

If you have a disk drive, you might want to convert your tape version of Tiny Pascal so that you can load and run it off a diskette. To convert the program, follow these steps:

1. Insert a system diskette into Drive 0. Insert the Tiny Pascal cassette into the cassette recorder, and make sure it is completely rewound and the "Play" key is down. Press the Reset button on your Model III.

2. After TRSDOS Ready apprears on your screen, type:

   ```
   TAPE (S=T D=D) <ENTER>
   ```

   Press <H> in response to the Cass? question. Your Model III will display:

   ```
   Device = Tape to Disk
   Press ANY key when Cassette ready
   ```

3. Press <ENTER>. As the computer transfers the Tiny Pascal System to the diskette, two asterisks will flash in the upper right hand corner of the screen. When it is finished, TRSDOS Ready reappears.

4. Type:

   ```
   RELO PASCAL/CMD (ADD=6400) <ENTER>
   ```

5. After TRSDOS Ready reappears, type:

   ```
   LOAD PASCAL/CMD <ENTER>
   ```

6. When TRSDOS Ready reappears, type:

   ```
   DEBUG <ENTER>
   ```

   The screen fills with numbers and letters. This is the DEBUG program.

7. Press <M>. DEBUG prompts you with M ADDRESS? =. Type 93B0 and then press the spacebar once. Now type in the following numbers (with no spaces between):

   ```
   F3 21 00 64 11 00 44 01 B0 2F ED B0 C3 00 46
   ```

   Double check what you have entered, and if it is correct, press <ENTER>. If it is not correct, you can use the arrow keys to space over to the incorrect data, type in the correction, and press <ENTER>.

8. Leave the DEBUG program by pressing <Q> for Quit. When TRSDOS Ready reappears, type:

   ```
   DUMP PASCAL (START=6400,END=93BE,TRA=93B0) <ENTER>
   ```

Now to run the Tiny Pascal Program, simply type PASCAL <ENTER> from TRSDOS Ready.

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE
PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A
RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. Except as provided herein, **RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".

NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA
280-316 VICTORIA ROAD
RYDALMERE, N.S.W. 2116

BELGIUM
PARC INDUSTRIEL DE NANINNE
5140 NANINNE

U. K.
BILSTON ROAD WEDNESBURY
WEST MIDLANDS WS10 7JN

982

Printed in USA