

225  
Larry Noonan

---

# Basic BASIC-English Dictionary

for the APPLE®, PET®, and TRS-80®

---

**Basic  
BASIC-English  
Dictionary  
for the Apple<sup>™</sup>, PET<sup>™</sup>  
and TRS-80<sup>™</sup>**



**Basic  
BASIC-English  
Dictionary  
for the Apple<sup>™</sup>, PET<sup>™</sup>  
and TRS-80<sup>™</sup>**

**Larry Noonan**



**dilithium Press**  
Beaverton, Oregon

## DEDICATION

To my daughter Kristyl, whose life will be influenced and enriched by the computer age.

© Copyright, dilithium Press, 1982

All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the publisher.

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Noonan, Larry, 1946-

Basic BASIC English dictionary for the Apple, PET, and TRS-80.

1. Basic (Computer program language) 2. Apple II (Computer) – Programming. 3. PET (Computer) – Programming. 4. TRS-80 (Computer) – Programming. I. Title.

QA76.73.B3N66

001.64'24

81-19507

ISBN 0-918398-54-1

AACR2

Cover: Marty Urman

Printed in the United States of America

dilithium Press

11000 S.W. 11th Street

Beaverton, Oregon 97005

\* Apple is a trademark of Apple Computer Inc. PET is a trademark of Commodore Business Machines. TRS-80 is a trademark of Tandy Corporation.

# PREFACE

Our society is now in the midst of the computer age. Computers influence all of our lives to a greater or lesser extent. Businesses use computers to bill us and issue our paychecks; doctors and lawyers use them to keep records on their patients and clients; pilots use them to help navigate their planes; shopkeepers use computers to manage their inventory records; NASA uses them for space exploration. Computers are used widely by hospitals, banks, newspapers, schools and research institutions. They are even used in the home to regulate appliances, to play video games, and to keep personal records such as addresses, Christmas card lists, recipes and the family budget.

We hear more and more often that people should become "computer literate" so that computers may be used and enjoyed by everyone. The *Basic BASIC-English Dictionary* tackles two aspects of computer literacy: computer language and programming. The BASIC language of the Apple, PET, and TRS-80 microcomputers are explained in detail, in the form of dictionary definitions, with a generous number of examples. The *Basic BASIC-English Dictionary* makes it possible to translate one BASIC to another, quickly and easily. Representations of the three computers help you compare the commands, statements, functions and operators of each computer's BASIC.



# CONTENTS

Introduction .....	1
Part I— Alphabetical listing of BASIC statements, functions, and commands .....	3
Alphabetical listing of operators .....	91
Part II— Summary of terms .....	103
Appendix A— ASCII Codes .....	113
Appendix B— Abbreviations of BASIC Words .....	119
Appendix C— Graphics .....	121
Appendix D— Reserved Words .....	139
Appendix E— Boolean Operators .....	145





# INTRODUCTION

Three computers are emerging as educational tools in many school systems. These same computers seem to monopolize much of the written material found in computer magazines. They are the Apple II, the PET, and the TRS-80.

All of these computers are used in my school. After using them for some time, it became obvious that there was a need for something that would help students, teachers and other computer users recognize the similarities and differences among the three BASIC languages used by these computers.

Too often it has been my experience to find a terrific program in a magazine that was just what I wanted—to play a game, teach a concept to my students, or help in my personal finances—but written in a BASIC different from the *dialect* used by my own computer. This book is an attempt to help you understand the three BASIC dialects, and thus make better use of your personal computer.

The book is divided into two parts. Part I is a dictionary of the more commonly used commands, functions, statements and operators used in the three BASIC dialects. If you come across a command, function, statement or operator that you do not understand, look it up in Part I. You will find the English definition with each new term printed next to it; if it is not used by that computer, the translation is printed instead. (See the example on the next page.)

Part II is a summary of terms. This part may be used as a quick reference when more detailed information is not needed.

The similarities and differences of the ASCII codes, abbreviated key words, reserved words, graphics and Boolean operators are covered in several appendices.

### EXAMPLE

**GET** statement

↑ word    ↗ how it is used (command, statement, function or operator)

A definition of the word appears here.

Also see: BASIC terms with similar meanings are listed here.

*Apple*    • The GET\$ statement is used only in Applesoft BASIC.  
GET\$      This sentence tells you that on the Apple II computer,  
            only Applesoft, and not Integer BASIC, uses the state-  
            ment GET\$.

*PET*        • The PET personal computer uses the statement GET.  
GET

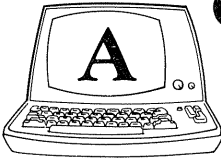
*TRS-80*    • The TRS-80 does not use GET, but it does have a state-  
INKEY\$    ment that means the same thing, or nearly the same  
            thing: INKEY\$. There will also be a sentence explaining  
            that only Level II, and not Level I, BASIC uses the IN-  
            KEY\$ statement.

If there are any peculiarities that appear in only one or two of the computers, it is written beside that computer's name.

If the word ANSI appears next to the word, it means that the word has a defined syntax in the ANSI standard for a minimal BASIC, ANS X360-1970.

# Part I

## Alphabetical Listing of BASIC Statements, Functions, and Commands



### **A.** function

ABS gives the absolute value of the specified number.

Format:  $10 A = \text{ABS}(N)$  means that A equals the absolute value of the number N.

The number returned is always positive.

EXAMPLE:  $\text{ABS}(-171)$  is 171.

Also see: ABS, AT

*Apple*     • The ABS function is used in both Applesoft and Integer  
ABS             BASIC.

*PET*         • The ABS function is used in PET BASIC  
ABS

*TRS-80*     • A. is used in Level I BASIC as an abbreviation of ABS  
A.             and AT. ABS and/or AT is used in Level II Basic.

### **ABS** (ANSI) function

ABS gives the absolute value of the specified number.

Format:  $10 A = \text{ABS}(N)$  means that A equals the absolute value of the number N. The number returned is always positive.

Example:  $\text{ABS}(-171)$  is 171.

*Apple*     • The ABS function is used in both Applesoft and Integer  
ABS             BASIC.

*PET*         • The ABS function is used in PET BASIC.  
ABS

*TRS-80*     • ABS is used only in Level II BASIC. In Level I BASIC A.  
ABS             may be used.

**AND** operator (See page 91.)

4 / ' (apostrophe)

' (**apostrophe**) (ANSI) statement

The ' (apostrophe) is used as a short form for the REM statement on the TRS-80.

On all three computers PRINTCHR\$(39) can be used to print an apostrophe on the screen.

*TRS-80* • The ' (apostrophe) is used in both Level I BASIC and Level II BASIC.

**ASC** function

The ASC function returns the integer ASCII code for the letter, number or character used as the argument of the function.

Format: ASC(A\$) returns the integer ASCII code for the first character of A\$. If A\$=A, then the decimal number 65 will be returned because 65 is the ASCII code for A.

ASC can be used with POKE to POKE a value directly into screen memory as shown below.

Also see: CHR\$, Appendix A for ASCII codes, and Appendix C for graphics.

*Apple* • The ASC function is used in both Applesoft and Integer  
ASC BASIC.

*PET* • The ASC function is used in PET BASIC. Appendix I  
ASC lists the special ASCII characters on the PET.  
POKE32970, ASC("\*\*") will poke a star or asterisk into  
screen memory address 32970.

Note: The cursor does not move when graphics or other characters are poked into screen memory.

*TRS-80* • The ASC function is used only in Level II BASIC.

ASC 5 A\$="\*\*"  
10 POKE15500,ASC(A\$) pokes a star or asterisk into  
memory location 15500.

\* (**asterisk**) (ANSI) operator (See page 91.)

**AT** function

The AT function is used after PRINT to indicate the starting location for the PRINT statement.

Example: 10 PRINT AT 10, "THE NUMBER IS"  
40 PRINT AT 40; "THANK YOU"

*Apple* • The AT function is used in both Applesoft and Integer  
AT BASIC.

*TRS-80* • The AT function is used in Level I BASIC only. In Level  
. AT II BASIC AT is replaced by the @.

@ (**AT**) operator (See page 92.)

**ATN** (ANSI) function

The ATN function calculates the arctangent of a number specified in radians.

Format:  $A = \text{ATN}(x/y)$

*Apple* • The ATN function is used in Applesoft BASIC, but not  
 ATN in Integer BASIC. Programs with ATN cannot be translated directly into Integer BASIC.

*PET* • The ATN function is used in PET BASIC.  
 ATN

*TRS-80* • The ATN function is used only in Level II BASIC.  
 ATN

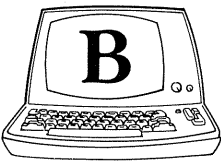
**AUTO** command

The AUTO command automatically provides numbers for the program lines.

Example: AUTO 10,5 will start the program lines at 10, and increment each line number by 5.

*Apple* • The AUTO command is used only in Integer BASIC.  
 AUTO Typing AUTO and pressing RETURN displays a 10, and puts the computer in the automatic line number mode. Typing CONTROL X (CTRL X), then MAN, returns the computer to the manual insertion of program lines mode.

*TRS-80* • The AUTO command is used only in Level II BASIC.  
 AUTO Typing AUTO and pressing the ENTER key places the computer in the automatic line numbering mode. Pressing the BREAK key returns the computer to the manual mode.

**BREAK** key

The BREAK key stops program execution. Resume program execution by entering the CONT command. The BREAK key also stops the scrolling of the program's listing.

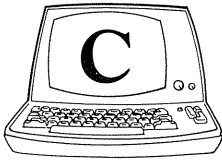
Also see: CONT, STOP, POKE

*Apple* • Applesoft uses the STOP key to halt program execution.  
 STOP

*PET* • Pressing the RUN/STOP key stops or breaks the program, and prints out the line number where the program stopped.

Example: BREAK IN LINE 10

- TRS-80* • The BREAK key is used in both Level I BASIC and Level II BASIC. To disable the BREAK key POKE 16396,23. To re-enable the BREAK key POKE 16396,201 or POKE 16396,20. See POKE for other methods of disabling functions of the computer.



**C.** command

C. command is the short form for CONT, and is used to restart a program or listing after it has been stopped by the BREAK key.

Also see: CONT, CON, BREAK key

- Apple* • The CON command is used in Integer BASIC. The CONT command is used in Applesoft BASIC.

- PET* • The CONT command is used in PET BASIC.

- TRS-80* • The C. command is used in Level I BASIC as an abbreviation for CONT. Level II BASIC uses the CONT command.

**CALL** command

CALL is a command which enables some computers to jump to a machine language program.

Also see: USR, SYS, SYSTEM

- Apple* • The CALL command is used in both the Applesoft and Integer BASIC languages. USR is also used in Applesoft.
- CALL A jumps program execution to a machine language subroutine beginning at memory location A.
- CALL-151 puts the computer into machine language mode. An \* (asterisk) prompt will appear.
- CALL-936 is used in Integer Basic to clear the screen.

- PET* • The USR command is used in PET BASIC.

- TRS-80* • The USR command is used in Level II BASIC only.

**CDBL** function

The CDBL function is used to change numbers or numeric variables from single-precision to double-precision (C-change;

DBL—double). Single-precision numbers contain 6 significant digits.

Example: .333333

Double-precision numbers contain 16 significant digits.

Example: .3333333333333333

Also see: CSNG

*TRS-80* • The CDBL function is used only in Level II BASIC.

```
CDBL      Example: 10 CLS
                20 X=6:Y=7
                30 PRINT" SINGLE PRECISION 6/7=";X/Y
                40 PRINT"DOUBLE PRECISION 6/7=";
                  CDBL(X)/CDBL(Y)
                50 END
```

When the program is run the following results:

```
SINGLE PRECISION 6/7 = .857143
DOUBLE PRECISION 6/7 = .8571428571428571
```

Note that in the single-precision, the last number is rounded off.

### **CHR\$** function

CHR\$( ) is a function which retrieves the character, letter or number represented by the decimal ASCII number code within the parentheses. It allows the computer to print characters not normally accessible from the keyboard.

```
Example: 10 INPUT L
          20 PRINT CHR$(L)
```

If the computer operator inputs the number 65, the letter 'A' will be printed on the video screen because 65 is the ASCII code for the letter A.

Also see: ASC, Appendix A, TEXT

*Apple* • The CHR\$ function is used only in Applesoft. Integer BASIC does not use CHR\$. However, this function can be simulated by creating a string with all the ASCII characters. (Note: Use a space for the quotation mark to avoid ending the string at that point.)

```
A$=" ! # $ % ' ( ) * + , - . ? 0 1 2 3 4 5 6 7 8 . . . .
. . . . UVWXYZ _ "
```

The first of the ASCII characters is a space, and its code number is 32. If we want an 'A' to be printed, we can subtract 31 from the code for 'A'. We will have the number representing the 'A' in our string. The ASCII code for 'A' is 65, so we have  $65 - 31 = 34$ . A\$(34,34) would print from the 34th character in the string to the 34th character in the string, which is, of course, only one character, the 'A'. We can say that  $\text{CHR}\$(65) = \text{A}\$(34,34)$  and in general  $\text{CHR}\$(N) = \text{A}\$(N - 31, N - 31)$ .



*PET*     • The CHR\$ function is used in PET BASIC. See Appendix A for special ASCII characters to be used with CHR\$ on the PET.

*TRS-80* • The CHR\$ function is used only in Level II BASIC. Since the left arrow is used for backspacing, the right arrow for TABbing and the down arrow for line feed, we must use their ASCII codes with CHR\$ when we want to print them on the screen.

Example: 10 PRINT CHR\$(93);CHR\$(94);CHR\$(92)

The line above would print the left arrow, right arrow, and down arrow. Graphics and TABs can also be programmed in this way (see Appendix A).

CHR\$(23) can be used to convert the video display from 64 characters per line to 32 characters per line.

Example: 10 PRINT CHR\$(23);"LARGE LETTERS"  
20 FOR A=1 TO 500: NEXT A  
30 CLS  
40 PRINT "I AM BACK TO NORMAL"  
50 FOR A=1 TO 500: NEXT A

Line 10 prints "LARGE LETTERS" in 32 characters per line mode. Lines 20 and 50 are timers. Line 30 clears the screen and returns the video display to the normal 64-character-per-line mode.

### **CINT** function

CINT converts numbers to their integer value (C—convert; INT—integer). The numbers assigned to the CINT function cannot be larger than +32767 or smaller than -32767. Variables used in the CINT function return to their original precision when they are used again in the program without the CINT function.

*TRS-80* • The CINT function is used only in Level II BASIC.  
CINT

ˆ (circumflex) operator (See page 92.)

### **CLEAR** command, statement

When not followed by an argument, CLEAR resets all variables to zero. It is also used to set aside bytes in memory storage when an argument is included after it.

Example: 10 CLEAR 150 reserves 150 bytes in memory for string storage.

Each character in a string, including spaces, takes up one byte of memory. It does not take long to use up the automatically allotted 50 bytes. When more than 50 bytes are required, the CLEAR(N) statement must be used or an OS (out of string space) error will occur.

Also see: ERASE, CLR, SHIFT→

*Apple*  
CLEAR • The CLEAR command or statement is used only in Applesoft. The CLR command or statement is used in Integer BASIC. CLEAR and CLR reset all variables to zero and all strings to null.

*PET*  
CLR • The CLR command or statement which is the short form for CLEAR is used in PET BASIC.

*TRS-80*  
CLEAR • The CLEAR command is used only in Level II BASIC. The CLEAR(N) statement automatically reserves N bytes for string storage. The ERASE statement cancels all the reserved memory space.

In the command mode CLEAR clears the display. If SHIFT→ has converted the display to 32 characters per line, the CLEAR returns the display to the 64-character per line format.

When the computer is turned on, 50 bytes of memory are reserved automatically, even without the CLEAR statement. All string variables are reserved by the single CLEAR statement.

### **CLEAR** key

The CLEAR key is used to clear the video screen of all ASCII codes and characters.

Also see: CLS, GR, CLR, CLR HOME key

*Apple*  
GR • GR is a graphics command which also clears the screen.

*PET*  
CLR/  
HOME • The PET CLR/HOME key, when used with the SHIFT key, clears the video screen. The clear screen command can be inserted into a program by using CHR\$(147).

*TRS-80*  
CLEAR • The CLS command and statement can be used to clear the video screen in both Level I and Level II BASIC.

key  
CLS Pressing the CLEAR key, then the ENTER key, clears the screen and places the cursor at the top left corner of the monitor.

If graphics blocks have been turned on with the SET statement, the CLEAR key will turn off all the blocks at once.

### **CLOAD** command

The CLOAD command is used to load a recorded program from tape to the computer (C—cassette; LOAD—load). CLOAD“A” loads program “A” from tape into computer memory. Other programs on the tape are ignored until “A” is found and loaded. CLOAD, with no specified program name, loads the first program that it encounters on the tape.

Also see: LOAD, VERIFY

- Apple*  
LOAD • The LOAD command is used in both the Applesoft and Integer BASIC.  
VERIFY can be used to check a SAVED program.
- PET*  
LOAD • The LOAD command is used in PET BASIC. VERIFY is used to check if the SAVED program is recorded properly.
- TRS-80*  
CLOAD • The CLOAD command is used in both Level I BASIC and Level II BASIC. The CLOAD?"A" command is used to compare a program in memory with one on tape. The message "BAD" is displayed if the recording is bad. The letter in parentheses is the name of the program.

**CLOSE** command, statement

CLOSE is a command and a statement that is used to close a file that was opened by the OPEN command. CLOSE stops the flow of information to a file or disk.

Format: CLOSE 1, file number

Also see: INPUT#, OPEN, PRINT#, STORE

- PET*  
CLOSE • The CLOSE command or statement is used in PET BASIC. To read from TAPE 1 use the following statements:  
OPEN3,1,0 opens file 3 on device 1 (tape recorder). The 0 is the secondary address that tells the computer to read a file.  
INPUT#3,A\$ reads back a value from the tape which is stored in the string A\$.  
CLOSE 3 closes file number 3.

*TRS-80*  
CLOSE • CLOSE is used in the DOS (disk operating system).

**CLR** command, statement

The CLR command and statement is an abbreviation for CLEAR, which is used to set all variables to zero.

Also see: CLS, CLEAR

- Apple*  
CLR • The CLR command and statement is used only in Integer BASIC. CLEAR is used in Applesoft.  
In Integer BASIC CALL 936 clears the screen.
- PET*  
CLR • The CLR command or statement is used in PET BASIC. CLR deletes all the stored references to variables, arrays, functions, GOSUB and FOR-NEXT statements.
- TRS-80*  
CLEAR • CLS in Level I BASIC clears the screen but does not set variables to zero. CLEAR and CLS are both used in Level II BASIC.

**CLR/HOME** key

The CLR/HOME key has two functions. When the key is pressed without the SHIFT key, the cursor returns home, that is, to the top left corner of the screen. The video display remains unchanged. When the CLR/HOME key is pressed with the SHIFT key, the screen is cleared of all alphanumeric characters and graphics. The cursor appears in the upper left corner of the blank screen.

Also see: CLS, CLEAR, CLR

*Apple*  
CLR • CLR is used in Integer BASIC, but it does something different. CLR is a short form for CLEAR, which sets all variables to zero.

CALL-936 is used in Integer BASIC to clear the screen.

*PET*  
CLR/  
HOME • The PET CLR/HOME key returns the cursor to the top left of the video screen. The characters on the screen remain the same.

The SHIFT and CLR/HOME keys, pressed at the same time, clear the screen and return the cursor to the top left of the screen.

The clear screen command can be inserted into a program by using CHR\$(147).

The home cursor command can be inserted into a program by using CHR\$(19).

*TRS-80*  
CLEAR • The CLS is used to clear the screen in both Level I and Level II BASIC.

key The CLEAR key and ENTER key, pressed in that order, clear the screen and place the cursor at the top left of the screen.

**CLS** command and statement

The CLS command and statement clears the video screen in a way similar to that of the CLEAR key. It may be used in a program line as in 10 CLS. This is useful when a blank screen is desired for a fresh video printout. It is often used in games.

Also see: SHIFT @, CLR, CLR/HOME, CHR\$( )

*Apple*  
ESC • Pressing the ESC key puts the computer into the edit mode. Pressing the SHIFT @ then clears the screen and places the cursor in the upper left corner of the screen.

The GR graphics command also clears the screen.

In Integer BASIC CALL-936 clears the screen.

*PET*  
CLR/  
HOME • The PET CLR/HOME key, pressed at the same time as the SHIFT key, clears the screen. If a clear screen is desired in a statement it can be achieved by typing PRINT "SHIFT CLR/HOME" or "SHIFT CLR/HOME".

In both of these a reverse heart will be displayed in the statement.

PRINT "!" or "?!"

The clear screen command can also be inserted into a program by using CHR\$(147).

*TRS-80* • The CLS command and statement is used in both Level I and Level II BASIC.

The CLS statement is also used to return the video display to the normal 64 character-per-line mode after it has been changed to 32 characters per line by the use of CHR\$(23). (See CHR\$)

**CMD** command

The CMD command is similar to PRINT# except that in this mode the computer remains connected to the external device. PRINT and LIST commands are then carried out on the external device instead of on the video screen. If the computer is on line to a printer then CMD LIST will make a hard copy of the program on the printer.

Also see: LLIST, PRINT#

*PET* • The CMD command is used in PET BASIC.

**CMD** Format: OPEN 3,4 opens file numbered 3.  
 CMD 3 the printer listens to number 3.  
 LIST the program is printed by the printer  
 (the printer continues to listen after the listing).  
 PRINT#3 stops the printer from listening  
 CLOSE 3 closes file number 3.

*TRS-80* • LLIST is a similar command in both Level I and Level II BASIC.

: (colon) operator (See page 93.)

**COLOR** command and statement

The COLOR command and statement is a special feature specifying the color to be displayed on the video screen.

Example: 10 color=3 produces a purple color on the screen

Also see: PLOT, VLIN, HLIN, HPLOT

*Apple* • Applesoft and Integer BASIC both use the COLOR command and statement in their low-resolution graphics.

**Color Codes**

- |               |                |            |
|---------------|----------------|------------|
| 0. black      | 6. medium blue | 11. pink   |
| 1. magenta    | 7. light blue  | 12. green  |
| 2. dark blue  | 8. brown       | 13. yellow |
| 3. purple     | 9. orange      | 14. aqua   |
| 4. dark green | 10. gray       | 15. white  |
| 5. gray       |                |            |

, **(comma)** (ANSI) operator (See page 93.)

### **CON** command

CON is an abbreviation for continue. This command continues a program's execution after it has been stopped by STOP, END, or CTRL C.

CON does not work after an error, editing, typing NEW, or after the execution of the program has ended normally.

Also see: C., CONT, STOP, END, CTRL C

*Apple* • The CON command is used in Integer BASIC. The  
CON CONT command is used in Applesoft.

*PET* • The CONT command is used in PET BASIC.  
CONT

*TRS-80* • The CONT command is used in Level II BASIC. In  
C. Level I BASIC C. is used as an abbreviation.  
CONT

### **CONT** command

CONT is an abbreviation for continue, and continues a program's execution after it has been stopped by STOP, END, the BREAK key or the STOP key. CONT does not work after an error, editing, typing NEW, or after the execution of the program has ended normally.

Also see: C., CON, STOP, END, BREAK key, STOP key, CLR

*Apple* • The CONT command is used only in Applesoft. Integer  
CONT Basic uses the CON command.

*PET* • The CONT command is used in PET BASIC. It may  
CONT only be executed in the direct mode. It does not work  
after the CLR command.

*TRS-80* • The CONT command is used in Level II BASIC. The C.  
CONT command is used in Level I BASIC.

### **COS** (ANSI) function

The COS(X) function returns the cosine of X. The argument "X" must be in radians.

*Apple* • The COS function is used only in Applesoft. Programs  
COS which use COS cannot be translated directly into In-  
teger BASIC.

*PET* • The COS function is used in PET BASIC.  
COS

*TRS-80* • The COS function is used only in Level II BASIC.  
COS

↑  
**CRSR** key  
↓

The CRSR key moves the cursor down one space. The SHIFT and CRSR keys pressed at the same time move the cursor up one space. Cursor up can be written in PRINT statements by the reverse field ↑. Type PRINT "SHIFT CRCSR". Cursor down can be written into PRINT statements by the reverse field ↓. Type PRINT "CRCSR".

Also see: ESC C, ESC D, ↓ (down arrow)

*Apple* • Pressing the ESC key followed by the C key moves the  
ESC C cursor down one space. Pressing the ESC key followed  
ESC D by the D key moves the cursor up one space.

*PET* •  
↑ The CRCSR key is unique to the PET.  
CRCSR ↓  
↓ Cursor up can be inserted into a program by using  
CHR\$(145).  
Cursor down can be inserted into a program by using  
CHR\$(17).

*TRS-80* • The ↓ (down arrow) moves the cursor down to the next  
↓ line.

**CRCSR** key

When the SHIFT and CRCSR keys are pressed at the same time, the cursor moves one space to the left. Cursor left can be written in PRINT statements by using the reverse field ←. Type PRINT "SHIFT CRCSR". Pressing the CRCSR key without the SHIFT key moves the cursor one space to the right. Cursor right can be placed in PRINT statements by using the reverse field →. Type PRINT "CRCSR".

Also see: ESC A, ESC B, ← (LEFT ARROW)

*Apple* • Pressing the ESC key and the A key moves the cursor  
ESC A one space to the right. Pressing the ESC key and the B  
ESC B key moves the cursor one space to the left. The → (right  
arrow) enters the character under the cursor into  
memory and moves one space to the right. The ← (left  
arrow) deletes one character from the line, and moves  
the cursor one space to the left.

*PET* • The CRCSR key is unique to the PET. Cursor left can be  
CRCSR inserted into a program by using CHR\$(157). Cursor  
right can be inserted into a program by using CHR\$(29).

*TRS-80* • The space bar moves the cursor one space to the right.  
SPACE The ← (left arrow) moves the cursor one space to the  
BAR left, deleting one character from the line.

**CSAVE** command

CSAVE is used to record a program in the computer's memory onto cassette tape. (C—cassette; SAVE—save)

CSAVE"A" gives the recorded program the name "A".

Also see: SAVE, LOAD, CLOAD, VERIFY, CLOAD?

*Apple* • The SAVE command is used in both Applesoft and SAVE Integer BASIC. VERIFY is used to check a SAVED program.

*PET* • The SAVE command is used in PET BASIC. VERIFY is SAVE used to check a SAVED program.

*TRS-80* • The CSAVE command is used in both Level I BASIC CSAVE and Level II BASIC. CLOAD? is used to check a SAVED program.

**CSNG** function

The CSNG function is used to change numbers from double-precision back to single-precision (C—change; SNG—single). Double-precision numbers are accurate to 16 digits, while single-precision are accurate to 6 digits.

Also see: CDBL

*TRS-80* • The CSNG function is used only in Level II BASIC.

CSNG Example: 10 CLS  
 20 X=1:Y=7  
 30 PRINT"DOUBLE PRECISION 1/7=";  
 CDBL(X)/CDBL(Y)  
 35 N=CDBL(X)/CDBL(Y)  
 40 PRINT" SINGLE PRECISION 1/7=";  
 CSNG(N)  
 50 END

When the program is run the following results:

DOUBLE PRECISION 1/7 = .1428571428571429

SINGLE PRECISION 1/7 = .142857

The double-precision number changes back to a single-precision number.

**CTRL C** command

The CTRL C command is used in the monitor mode to halt the program or listing of the program.

Also see: RVS, SHIFT@

*Apple* • CTRL C is used in Applesoft.

CTRL C

*PET* • The RVS key, when held down, slows the listing of a RVS program, which allows it to be read and studied.

*TRS-80* • Pressing the SHIFT@ keys at the same time stops the SHIFT@ listing or execution of a program. Pressing any letter will then resume the listing or execution.



### **CTRL G and CTRL J** commands

#### **CTRL G**

Pressing the CTRL (control) key and the G key simultaneously causes the speaker to emit a beep sound.

#### **CTRL J**

Pressing the CTRL (control) key and the J key simultaneously causes a line feed.

*Apple* • The CTRL G command causes the speaker to emit a beep sound. CTRL J causes a line feed. CALL-922 does the same thing during the execution of a program.  
**CTRL J** CHR\$(10) causes a line feed during the execution of a program.

*TRS-80* • The OUT command can be used in Level II BASIC to make a noise on the tape recorder.  
**OUT**

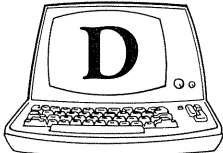
### **CTRL X** command

The CTRL X command deletes the entire line being typed. After a line is entered into memory, it may be deleted by typing the line number and ENTER or RETURN.

Also see: SHIFT

*Apple* • The CTRL X command is used only in Applesoft.  
**CTRL X**

*TRS-80* • Pressing the SHIFT and the ← (left arrow) keys at the same time deletes the entire line being typed.  
**SHIFT←**



### **D.** statement

D. is a Level I BASIC abbreviation for the DATA statement. DATA statements may be placed anywhere in a program. Elements in a DATA statement must be separated by a comma.

*Apple* • The DATA statement is used only in Applesoft.  
**DATA**

*PET* • The DATA statement is used in PET BASIC.  
**DATA**

*TRS-80* • The D. abbreviation is used in Level I BASIC. DATA is used in Level II BASIC. In Level II BASIC the D without a period following is used to define a variable as double-precision with scientific notation.  
**D.**

### **DATA** (ANSI) statement

DATA statements contain data elements that can be read by READ statements. Each element in the DATA statement must be

separated by a comma. Commas and ends of lines are considered terminators for an element of data. To enter DATA statements you first type the line number, then the word DATA, then the first data value, then a comma, then the second data value, and so on to the last data value for the line. No comma is entered at the end of the line.

Also see: D., , comma, READ

*Apple* • The DATA statement is used only in Applesoft.  
DATA Example: 10 DATA A, "B", "C"

Some DATA statements can be translated into Integer BASIC.

Example: 1000 DATA 6,20, - 1,6,4,3 can be translated into Integer Basic by typing 1000 A(1)=6:A(2)=20:A(3)= - 1:A(4)=6: A(5)=4:A(6)=3

*PET* • The DATA statement is used in PET BASIC. Blanks and graphics characters are not considered DATA unless they are surrounded by quotation marks.

DATA Example: 1000 DATA NAME is legal  
2000 DATA ❄, , IS ILLEGAL

When loading blanks, colons or commas into DATA statements, treat them as part of a longer string and enclose them in quotation marks.

Example: 3000 DATA "NAME: AGE"

*TRS-80* • The DATA statement is used in Level II BASIC. The abbreviation D. is used in Level I BASIC.  
DATA

## **DEF** (ANSI) statement

The DEF statement allows you to define new functions. The name of the function is FN, followed by a legal variable name. This statement is useful in defining functions which are not intrinsic to a particular computer's BASIC language.

Examples:

1. To define a SECANT, SEC(X) function DEF FN S(X)=1/COS(X) where X is in radians. FN S(X) is used for SEC(X) from then on in the program.

2. To define the INVERSE COSECANT, ARCCSC(X) DEF FN C(X)=ATN(1/SQR(X\*X-1))+(SGN(X)-1)\*3.14/2. FNC(X) is then the same as ARCCSC(X).

Also see: DEFDBC, DEFINT, DEFSNG, DEFSTR

*Apple* • The DEF statement is used only in Applesoft. DEF FN A(M)=(N-3)\*9 defines a function FNM which can be substituted for M in the defined expression.

Example: FNM(3) would return the number 0, while FNM(5) would return 18. In the first example (3-3)\*9=0 while in the second example (5-3)\*9=18.

*PET* • The DEF statement is used in PET BASIC. DEF FN(M) is acceptable, where FN A6—letter and number and FN AO—letter and letter are both acceptable defined functions.

The user-defined functions must be contained in one line with not more than 80 characters. String functions cannot be defined.

*TRS-80* • Level II BASIC has a number of built-in defined functions. See: DEFDBL, DEFINT, DEFSNG, DEFSTR

### **DEFDBL** statement

The DEFDBL statement defines a variable or number as being double-precision. Double-precision numbers are accurate to 17 digits. (DEF—define; DBL—double)

Example: 10 DEFDBL A,D defines the variables A and D as being double-precision.

Also see: CDBL, CSNG

*Apple* • The DEF statement is used only in Applesoft.  
DEF

*PET* • The DEF statement is used in PET BASIC.  
DEF

*TRS-80* • The DEFDBL is used only in Level II BASIC.  
DEFDBL

### **DEFINT** statement

The DEFINT statement defines variables as integers. DEFINT can be used with multiple variables if they are separated by a comma (DEF—define; INT—integer).

Examples: DEFINT A,B,C, defines A, B, C as integer.  
DEFINT A-F defines A, B, C, D, E and F as integers.

Also see: DEF

*Apple* • The DEF statement is used only in Applesoft.  
DEF

*PET* • The DEF statement is used in PET BASIC.  
DEF

*TRS-80* • DEFINT is used only in Level II BASIC. The Z-80 micro-processor, like the 6502, allows only integers from -32767 to +32767. Less memory is required to store integer values than non-integer, which is an advantage in longer programs.

### **DEFSNG** statement

The DEFSNG statement re-defines variables as single-precision in programs where the DEFDBL statement has been used to make the

variables double-precision (DEF—define; SNG—single). Multiple variables may be used after the DEFSNG statement, if separated by a comma.

Examples: DEFSNG A,B,C re-defines A, B and C as single-precision. DEFSNG A-F re-defines A, B, C, D, E and F as single-precision.

Also see: DEF, CDBL, CSNG

*Apple* • The DEF statement is used only in Applesoft. Applesoft  
DEF can deal with numbers with 9-digit precision in the range  $\pm 10$  to the power of 38.

*PET* • The DEF statement is used in PET BASIC.  
DEF

*TRS-80* • DEFSNG is used only in Level II BASIC.  
DEFSNG

### **DEFSTR** statement

The DEFSTR statement is used to define variables as string variables (DEF—define; STR—string). The DEFSTR statement may be used with multiple variables if they are separated by a comma.

Examples: DEFSTR A,B,C defines A, B and C as string variables. DEFSTR A-F defines A, B, C, D, E and F as string variables. The \$ sign declaration may then be omitted.

Also see: DEF, DIM

*Apple* • The DEF statement is used only in Applesoft.  
DEF

*PET* • The DEF statement is used in PET BASIC.  
DEF

*TRS-80* • The DEFSTR statement is used only in Level II BASIC.  
DEFSTR When using the DEFSTR and DIM statements in the same program, always place the DIM statement after the DEFSTR because the DEFSTR statement resets the array depth to 10. This would cause problems if the desired size of the array was more than 10.

Example: 10 DIMAB(30,20) sets array AB to 30×20. 20 DEFSTRA resets all arrays to 10, which results in array AB being set to 10×10. This would cause more than half the stored information to be lost.

### **DEL** command

DEL is an abbreviation for DELETE.

Format: DEL 10 deletes line 10. An alternate way to delete is to type each line number, then press the ENTER or RETURN key. Typing NEW eliminates all line numbers, that is, the whole program is erased from memory.

Also see: DELETE, INST/DEL

- Apple* • The DEL command is used in both Applesoft and Integer BASIC. DEL A,D deletes lines A and D.
- PET* • The PET uses the INST/DEL key to delete lines. INST/DEL CHR\$(20) can be used to delete lines during a program run.
- TRS-80* • The DELETE command is used only in Level II BASIC. DELETE A-D deletes lines A to D inclusive.

**DELETE** command

Format: DELETE 10 deletes line 10. DELETE # to # deletes specified lines.

Example: DELETE 20-40 deletes all the lines from 20 to 40 inclusive. An alternate way to delete is to type each line number, then RETURN or ENTER. Typing NEW eliminates all line numbers by erasing the whole program, which is in memory.

*Apple* • The DEL command is used in both Applesoft and Integer BASIC.

*PET* • The PET uses the INST/DEL key to delete lines. INST/DEL CHR\$(20) can be used to delete lines during a program run.

*TRS-80* • The DELETE command is used in Level II BASIC. DELETE

**DIM** (ANSI) statement

The DIM or dimension statement establishes the number of elements in a numeric or string array.

```
DIM   A   (15)
dimension array number of elements assigned to the array
name
```

When the DIM statement is executed, all values stored in each designated array element are set to zero. Space is set aside for the specified array with subscripts as in the parentheses.

*Apple* • The DIM statement is used in both Integer BASIC and Applesoft.

Example: DIM NAME\$(40) sets aside space for specified array with subscripts within range.

In Integer BASIC strings must be dimensioned. If A\$ is to be 30 characters long, the user must type: DIM A\$(30) at the beginning of the program to avoid a RANGE ERR message.

*PET* • The DIM statement is used in PET BASIC. 10 DIM A(5), DIM B(6) dimensions matrix A so that it has 5 subscripts and matrix B so that it has 6 subscripts. BASIC automatically sets up space for ten elements in an array if a one-dimensional array is used without a DIM statement.

Space for a ten by ten array is set up automatically if a two-dimensional array is used without a DIM statement. The DIM statement must be used if more space is needed.

*TRS-80* • The DIM statement is used only in Level II BASIC. DIM  
DIM A\$(100) creates an array of 101 string variables from A\$(0) through A\$(100).

/ (division sign) (ANSI) operator (See page 93.)

\$ (dollar sign) (ANSI) operator (See page 94.)

↓ (down arrow) key

The ↓ (down arrow) is a line feed, which makes the cursor move down to the next line.

Also see: ESC, C, CRSR

*Apple* • Pressing the ESC key followed by the C key moves the  
ESC C cursor down one line. CTRL J causes a line feed.  
CTRL J CALL-922 does the same thing during the execution of a  
program. PRINT CHR\$(10) can also be used to create a  
line feed during execution of a program.

*PET* • The PET CRSR key moves the cursor down one line.  
CRSR

*TRS-80* • The ↓ (down arrow) is used in both Level I and Level II  
↓ BASIC.

**DRAW AT** statement, command

The DRAW AT command draws a defined shape (loaded from cassette by the SHLOAD command) at a specified location.

Format: DRAW N AT X,Y

Example: 10 DRAW 1 AT 30,40 draws the shape which has the number 1 in the shape table starting at the location 30 (X co-ordinate), 40 (Y co-ordinate).

The color is set by the HCOLOR command.

Also see: HCOLOR, SHLOAD, XDRAW AT, SET, RESET, PRINT AT, PRINT@

*Apple* • The DRAW AT command is used only in Applesoft  
DRAW BASIC.  
AT

*PET* • The PET cursor keys can be programmed to draw  
graphics at specific locations on the screen.

*TRS-80* • The SET and PRINT@ commands can be used to print  
SET or place graphics at specified locations on the screen in  
PRINT@ both Level I and Level II BASIC.

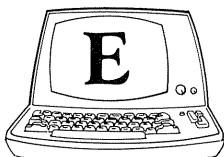
**DSP** statement

The DSP statement displays a specific variable and its value every time the variable is given a new value. The line number containing the variable is also shown. NO DSP or TROFF is used to stop this process.

Also see: TRACE, NO TRACE, TRON, TROFF

*Apple* • The DSP statement is used only in Integer BASIC. DSP TRACE is similar in that it displays the line numbers as the program is executed. Unlike DSP, TRACE does not display the values of the variables. NO DSP can be used to stop the display of a number and its value.

*TRS-80* • The TRON statement is used only in Level II BASIC. It TRON turns on, or displays, the line numbers as each line is executed. It does not, however, give the value of the variables used in the program.

**E.** statement

E. is the Level I BASIC abbreviation for the END statement.

Also see: END

*Apple* • The END statement is used in both Applesoft and Integer BASIC.

*PET* • The END statement is used in PET BASIC. END

*TRS-80* • The E. statement is used only in Level I BASIC. Level II E. BASIC uses the END statement.

**EDIT** command

The EDIT command is used to start editing a specified line.

Example: EDIT 10 allows the operator to edit line 10 using all the edit commands. EDIT (no line specified) allows the operator to edit the first line in the program.

Also see: ESC key

*Apple* • ESC is used only in Applesoft BASIC. Pressing the ESC ESC key puts the computer into the edit mode.

*PET* • Using the PET INST/DEL key is the fastest way to edit a INST/ line. The PET has two other screen edit keys. They are: DEL



*TRS-80* • The EDIT command is used only in Level II BASIC. EDIT Some of the editing commands are

C, change the next character  
 D, delete the next character;  
 I, insert one or more characters  
 SHIFT I, terminate the insert command  
 L, list the line that is being edited but do not terminate editing  
 S, search for a specified character  
 ENTER, terminate editing.

### **ELSE** statement

The ELSE statement is used after the IF-THEN statement. When the IF condition is not met the ELSE statement takes over, and program execution goes to the line number specified.

Example: 50 IF X=1 THEN 10 ELSE 20

If the X does equal 1 the program goes to line 10, but if X does *not* equal 1 the program continues at line 20.

Without the ELSE the program continues at the next line when the condition in the IF-THEN statement is not met.

Also see: IF-THEN

*The Apple and PET* get the same results by placing a GOTO statement in the line after the IF-THEN statement.

Example: 10 IF X=5 THEN 100  
20 GOTO 200

This would be the same as

10 IF X=5 THEN 100 ELSE 200.

Using the ELSE statement often saves memory space.

TRS-80 • The ELSE statement is used only in Level II BASIC.  
ELSE

### **END** (ANSI) statement

The END statement is used to end the execution of the program without printing the last line number. This is different from the STOP statement.

Example: 10 REM  
20  
:  
:  
999 END  
1000 REM SUBROUTINE STARTS HERE

The END statement keeps the subroutine safe from accidental execution with the main part of the program. Using CONT after an END statement causes the program to resume.

Also see: STOP

Apple • The END statement is used in both Applesoft and in Integer BASIC.  
END



*PET*     • The END statement is used in PET BASIC.  
END

*TRS-80* • END is used in Level II BASIC. In Level I BASIC E. may  
END       be used as an abbreviation.

**ENTER** key

The ENTER key is used to signify the end of an input line. On all three computers CHR\$(13) can be used to generate a return during a program run.

Also see: RETURN

*Apple*   • The RETURN key is used on the Apple II computer.  
RETURN

*PET*       • The RETURN key is used on the PET computer.  
RETURN

*TRS-80* • The ENTER key is used on the TRS-80 computer.  
ENTER

=   **(equal sign)** (ANSI) operator (See page 94.)

**ERASE** statement

The ERASE statement cancels all the reserved space in memory. It cancels the CLEAR(N) statement.

Also see: CLEAR

*TRS-80* • The ERASE statement is used only in Level II BASIC.  
ERASE

**ERL** function

The ERL function is used with the ON-ERROR statement to identify the line in which an error occurred. The line number contained in the ERL function changes each time an error occurs.

Example: 10 ON ERROR GOTO 100

100 PRINT "THERE IS AN ERROR IN LINE";ERL

Also see: ON ERROR

*TRS-80* • The ERL function is used only in Level II BASIC.  
ERL

**ERR** function

The ERR function is used to identify the error code of the last error. It changes each time an error occurs.

*TRS-80* • The ERR function is used only in Level II BASIC. The  
ERR       ERR value must be divided by 2, and then 1 added to  
          the answer, to get the actual error code.

**ERROR** command, statement

ERROR is used to simulate a program error.

Example: ERROR A causes an error message to be printed.

The type of error is specified by the error code, represented by the operand A.

Example: 10 ERROR 14 prints OS ERROR IN 10

See your own manual for error codes and statements for error handling.

*TRS-80* • The ERROR command and statement is used only in ERROR Level II BASIC.

**ESC A** command

Press the ESC key to get into edit mode. The A key can then be pressed to move the cursor one space to the right. Press the REPT key to repeat the movement.

Also see: SPACE BAR,  $\overline{\text{CR}}\overline{\text{SR}}$ , REPT, EDIT

*Apple* • The Apple II computer uses the ESC A command to  
ESC A move the cursor one space to the right.

*PET* • Pressing the PET cursor key ( $\overline{\text{CR}}\overline{\text{SR}}$ ) without a SHIFT  
 $\overline{\text{CR}}\overline{\text{SR}}$  moves the cursor one space to the right.

*TRS-80* • The TRS-80 SPACE BAR or  $\rightarrow$  (right arrow) key moves  
SPACE the cursor one space to the right. CHR\$(25) can also be  
BAR used for the  $\rightarrow$  (right arrow).

**ESC B** command

Pressing the ESC key puts the computer in the edit mode. If the B key is then pressed the cursor moves one space to the left.

Also see: CR $\overline{\text{SR}}$ ,  $\leftarrow$  (left arrow)

*Apple* • The Apple II Computer uses the ESC B command to  
ESC B move the cursor one space to the left.

*PET* • Pressing the PET cursor key  $\overline{\text{CR}}\overline{\text{SR}}$  and the SHIFT key  
 $\overline{\text{CR}}\overline{\text{SR}}$  at the same time moves the cursor one space to the left.  
CHR\$(157) can also be used to move the cursor one  
space to the left.

*TRS-80* • Pressing the TRS-80  $\leftarrow$  (left arrow) key moves the  
 $\leftarrow$  cursor one space to the left. CHR\$(24) and CHR\$(8) can  
also be used for the  $\leftarrow$  (left arrow).

**ESC C** command

Pressing the ESC key and the C key moves the cursor down one line. ESC puts the computer into the edit mode, so that other commands can be executed.

Also see: down arrow,  $\uparrow$   
 $\downarrow$  CRSR

- Apple* • The Apple II computer uses the ESC C command to  
 ESC C move the cursor down one line.  $\uparrow$
- PET* • Pressing the PET cursor key (CRSR) once moves the  
 $\uparrow$   
 CRSR cursor one line down. CHR\$(17) can also be used to  
 $\downarrow$  move the cursor one line down.
- TRS-80* • Pressing the TRS-80  $\downarrow$  (down arrow) key moves the cur-  
 $\downarrow$  sor one line down. The down arrow is a line feed.  
 CHR\$(26) can also be used for the  $\downarrow$  (down arrow).

**ESC D** command

Pressing the ESC key puts the computer into the edit mode; then pressing the D key moves the cursor one line up.

Also see:  $\uparrow$   
 $\downarrow$  CRSR

- Apple* • The Apple II computer uses the ESC D command to  
 ESC D move the cursor up one line.
- PET* • Pressing the SHIFT key and the cursor key at the same  
 $\uparrow$   
 CRSR time moves the cursor one line up. CHR\$(145) can also  
 $\downarrow$  be used to move the cursor one line up.
- TRS-80* • Pressing the TRS-80  $\uparrow$  (up arrow) key moves the cursor  
 $\uparrow$  one line up. CHR\$(27) can also be used for the  $\uparrow$  (up  
 arrow).

**ESC E** command

**ESC F** command

Pressing the ESC key puts the computer in the edit mode. Pressing the E key then clears the screen from the position of the cursor to the end of the line.

CALL-868 will do the same thing during execution of a program.

Pressing the F key after the ESC key clears the screen from the cursor position to the bottom of the page.

CALL-958 will do the same thing during execution of a program.

- Apple* • The Apple II Computer uses the ESC E command to  
 ESC E clear the screen from the current cursor position to the  
 ESC F end of the current line. The ESC F command clears the  
 screen from the current cursor position to the end of the  
 screen.
- PET* • The SYS command can be used to simulate an escape  
 SYS back to a specific line (example a menu) by placing the

following line after every INPUT line: IF A\$="@  
THEN POKE167,1:SYS50583: GOTO 1000 where A\$  
changes to correspond to the same variable used in the  
INPUT statement, and 1000 is the line to which the user  
wants to escape. The "@" can be any key the user wants.  
The "@" key is not often used, and therefore is a good  
choice.

*TRS-80* • In Level II BASIC CHR\$(30) clears the screen to the end  
of the line. CHR\$(31) clears the screen to the end of the  
page.

! (exclamation mark) operator (See page 95.)

### EXP (ANSI) function

The EXP function is the opposite of the LOG function. EXP(n) com-  
putes the natural logarithm's base value e (2.718289) raised to the  
power of (n).

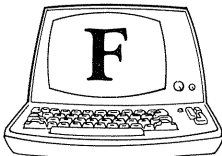
Example: A = EXP(2) is A = 2.718289 \* 2.718289 or  
A = (2.718289)<sup>2</sup>

Also see: LOG

*Apple* • The EXP function is used only in Applesoft. Programs  
EXP using EXP cannot be translated directly into Integer  
BASIC.

*PET* • The EXP function is used in PET BASIC.  
EXP

*TRS-80* • The EXP function is used only in Level II BASIC.  
EXP



### F. statement

The F. statement is an abbreviation for the FOR statement, and is  
part of a FOR-TO-NEXT statement. It assigns numbers to numeric  
variables.

Example: 10 F. A = 1 TO 10  
20 PRINT"HELLO"  
30 NEXT A

In this example, A is given the value of 1 and HELLO is printed.  
When the NEXT A is encountered in line 30, A=2 and HELLO is  
printed again. This continues until HELLO has been printed ten  
times.

Also see: FOR, FOR-TO-NEXT

- Apple* FOR • The FOR statement is used in both Applesoft and Integer BASIC.
- PET* FOR • The FOR statement is used in PET BASIC.
- TRS-80* F. • The FOR statement is used in Level II BASIC. Level I BASIC uses the F. statement or the FOR statement.

**FIX** function

The FIX function removes all numbers to the right of the decimal point. Unlike the INT function, it does not round negative numbers.

```
Example: 10 A=3.7865           10 A=-3.7865
          20 B=FIX(A)          20 B=FIX(A)
          30 PRINT B           30 PRINT B
```

The program in the first example prints the number 3. The program in the second example prints the number -3. In both cases the .7865 has been removed.

Also see: INT

- TRS-80* FIX • The FIX function is used in Level II BASIC.

**FLASH** command, statement

The FLASH command causes the output on the screen to flash, alternately showing white on black and then black on white. The NORMAL command is used to return to the non-flashing display.

Also see: NORMAL, CHR\$, OFF/RVS

- Apple* FLASH • The FLASH command or statement is used only in Applesoft.
- PET* RVS • The RVS key is used in PET BASIC. It reverses the video display from the usual white on black to black characters on a white background. The RVS key can be programmed along with its opposite, the SHIFT RVS key, which returns the display to normal. Flashing can be programmed by using the RVS and SHIFT RVS in turn.

```
Example: 10 PRINT" RVS THIS IS A DEMO"
          20 FOR A=1 TO 500: NEXTA
          30 PRINT" HOME key SHIFT RVS THIS
             IS A DEMO"
          40 FOR A=1 TO 500: NEXTA
          50 GOTO 10
```

CHR\$ can also be used to reverse the field and return to normal.

RVS on CHR\$(18); RVS off CHR\$(146)

**FN** function

The FN function is used with the DEF statement to specify the defined variable of the operator.

```
Example: 10 DEF FNA(R)=3.14*R*R
          20 PRINT"ENTER A RADIUS OF A CIRCLE"
          30 INPUT R
          40 C=FNA(R)
          50 PRINT"THE CIRCUMFERENCE OF A CIRCLE
              WITH RADIUS";R;"IS";C
```

Also see: DEF, DEFDBL, DEFINT, DEFSNG, DEFSTR

- Apple* • The FN function is used only in Applesoft.
- FN** DEF FN A(X)=X\*2+3 defines a function FNA. The FNA is substituted for X in the defined expression. FNA(5) gives an answer of 13.
- PET* • The FN function is used in PET BASIC. Other ways to write the function are FNA6 (letter and number), and FNAO (letter and letter).
- FN** The user-defined functions must be contained on one line (80 characters). String functions cannot be defined.
- TRS-80* • Level II BASIC has a number of built-in defined functions that are similar to the FN functions. See DEFDBL, DEFINT, DEFSNG, and DEFSTR.

**FOR** (ANSI) statement

The FOR statement is part of a FOR-TO-STEP-NEXT statement, and assigns numbers to specified numeric variables. Each time NEXT is encountered the number after FOR is incremented by the STEP value (one if STEP was omitted). When the number following TO is exceeded, program execution continues with the line following the one which contains the NEXT statement.

Also see: NEXT, STEP

- Apple* • The FOR statement is used in both Applesoft and Integer BASIC.
- FOR**
- PET* • The FOR statement is used in PET BASIC.
- FOR**
- TRS-80* • The FOR statement is used in both Level I BASIC and Level II BASIC.
- FOR**

**FRE** function, statement

The FRE function is used to determine the amount of memory space available in the computer. FRE(0) returns the total amount of free memory space. FRE(A\$) is used to tell the operator how many bytes of total string space are free in the computer's memory. The argument must be enclosed in parentheses.

Also see: MEM

*Apple* • The FRE(0) function is used only in Applesoft.  
FRE

*PET* • The FRE(0) function is used in PET BASIC.  
FRE  
10 PRINT " " FOR A = 1 TO 10  
20 INPUT "ENTER A STRING";A\$(A)  
50 PRINT "THERE IS NOW ";FRE(0);" BYTES OF  
SPACE LEFT"  
60 NEXTA

Try the above program and experiment with it to discover the amount of space that is used by strings.

*TRS-80* • The FRE function is used in LEVEL II BASIC.  
FRE



**G.** statement

G. is an abbreviation for the GOTO statement.

Also see: GOTO

*Apple* • The GOTO statement is used in both Applesoft and Integer BASIC.  
GOTO

*PET* • The GOTO statement is used in PET BASIC.  
GOTO

*TRS-80* • The G. statement is used in Level I BASIC. Level II BASIC uses the GOTO statement.  
G.

**GET** command, statement

The GET command scans the keyboard until a character is typed. It waits for the operator to type, and then continues with the execution of the program.

Example: 10 GET X  
20 IF X = " " THEN 10

Line 20 tells the computer that if X is null, or if nothing has been typed on the keyboard, THEN return to line 10 and try again. This continues until the X condition in the IF statement is met.

Also see: INKEY\$

*Apple* • The GET\$ command is used only in Applesoft. In a program, GET\$ waits for the operator to type a one-character value for A\$. Pressing the RETURN key is not necessary.

Example: 10 GET A\$: IF A\$ = " " THEN 10

GET ANS\$ can suspend program execution until a character is typed on the keyboard. The character is not displayed on the screen.

- PET*  
GET • The GET statement is used in PET BASIC. Only numeric values are accepted as input for the GET statement. If GET variable \$ is used, then a numeric or string character can be entered.

Example: 10 GET V\$: IF V\$=" " THEN 10 or  
10 GET Z\$: IF Z\$=" " GOTO 10

In both examples, if a numeric or string character is not typed on the keyboard, execution stays in the same line. Program execution continues when a character is typed.

- TRS-80*  
INKEY\$ • The INKEY\$ statement is used in Level II BASIC. It performs the same way as GET, except string characters are also acceptable.

### **GET#** command

The GET# retrieves a single character at a time from a cassette tape. The character is placed in the field specified following the GET#.

Format: GET# file, field

Also see: INPUT#

- PET*  
GET# • The GET# command is used in PET BASIC. INPUT# can also be used.

- TRS-80*  
INPUT# • INPUT #-1 is used in both Level I and Level II BASIC.

-1

### **GOS.** statement

GOS. is an abbreviation for the GOSUB statement.

Also see: GOSUB

- Apple*  
GOSUB • The GOSUB statement is used in both Applesoft and Integer BASIC. In Applesoft, the GOSUB statement must be followed by a line number. In Integer BASIC, GOSUB may be followed by a variable or expression.

- PET*  
GOSUB • The GOSUB statement is used in PET BASIC.

- TRS-80*  
GOS. • The GOS. statement is used in Level I BASIC. Level II BASIC uses the GOSUB form.

### **GOSUB** (ANSI) statement

The GOSUB statement transfers execution of the program to a subroutine at the line specified after the GOSUB.

Example: 5 GOSUB 1000



In the example, execution of the program branches to line 1000 every time line 5 is executed. The subroutine which starts at line 1000 must end with a RETURN statement so that execution returns to the main program.

Also see: GOS., RETURN

*Apple*  
GOSUB • The GOSUB statement is used in both Applesoft and Integer BASIC. In Applesoft, the GOSUB statement must be followed by a line number. In Integer BASIC, the GOSUB statement may be followed by a variable or expression.

*PET*  
GOSUB • The GOSUB statement is used in PET BASIC.

*TRS-80*  
GOSUB • The GOSUB statement is used in Level II BASIC. Level I BASIC may also use the abbreviation for GOSUB, which is G.

### **GOTO** (ANSI) statement

The GOTO statement transfers execution of the program to a specified line and continues there.

Example: 10 GOTO 100

When line 10 is reached, the program jumps to line 100 and continues execution.

Also see: G.

*Apple*  
GOTO • The GOTO statement is used in both Applesoft and Integer BASIC. In Applesoft, the GOTO statement must be followed by a line number. In Integer BASIC, the GOTO statement may be followed by a variable or expression.

*PET*  
GOTO • The GOTO statement is used in PET BASIC.

*TRS-80*  
GOTO • The GOTO statement is used in Level II BASIC. In Level I BASIC GOTO or G. may be used.

### **GR** command, statement

The GR command or statement puts the computer into the low resolution graphics mode. Each time GR is encountered in a program, the top 40 by 40 character area is cleared to black. The bottom 4 lines can be used for text.

Also see: PLOT, HLINE-AT, VLINE-AT, HCOLOR, HGR, COLOR, SCRN, STRING\$, CHR\$( ), APPENDIX C

*Apple*  
GR • The GR statement is used in both Applesoft and Integer BASIC.

### Low Resolution Graphics Commands

GR sets graphics.  
 COLOR sets color (0 to 15)  
 PLOT places dot.  
 HLIN draws a horizontal line.  
 VLIN draws a vertical line.  
 SCRN returns color to the screen at a particular point.

- PET* • The PET has keys which produce screen graphics. Pressing simultaneously the SHIFT key and a graphic key produces graphics on the screen.  
 Example of keys:



- TRS-80* • The TRS-80 uses SET, RESET and STRING\$ for graphics.  
 Changing the video display to 32 characters per line from 64 characters per line can be an effective graphics function. This is done by placing CHR\$(23) in a PRINT statement.

Example: 10 PRINT CHR\$(23);"XXXXXXXXXX"

Return to the normal 64 characters per line mode by using the CLS command. (See CHR\$)

> (**greater than**) (ANSI) operator (See page 95.)

>= (**greater than or equal**) operator (See page 95.)



### HCOLOR command

The HCOLOR command is used in high resolution graphics to set the color for the PLOT statement. The colors are numbered from 0 to 7, not 0 to 15 as in low resolution graphics.

Format: HCOLOR=n where n is a number from 0 to 7

Also see: COLOR, HGR

- Apple* • The HCOLOR command is used only in Applesoft.

### HCOLOR Colors For High Resolution

0 Black 1	4 Black 2
1 Green	5 Orange
2 Violet	6 Blue
3 White 1	7 White 2

**HGR** command

The HGR command puts the computer in the high resolution graphics mode. It clears the top 280-by-160 character area to black, and reserves the bottom four lines for text.

The HGR2 command (used only in Applesoft) clears the entire 280-by-192 character area of the screen to black.

Also see: HCOLOR, HPLOT, SHLOAD, DRAW, ROT, SCALE

*Apple* • The HGR command is used only in Applesoft.

HGR

### High Resolution Graphics Commands and Statements

HGR2

HCOLOR sets the color (0 to 7).

HPLOT draws a horizontal line.

SHLOAD loads a shape table from cassette.

DRAW draws a shape.

ROT sets rotation.

SCALE sets scale.

**HIMEM:** command

The HIMEM: command sets the highest memory address available for the Applesoft program use.

*Apple* • The HIMEM: command is used only in Applesoft.

HIMEM:

**HLIN-AT** statement

The HLIN-AT statement displays a horizontal line at a specified row on the screen.

The GR statement must be executed before the HLIN-AT, or any other graphics statement, can be implemented. The color of the line is controlled by the COLOR statement.

Also see: SET, STRING\$, GR, COLOR

*Apple* • HLIN is used in both Applesoft and Integer BASIC.

HLIN-AT HLIN Y1,Y2 AT X draws a horizontal line from the point Y1,X to the point Y2,X. HLIN is used in Low Resolution Graphics.

*PET* • The PET uses the special graphics keys to draw lines.

*TRS-80* • Level I and Level II BASIC use the SET command and also the STRING\$(number, ASCII code) function to draw lines.

**HOME** command

The HOME command clears the screen and puts the cursor at the top left corner of the screen.

Also see: CALL, CLS, CLEAR, CLR/HOME key, ESC E, ESC F

*Apple*  
HOME • The HOME command is used only in Applesoft. Integer BASIC uses the CALL-936 command. (Note: Applesoft can also use the CALL-936 command.)

To clear part of the screen first press ESC to get into the edit mode. Then move the cursor to the desired character and press the F key to erase or clear the rest of the screen (down to the bottom). Pressing the ESC key and then the E key erases the rest of the line in which the cursor is located.

*PET*  
HOME key • The PET HOME key puts the cursor at the top left corner of the screen. In a PRINT statement, the cursor can be returned home by the use of the reverse field character  $\overleftarrow{\square}$ . Type PRINT"CLR/HOME" or "?CLR/HOME"

*TRS-80*  
CLS  
CLEAR  
key • The Level I and Level II BASIC CLS command or statement and the TRS-80 CLEAR key both clear the screen and place the cursor at the top left corner of the screen. The ENTER key must be pressed after the CLEAR key.

### **H PLOT** statement

The H PLOT statement places a colored graphics block at a specified location.

Example: 10 H PLOT 30,20 places a colored graphics block at the horizontal co-ordinate 30 and the vertical co-ordinate 20.

The horizontal co-ordinate can be any point from 0 to 279. The vertical co-ordinate can be any point from 0 to 159 in the HGR mode and from 0 to 191 in the HGR2 mode.

Point 0,0 is located in the upper left corner of the screen.

On all three computers the POKE command and CHR\$ function can be used with the PRINT statement to place graphics characters on the screen.

Also see: HGR, COLOR HGR2, PRINT@, PRINT AT, SET, RESET, CHR\$, POKE

*Apple*  
H PLOT • The H PLOT statement is used only in Applesoft.

*PET* • The PET uses cursor control and graphics keys to draw graphics on the screen. Placing the cursor control keys inside the program allows the placement of graphics at specific locations on the screen.

*TRS-80* • The SET, RESET, PRINT@ or PRINTAT statements in Level I and Level II BASIC can be used to place graphics and other characters at specific locations on the screen.

**HPLOT TO** statement

The HPLOT TO statement plots or draws a line from the first specified point to the second specified point.

Format: HPLOT X1,Y1 TO X2,Y2

Example: 10 HPLOT 20,30 TO 40,50 draws a line from point 20,30 to point 40,50.

The statement may draw to multiple points.

Example: 10 HPLOT 20,30 TO 40,50 TO 60,70 TO 100,200

Also see: COLOR, HGR, HGR2, HLINE, VLIN, POKE, STRING\$

*Apple* • The HPLOT TO statement is used only in Applesoft HPLOT TO BASIC.

*PET* • The POKE statement can be used in PET BASIC to print POKE lines of graphics.

*TRS-80* • The STRING\$ statement can be used in Level I and STRING\$ Level II BASIC to print graphics in a horizontal line POKE across the screen. The POKE statement can also be used to print lines of graphics.

**HTAB** command

Like VTAB, HTAB starts a printed line at a specified location on the screen.

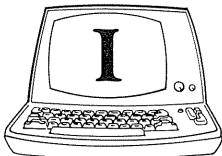
Example: 10 HTAB 14 places the cursor at column 14 (there are 40 columns). Unlike TAB, the HTAB command does not have to be used in a PRINT statement.

Also see: TAB, VTAB, SPC

*Apple* • The HTAB command is used only in Applesoft BASIC. HTAB TAB is used in Integer BASIC. TAB can also be used in Applesoft BASIC. HTAB 5 = TAB 5

*PET* • The TAB and SPC commands are used in PET BASIC to TAB skip a number of spaces during PRINTing.

*TRS-80* • The TAB command is used in Level II BASIC. T. is used TAB in Level I BASIC.

**I.** function, statement

I. is an abbreviation for the INT function.

Also see: INT

- Apple*    • The INT function is used in Applesoft BASIC.  
INT
- PET*        • The INT function is used in PET BASIC.  
INT
- TRS-80*    • The I. function is used in Level I BASIC. Level II BASIC  
I.            uses INT.

**IF (ANSI) statement**

The IF statement is used in conditional branching statements such as IF-THEN, IF-GOSUB, IF-LET, IF-GOTO, IF-GOS., etc. It tests expression values using the relational operators =, >, <, =<, =>, and <>. If the test is passed or the assertion (e.g. X=3) is true, execution goes to the specified line after THEN, GOSUB, GOTO, etc. If the test fails or the assertion is false, then execution jumps to the next numbered line.

Example:    10 IF X=0 GOTO 1000  
              20 PRINT"X DOES NOT EQUAL 0"  
              1000 PRINT"X EQUALS 0"

If the X=0 condition is met, then execution goes to line 1000. If X<>0, then execution continues to the next numbered line—that is to line 20.

Also see: THEN, GOTO, LET, GOS., G., GOSUB

- Apple*    • The IF statement is used in both Applesoft BASIC and  
IF            Integer BASIC.
- PET*        • The IF statement is used in PET BASIC.  
IF
- TRS-80*    • The IF statement is used in both Level I and Level II  
IF            BASIC.

**IF-G. statement**

The IF-G. statement is an abbreviation for the IF-GOTO statement.

Also see: IF-GOTO

- Apple*    • The IF-GOTO statement is used in both Applesoft and  
IF-GOTO    Integer BASIC.
- PET*        • The IF-GOTO statement is used in PET BASIC.  
IF-GOTO
- TRS-80*    • The IF-G. statement is used in Level I BASIC. Level II  
IF-G.        BASIC uses IF-GOTO.

**IF-GOS. statement**

The IF-GOS. statement is an abbreviation for the IF-GOSUB statement.

Also see: IF-GOSUB, IF-THEN

*Apple* • The IF-GOSUB statement is used in both Applesoft and IF-Integer BASIC.

GOSUB

*PET* • The IF-GOSUB statement can be simulated in PET IF THEN BASIC by the use of the IF-THEN statement and the GOSUB GOSUB statement.

Example: 10 IF N=2 THEN GOSUB 1000. This statement would be the same as 10 IF N=2 GOSUB 1000

*TRS-80* • The IF-GOS. statement is used in Level I BASIC. Level IF-GOS. II BASIC uses IF-GOSUB.

### **IF-GOSUB** statement

IF-GOSUB is a condition branching statement which uses one of the relational operators =, <=, >=, <>, >, <. If the condition in the IF-GOSUB statement is met, then execution continues at the specified line number, which is found after GOSUB. If the condition is false, then execution continues at the next numbered line. To return to the main part of the program a RETURN command must be used at the end of the subroutine.

Example: 10 IF X = 1 GOSUB 1000

The subroutine which starts at line 1000 will be executed only if X = 1. If X equals anything else, execution of the program continues at the next numbered line.

Also see: IF-GOS., GOSUB, IF-THEN, IF-GOTO

*Apple* • The IF-GOSUB statement is used in both Applesoft and IF-Integer BASIC.

GOSUB

*PET* • The IF-GOSUB statement can be simulated in PET IF THEN BASIC by the use of the IF-THEN statement and the GOSUB GOSUB statement, or by the use of the IF-GOTO statement.

Example: To simulate IF N=2 GOSUB 1000 use the following lines.

10 IF N=2 THEN GOSUB 1000

or

10 IF N=2 GOTO 100

100 GOSUB 1000

*TRS-80* • The IF-GOSUB statement is used in both Level I and IF-Level II BASIC. Level I BASIC also uses If-GOS.

GOSUB

### **IF-GOTO** statement

IF-GOTO is the same as IF-THEN, but IF-GOTO must be followed by a line number.

Format: IF (condition) GOTO (program line number)

Also see: IF-THEN, IF-G.

*Apple* • The IF-GOTO statement is used in both Applesoft and Integer BASIC.

*PET* • The IF-GOTO statement is used in PET BASIC.  
IF-GOTO

*TRS-80* • The IF-GOTO statement is used in both Level I IF-GOTO and Level II BASIC. Level I BASIC also uses the IF-G. statement.

### **IF-T.** statement

IF-T. is an abbreviation for the IF-THEN statement.

Also see: IF-THEN, THEN, IF

*Apple* • The IF-THEN statement is used in both Applesoft and IF-THEN Integer BASIC.

*PET* • The IF-THEN statement is used in PET BASIC.  
IF-THEN

*TRS-80* • The IF-T. statement is used in Level I BASIC. Level II IF-T. BASIC uses the IF-THEN statement.

### **IF-THEN** (ANSI) statement

IF-THEN is a conditional branching statement which uses one of the relational operators =, <=, >=, <>, >, <. If the condition is met, then execution continues at the line specified after THEN. If the condition is not met, then execution continues at the next numbered line. (Exception: See Integer BASIC below.) Commands may be used after THEN.

Example: 10 IF A=6 THEN STOP

In the example, if A=6 the program stops, but if A equals anything else, the program continues at the next numbered line.

Format: IF (condition) THEN (program line number)  
IF (condition) THEN (statement or command)

Also see: IF, THEN, IF-T.

*Apple* • The IF-THEN statement is used in both Applesoft and IF-THEN Integer BASIC.

Example: 50 IF A=10 THEN A=A+1: B=A+10

In most BASICs, including Applesoft, if A=10 then A=A+1 and B=A+10. However, if A<>10, then execution continues at the next numbered line. In Integer BASIC, if A#10, then the first statement (A=A+1) is ignored, *but* the second statement (B=A+10) is executed. Only in Integer BASIC is the first statement ignored when the IF THEN statement is false. Change our example to the following:

```
50 IF A#10 THEN 60: A=A+1: B=A+10
60 . . . .
```



Here if A is not equal to 10, then the rest of the line is skipped.

*PET* • The IF-THEN statement is used in PET BASIC.  
IF-THEN

*TRS-80* • The IF-THEN statement is used in both Level I  
IF-THEN and Level II BASIC. Level I BASIC also uses the IF-T.  
statement.

**IN.** statement

IN. is an abbreviation for the INPUT statement.

Also see: INPUT

*Apple* • INPUT is used in both Integer BASIC and in Applesoft.  
INPUT

*PET* • The INPUT statement is used in PET BASIC.  
INPUT

*TRS-80* • The IN. statement is used in Level I BASIC. Level II  
IN. BASIC uses INPUT.

**IN#** command

The IN# command is used to direct the program to take input data from a slot number where a peripheral is connected.

Example: IN# 4 tells the computer that a peripheral is connected to slot #4, and that future input will come through slot #4 and not the keyboard. To reset to normal, enter the command IN#0.

*Apple* • IN# is used in both Applesoft and Integer BASIC.  
IN#

**INKEY\$** function

The INKEY\$ function reads a character from the keyboard without stopping execution, while waiting for the ENTER key to be pressed. The computer scans the keyboard until it receives a message.

Example: 10 IF INKEY\$="A" THEN 100 ELSE 10

When the A key is pressed, the program jumps to line 100. If no key, or the wrong key, is pressed then the program loops back to line 10 (ELSE 10). This continues until the proper character is entered.

Also see: GET\$, GET, ENTER

*Apple* • The GET\$ statement is used only in Applesoft BASIC.  
GET\$

*PET* • The GET statement is used in PET BASIC.  
GET

*TRS-80* • The INKEY\$ statement is used in Level II BASIC.  
INKEY\$

**INP** statement

The INP statement inputs a decimal value, from 0 to 255, from a port.

*TRS-80* • The INP statement is used in Level II BASIC.

INP

**INPUT** (ANSI) statement

The INPUT statement is used to input a numeric or string variable from the keyboard. A question mark (?) is printed on the screen and execution halts until a response is entered. ENTER or RETURN must be pressed when the input is complete.

Example: 10 PRINT"WHAT IS YOUR NAME"  
20 INPUT A\$

The INPUT has a built-in ability to output text for prompts. The above example could be shortened to 10 INPUT"WHAT IS YOUR NAME";A\$.

One INPUT statement could ask for 2 or more responses from the user.

Example: 10 INPUT"WHAT IS YOUR NAME AGE AND  
GRADE";A\$,B,C

A comma must separate each of the INPUTs. When an INPUT statement is used, as in the last two examples, a semicolon separates it from the string variable. Integer BASIC, however, uses a comma.

*Apple* • The INPUT statement is used in both Applesoft and Integer BASIC. Applesoft can use a string response to the INPUT statement.

Applesoft BASIC uses a semicolon after INPUT as in 50 INPUT "ANSWER";N. Integer BASIC uses a comma after the INPUT statement: 50 INPUT "ANSWER",N

*PET* • The INPUT statement is used in PET BASIC. The maximum length of the INPUT must not exceed 79 characters. INPUT can have a statement contained within it: 50 INPUT "5\*2 = ";A

*TRS-80* • The INPUT statement is used in both Level I and Level II BASIC. The INPUT statement accepts letters, numbers, or a combination of both.

**INPUT#** statement

The INPUT# statement is used when taking input data from a cassette tape. The number specified after the # (number sign) indicates the number of the INPUT device (usually 1 for the cassette). If the DATA is not loaded properly, an error message is printed.

Also see: CLOSE, OPEN, PRINT#, GET#

- PET* • The INPUT# statement is used in PET BASIC.
- INPUT# INPUT#1,variable (the variable can be numeric or string)  
 Example: 10 OPEN1,1,0"READ" opens file 1 entitled READ  
 20 INPUT#1,A inputs the value of A  
 30 CLOSE 1 closes file 1

If the data is not properly loaded into the computer, an error message is printed: ?BAD DATA ERROR.  
 GET is also used to input data in PET BASIC.

- TRS-80* • INPUT#-1, variable (the variable can be numeric or string)  
 INPUT#-1 is used in both Level I and Level II BASIC.

### **INST/DEL** key

The INST/DEL key moves all characters to the right of the cursor, one space to the right, replacing the character to the left of the cursor. One character is deleted at a time.

One space is inserted by pressing the SHIFT and INST/DEL keys. This space may then be filled with a character.

Also see: DELETE, DEL, (left arrow)

- Apple* • The DEL key on the Apple II computer is used to delete DEL characters.

- PET* • The INST/DEL key on the PET computer is used to insert or delete characters. CHR\$(20) can be used to delete during a program run. CHR\$(148) can be used to insert during a program run.

- TRS-80* • Level II BASIC uses the DELETE command to delete DELETE characters from memory.

### **INT** (ANSI) function

The INT function is used to find the integer value of a number. The numbers are rounded down to the largest integer equal to or less than the argument. Only numbers between -32767 and +32767 can be used.

Example: 10 INT(-3.5) prints -4  
 20 INT(9.9) prints 9  
 30 INT(.25) prints 0

- Apple* • The INT function is used only in Applesoft. Integer INT BASIC has no INT function because all values are already integers. When translating a program to Integer BASIC, delete all INT( ) functions.

*PET* • The INT function is used in PET BASIC.  
INT

*TRS-80* • The INT function is used in both Level I and Level II  
INT BASIC.

**INVERSE** command, statement

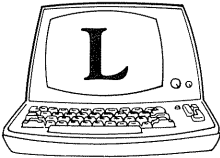
The INVERSE command sets the screen output to black on white instead of the usual white on black. The NORMAL command returns the screen output to white characters on black.

Also see: OFF/RVS key, SET

*Apple* • The INVERSE command is used only in Applesoft. In-  
INVERSE teger BASIC uses the POKE 50,127 statement to achieve the same result.

*PET* • The PET uses the RVS key to produce the reverse field.  
RVS key The reverse field can be inserted into a program run by using CHR\$(18). The reset reverse field can be inserted into a program by using CHR\$(146).

*TRS-80* • A reverse field can be achieved in graphics by using the SET command.



**L.** command

The L. command is an abbreviation for the LIST command.

Also see: LIST

*Apple* • The LIST command is used in both Applesoft and In-  
LIST teger BASIC.

*PET* • The LIST command is used in PET BASIC.  
LIST

*TRS-80* • The L. command is used in Level I BASIC. Level II  
L. BASIC uses the LIST command.

← (**left arrow**) key

The ← (left arrow) deletes one character from the line being typed, and moves the cursor one space to the left.

Also see: CRSR

*Apple* • The ← (left arrow) key is used on the Apple II computer  
← to delete characters to the left of the cursor.

- PET*  
*CRSR*
- Pressing the SHIFT key and the cursor  $\overline{\text{CRSR}}$  key at the same time moves the cursor one space to the left. To delete or change a character, another character must be typed when the cursor is on top of the one that is to be replaced or changed. The reverse field character  $\overline{\text{||}}$  can be used in PRINT statements to achieve the same results.
- TRS-80*  
←
- The ← (left arrow) key is used on the TRS-80 computer. A number specified before the ← (left arrow), while in edit mode, moves the cursor the specified number of spaces to the left, and erases all the characters in between. Pressing the SHIFT ← (left arrow) deletes the entire line currently being typed.

**LEFT\$** function

The LEFT\$ function is used to select part of a string for use elsewhere. The LEFT\$("string", number) function returns the specified number of string characters, starting from the left side of the string. A comma must separate the string and the number, and quotation marks must be used around the printed string ("ARTIST") if it is not a string variable (A\$).

Example 1: 10 PRINT LEFT\$ ("ARTIST",3) returns the first 3 letters or ART.

Example 2: 10 A\$="ARTIST"  
20 PRINT LEFT\$(A\$,3)

This also returns the first three letters or ART. The string and/or number inside the argument may be variables.

Example 3: 10 LEFT\$(A\$,X) where both A\$ and X are variables that are stored elsewhere in the program

If the number inside the argument is larger than the number of characters in the string, the whole string will be returned.

Also see: MID\$, RIGHT\$

*Apple*  
LEFT\$

- The LEFT\$ function is used only in Applesoft. Though LEFT\$ is not used in Integer BASIC, there are other string handling notations. A\$(N) prints all the characters of A\$ starting with the Nth character. A\$(N,P) prints all the characters starting with the Nth character, and ending with the Pth character.

To translate LEFT\$(A\$,3) to Integer BASIC the user would type A\$(1,3). In general LEFT\$(A\$,N) = A\$(1,N).

*PET*  
LEFT\$

- The LEFT\$ function is used in PET BASIC. If the specified number used after the LEFT\$ is negative, zero or greater than 255 then an ILLEGAL QUANTITY ERROR is printed.

*TRS-80*  
LEFT\$

- The LEFT\$ function is used only in Level II BASIC.

**LEN** function

The LEN function measures the length of strings by counting the number of characters in the string variables. Non-printable characters and blanks are also counted.

Example 1: 10 PRINT LEN("ARTIST") prints 6

Example 2: 10 PRINT "WHAT IS YOUR NAME"  
 20 INPUT A\$  
 30 PRINT"YOUR NAME CONTAINS";  
 40 PRINT LEN(A\$); " CHARACTERS"

The program in Example 2 prints the number of characters in your name (A\$).

*Apple* • The LEN function is used only in Applesoft.  
 LEN

*PET* • The LEN statement is used in PET BASIC. It may be  
 LEN used as part of a numeric expression.

*TRS-80* • The LEN function is used only in Level II BASIC.  
 LEN

< (**less than**) (ANSI) operator (See page 96.)

< = (**less than or equal**) operator (See page 96.)

**LET** (ANSI) statement

The LET statement is used to assign numeric values to variables.

Example: 10 LET A = 15

LET is optional LET A = B produces the same result as A = B.

*Apple* • The LET statement is used in both Applesoft and In-  
 LET teger BASIC.

*PET* • The LET statement is used in PET BASIC.  
 LET

*TRS-80* • The LET statement is used in both Level I and Level II  
 LET BASIC.

**LIST** command

The LIST command displays a program listing starting with the lowest number. The listing normally scrolls to the end unless it is stopped. LIST 10 lists only line 10. LIST 10-100 lists all lines from 10 to 100 inclusive.

Also see: L.

*Apple* • The LIST command is used in both Applesoft and In-  
 LIST teger BASIC.

*PET* • The LIST command is used in PET BASIC. This com-  
 LIST mand may be entered as LIST or LIST- to list the entire program. LIST A- lists the program from line A to the end. LIST -A lists the program up to line A.

*TRS-80* • The LIST command is used in Level II BASIC. In Level I LIST BASIC L. is used.

Pressing the SHIFT@ at the same time stops scrolling. The listing of the program continues when any character key is pressed.

### **LOAD** command

The LOAD command is used to load a program from tape into the computer. The program in computer memory is erased as the new program is loaded.

Also see: CLOAD

*Apple* • The LOAD command which loads a program from tape LOAD is used in both Applesoft and Integer BASIC. Only RESET can interrupt a LOAD.

*PET* • The LOAD command is used in PET BASIC. It has LOAD several forms:

LOAD loads the first program encountered on the tape.

LOAD"GAME" searches for the file named "GAME" and then loads it.

LOAD"GAME",2 searches for the file on input device #2; when found, it is loaded.

10 LOAD"GAME" is used in a program. Execution of the program stops until "GAME" is loaded. The new program starts execution from its lowest number.

Pressing L and then SHIFT O starts the loading process without printing out the whole word on the keyboard.

Pressing the SHIFT RUN/STOP keys at the same time also starts loading from the cassette tape (2.0 ROM's).

*TRS-80* • The CLOAD command is used in Level I and Level II CLOAD BASIC. CLOAD"A" searches for the program named "A"; when found, it is loaded.

### **LOG** (ANSI) function

The LOG(n) function computes the natural logarithm of the specific argument (number inside the brackets). To compute common logs from natural logs, multiply the natural log by .434295. To compute the natural log from a common log, multiply the common log by 2.3026.

Example: 10 X=LOG(2)

computes the natural log of the number 2.

*Apple* • The LOG function is used only in Applesoft. Programs LOG using LOG cannot be translated directly into Integer BASIC.

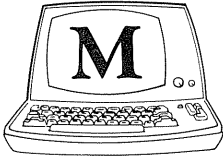
*PET*     • The LOG function is used in PET BASIC.  
LOG

*TRS-80*   • The LOG function is used in Level II BASIC.  
LOG

**LOMEM:** command

The LOMEM: command sets the lowest memory address available to the Applesoft program user.

*Apple*     • The LOMEM: command is used only in Applesoft.  
LOMEM:



**M.** function, command, statement

M. is an abbreviation for the MEM command and statement.

Also see: MEM, FRE

*Apple*     • The FRE(0) function is used only in Applesoft. It returns  
FRE(0)     the amount of memory still available to the operator.

*PET*       • The FRE(0) function is used in PET BASIC. It reports  
FRE         the total amount of unused memory space.

*TRS-80*   • The M. function is used in Level I BASIC. In Level II  
M.          BASIC MEM is used. FRE(0) and FRE(A\$) are also used  
            in Level II BASIC.

**MAN** command

If the computer is in the automatic line numbering mode, the operator must press control X (CTRL X), then MAN and RETURN to begin inserting program lines manually.

Also see: AUTO

*Apple*     • The MAN command is used only in Integer BASIC.  
MAN

*TRS-80*   • In Level II BASIC if the computer is in the automatic  
BREAK     line numbering mode, pressing the BREAK key returns  
key         the computer to the monitor or command mode, which  
            requires the manual insertion of line numbers.

**MEM** command, statement

MEM is used with the PRINT command to tell the user the number of unused bytes of memory remaining in the computer. It can also be used in a program line or subroutine such as

```
1000 PRINT MEM;"BYTES OF MEMORY REMAINING"
1001 RETURN
```



In longer programs, this helps keep track of how much memory space is left.

Also see: M., FRE

- Apple* • The FRE(0) function is used only in Applesoft. It returns  
FRE(0) the amount of memory still available.
- PET* • The FRE(0) command is used in PET BASIC. It reports  
FRE(0) the total amount of unused memory space.
- TRS-80* • In Level II BASIC PRINT MEM displays the amount of  
MEM memory space still available. If the statement PRINT  
MEM is used just after the computer is powered up, the  
following will appear:  
for 4K—3284 bytes  
for 8K—7428 bytes  
for 16K—15,572 bytes  
for 32K—31,288 bytes  
for 64K—63,274 bytes

Level II also uses FRE(0) and FRE(A\$).

### **MID\$** function

The MID\$ function selects characters from the *middle* of a string. The MID\$("STRING", number over, number of letters) function displays a specified number of letters or elements in a string. The starting letter or element is specified by the number over from the beginning of the string. The string must be in quotation marks (unless it is a variable like A\$). The string and numbers must be separated by commas.

Example 1: 10 PRINT MID\$("AUTOMATIC",5,3)

Example 1 prints MAT, which starts with the 5th element of the string and is 3 letters long.

Example 2: 10 A\$="AUTOMATIC"  
20 PRINT MID\$(A\$,5,3)

This also prints MAT.

As with the LEFT\$ and RIGHT\$ functions, the arguments may all be variables as in MID\$(A\$,X,Y). The A\$ must be a stored string, while X and Y must be numeric variables stored earlier in the program.

Also see: LEFT\$, RIGHT\$

- Apple* • The MID\$ function is used only in Applesoft. Integer  
MID\$ BASIC does not use MID\$; however there are other  
string handling notations. A\$(N) prints all the  
characters of A\$ starting with the Nth character.  
A\$(N,P) prints all the characters starting with the Nth  
character and ending with the Pth character.

To translate  $MID\$(A\$,2,3)$  to Integer BASIC, the user would type  $A\$(2,4)$ . In general  $MID\$(A\$,N,P) = A\$(N,N+P-1)$ .

*PET*     • The  $MID\$(A\$,N,P)$  function is used in PET BASIC. If either of the specified numbers are less than zero or greater than 255, an **ILLEGAL QUANTITY ERROR** is printed on the screen.

On the PET,  $MID\$(A\$,N,P)$  may be written as  $MID\$(STRING, number)$ . In this example all the characters from the specified number to the end will be written. This makes it similar to the  $RIGHT\$(A\$,N)$  function.

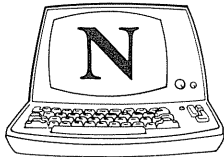
*TRS-80*     • The  $MID\$(A\$,N,P)$  function is used only in Level II BASIC.  
*MID\$*

### **MOD** function

$A \text{ MOD } B$  computes the remainder when  $A$  is divided by  $B$ .

Example: `10 PRINT 11 MOD 5` prints the number 1. (1 is the remainder when 11 is divided by 5.)

*Apple*     • The MOD function is used only in Integer BASIC.  
*MOD*



**N.** command, statement

$N.$  is used as an abbreviation for the **NEW** command and the **NEXT** statement.

Also see: **NEW**, **NEXT**

*Apple*     • **NEW** and **NEXT** are both used in Applesoft and Integer BASIC.  
*NEW*  
*NEXT*

*PET*     • **NEW** and **NEXT** are both used in PET BASIC.  
*NEW*  
*NEXT*

*TRS-80*     • The  $N.$  command and statement is used in Level I BASIC. Level II uses **NEW** and **NEXT**.

### **NEW** command

The **NEW** command erases the resident BASIC program from computer memory. It deletes all program lines and all variables.

Also see:  $N.$ ,  $CLEAR\{n\}$ ,  $CLR/HOME$

- Apple*  
NEW • The NEW command is used in both Applesoft and Integer BASIC. It deletes the current program in memory so that a new one may be entered.
- PET*  
NEW • The NEW command is used in PET BASIC. It deletes the resident program and all variables. The screen memory is not affected by the NEW command. Even though the program is erased from memory, the screen still shows the last video monitor display from that program.
- TRS-80*  
NEW • The NEW command is used in Level II BASIC. Level I BASIC uses N. NEW does not change the string space allocated by a previously executed CLEAR(n) statement.
- When the NEW command is executed, the screen memory is cleared, and the screen goes blank, except for the cursor in the upper left hand corner.

**NEXT** (ANSI) statement

The NEXT statement is used in a FOR-NEXT loop to return execution of the program to the first statement of the loop. When the loop value exceeds the limit specified, the loop terminates and the statement following the NEXT statement is executed.

Example: 10 FOR A=1 TO 15  
20 PRINT A  
30 NEXT A

NEXT A causes the program to jump back to line 10. In this program the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15 are printed. The A is optional in some BASIC languages.

Also see: N., FOR

- Apple*  
NEXT • The NEXT statement is used in both Applesoft and Integer BASIC. In Integer BASIC the NEXT statement must be followed by a variable. In Applesoft the variable name is optional.
- PET*  
NEXT • The NEXT statement is used in PET BASIC.
- TRS-80*  
NEXT • The NEXT statement is used in Level II BASIC. In Level I BASIC N. may be used.

**NORMAL** command, statement

The NORMAL command or statement returns the screen to the normal display, which is white on black. It turns off the INVERSE and/or FLASH commands.

Also see: INVERSE, FLASH, OFF/RVS key, CHR\$( ), CLS

- Apple* • The NORMAL command is used only in Applesoft. In NORMAL Integer BASIC POKE 50,255 achieves the same result.

*PET* • The PET RVS key produces a black on white display.  
*SHIFT* Pressing the SHIFT key and the RVS key at the same  
*RVS* time returns the display to the normal white on black.

The RVS key and SHIFT RVS key can be programmed by putting them inside quotation marks. See OFF/RVS for more details on the RVS key.

*TRS-80* • In Level II BASIC the CHR\$(23) function and CLS com-  
*CHR\$(23)* mand are used to change from one screen format to  
*CLS* another.

CHR\$(23) changes the screen display to 32 characters per line. The CLS command returns the display to the normal 64 characters per line.

**NOT** operator (See page 96.)

< > (**not equal**)(ANSI) operator (See page 97.)

**NOTRACE** command, statement

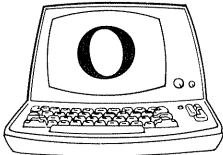
The NOTRACE command stops the TRACE function. Both TRACE and NOTRACE may be used in program statements. TRACE displays the program lines as they are executed.

Also see: TRACE, TROFF, TRON

*Apple* • The NOTRACE command is used in both Applesoft and  
*NO-* Integer BASIC.  
*TRACE*

*TRS-80* • The TROFF command is used in Level II BASIC to turn  
*TROFF* the TRACE off.

# (**number sign**) operator (See page 97.)



**OFF/RVS** key

The OFF/RVS key, when pressed, displays the remainder of the line being typed in reverse field. Reverse field is black on white instead of the usual white on black. The SHIFT OFF/RVS key turns the reverse field off, or turns it back to normal.

The reverse field can be placed in a PRINT statement by placing the reverse field character R in quotation marks.

Example: 25 PRINT " OFF/RVS " is typed but a reverse field R appears in the quotation marks.

The reset can be placed in a PRINT statement by typing PRINT "SHIFT OFF/RVS".

Also see: INVERSE, NORMAL, FLASH

*Apple* • The INVERSE command is used only in Applesoft. In INVERSE teiger BASIC uses POKE 50,127.

*PET* • The PET uses the OFF/RVS key.

OFF/RVS The reverse field command can be inserted into a program by using CHR\$(18).

The off reverse, or reset reverse field command, can be inserted into a program by using CHR\$(146). This key can also be used in a program.

Example: 10 PRINT"RVS THIS IS A DEMO  
SHIFT RVS"

In this example the words THIS IS A DEMO will be displayed in reverse field, black on white. SHIFT RVS returns the display to normal.

□ represents a single or double key, not individual characters.

### **ON ERROR GOTO** statement

The ON ERROR GOTO statement is used to branch to an error recovery routine when an error is encountered during execution. This prevents the program from stopping.

Example: 10 ON ERROR GOTO 100  
20 PRINT "7 DIVIDED BY 0 IS"  
30 A = 7/0  
100 PRINT "DIVISION BY ZERO IS IMPOSSIBLE."

When the error in line 30 is encountered, the program jumps to line 100.

Also see: ONERR GOTO, ST.

*Apple* • The ONERR GOTO statement is used in Applesoft.  
ONERR  
GOTO

*PET* • The ST statement is used by the PET to check the status of an I/O (INPUT/OUTPUT) operation. It can be used as an ON ERROR GOTO statement. If ST=0, then there are no errors in I/O. We can use the statement 100 IF ST < > 0 THEN 1000. If the status is not equal to zero (e.g. there is an error), jump to line 1000.

See the ST. statement for information.

*TRS-80* • The ON ERROR GOTO statement is used in Level II BASIC.  
ON  
ERROR  
GOTO

**ON G.** statement

The ON G. statement is an abbreviation for the ON GOTO statement.

Also see: ON GOTO

*Apple* • The ON GOTO statement is used only in Applesoft.

ON  
GOTO

*PET* • The ON GOTO statement is used in PET BASIC.

ON  
GOTO

*TRS-80* • The ON G. statement is used in Level I BASIC. Level II  
ON G. BASIC uses ON GOTO.

**ON GOSUB** statement

ON GOSUB is a branching statement which evaluates the expression after ON, and then goes to one of the subroutines specified by the line numbers after GOSUB.

Example 1: 10 ON A GOSUB 1000,2000,4000

If A is 1, execution goes to line 1000; but if A is 2, it goes to line 2000, and if A is 3, execution jumps to the subroutine starting at line 4000. If the integer value of A is less than 1 or more than 3, the test fails and the program does not branch to any subroutine. The program may continue at the next line, or it may even crash.

Example 2: 100 ON A GOSUB 15000,20000

In Example 2, the integer value of A must be 1 or 2 to insure that one of the subroutines will be executed. A RETURN is located at the end of the subroutine.

Also see: GOSUB, RETURN

*Apple* • The ON GOSUB statement is used only in Applesoft.

ON  
GOSUB

*PET* • The ON GOSUB statement is used in PET BASIC.

ON  
GOSUB

*TRS-80* • The ON GOSUB statement is used in both Level I and  
ON Level II BASIC.

GOSUB

**ON GOTO** statement

The ON GOTO statement is a multiple branching statement.

Example: 10 ON Z GOTO 1000,2000,4000

Z is a numeric variable. If Z is 1, GOTO 1000 is executed. If Z is 2 or 3, execution jumps to line 2000 or 4000 respectively. If the integer value of Z is less than 1 or more than 3, the test in the ON GOTO statement fails, and the program may either proceed at the next line or crash.

Also see: GOTO, ON GOSUB

*Apple* • The ON GOTO statement is used only in Applesoft  
ON BASIC.

GOTO Example: 100 ON N GOTO 1000,2000,3000

Integer BASIC can simulate ON GOTO with the following:

```
100 IF N=1 THEN 1000
```

```
101 IF N=2 THEN 2000
```

```
102 IF N=3 THEN 3000
```

Another method is to use 100 GOTO N\*1000. This can be used when the lines to GOTO are incremented evenly.

*PET* • The ON GOTO statement is used in PET BASIC.

ON  
GOTO

*TRS-80* • The ON GOTO statement is used in both Level I and  
ON Level II BASIC.

GOTO

### **ONERR GOTO** statement

The ONERR GOTO statement causes the program to branch to an error recovery routine when an error occurs. This prevents the program from stopping and an error message being printed.

Also see: ON ERROR GOTO, ST.

*Apple* • The ONERR GOTO statement is used only in  
ONERR Applesoft.

GOTO

*PET* • The ST statement is used by the PET to check the status  
ST of an I/O (INPUT/OUTPUT) operation. It can be used as an ONERR GOTO statement when I/O operations are involved. If ST=0 then there are no errors in I/O. We can use the statement 100 IF ST<>0 THEN 1000. If the status is not equal to zero (e.g., there is an error), execution of the program jumps to line 1000.

See the ST. statement for more information.

*TRS-80* • The ON ERROR GOTO statement is used in Level II  
ON BASIC.

ERROR  
GOTO

**OPEN** command, statement

The OPEN command is used to open a file so that DATA can be read in.

Format: 10 OPEN 1,1,0"READ"

file #1	1 input device (1 cassette)	open tape to look for file entitled "READ"
---------	--------------------------------	--

CLOSE is used to CLOSE the file.

Also see: CLOSE

*PET* • The OPEN statement is used in PET BASIC. The OPEN parameters for the OPEN statement are OPEN F, D, S, F\$.

F—File number to be used in INPUT#, PRINT# and CLOSE statements

D—Physical device number

0 = keyboard

1 = 1st cassette tape recorder

2 = 2nd cassette tape recorder

4 = printer

8 = disk drive

S—Secondary address for tape recorder

0 = read file from the tape

1 = write file to the tape

2 = write file (end with the end of the tape)

To write to TAPE 1 the following would be used in the program.

OPEN 3,1,2,"name"

PRINT#3,A\$ prints the value of A\$ along channel 3 to the tape recorder and records it.

CLOSE 3 closes file number 3

**OR** operator (See page 98.)

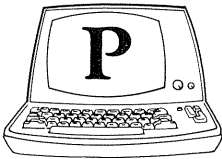
**OUT** statement

The OUT statement sends a number to a specified computer output port. This is often used for sound generation or other types of control operations. The format is OUT port number, byte number.

Example: 10 OUT 255,1

*TRS-80* • The OUT statement is used only in Level II BASIC. In a program, OUT 255,4 turns the cassette recorder on for music or other effects. OUT 255,0 turns the cassette recorder off.



**P.** statement

The P. statement is an abbreviation for the PRINT statement and for POINT.

Also see: PRINT, POINT

*Apple* • The PRINT statement is used in both Applesoft and Integer BASIC.  
PRINT

*PET* • The PRINT statement is used in PET BASIC.  
PRINT

*TRS-80* • The P. statement is used in Level I BASIC. Level II P. BASIC uses the PRINT statement and POINT.

**P.A.** statement

P.A. is the abbreviation for PRINT AT and PRINT@.

Also see: PRINT AT, PRINT@

*TRS-80* • The P.A. statement is used in Level I BASIC. Level I P.A. BASIC can also use the PRINT AT statement to produce the same results.

Level II BASIC uses the PRINT@ statement.

( ) (**parentheses**) (ANSI) operator (See page 98.)

**PDL** function

The PDL function is used with game paddles. The control units are identified as PDL(0) and PDL(1).

*Apple* • The PDL function is used in both Applesoft and Integer PDL BASIC.

PDL stands for game paddle. There can be up to 3 paddle controls. PDL 1, 2 or 3 returns the setting from 0 to 255 of the specified game control paddle.

Format: PDL 1 returns the setting of game control paddle number 1.

**PEEK** statement

The PEEK statement is used to read the contents of a specified address in the computer's memory. (See individual manuals for memory address maps.)

Example: PEEK(A) prints the contents of memory location A. This will be a number between 0 and 255. PEEK is sometimes used

to see what number was POKEd into a memory address, and both PEEK and POKE are often used with the USR(X) statement to start a machine language subroutine.

Also see: POKE, USR( )

*Apple*  
PEEK • The PEEK statement is used in both Applesoft and Integer BASIC. PEEK(-16336) causes a click on the speaker. In PEEK(X-A) if the value of (X-A) is less than 127, the button on the game control X is being pressed.

*PET*  
PEEK • The PEEK statement is used in PET BASIC. In the PET, the contents of any address greater than hexadecimal C000 is always returned as zero.

PEEK can be used to disable the BREAK or STOP key on the 2.0 or 4.0 ROM PETs. The following line disables the STOP on both these memory types: 1 POKE144, PEEK(144)+3. To re-enable the STOP key use this line: 2 POKE144, PEEK(144)-3.

*TRS-80*  
PEEK • The PEEK statement is used in Level II BASIC.

% (percent) operator (See page 99.)

. (period) operator (See page 99.)

**PI**  $\pi$  key

The  $\pi$  key returns a constant of 3.14159265. This is used in numeric operations.

*PET*  
 $\pi$  • The  $\pi$  key is used only on the PET. CHR\$(222) or CHR\$(255) can be used to print out the symbol on the screen.

Example: READY.

```
10 PRINT"THIS IS PI ";CHR$(222)
READY.
```

When the program is run the result is: THIS IS PI  $\pi$ .

**PLOT** statement

The PLOT statement is used to turn on a color graphics block on the screen. The block is specified by two numbers following the plot statement.

Example: PLOT 10,5. The first number (10) indicates the column and the second number (5) indicates the row. The numbers can be from 0 to 39 with 0,0 being the top left block on the screen. The color is black unless otherwise specified.

To turn the color block off, either color the block black (color number 0), or execute the GR statement. The GR statement erases the whole screen.

Also see: GR, SET, S., COLOR

*Apple PLOT* • The PLOT statement is used in both Applesoft and Integer BASIC. It is used in low resolution graphics.

*PET* • The PET uses the graphics keys and characters to light up specific locations, and to do graphics.

*TRS-80 SET(X,Y)* • The SET statement is used in Level I and Level II BASIC. The SET(X,Y) command lights up the specified graphics block. In Level I BASIC S. can be substituted for SET.

+ (**plus sign**) (ANSI) operator (See page 100.)

### **POINT** statement

The POINT statement is used with the IF THEN statement to find out if a specified graphics block is on. A -1 or 1 is returned if the block is on, and a 0 if it is off.

Example: 10 IF POINT(1,6) = -1 THEN 100

If the graphics block (1,6) is on, program execution jumps to line 100. If the test fails, execution proceeds to the next numbered line.

PEEK can be used to look into a screen memory location to discover if a particular character is there.

A = PEEK 15950 sets A equal to the contents of screen memory location 15950 (on the TRS-80). In a baseball game an asterisk (ASCII 42) could be the ball, and the screen location 15950 could be the plate. When the key is pressed to swing the bat, the computer would do the following: 500 IF A = 42 THEN PRINT "A HIT", that is, if the ball is over the plate there is a hit.

Also see: SCRN, PEEK, Appendix A, Appendix C

*Apple SCRN* • SCRN is used to determine the color of a graphics block. If it is not on, a 0, which is the color code for black, is returned.

*TRS-80 POINT* • The POINT statement is used in both Level I and Level II BASIC. A 1 is returned if the block is on, and a 0 if it is off.

In Level I BASIC, a P. can be used.

### **POKE** statement

The POKE statement puts values from 0 to 255 into specified memory locations.

Example: 10 POKE 89,255 pokes the number 255 into memory location 89.

Also see: PEEK, TEXT, STOP, BREAK key, C., CON, CONT

*Apple POKE* • The POKE statement is used in both Applesoft and Integer BASIC. POKE A,65 pokes the value 65 into memory location A.

POKE 50,127 is the Integer BASIC equivalent to the Applesoft INVERSE.

POKE 50,255 is the Integer BASIC equivalent to the Applesoft NORMAL.

POKE 33,33 sets the width of the screen to 33 columns from the normal 40 columns.

Use POKE 33,40, or type TEXT, to reset the column number to 40.

- PET*  
POKE
- The POKE statement is used in PET BASIC.  
The memory address may be from 0 to 65536, and the bytes to be stored must be from 0 to 255 inclusive.  
Format: POKE memory address, number to be poked.  
POKE 59468,14 converts all graphic characters to lower case letters.  
POKE 59486,12 and/or turning the computer off and then on again, returns the PET to its normal graphic character mode.  
POKE 167,0 turns the cursor on (\*548,0)\*Old ROM's  
POKE 167,1 turns the cursor off (\*548,1)
- |                         |                           |
|-------------------------|---------------------------|
| To disable the STOP key | To re-enable the STOP key |
| ROM 1.0 POKE 537,163    | POKE 537,133              |
| ROM 2.0 POKE 144,49     | POKE 144,46               |
| ROM 4.0 POKE 144,88     | POKE 144,85               |
- On the 2.0 and 4.0 ROM PETs POKE 144,PEEK(144) + 3 disables the STOP key. POKE 144,PEEK(144) - 3 re-enables the STOP key.

- TRS-80*  
POKE
- The POKE statement is used in Level II BASIC.  
POKE 16405,0 disables the keyboard.  
POKE 16405,1 re-enables the keyboard. Do not use these except inside program lines.  
POKE 16413,0 disables the video monitor.  
POKE 16413,7 re-enables the video monitor.  
POKE 16396,23 disables the break key.  
POKE 16396,20 (or 16396,201) re-enables the break key.  
POKE 16421,0 disables the printer.  
POKE 16421,6 re-enables the printer.

### **POP** command

The POP command removes one address from the RETURN address stack.

- Apple*  
POP
- The POP command is used in both Applesoft and Integer BASIC.

**POS** command

The POS command is used to determine the next position available to print a character. Any argument can be used. We call this type of argument a dummy argument.

The POS command can be used in a program line.

Example: 100 PRINT "HERE" TAB(POS(0)+10) "TO HERE IS TEN SPACES"

The cursor moves ten spaces more than the position (POS) of the cursor after the first "HERE" is printed.

Also see: TAB, SPC

*Apple*     • The POS( ) command is used only in Applesoft. It returns the current horizontal position of the cursor. The value can be from 1 to 39.

*PET*       • The POS ( ) command is used in PET BASIC. It returns the present horizontal position of the cursor.

*TRS-80*   • The POS command is used in Level II BASIC. It returns a number from 0 to 63 indicating the current position of the cursor on the video screen.

**PR** command

The PR command sends output to a peripheral connected to the specified slot instead of the T.V. screen.

Example: PR#4 sends output through slot #4.

To reset to normal, enter the PR#0 command.

Also see: OUT

*Apple*     • The PR command is used in both Applesoft and Integer PR BASIC.

*TRS-80*   • The OUT statement is used in Level II BASIC. OUT

**PRINT** (ANSI) command, statement

PRINT is used to print numbers or strings on the screen.

Example 1: 10 PRINT A prints the numeric value of the variable A.

Example 2: 10 PRINT A\$ prints the characters stored in the string variable A\$.

Example 3: 10 PRINT "GOOD" prints the contents of the quotation marks.

The use of commas (,) in PRINT statements spaces the output into horizontal zones.

The semicolons (;) used in PRINT statements leave no spaces between letters or words that are separated by them.

TAB is often used with the PRINT statement to insert a number of spaces before the statement is printed.

PRINT by itself on a line leaves an empty line during the run of the program. The ? can be used as a short form for PRINT.

Also see: PRINT AT, TAB, SPC

*Apple* • The PRINT statement is used in both Applesoft and Integer BASIC.  
PRINT

The ? can be used as a short form for PRINT.

SPC(X) is used in Applesoft with the PRINT statement to put X spaces between the last character printed and the next one.

Semicolons (;) are used to concatenate printed items.

Commas (,) separate items into three tab fields on the screen.

*PET* • The PRINT statement is used in PET BASIC.  
PRINT The graphics keys can be used in PRINT statements. This is a good way to do graphics.

*TRS-80* • The PRINT statement is used in both Level I and Level II BASIC. Level I BASIC can use P.  
PRINT

PRINT AT is used in Level I BASIC.

PRINT@ is used in Level II BASIC.

The PRINT USING statement is used in Level II BASIC to print numbers or strings with a particular format.

### **PRINT AT** statement

The PRINT AT statement specifies the starting location of a string or numeric variable to be printed on the screen.

Example: 10 PRINT AT 10, "HELLO";A\$ prints HELLO and the string (in this case a person's name) at location 10 on the screen.

Also see: AT, @, PRINT, SPC( ), TAB

*Apple* • The PRINT AT statement is used in both Applesoft and Integer BASIC. The SPC( ) command is used in Applesoft with a PRINT statement to insert a specified number of spaces between the last character printed and the next one. The TAB function used with the PRINT statement can be used in Applesoft. The TAB statement can be used in Integer BASIC.  
SPC( )

*PET* • The PRINT statement is used in PET BASIC.  
TAB PRINT TAB(N),"character" can be used to place spaces before the desired printout. SPC can also be used on the  
SPC PET to count a number of spaces from the last cursor position before printing the next character, word or sentence.

Using the cursor controls inside quotation marks will also allow the programmer to PRINT at specific locations on the screen.

Example: 10 PRINT"home keycursor downcursor downcursor rightHERE"

This line will cause the word "HERE" to be printed two lines down and one space right from the home position, which is the top left-hand side of the monitor.

TRS-80 • The PRINT AT statement is used in Level I BASIC. A PRINT AT short form for PRINT AT is P.A.

Level II BASIC uses the PRINT@ statement.

If the user hits SHIFT and @ at the same time, @ appears as usual on the screen, but the ASCII code number is not correct, and a syntax error will occur when the program is run.

PRINT AT location 960 is at the bottom left corner of the video screen. PRINT@960, "down arrow" will cause a line feed and scroll the whole screen upward. PRINT-ing a space into location 1023 (bottom right corner) will also result in a line feed.

See the TRS-80 PRINT AT & TAB GRAPHICS WORKSHEET.

### PRINT USING statement

The PRINT USING statement allows printed output to be displayed in a specified format. The number sign (#) is used to reserve a position for each number.

Format: PRINT USING "string"; value

Example: 5 X = 262.2

```
10 PRINT USING "***#####.##";X
```

This prints 262.20. Since there are two #'s after the decimal in line 10, a zero is added after the 2 to fill the space. See the TRS-80 Level II BASIC Manual for more PRINT USING examples.

The PRINT USING operators, numbers or strings can be specified as variables.

```
10 A$ = "***##,###.##"
```

```
20 X = 1276.5
```

```
30 PRINT USING A$,X
```

This would result in \*\*1,276.50 being printed. Notice that more than one of the formats were linked together in line 10.

Also see: \$,%

TRS-80 • The PRINT USING statement is used only in Level II PRINT BASIC. USING

### PRINT# statement

The PRINT# statement is used to store data on cassette.

Example: 1000 PRINT#1,A;"",B stores the value of A and B on cassette recorder #1. This can later be read back into the computer

by using the INPUT# statement. The values in the PRINT# list cannot exceed 255 characters.

Also see: INPUT#, OPEN, CLOSE, CMD

*Apple* • The STORE statement is used only in Applesoft.  
STORE

*PET* • The PRINT# statement is used in PET BASIC.  
PRINT# Format: PRINT#1, data

Data is transferred, one character at a time, to the tape.

Example: 10 OPEN1,4,0 file#1, device 4 (printer),  
0 (start at the beginning)

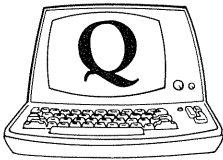
20 PRINT#1,K prints value K on printer

30 CLOSE1 closes file #1

CMD may be used in place of PRINT#.

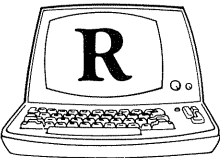
*TRS-80* • The PRINT# statement is used in Level I BASIC. Level  
PRINT# II uses PRINT#-1.

Example: PRINT#-1, A\$,B\$,C,D,E stores on cassette  
the strings A\$ and B\$, and the numeric values assigned  
to C, D, and E.



? (question mark) operator (See page 100.)

“ (quotation marks) operator (See page 101.)



**R.** command, statement, function

R. is an abbreviation for RUN, RND (Random) and RESET. (RESET is used in graphics to turn off a graphics block.)

R. is recognized as RUN when used as a command.

R. is recognized as the RND function when followed by (X) where X is a numeric variable.

R. is recognized as the RESET statement when followed by (X,Y) where X and Y are the co-ordinates of the graphics block that is to be shut off.

Also see: RUN, RND, RESET



- Apple* • The RUN command is used in both Applesoft and Integer BASIC. The RND function has no short form in Applesoft or Integer BASIC.
- RUN*
- RND*
- PET* • The RUN command is used in PET BASIC. The RND function has no short form on the PET.
- RUN*
- RND*
- TRS-80* • The R. command is used in Level I BASIC for RUN, R. RND and RESET. Level II BASIC uses RUN, RND and RESET.
- R.*

**RANDOM** statement

The RANDOM statement randomizes a set of numbers which are selected by the RND function. RANDOM insures that a new set of random numbers will be generated for the RND function each time the program is run. RANDOM must be placed before the RND functions in a program.

Also see: RND

- Apple* • The RND statement is used in both Applesoft and Integer BASIC.
- RND*
- PET* • The RND function is used in PET BASIC.
- RND*
- TRS-80* • The RANDOM statement is used only in Level II RANDOM BASIC.

**REA.** statement

REA. is an abbreviation for the READ statement, which is used to read stored information from DATA lines.

Also see: READ

- Apple* • The READ statement is used only in Applesoft.
- READ*
- PET* • The READ statement is used in PET BASIC.
- READ*
- TRS-80* • The REA. statement is used in Level I BASIC. Level II REA. BASIC uses READ.

**READ** (ANSI) statement

The READ statement reads numeric or string DATA from DATA lines, and assigns that DATA to variables. Each time the READ statement is executed, the next element in a DATA line is read. When all the DATA is read, it must be restored, or an "OUT OF DATA ERROR" occurs.

Example 1: 10 READ A,N\$,C  
 150 DATA 6, "JOHN",14 assigns 6 to A, JOHN to N\$ and 14 to C.

Example 2: Numbers can be assigned to subscript variables as in  
 90 FOR A=1 TO 6: READ N(A): NEXT A  
 100 DATA 6,20,-1,6,4,3

In example 2 N(1) equals 6, N(2) equals 20, N(3) equals -1, etc.

Also see: RESTORE, REA., DATA

*Apple* • The READ statement is used only in Applesoft. READ  
 READ X\$ assigns the next DATA character or string to X\$.

Integer BASIC can simulate READ...DATA statements by writing out all the DATA in variable statements.

Example: 90 FOR A=1 TO 6: READ N(A): NEXT A  
 100 DATA 6,20,-1,6,4,3

These lines can be changed for Integer BASIC.

100 A(1)=6:A(2)=20:A(3)=-1:A(4)=  
 6:A(5)=4:A(6)=3

*PET* • The READ statement is used in PET BASIC.  
 READ

*TRS-80* • The READ statement is used in Level II BASIC. REA. is  
 READ used in Level I BASIC.

## **RECALL** command

The RECALL command loads an array back from tape after it has been saved by the STORE command.

Example: 100 RECALL X loads array X into computer memory.

Note: The array must have been dimensioned (DIM) correctly.

Also see: DIM, STORE

*Apple* • The RECALL command is used only in Applesoft.  
 RECALL

## **REM** (ANSI) statement

The REM statement is used for program comments. They have no effect on program execution. REM is often used to identify different sections of the program, such as subroutines. If more than one line is required for the remark or comment, REM must precede each line.

Example: 10 REM THE PROGRAM STARTS HERE  
 100 REM GAME SUBROUTINE

*Apple* • The REM statement is used in both Applesoft and In-  
 REM ter BASIC.

*PET* • The REM statement is used in PET BASIC.  
 REM

*TRS-80* • The REM is used in both Level I and Level II BASIC.  
 REM The apostrophe (') can be used as a short form for REM.  
 Example: 10 ' THE PROGRAM BEGINS

**REPT** key

Pressing and holding the REPT key after pressing and holding any other key causes that key to be repeated.

Example: Pressing the REPT and X keys down at the same time results in the following:

XX

The X's continue until the keys are released.

*Apple* • The Apple II REPT key is used to enter multiple  
REPT characters.

*TRS-80* • The Model II TRS-80 has a REPT key.  
REPT

**RESET** command

The RESET command puts the computer into the monitor mode. CTRL C or G returns control to Applesoft.

*Apple* • The RESET command is used in Applesoft.  
RESET

**RESET** statement

The RESET statement is used to turn off a graphics block previously turned on by the SET command.

Example: 500 RESET(X,Y) where X and Y represent the co-ordinates of a graphics block on the screen. X is the column and Y is the row.

Also see: SET, PLOT, HPLOT, Appendix C

*Apple* • PLOT is used in both Applesoft and Integer BASIC.  
PLOT HPLOT is used only in Applesoft. PLOT X,Y is used in low resolution graphics and HPLOT X,Y in high resolution graphics to place a colored graphics block at a location specified by the X and Y.

There is a RESET command in Applesoft (see the previous reference).

*PET* • PRINT statements and the PET graphic keys are used to place graphics in a program.

*TRS-80* • The RESET statement is used in both Level I and Level  
RESET II BASIC.

**REST.** statement

The REST. statement is an abbreviation for the RESTORE statement.

Also see: RESTORE

*Apple* • The RESTORE statement is used only in Applesoft.  
RESTORE

*PET* • The RESTORE statement is used in PET BASIC.  
RESTORE

*TRS-80* • The REST. statement is used in Level I BASIC. Level II  
REST. BASIC uses RESTORE.

### **RESTORE** (ANSI) statement

The RESTORE statement restores all the DATA in the DATA statements so they can be used again when another READ statement is encountered in the program. It also starts the READING of DATA at the beginning of a program.

Also see: REST., DATA, READ

*Apple* • The RESTORE statement is used only in Applesoft.  
RESTORE

*PET* • The RESTORE statement is used in PET BASIC.  
RESTORE

*TRS-80* • The RESTORE statement is used in both Level I and  
RESTORE Level II BASIC. In Level I BASIC REST. may be used.

### **RESUME** statement

The RESUME statement is used with the ON ERROR GOTO routine. The program resumes at a line specified after the RESUME statement.

Example: 1000 RESUME 100. This causes the program to resume at line 100 without stopping and without producing an error message. If no line is specified, the program resumes at the line where the error occurred.

*Apple* • The RESUME statement is used only in Applesoft.  
RESUME

*TRS-80* • In Level II BASIC RESUME NEXT causes the program  
RESUME to resume on the line following the line where the error occurred. RESUME NEXT must be in an error handling subroutine.

### **RET.** statement

The RET. statement is an abbreviation for RETURN.

Also see: RETURN

*Apple* • The RETURN statement is used in both Applesoft and  
RETURN Integer BASIC.

*PET* • The RETURN statement is used in PET BASIC.  
RETURN

*TRS-80* • The RET. statement is used in Level I BASIC. Level II  
RET. BASIC uses the RETURN statement.

**RETURN** key

The RETURN key is used to signify the end of an input line. On all three computers CHR\$(13) can be used to generate a return during a program run.

Also see: ENTER

*Apple* • The RETURN key is used on the Apple II computer.  
RETURN

*TRS-80* • The RETURN key is used on the PET computer.  
RETURN

*TRS-80* • The ENTER key is used on the TRS-80 computer.  
ENTER

**RETURN** (ANSI) statement

The RETURN statement is used at the end of a subroutine. After a GOSUB is executed and the subroutine is completed, the RETURN statement returns execution of the program to the line following the GOSUB. If a RETURN statement without a GOSUB is encountered then execution halts and an error message is printed.

Also see: RET., GOSUB

*Apple* • The RETURN statement is used in both Applesoft and RETURN Integer BASIC.

*PET* • The RETURN statement is used in PET BASIC.  
RETURN

*TRS-80* • The RETURN statement is used in both Level I and RETURN Level II BASIC. Level I BASIC may also use RET. as a short form.

**→(right arrow)** key

The → (right arrow) is a cursor control character used to move the cursor one or more spaces to the right.

Also see:  $\overline{\text{CR}}\overline{\text{SR}}$

*Apple* • The Apple II → (right arrow) enters the character under the cursor into memory, and moves the cursor one space to the right.  
→

*PET* • Pressing the PET cursor  $\overline{\text{CR}}\overline{\text{SR}}$  key once moves the cursor one space to the right and enters the character under it into memory. (In PRINT statements the reverse character  $\overleftarrow{\text{R}}$  can be used.)  
 $\overline{\text{CR}}\overline{\text{SR}}$

Cursor right can be inserted into a program by using CHR\$(29).

*TRS-80* • The TRS-80 right arrow (→) moves the cursor to the next tab stop. The tab stops are located at positions 0, 8, → 16, 24, 32, 40, 48 and 56.

SHIFT→ converts the display from 64 characters per line to 32 characters per line.

CHR\$(9) can be used for the →(right arrow).

### **RIGHT\$** function

The RIGHT\$ function is used to select a number of string characters in a string. The format is RIGHT\$("string",number). The number is counted from the right side of the string.

Example: 10 PRINT RIGHT\$("UPSTAIRS",6)

This prints the last six characters (STAIRS). A string variable may also be used.

Example: 10 PRINT RIGHT\$(A\$,3) where A\$ has already been assigned a string value.

The number of characters can also be expressed as a variable or arithmetic expression.

Example: 10 PRINT RIGHT\$("UPSTAIRS",X + 1)

Note: A comma must separate the string and the number. If the number inside the argument is larger than the number of characters in the string, the whole string will be returned.

Also see: LEFT\$, MID\$

*Apple* • The RIGHT\$ function is used only in Applesoft. In Integer BASIC RIGHT\$ is not used. There are, however, other string handling notations. A\$(N) prints all the characters of A\$ starting with the Nth character. A\$(N,P) prints all the characters starting with the Nth character and ending with the Pth character.

To translate RIGHT\$(A\$,6) the user would type A\$(LEN(A\$) - 5,LEN(A\$))

In general RIGHT\$(A\$,N) = A\$(LEN(A\$) - N + 1, LEN(A\$)).

*PET* • The RIGHT\$ function is used in PET BASIC. If the RIGHT\$ specified number is zero, less than zero or greater than 255, an ILLEGAL QUANTITY ERROR is printed.

*TRS-80* • The RIGHT\$ function is used only in Level II BASIC. RIGHT\$

### **RND** (ANSI) function

The RND function generates random numbers. This is often used in games.

Also see: R.

*Apple* • The RND function is used in both Applesoft and Integer RND BASIC.

RND(1) returns a random real number between 0 and 0.99999999 inclusive each time it is used.

RND(0) returns the last random number again.

RND(negative number) returns a different fixed number for each negative number.

RND(positive number) returns a random number every time it is used.

In Integer BASIC decimals are not used, so RND(0) returns a random number integer between 0 and 9. To get a random number between 0 and X, Applesoft uses  $\text{INT}(\text{RND}(0) * X + 1)$ .

Integer BASIC uses  $\text{RND}(X) + 1$ .

**PET**  
**RND**

- The RND function is used in PET BASIC.

RND(0) gives the same sequence of random numbers for each RND call.

RND(negative number) returns the same random numbers each time it is called.

RND(positive number) gives a new sequence of random numbers each time RND is called. The range of numbers generated is from 0.000000000 to 0.999999999.  $\text{INT}(\text{RND}(1) * 10)$  gives a random number between 1 and 10.

RND(variable) can be used if that variable has been defined elsewhere in the program. If the variable has not been defined, it is given the value zero, and the resultant numbers are the same as for RND(0).

**TRS-80**  
**RND**

- The RND function is used in both Level I and Level II BASIC.

RND(0) returns a random number between 0 and 0.999999999999999 inclusive each time it is used.

RND(integer) returns an integer between 1 and the integer.

**ROT** statement

The ROT statement sets the rotation angle of the shape from DRAW or XDRAW.

Format: ROT=N where N is a number.

Example: 40 ROT=0 sets rotation of the shape in a vertical direction. ROT=16 is for 90 degrees clockwise. ROT=32 is for 180 degrees clockwise.

A series of ROT's could change the position of the shape quickly so the shape would appear to be spinning.

**Apple** • The ROT statement is used only in Applesoft.  
**ROT**

**RTS** command

The RTS command must be the last instruction in a machine language program that has been accessed from BASIC. RTS causes the program to return to the original BASIC program.

*PET*       • The RTS command is used in PET BASIC.  
**RTS**

**RUN** command

The RUN command is used to initiate the execution of the program in memory, starting at the lowest line number.

A specific line number can be placed after the RUN command.

Example: RUN 100. This causes the execution of the program to start at line 100.

Also see: R.

*Apple*     • The RUN command is used in both Applesoft and Integer  
**RUN**       RUN       BASIC. The RUN command clears all variables and begins execution at the specified line. If no line number is specified, execution starts at the lowest line number in the program.

*PET*       • The RUN command is used in PET BASIC. Pressing the  
**RUN**       RUN       R key, followed by the SHIFT and U keys, results in the program starting execution.

*TRS-80*   • The RUN command is used in both Level I and Level II  
**RUN**       RUN       BASIC. R. may also be used in Level I BASIC.

**RUN/STOP** key

Pressing the RUN/STOP key stops program execution and prints out the line number where it stopped. Pressing RUN/STOP in conjunction with the SHIFT key starts the loading process.

Also see: LOAD, RUN, BREAK key, STOP

*Apple*     • The RUN and STOP commands are used in Applesoft  
**RUN**       RUN       and Integer BASIC. Both commands must be entered  
**STOP**      STOP      letter by letter.

*PET*       • The PET RUN/STOP key causes the program to STOP  
**RUN/**      RUN/     and print out the line number where it stopped.  
**STOP**      STOP     Example: BREAK IN LINE 20

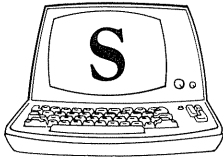
SHIFT and RUN/STOP pressed at the same time starts the LOADING process.

*TRS-80*   • The RUN and STOP commands are both used in both  
**RUN**       RUN       Level I and Level II BASIC. Both commands must be  
**STOP**      STOP      entered letter by letter.

The BREAK key also stops execution and print the line where it stopped.

Example: BREAK IN LINE 20





**S.** function, command, statement

The S. statement is an abbreviation for the STEP and graphics SET statements. S. is recognized as SET when followed by two numbers in brackets. The numbers are the co-ordinates of the graphics block, which is to be turned on.

Example: 10 S. (5,12)

S. is recognized as STEP when it is used with the FOR TO NEXT statement.

Example: 10 FOR X= 1 TO 100 S.2  
20 NEXT X

In this example the STEP is equal to 2.

Also see: STEP, PLOT, SET

- Apple*     • The S. statement is used in both Applesoft and Integer  
STEP       BASIC. The STEP statement replaces S. when S. is used  
PLOT       as a short form for STEP.  
             PLOT replaces S. when S. is used as a short form for  
             SET.
- PET*       • The S. statement is not used in PET BASIC. STEP re-  
STEP       places S. when S. means STEP, and the graphics keys  
             replace S. when S. means SET.
- TRS-80*   • The S. statement is used in Level I BASIC. Level II  
S.           BASIC uses either STEP or SET.

### **SAVE** command

The SAVE command is used to save programs by transferring them from the computer's memory to cassette tape.

SAVE"name" gives a program a name on the tape. LOAD"name" loads that program back into computer memory from cassette tape. If more than one program is recorded on the tape, LOAD"name" searches and LOADs only the program that is named.

Also see: CSAVE, LOAD

- Apple*     • The SAVE command is used in both Applesoft and In-  
SAVE       teger BASIC.
- PET*       • The SAVE command is used in PET BASIC. It saves the  
SAVE       program on cassette tape.  
             SAVE"GAME" saves and names the file on cassette  
             tape.

SAVE"GAME",2 saves and names the file on cassette recorder #2. When a number is not specified, recorder #1 is assumed.

SAVE"GAME",2,1 saves the named file on cassette recorder #2, and writes an end of tape block.

*TRS-80* • The CSAVE statement is used in both Level I and Level  
CSAVE II BASIC. (C—cassette; SAVE—save)

### **SCALE** statement

The SCALE statement sets the scale (from 0 to 255) of the shape to be drawn on the screen from the DRAW or XDRAW commands.

Also see: DRAW, XDRAW

*Apple* • The SCALE statement is used only in Applesoft.  
SCALE

### **SCRN** function

The SCRN function returns the color of the graphics block specified by two numbers in brackets and separated by a comma.

Example: 200 IF SCRN(10,10) = 15 GOTO 10

In the example, if the color of point (10,10) is white (15 = white), the program continues at line 10.

Also see: COLOR, POINT

*Apple* • The SCRN function is used for low resolution graphics  
SCRN in both Applesoft and Integer BASIC.

*TRS-80* • The POINT statement is used in both Level I and Level  
POINT II BASIC. A 1 is returned if the block is on, and a 0 is returned if it is off.

; (**semicolon**) (ANSI) operator (See page 101.)

### **SET** statement

The SET statement turns on a graphics block. The location on the screen is specified by the co-ordinates which are in brackets after SET.

Example: 10 SET (5,9) turns on the graphics block located at the intersection of the 5th column and the 9th row. The RESET statement turns off the graphics block.

Also see: PLOT, RESET

*Apple* • The PLOT statement is used in both Applesoft and In-  
PLOT teger BASIC. It is used in low resolution graphics.

HPLLOT HPLLOT is used for high resolution graphics in  
Applesoft.

*PET* • The PET graphics keys are used for program graphics.

*TRS-80* • The SET statement is used in both Level I and Level II  
SET BASIC. Level I BASIC can also use the S. form of SET.

**SGN** function

The SGN function determines the sign of a number (positive or negative). If the number is less than 0 (<0), then -1 is returned. If the number is greater than 0 (>0), then 1 is returned. If the number equals 0 (=0), then 0 is returned.

*Apple* • The SGN function is used in both Applesoft and Integer  
SGN BASIC.

*PET* • The SGN function is used in PET BASIC.  
SGN

*TRS-80* • The SGN function is used only in Level II BASIC.  
SGN

**SHIFT@** keys

SHIFT@ halts the execution or listing of a program. Pressing any key causes execution or listing to continue.

Also see: RVS

*Apple* • The Apple II CTRL C keys, when pressed at the same  
CTRL C time, stop the listing or the execution of a program.  
SHIFT@ When the computer is in edit mode, pressing SHIFT@  
at the same time clears the screen and places the cursor  
in the upper left corner of the screen.

*PET* • Pressing and holding down the PET RVS key during  
RVS listing of a program, slows the scrolling so that it is  
easier to read and study. Pressing the STOP key stops  
the listing, but when more listing is desired, the  
operator must start again from the beginning. This  
can be very inconvenient when working with a long  
program.

*TRS-80* • The TRS-80 SHIFT@ keys are used in both Level I and  
SHIFT@ Level II BASIC to stop the program listing or program  
execution.

**SHIFT INST/DEL** key

The SHIFT INST/DEL keys, pressed simultaneously, insert spaces.

*PET* • Pressing the PET SHIFT and INST/DEL keys at the  
SHIFT same time inserts spaces. SHIFT RETURN moves the  
INST/DEL cursor to the next line without entering the line into  
memory; that line will be ignored during running or  
listing.

*TRS-80* • Entering EDIT, pressing the "I" key and then the space  
EDIT I bar, inserts spaces.  
space bar

**SHIFT ←(left arrow)** key

The SHIFT ← (left arrow) deletes the entire line currently being typed.

*Apple* • The Apple II CTRL X key combination deletes the entire line currently being typed.

*TRS-80* • The TRS-80 SHIFT ← (left arrow) keys are used in Level I and Level II BASIC to delete the line currently being typed.

**SHIFT → (right arrow)** key

The SHIFT → (right arrow) converts the display to a 32 character-per-line format from the usual 64 character-per-line format.

Clearing the screen using the CLEAR key or the CLS statement returns the screen to a 64 character-per-line format.

Also see: CLEAR, CLS

*TRS-80* • The TRS-80 SHIFT → (right arrow) keys are used in Level I and Level II BASIC to convert the display to the 32 character-per-line mode.

**SHLOAD** command

The SHLOAD command loads a shape table from cassette. The shape table may have a number of shapes, each of which are given a number so they can be drawn by the DRAW command.

Also see: DRAW

*Apple* • The SHLOAD command is used only in Applesoft.  
SHLOAD

**SIN** (ANSI) function

The SIN(X) function calculates the Sine of the angle X expressed in radians.

$$\text{SINE } X = \frac{\text{length of opposite side}}{\text{length of hypotenuse}}$$

To convert radians to degrees, multiply the number of radians by 57.29578; to convert degrees to radians divide by 57.29578.

Example: to find the sine of 35°, 10 X = SIN(35/57.29578).

*Apple* • The SIN function is used only in Applesoft. Programs containing SIN cannot be translated directly into Integer BASIC.

*PET* • The SIN function is used in PET BASIC.  
SIN

*TRS-80* • The SIN function is used only in Level II BASIC.  
SIN

**SPC** function

The SPC function places a specified number of spaces, or skips characters, before a string or number to be printed.

Example: 10 PRINT SPC(5); "WHAT IS YOUR NAME"

In the example, 5 spaces or skip characters are placed before the sentence to be printed.

Also see: TAB

*Apple*  
SPC • The SPC function is used only in Applesoft. TAB may be used with the same result. SPC(N) is used only in the PRINT statement, and puts the specified number (N) of spaces between the last element and the next.

*PET*  
SPC • The SPC function in PET BASIC places the specified number of skip characters, not spaces, between the last character and the next. The range of values are from 0 to 255 inclusive.

TAB may be used with the same effect, except TAB counts from the left margin each time it counts spaces.

*TRS-80*  
TAB • The TAB function is used in both Level I and Level II BASIC.

**SPEED** command, statement

The SPEED command sets the speed of character output onto the screen. The speed is specified by the numbers between 0 and 255.

Example: SPEED=100

SPEED can be used to scroll the listing of a program more slowly so it can be studied.

*Apple*  
SPEED • The SPEED command is used only in Applesoft.

*PET*  
POKE 59458,62 • On the PET, POKE 59458,62 is a speed-up for the 2.0 ROM PETs.

*TRS-80*  
POKE 32763,n • In Level II BASIC, POKE 32763, 1 to 255 controls the program speed or listing. The higher the number poked into memory location 32763, the slower the execution of the program or the listing. To return the computer to normal speed use POKE 32763,0.

**SQR** (ANSI) function

The SQR function calculates the square root of a positive number.

Example: 10 PRINT "THE SQUARE ROOT OF ";X;" IS ";SQR(X)

*Apple*  
SQR • The SQR function is used only in Applesoft. Programs using SQR cannot be translated directly into Integer BASIC.

- PET*     • The *SQR* function is used in PET BASIC.  
*SQR*
- TRS-80* • The *SQR* function is used only in Level II BASIC.  
*SQR*

**ST.** statement

The ST. statement is an abbreviation for the STOP statement.

Also see: STOP

- Apple*     • The STOP statement is used only in Applesoft.  
 STOP
- PET*     • The PET uses the RUN/STOP key.  
*RUN/*     The PET uses ST to check the status of an I/O (In-  
*STOP*     put/Output). Some of the common values for ST are  
*ST*       listed here.  
           ST=0 no error  
           ST=4 short block  
           ST=8 long block  
           ST=16 unrecoverable READ error  
           ST=32 checksum error  
           ST=64 end of file is detected  
           ST=128 the end of the tape is detected.  
           Example of use: 100 IF ST=16 PRINT  
                           "UNRECOVERABLE READ ERROR"
- TRS-80* • The ST. statement is used in Level I BASIC. Level II  
*ST.*     BASIC uses STOP.

**STEP** (ANSI) function

The STEP function is used with the FOR-TO-NEXT statement to specify the size of the step increment. Positive or negative numbers may be used after the STEP. If no number is specified + 1 is used as the STEP value.

Example: 10 FOR X=100 TO 10 STEP -10  
 20 NEXT X

The number after the STEP can be a variable.

Example: 10 FOR X=100 TO 10 STEP A  
 20 NEXT X

Also see: S., FOR, NEXT

- Apple*     • The STEP statement is used in Applesoft and Integer  
*STEP*     BASIC.
- PET*     • The STEP statment is used in PET BASIC.  
*STEP*
- TRS-80* • The STEP statement is used in Level II BASIC. Level I  
*STEP*     BASIC may also use STEP and the short form S.

**STOP** (ANSI) statement

The STOP statement halts execution of the program, and prints out the line number where the stop occurred. The STOP statement can be placed anywhere in a program.

The purpose of the STOP statement is to stop the program from going into subroutines or other areas of the program by mistake. STOP is often placed on the line before a subroutine begins so that the only way to get to the subroutine is by a GOSUB statement.

Also see: BREAK key, GOSUB, POKE, RUN/STOP key

*Apple*  
STOP • The STOP statement is used only in Applesoft. CONT continues a program's execution after it has been STOPped.

*PET*  
RUN/  
STOP • The operator may use the RUN/STOP key to stop a program's execution. The program will respond with BREAK IN LINE N. N is the number of the line where the program was stopped.

CONT after STOP continues the execution at the point following STOP.

POKE144,49 disables the STOP key.

POKE144,46 re-enables the STOP key.

Disabling the STOP key prevents accidental BREAKing of the program.

See POKE for more details on disabling the STOP key for different ROM sets.

*TRS-80*  
STOP • The STOP statement is used in Level I and Level II BASIC. In Level II BASIC CONT after the stop continues the execution.

The BREAK key can also stop a program, but it cannot be used within a program.

**STORE** command

The STORE command saves a numeric array on a tape.

Example: STORE X stores or saves on tape the numeric array X.

Note: STORE cannot be used to store or save string arrays.

Also see: PRINT#, CMD

*Apple*  
STORE • The STORE command is used only in Applesoft.

*PET*  
PRINT# • The PRINT# statement is used in PET BASIC. The format is PRINT#1, data. The data is transferred to the tape a single character at a time.

Example: 10 OPEN 1,4,0 FILE#1, device 4 (printer)  
start at beginning

20 PRINT#1,K prints K on printer

30 CLOSE 1 close file #1

The CMD command may also be used in place of the PRINT# statement.

*TRS-80* • The PRINT# statement is used in Level I BASIC. Level  
PRINT# II BASIC uses PRINT # - 1.

Example: PRINT# - 1, A\$, B\$, C, D, E stores on cassette tape the strings A\$ and B\$ and the numeric values of C, D, and E.

### **STR\$** function

The STR\$ function converts a numeric value into a string so that the RIGHT\$, LEFT\$, MID\$ and other string functions can operate on it.

Example: 10 STR\$(20). The number 20 can now be handled as a string.

Also see: RIGHT\$, LEFT\$, MID\$

*Apple* • The STR\$ function is used only in Applesoft.  
STR\$

*PET* • The STR\$ function is used in PET BASIC.  
STR\$

*TRS-80* • The STR\$ function is used in Level II BASIC.  
STR\$

### **STRING\$** function

The STRING\$(number, ASCII code) function is used with the PRINT statement to print the ASCII character the number of times specified.

Example: 10 PRINT STRING\$(5,42) prints the ASCII character "\*" five times.

The STRING\$ function is useful in graphics. The string that is created can have up to 255 repeated characters. Any characters on the keyboard may be used, except the quotation mark, comma and colon, which must be printed by using their ASCII code numbers.

Also see: STR\$

*Apple* • The PRINT statement is used in Applesoft and Integer  
PRINT BASIC.

*PET* • The PRINT statement is used in PET BASIC.  
PRINT

*TRS-80* • The STRING\$ function is used in Level II BASIC. Level  
STRING\$ II BASIC allows the string characters to be enclosed in quotation marks, or even to be string variables.

Example 1: 10 PRINT STRING\$(5, "Z")

Example 2: 10 A\$ = "Z"

20 PRINT STRING\$(5, A\$)

In both examples Z is printed five times.



- (**subtraction sign**) (ANSI) operator (See page 102.)

**SYS** command

The SYS command transfers control of the program to a machine language program.

Format: SYS(starting address)

To return from the machine language program, the last instruction in the machine language must be an RTS command.

Also see: SYSTEM, PEEK, POKE, RTS, USR, CALL

*Apple* • The CALL function is used in both Applesoft and Integer CALL BASIC. Applesoft also uses the USR command.

The CALL N function transfers the program to a machine-language subroutine starting at memory location N.

*PET* • The SYS(N) command is used in PET BASIC. It SYS transfers control to the machine language program starting at N. Starting address must not be greater than 65535.

DATA transfer to and from the machine language program must be done by PEEKing and POKEing.

The SYS command can simulate an escape key back to a specific line, by placing the following line after every INPUT line.

```
IF A$="@ THEN POKE167,1: SYS50583: GOTO 1000
```

where A\$ changes to correspond to the same variable as used in the INPUT statement, and 1000 is the line to which the user wishes to escape.

Typing SYS4 gets the user into the monitor program and makes it possible to study the internal codes used by the PET.

*TRS-80* • The USR function is used in Level II BASIC. It calls a USR machine language subroutine.

**SYSTEM** command

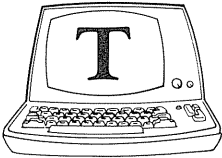
The SYSTEM command allows machine language programs to be loaded from cassette. Enter SYSTEM and the \*? appears. Enter the file name and the program starts loading. When loading is complete, another \*? appears. Enter a slash / and starting address. If no address is entered, the program starts at the address specified in the program.

Also see: SYS, CALL, PEEK, POKE, USR, RTS

*Apple* • The CALL function is used in both Applesoft and Integer CALL BASIC. The CALL N function transfers the program to a machine language subroutine starting at memory location N.

*PET*     • The SYS command is used in PET BASIC. The SYS(N)  
 SYSTEM   command transfers the program to a machine language  
 SYS       program starting at memory location N.

*TRS-80* • The SYSTEM command is used only in Level II BASIC.  
 SYSTEM



**T.** function, statement

T. is an abbreviation for the TAB function and the THEN statement. T. replaces TAB when it is followed by a number enclosed in parentheses. T. replaces THEN when used with an IF statement, and when it is not followed by a number in parentheses.

Also see: TAB, THEN, IF-THEN

*Apple*     • TAB and THEN are both used in both Applesoft and In-  
 TAB        teger BASIC.  
 THEN

*PET*       • TAB and THEN are both used in PET BASIC.  
 TAB  
 THEN

*TRS-80*   • The T. statement abbreviation is used only in Level I  
 T.         BASIC. Level II BASIC must use TAB and THEN.

**TAB** (ANSI) function

The TAB function is used with the PRINT statement to insert a specified number of spaces before the characters to be printed.

Format: 10 PRINT TAB(N)"string";TAB(N)"string"

Example: 10 PRINT TAB(10)"HELLO" prints 10 spaces before the HELLO is printed. As seen in the format, more than one TAB can be placed in a statement.

A numeric variable or expression may be used inside the parentheses.

If the print position is greater than the TAB value, the TAB is ignored.

Example: 10 PRINT"0123456789";TAB(5). The TAB has no effect.

Also see: T., SPC

*Apple*     • The TAB function is used in both Applesoft and Integer  
 TAB        BASIC. In Applesoft HTAB can also be used; the PRINT  
            statement is not necessary.

There is no TAB function in Integer BASIC; the TAB statement is used instead.

Applesoft: 10 PRINT TAB(6);"\*\*"  
 Integer BASIC 10 TAB 6: PRINT"\*\*"

- PET*  
 TAB
- The TAB function is used in PET BASIC. TAB places the cursor at the column specified in the argument. The range of values can be between 0 and 255 inclusive.  
 Format: TAB(X)  
 If X>255 TAB is ignored.  
 TAB uses skip characters, not spaces, in placing the cursor.  
 Example: 10 PRINT TAB(9);"NAME"  
 SPC may be used in a similar way, except SPC counts spaces from the last cursor position and not from the margin, as with the TAB function.
- TRS-80*  
 TAB
- The TAB function is used in both Level I and Level II BASIC. Level I BASIC may also use the T. function.  
 CHR\$(9) can be used to TAB during a program.

### **TAN** (ANSI) function

The TAN(X) function calculates the tangent of the specified angle in radians.

$$\text{TAN}(X) = \frac{\text{length of opposite side}}{\text{length of adjacent side}}$$

To convert from radians to degrees, multiply by 57.29578.

To convert from degrees to radians, multiply by 0.0174533. These are useful because you must input the angle expressed in radians into the formula, and then convert back when the computer gives the result.

- Apple*  
 TAN
- The TAN function is used only in Applesoft. Programs using TAN cannot be translated directly into Integer BASIC.
- PET*  
 TAN
- The TAN function is used in PET BASIC.
- TRS-80*  
 TAN
- The TAN function is used only in Level II BASIC.

### **TEXT** command, statement

The TEXT command or statement is used to return from the graphics mode to normal text mode. The GR command is used to enter the graphics mode.

Also see: GR, SHIFT→, POKE, CHR\$

- Apple*  
 TEXT
- The TEXT command is used in both Applesoft and Integer BASIC. The TEXT command is used to place the

computer in a non-graphics text mode. This allows 40 characters per line and 24 lines.

POKE 33,33 changes 40 characters per line to 33 characters per line. To reset this to the normal 40 characters enter POKE 33,40, or type TEXT once again.

*PET* • PET changes modes in only one instance. The regular graphics mode can be changed to the upper and lower case mode by typing POKE 59468,14. POKE 59468,12 changes the computer back to the normal graphics character mode.

*TRS-80* • The use of the SHIFT→ (right arrow) converts the video display to 32 characters per line from the usual 64 characters per line.

You can also use CHR\$(23) to convert to 32 characters per line. Clearing the screen with CLS then returns the display to 64 characters per line. Try this:

```
10 PRINT CHR$(23);"TITLES"
20 FOR A=1TO500:NEXT A
30 CLS
40 PRINT"I AM NOW BACK TO NORMAL"
50 FOR A=1TO500:NEXTA
```

### **THEN** (ANSI) statement

The THEN statement is part of the IF-THEN statement. If the condition in the IF part is met, the statement(s) following the THEN are executed.

Example 1: 10 IF X=1 THEN 100 transfers execution to line 100 only if X=1. If X≠1, then the program continues at the next numbered line.

Example 2: 10 IF X=1 THEN A=5

If X=1 then A is assigned the value.

Also see: IF, IF-THEN

*Apple* • The THEN statement is used in both Applesoft and Integer BASIC.

*PET* • The THEN statement is used in PET BASIC.

*TRS-80* • The THEN statement is used in both Level I and Level II BASIC. T. can be used in Level I BASIC.

### **TI** function

The TI function is an abbreviation for the TIME function. The TI function indicates elapsed time.

Also see: TIME

*PET* • The TI function is used in PET BASIC.

TI

**TI\$** function

The TI\$ function is an abbreviation for the TIME\$ function. It keeps track of time in hours, minutes and seconds.

Also see: TIME\$

*PET* • The TI\$ function is used in PET BASIC.

**TI\$** If you want to display the time during your program, place this line near the beginning of the program: TI\$="000000". At the part of the program where you want the time displayed, insert this line: PRINT "␣"; TI\$. The "␣" homes the cursor, and the TI\$ prints the time, since the time was set to zero.

**TIME** function

The TIME function tells how much time has elapsed since the computer was turned on.

Example: PRINT TIME will give the length of time since the computer was turned on.

*PET* • The TIME command is used in PET BASIC. The TIME  
**TIME** is incremented 60 times per second, and is expressed as a six-digit number. It cannot be reset to zero without turning the machine off.

*TRS-80* • There is a clock feature when the expansion interface is used.

**TIME\$** function

The TIME\$ function is used to indicate the time of day. The time is expressed in 6 digits, and is in a 24-hour-clock format.

Example: TIME\$="163921" (hhmmss)

This means that it is (16-12) = 4 o'clock, 39 minutes, and 21 seconds, or 4:39:21 p.m.

*PET* • The TIME\$ function is used in PET BASIC.

**TIME\$**

**TRACE** command, statement

The TRACE command prints each program line as it is executed. It is often used in debugging programs. NOTRACE turns off the TRACE. TRACE is not turned off by RUN, CLEAR, NEW, DEL or RESET. TRACE can be used as a statement to turn the trace on at specific locations in the program.

Also see: TRON, NOTRACE, TROFF

*Apple* • The TRACE command is used in both Applesoft and In-  
**TRACE** teger BASIC.

*TRS-80* • The TRON command is used only in Level II BASIC.  
**TRON**

**TROFF** command, statement

The TROFF command turns off the trace of program line numbers as they are executed. As with the TRON command, TROFF can be used as a statement to turn the trace on and off in specific parts of the program.

Also see: TRACE, NOTRACE, TRON

*Apple* • The NOTRACE command is used by both Applesoft NO- and Integer BASIC.  
TRACE

*TRS-80* • The TROFF command is used in Level II BASIC to turn TROFF off the trace (TRON).

**TRON** command, statement

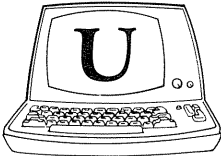
The TRON command is used to display each program line as it is executed. It is used in debugging.

TRON can be used as a statement to display the line numbers of only certain parts of a program. To turn off the trace, use the TROFF command or statement.

Also see: TROFF, TRACE, NOTRACE

*Apple* • The TRACE command is used in both Applesoft and In-TRACE teger BASIC.

*TRS-80* • The TRON command is used only in Level II BASIC.  
TRON



↑ (**up arrow**) (ANSI) operator (See page 102.)

**USR** function, command

The USR function calls a machine language subroutine from the computer's memory and executes that program.

A machine language program may be loaded into memory by using the POKE statement, or by using the SYSTEM command to load it from tape.

The USR function can also execute a machine language program that transfers the specified value to a machine language subroutine. If the argument is not needed, a dummy argument can be used.

Also see: SYS, SYSTEM, POKE, PEEK, CALL

*Apple* • The CALL command is used in both Applesoft and In- CALL N teger BASIC; USR is used only in Applesoft.

USR

The CALL N function transfers the program to a machine language subroutine, which starts at memory location N.

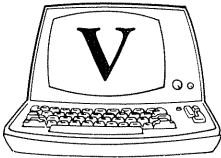
The USR(X) function passes a specified value (X) to a machine language subroutine.

*PET*     • The USR function is used in PET BASIC. It passes a  
*SYS*     value into a machine language program.  
*USR*

The SYS function runs a machine language program starting at a specified address.

Example: 10 SYS(634)

*TRS-80* • The USR function is used only in Level II BASIC.  
*USR(N)*    *USR(0)* to *USR(9)* can be used.



### **VAL** function

The VAL function converts numbers (which are considered strings) back into numeric notation. VAL is therefore the complement of STR\$. Try this:

10 A\$="XYZ":B\$="XYZ123":C\$="123XYZ"	The results are:
20 PRINTVAL(A\$)	0
30 PRINTVAL(B\$)	0
40 PRINTVAL(C\$)	123

The zeros result because VAL ignores the rest of the instruction once a letter is encountered. The third one is a number because the number part of the string comes first.

*Apple*     • The VAL function is used only in Applesoft.  
*VAL*

*PET*     • The VAL function is used in PET BASIC. When using  
*VAL*     *VAL*, if the string is not numeric, a zero (0) is returned.

*VAL* may be used in a numeric expression.

*TRS-80* • The VAL function is used only in Level II BASIC.  
*VAL*

### **VARPTR** function

The VARPTR function returns the address where a specified variable and its value are stored in memory. If the variable has not been given a value, an error message is printed.

Format: VARPTR(variable)

*TRS-80* • The VARPTR function is used only in Level II BASIC.  
*VARPTR*

**VERIFY** command

VERIFY checks to see if the program SAVED on cassette is the same as the one in memory.

VERIFY "GAME" compares the file "GAME" with the program in memory, and reports on the success or failure of the attempt to SAVE that file.

Also see: CLOAD?, SAVE, CSAVE, CLOAD

*PET* • The VERIFY command is used in PET BASIC. It compares the program in memory to the program on tape. If there is a mistake a ?VERIFY ERROR is printed, and it would be wise to re-SAVE the program and VERIFY it again.

Pressing the V key and then the SHIFT E will start the verifying process.

*TRS-80* • The CLOAD? command is used only in Level II BASIC. CLOAD? If the CSAVE was not successful, a BAD message appears. CSAVE the program again.

**VLIN AT** statement

The VLIN AT statement displays a vertical line at a specific location. The line length can be from 1 to 39 units long.

Example: 10 VLIN 10,20 AT 30 draws a line from row 10 to row 20 at column 30.

The GR statement must be executed before the VLIN statement. The COLOR statement determines the color of the line.

Also see: GR

*Apple* • The VLIN AT statement is used in both Applesoft and VLIN AT Integer BASIC. It is used in low resolution graphics.

**VTAB** statement

The VTAB statement tells the computer the line where the operator wants a PRINT statement to start.

Example: 10 VTAB 6  
20 PRINT"HELLO"

The HELLO is printed on the 6th line.

VTAB moves the cursor up or down, but not left or right.

Also see: TAB, PRINT AT, PRINT@, @, PRINT

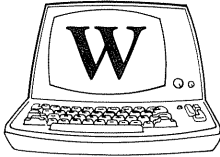
*Apple* • The VTAB statement is used in both Applesoft and VTAB Integer BASIC. VTAB values can be from 1 to 24, representing the screen's 24 lines.

*PET* • The TAB statement is used in PET BASIC. It places a specified number of skip characters between the last character printed and the next. Using PRINT  
TAB



statements with nothing after them moves the cursor down.

- TRS-80* • The TAB statement is used in both Level I and Level II BASIC. TAB can be used to PRINT statements on different lines on the screen. PRINT statements and PRINT AT or PRINT@ will also PRINT at desired locations.



**WAIT** command, statement

The WAIT statement causes the program to wait until the contents of a specified location is equal to a particular number.

Also see: FOR-TO-NEXT

- Apple* • The WAIT command or statement is used only in Applesoft.  
**WAIT** Example: WAIT A,B,C

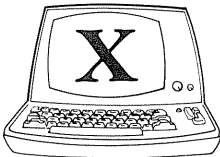
In this example the computer waits until the contents of location A is A ORed with C and ANDed with B and gives a non-zero result.

- PET* • The WAIT statement is used in PET BASIC.  
**WAIT** Example: WAIT A,B,C

The status of memory location A is read and ORed with B and then ANDed with C until the result is a non-zero number. The program then continues.

- TRS-80* • The best way to WAIT in Level I or Level II BASIC is to use a FOR-TO-NEXT statement.  
 Example: 50 FOR X = 1 TO 100: NEXT X

The program waits until the computer counts from 1 to 100. The length of time waited can be increased by raising the number after TO.



**XDRAW AT** statement

The XDRAW AT statement draws a defined shape loaded from tape by the SHLOAD command. The drawing starts at a specified location. The difference between XDRAW and DRAW is that the

XDRAW colors complement the colors that are already on the screen at that point.

Example: XDRAW 1 AT 30,40

This draws a shape, given the label number 1, starting at the point 30,40.

Also see: HCOLOR, SHLOAD, DRAW AT

*Apple* • The XDRAW AT statement is used only in Applesoft.  
 XDRAW  
 AT

### **XPLOT** statement

XPLOT is a graphics statement that is a reserved word in Applesoft. It does not correspond to current command; it is a word without function or meaning at the present time. Presumably this statement will have some meaning in future versions of Applesoft.

Also see: SET, RESET, PLOT

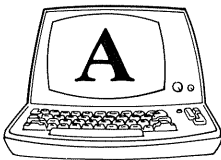
*Apple* • XPLOT is a reserved word in Applesoft.  
 XPLOT

*PET* • The PET uses its special graphics keys and characters for plotting.

*TRS-80* • Level I and Level II BASIC use the SET statement to turn on graphics blocks.  
 SET



# Alphabetical Listing of Operators



## **And** operator

AND is a mathematical operator often used in IF-THEN statements. It is usually referred to as a logical AND.

Example 1: IF X = 5 AND Y = 3 THEN 100

X must equal 5 *and* Y must equal 3 before the program branches to line 100; otherwise the next line is executed.

Example 2: IF A > 3 AND B > 5 THEN 100

In Example 2, if A is greater than 3 *and* B is greater than 5, then, and only then, program execution jumps to line 100. AND, along with OR and NOT, are Boolean Operators. See Appendix E for more information on Boolean Operators.

Also see: \* (asterisk), Appendix E, NOT, OR

*Apple*     • The AND operator is used in both Applesoft and Integer  
AND         BASIC.

*PET*         • The AND operator is used in PET BASIC.  
AND

*TRS-80*    • AND is used in Level II BASIC. The \* (asterisk) is used  
AND         in Level I BASIC to represent AND.

## **\* (asterisk)** (ANSI) operator

The \* (asterisk) is used by all three computers as a multiplication sign.

On all three computers PRINT CHR\$(42) can be used to print an asterisk (\*) on the screen.

Also see: AND

- Apple* • The \* (asterisk) operator is used in both Applesoft and Integer BASIC. An \* (asterisk) appears when a CALL-151 is executed. This indicates that the computer is in the mode to accept machine language instructions.
- PET* • The \* (asterisk) operator is used in PET BASIC.
- TRS-80* • The \* (asterisk) operator is used in Level I BASIC to represent the logical AND. In Level II BASIC AND must be used.

### @ (AT) operator

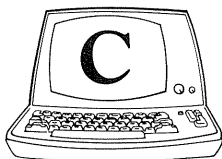
The @ operator specifies the starting location on the screen for a PRINT statement. The @ argument must be followed by a comma.

Example: 10 PRINT @ 30,"WHAT IS YOUR NAME"

On all three computers PRINT CHR\$(64) can be used to print an @ sign on the screen.

Also see: AT, PRINT, PRINT AT

- Apple* • The AT function is used in both Applesoft and Integer BASIC.
- TRS-80* • AT or A. must be used in Level I BASIC. The @ operator is used in Level II BASIC. The argument following the @ can be numbered from 0 to 1023.



### ^ (circumflex) operator

The ^ (circumflex) operator is a symbol for exponentiation.

Example: 3^2 is the same as 3<sup>2</sup> or 9.

Also see: ↑ (up arrow)

- Apple* • The ^ (circumflex) operator is used in Applesoft and Integer BASIC. CHR\$(94) or CHR\$(222) may be used in a program to represent a circumflex.
- PET* • The ↑ (up arrow) operator is used in PET BASIC. CHR\$(94) can be used in a program to print out an ↑ (up arrow).
- TRS-80* • The ↑ (up arrow) operator is used in both Level I and Level II BASIC. CHR\$(91) can be used in a program to print out an ↑ (up arrow).

**:** (Colon) operator

The : (colon) allows the placing of more than one statement on one program line. This saves memory space.

Example: 10 CLS: PRINT "RULES": GOSUB 1000

On all three computers PRINT CHR\$(58) can be used to print a : (colon) on the screen.

*Apple* • The : (colon) operator is used in Applesoft and Integer BASIC.

*PET* • The : (colon) is used in PET BASIC.

:

*TRS-80* • The : (colon) operator is used in both Level I and II BASIC.

**,** (comma) (ANSI) operator

The , (comma) operator is used in PRINT statements to separate the elements so that they will be printed in separate zones. The , (comma) is also used in DATA, DIM, INPUT, ON-GOTO, ON-GOSUB and READ statements to separate the different elements.

Also see: PRINT, PRINT@, PRINT AT, DATA, DIM, INPUT, ON-GOTO, ON-GOSUB, READ

*Apple* • The , (comma) operator is used in both Applesoft and Integer BASIC. The , (comma) may be printed by using PRINT CHR\$(44) as one of the program lines. In Integer BASIC a comma is used after the INPUT statement.

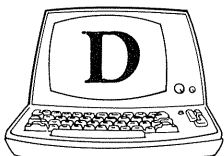
Example: 50 INPUT "ANSWER",N

*PET* • The , (comma) operator is used in PET BASIC. The , (comma) may be printed by using the statement PRINT CHR\$(44) as one of the program lines.

As with the other computers, the "," is a print delimiter.

*TRS-80* • In Level II BASIC the , (comma) is used after the PRINT@ statement.

PRINT CHR\$(44) can be used to print a comma on the screen.

**/** (division sign) (ANSI) operator

The / (division sign) operator is used in the arithmetic operation of division.

Example: 10 A = B/2 reads as A equals B divided by two.

On all three computers PRINT CHR\$(47) may be used to print a / sign on the screen.

Also see: Appendix I

*Apple* • The / (division sign) operator is used in Applesoft and in / Integer BASIC.

*PET* • The / (division sign) operator is used in PET BASIC. /

*TRS-80* • The / (division sign) operator is used in both Level I and / Level II BASIC.

**\$ (dollar sign)** (ANSI) operator

The \$ (dollar sign) operator designates a variable as a string variable.

Example: 10 INPUT"WHAT IS YOUR NAME";A\$. The program expects a string input for the person's name.

On all three computers PRINT CHR\$(36) may be used to print a \$ sign on the screen.

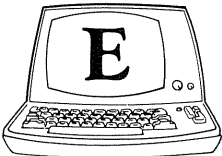
Also see: Appendix I, PRINT USING

*Apple* • The \$ (dollar sign) operator is used in both Applesoft and \$ Integer BASIC.

*PET* • The \$ (dollar sign) operator is used in PET BASIC. \$

*TRS-80* • Level I BASIC has only two string variables: A\$ and B\$. \$ Level II BASIC has many more string variables.

The PRINT USING statement uses the \$ sign to format lines that contain dollars.



**= (equal sign)** (ANSI) operator

The = (equal sign) is used in arithmetic operations as an equal sign. It is also used to give variables a designated value.

Example: 10 A\$ = "Yes"  
20 B = 15

The = (equal sign) can also be used in conditional IF-THEN, IF-GOTO or IF-GOSUB statements to test for equal expressions.

Example: IF A = 3 THEN 100

Also see: IF-THEN, IF-GOTO, IF-GOSUB

*Apple* • The = (equal sign) operator is used in both Applesoft and = Integer BASIC.

*PET*     • The = (equal sign) operator is used in PET BASIC.

=

*TRS-80* • The = (equal sign) operator is used in both Level I and  
=           Level II BASIC.

**! (exclamation mark) operator**

The ! (exclamation mark) is used to change double-precision variables back to single-precision.

On all three computers CHR\$(33) can be used to print an ! (exclamation mark) on the screen.

Also see: PRINT USING, CSNG, CDBL

*TRS-80* • The ! (exclamation mark) is used in Level II BASIC. The  
!           PRINT USING statement prints only the left-most  
            character in a string.



**> (greater than) (ANSI) operator**

The > (greater than) operator is used in conditional operations to indicate that one expression is greater than another. It is used in IF-THEN, IF-GOTO, and IF-GOSUB statements.

Example: IF X > Y THEN 1000

*Apple*     • The > (greater than) operator is used in Applesoft and  
>           Integer BASIC.

*PET*       • The > (greater than) operator is used in PET BASIC.

>

*TRS-80* • The > (greater than) operator is used in both Level I  
>           and Level II BASIC.

**> = (greater than or equal) operator**

The > = (greater than or equal) operator is used in conditional operations to indicate that one expression is greater than or equal to another. It is used in conditional IF-THEN, IF-GOTO, and IF-GOSUB statements.

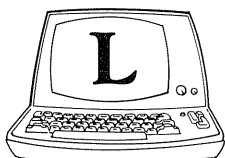
Example: 10 IF X > = Y THEN 1000

*Apple*     • The > = (greater than or equal) operator is used in both  
> =         Applesoft and Integer BASIC.

*PET*       • The > = (greater than or equal) operator is used in PET  
> =         BASIC.

*TRS-80* • The > = (greater than or equal) operator is used in both  
> =         Level I and Level II BASIC.





### < (less than) (ANSI) operator

The < (less than) operator is used in conditional operations to indicate one expression is less than another.

Example:  $A < B$

The < (less than) operator is used in IF-THEN, IF-GOTO, IF-GOSUB statements.

*Apple*     • The < (less than) operator is used in both Applesoft and  
<             Integer BASIC.

*PET*        • The < (less than) operator is used in PET BASIC.  
<

*TRS-80*    • The < (less than) operator is used in both Level I and  
<             Level II BASIC.

### <= (less than or equal) operator

The <= (less than or equal) operator is used in conditional operations to show that one expression is less than or equal to another.

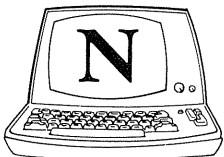
Example: IF  $X <= Y$  THEN 1000

In the above example if X is less than or equal to Y, then execution jumps to line 1000. The <= operator is used in the IF-THEN, IF-GOTO and IF-GOSUB statements.

*Apple*     • The <= (less than or equal) operator is used in both  
<=           Applesoft and Integer BASIC.

*PET*        • The <= (less than or equal) operator is used in PET  
<=           BASIC.

*TRS-80*    • The <= (less than or equal) operator is used in Level I  
<=           and Level II BASIC.



### NOT operator

NOT is used as a logical NOT in relational expressions between two numbers or variables.

Example: 10 IF NOT  $A = 3$  THEN 100

This means that if A is NOT equal to 3, THEN program execution jumps to line 100. If the logical NOT is false, then program execution continues at the next numbered line. In our example, if A is equal to 3, then the program continues at the next numbered line.

NOT, along with AND and OR, are examples of Boolean Operators. See Appendix E for more information on Boolean Operators.

Also see: AND, OR, Appendix E

*Apple*     • NOT is used in both Applesoft and Integer BASIC.  
NOT

*PET*       • The NOT operator is used in PET BASIC.  
NOT

*TRS-80*   • The NOT operator is used in both Level I and Level II  
NOT        BASIC.

#### < > (not equal) (ANSI) operator

The < > (not equal) operator is used to indicate inequality.

Example: IF X < > Y THEN 1000

The < > operator is used in conditional IF-THEN, IF-GOTO and IF-GOSUB statements.

*Apple*     • The < > (not equal) operator is used only in Applesoft.  
< >        Integer BASIC uses the # (number sign) for not equal.

*PET*       • The < > (not equal) operator is used in PET BASIC.  
< >

*TRS-80*   • The < > (not equal) operator is used in Level I and  
< >        Level II BASIC.

#### # (number sign) operator

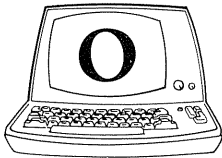
The # (number sign) has different meanings for all three machines.

*Apple*     • In Integer BASIC the # (number sign) operator is used  
#           for the not equal operator. In Applesoft the < > (not  
            equal) sign is used.

*PET*       • The # (number sign) operator is used for DATA input.  
#           Format: INPUT#1, variable

*TRS-80*   • In Level I and Level II BASIC the # (number sign) is  
#           used in PRINT#-1 and INPUT#-1 statements.  
            Format: INPUT#-1, variables.

It also defines a variable as double-precision in Level II BASIC.



### **OR** operator

The OR operator is the logical OR.

Example: 10 IF A=1 OR A=2 THEN 1000

In the example, the A can be equal to 1 *or* it can be equal to 2, in order for the program to branch to line 1000. If A is equal to anything else, execution of the program continues at the next numbered line.

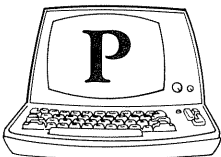
OR, along with AND and NOT, are examples of Boolean Operators. See Appendix E for more information on Boolean Operators.

Also see: AND, NOT Appendix E

*Apple*     • The OR operator is used in both Applesoft and Integer  
OR            BASIC.

*PET*         • The OR operator is used in PET BASIC.  
OR

*TRS-80*    • The OR operator is used in Level II BASIC. In Level I  
OR            BASIC the + (plus sign) is used for the logical OR.



### **( ) (parentheses)** (ANSI) operator

The ( ) (parentheses) operator is used to surround the arguments in function calls. They are used in PEEK ( ), TAB( ), FRE( ) LEN( ), LEFT\$( ), RIGHT\$( ), MID\$( ), CHR\$( ), ASC( ), DIM( ), SIN( ), COS( ), ABS( ), RND( ), SGN( ), SQR( ), EXP( ), LOG( ), TAN( ), ATN( ), INT( ), PDL( ), USR( ), etc.

The ( ) (parentheses) operator is also used to determine the order of operations in a mathematical equation. Expressions inside parentheses are evaluated first.

Example 10 A=(3+2)\*5. The part inside parentheses is performed first. The result is 25. If the parentheses were not included, the multiplication would be performed first, and the result would be 13.

*Apple*     • The ( ) (parentheses) operator is used in both Applesoft  
( )            and Integer BASIC.

- PET*     • The ( ) (parentheses) operator is used in PET BASIC.  
( )
- TRS-80* • The ( ) (parentheses) operator is used in both Level I  
( )         and Level II BASIC.

**% (percent) operator**

The % (percent) operator is used to define a variable as integer. On all three computers CHR\$(37) can be used to print a % sign on the screen.

Also see: INT, PRINT USING

- Apple*     • The % (percent) operator is used in Applesoft to in-  
%         dicate an integer variable.  
           Example: A%

In Integer BASIC A is sufficient to indicate an integer because all letters are integer variables.

- PET*         • The % (percent) operator is used in PET BASIC to  
%         define a variable as integer.
- TRS-80*    • In Level II BASIC the % (percent) operator is used in the  
%         PRINT USING statement. It allows the printing of a  
           specified number of left-most letters or characters in a  
           string. (See PRINT USING)  
           % also defines a variable as an integer in Level II  
           BASIC.

**. (period) operator**

The . (period) operator is used in abbreviations, and it is used as a decimal in mathematical operations.

On all three computers CHR\$(46) can be used to print a period on the screen during a program run.

- Apple*     • The . (period) is used in both Applesoft and Integer  
           BASIC. The . (decimal) is used in Applesoft.

Integer BASIC has no floating point. Try 1 divided by 3; the answer is 0. Entering 2.6 results in a SYNTAX ERR. Because of this, Integer BASIC has no SIN( ), COS( ), TAN( ), ATN( ), SQR( ), LOG( ) and EXP( ) functions. Programs with these functions cannot be translated directly into Integer BASIC.

- PET*         • The . (period) is used in PET BASIC.  
           .
- TRS-80*    • The . (period) operator is used in both Level I and Level  
           II BASIC. In Level II BASIC. LIST. and EDIT. cause the  
           LISTing and EDITing of the last program line entered.

It also LISTS the line number that caused an error message to be printed.

Example: LIST. ENTER  
or EDIT. ENTER  
or. ENTER

In the third example, entering a period after an error message appears causes that line to list.

**+ (plus sign)** (ANSI) operator

The + (plus sign) operator is used for the arithmetic addition operation. The + (plus sign) is also used for concatenation of strings (the linking together of strings).

Example: 10 PRINT "UP"+"STAIRS"

In the example, the two words are linked or concatenated to form one word, which is UPSTAIRS.

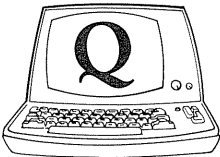
On all three computers CHR\$(43) can be used to print a + sign on the screen.

Also see: OR

*Apple* • The + (plus sign) operator is used in both Applesoft and Integer BASIC.

*PET* • The + (plus sign) operator is used in PET BASIC.

*TRS-80* • The + (plus sign) operator is used in both Level I and Level II BASIC. In Level I, the + (plus sign) is used as a short form for the logical FOR. In Level II BASIC, the FOR must be used.



**? (question mark)** operator

The ? (question mark) is used by all three computers as a short form for the PRINT statement.

On all three computers CHR\$(63) can be used to print a ? (question mark) on the screen.

Also see: PRINT

*Apple* • The ? (question mark) operator is used in both Applesoft and Integer BASIC.

*PET* • The ? (question mark) operator is used in PET BASIC.

? When the ? is used as a short form for the PRINT statement, the long form (the word PRINT) is automatically placed in the program.

**TRS-80** • The ? (question mark) operator is used in both Level I and Level II BASIC. As is the case with the PET, the long form, PRINT, appears in the program when ? is used as a short form of PRINT.

**“ (quotation marks) operator**

“ (quotation marks) are used in PRINT statements to indicate what is to be printed. If the quotation marks are not included, all letters are treated as variables, and their assigned values are printed.

Example: 10 A=5

20 PRINT“A” prints the letter A

30 PRINT A prints the value 5 since there are no quotation marks around the A, and in line 10 A is given the value 5.

On all three computers, CHR\$(34) can be used to print the quotation marks on the screen.

Example: 10 PRINT CHR\$(34);“HERE”;CHR\$(34) prints “HERE” on the video monitor.

Also see: CLOAD, CSAME, SAVE, LOAD, PRINT, CHR\$( ), \$ dollar sign, Appendix A

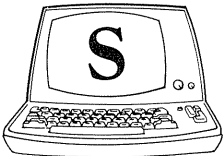
**Apple** • The “ (quotation marks) operator is used in both Applesoft and Integer BASIC.

**PET** • The “ (quotation marks) operator is used in PET BASIC. LOAD “GAME” searches for the file named “GAME”, and then loads it.

Quotation marks are used in LOADING as seen above, but they are also used in SAVEing a program. SAVE “GAME” would SAVE the program “GAME” on tape.

**TRS-80** • Level I BASIC uses “ (quotation marks) in the PRINT# statement. Level II BASIC uses “ (quotation marks) in the CSAVE and CLOAD commands to give the program a specific name. Unlike the PET computer, which allows a word or even group of words for the title, the TRS-80 allows only single letters for titles.

Example: SAVE“A” saves the program named “A” on tape.



**;(semicolon) (ANSI) operator**

The ; (semicolon) operator is used in PRINT statements to print values without intervening spaces.

Example: 10 PRINT"INSTRUCTIONS FOR THE GAME";  
20 PRINT"ARE PRINTED BELOW"

Lines 10 and 20 result in the whole sentence being printed on one line: INSTRUCTIONS FOR THE GAME ARE PRINTED BELOW.

On all three computers CHR\$(59) can be used to print a ;.

*Apple* • The ; (semicolon) operator is used in both Applesoft and Integer BASIC.

*PET* • The ; (semicolon) operator is used in PET BASIC.

*TRS-80* • The ; (semicolon) operator is used in both Level I and Level II BASIC.

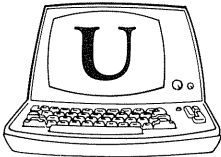
### - (subtraction sign) (ANSI) operator

The - (subtraction sign) operator is used for the arithmetic operation of subtraction. The - (minus sign) is also used as a sign of negation.

*Apple* • The - (subtraction sign) operator is used in both Applesoft and Integer BASIC.

*PET* • The - (subtraction sign) operator is used in PET BASIC.

*TRS-80* • The - (subtraction sign) operator is used in both Level I and Level II BASIC.



### ↑ (up arrow) (ANSI) operator

The ↑ (up arrow) is used to indicate exponentiation.

Example: 3↑2 is the same as 3<sup>2</sup> or 9.

Also see: ^ (circumflex)

*Apple* • The ^ (circumflex) is used by Applesoft and Integer BASIC.

*PET* • The ↑ (up arrow) operator is used in PET BASIC.

*TRS-80* • The ↑ (up arrow) is used in Level I and Level II BASIC.

# Part II

## Summary of Terms

Terms	Applesoft BASIC	Integer BASIC	PET BASIC	TRS-80 BASIC (Level I)	TRS-80 BASIC (Level II)	Meaning
A.				X		absolute value
ABS	X	X		X		absolute value
' (apostrophe)				X	X	short form for REM
ASC	X		X		X	return ASCII code
AT	X	X		X		start PRINT location
ATN	X		X		X	arctangent
AUTO		X			X	automatic line numbering
BREAK key				X		stops AUTO, Program RUN or LISTing
C.				X		continue program execution
CALL	X	X				machine language
CALL-936		X				cursor to top left
CDBL				X		single to double precision
CHR\$	X		X		X	return ASCII character
CINT				X		convert to integer
CLEAR	X				X	reset variables to zero
CLEAR key				X	X	cursor to top left and clear screen



Terms	A	I	P	T-I	T-II	Meaning
CLEAR key CLS and reverse CLEAR key			X			clear video screen
CLOAD				X	X	load program
CLOSE			X			close file
CLR		X	X			reset variables to zero
CLR/HOME key			X			cursor to top left and/or clear screen
CLS				X	X	clear video screen
CMD LIST			X			output to line printer
COLOR	X	X				specify color
CON		X				continue program execution
CONT	X		X	X		continue program execution
COS	X		X	X		cosine
CSAVE				X	X	save program
CSNG					X	double to single precision
CTRL C key	X					halt program or program listing
CTRL G key	X					beep sound
CTRL J key	X					line feed
CTRL X	X					delete line
cursor keys	X	X	X	X	X	move cursor
D.				X		list data
DATA	X		X	X		list data
DEF	X		X			define functions
DEFDBL				X		define double precision variable
DEFINT				X		define integer variable
DEFSNG				X		redefine variable as single precision
DEFSTR				X		define string variable
DEL	X	X				delete

Terms	A	I	P	T-I	T-II	Meaning
DEL key				X		delete
DELETE					X	delete
DIM	X	X	X		X	dimension arrays
DRAW AT	X					position for drawing
DSP			X			variable value displayed
E.					X	end a program
EDIT					X	edit
ELSE					X	use after IF-THEN
END	X	X	X		X	end a program
ENTER key				X	X	end of input line
ERASE					X	cancel reserved space
ERL					X	identify error line
ERR					X	identify error code
ERROR					X	simulate error
ESC	X	X				put into edit mode
ESC A	X	X				move cursor right
ESC B	X	X				move cursor left
ESC C	X	X				move cursor down
ESC D	X	X				move cursor up
ESC E	X	X				delete from cursor to end of line
ESC F	X	X				delete from beginning of line to cursor
EXP	X		X		X	compute natural log of e
F.					X	for
FIX					X	remove numbers after decimal
FLASH	X					flash display
FN	X		X			specify defined variable of the operator
FOR	X	X	X		X	for
FRE	X		X		X	indicate unused string space

Terms	A	I	P	T-I	T-II	Meaning
G.				X		goto
GET	X		X			scan keyboard for input
GET#			X			input from a peripheral
GOS.				X		gosub
GOSUB	X	X	X	X		gosub
GOTO	X	X	X	X		goto
GR	X					graphics
HCOLOR	X					set high resolution color
HGR	X					high resolution graphics mode
HIMEM	X					set highest memory address available
HLIN-AT	X	X				draw horizontal line
HOME	X					clear screen and cursor to top left
HLOT	X					graphics
HLOT TO	X					graphics
HTAB	X					start print at specific location
I.				X		find integer value
IF	X	X	X	X	X	if
IF-G.				X		if goto
IF-GOS.				X		if gosub
IF-GOSUB	X	X		X		if gosub
IF-GOTO	X	X	X	X		if goto
IF-T.				X		if then
IF-THEN	X	X	X	X		if then
IN.				X		input
IN#	X	X				input from a peripheral
INKEY\$				X		scan keyboard for input
INP				X		input a decimal value from a port
INPUT	X	X	X	X		input

Terms	A	I	P	T-I-T-II	Meaning
INPUT#			X	X X	input from a peripheral
INST/DEL key			X		insert or delete
INT	X		X	X	find integer value
INVERSE	X				black on white display
L.				X	list program
LEFT\$	X		X	X	examine left of string
LEN	X		X	X	length of string
LET	X	X	X	X X	let
LIST	X	X	X	X	list program
LOAD	X	X	X		load program
LOG	X		X	X	natural log
LOMEM:	X				set lowest memory location
M.				X	unused memory
MAN		X			with CTRL X, stops AUTO
MEM				X	unused memory
MID\$	X		X	X	examine middle of string
MOD		X			remainder in division
N.				X	new or next
NEW	X	X	X	X	delete entire program
NEXT	X	X	X	X	next
NORMAL	X				white on black display
NOTRACE	X				turn trace off
OFF/RVS key					black on white display
ON ERROR GOTO				X	on error goto
ON G.				X	on goto
ON GOSUB	X		X	X	on gosub
ON GOTO	X		X	X	on goto
ONERR GOTO	X				on error goto
OPEN			X		open file

Terms	A	I	P	T-I-T-II	Meaning
OUT				X	send number to output port
P.				X	print or point
P.A.				X	print at
PDL	X	X			game paddles
PEEK	X	X	X	X	peeking into memory
Pi $\pi$ key					Pi, 3.14159265
PLOT	X	X			graphics
POINT				X	see if graphics block is on
POKE	X	X	X	X	poke into memory
POP	X	X			removes one address from return address stack
POS	X		X	X	current position of cursor
PR	X	X			send output to a peripheral
PRINT	X	X	X	X	print
PRINT AT				X	print at specified location
PRINT USING				X	print output in specified format
PRINT#			X	X	store data on cassette
R.				X	run, rnd or reset
RANDOM				X	initial randomize
REA.				X	read data lines
READ	X		X	X	read data lines
RECALL	X				load an array from tape
REM	X	X	X	X	remark
REPT key	X	X		X	enter multiple characters
RESET	X				monitor mode
RESET				X	X turns off graphics block
REST.				X	restore data
RESTORE	X		X	X	restore data
RESUME	X			X	resume after error
RET.				X	return from subroutine

Terms	A	I	P	T-I	T-II	Meaning
RETURN	X	X	X		X	return from subroutine
RETURN key	X	X	X			signify end of input line
RIGHT\$	X		X		X	examine right of string
RND	X		X	X	X	random function
ROT	X					rotation angle
RTS			X			signals end of machine language subroutine
RUN	X	X	X	X	X	run program
RUN/STOP key			X			stop program execution
S.				X		step or set
SAVE	X	X	X			save program
SCALE	X					set scale of drawing
SCRN	X	X				returns color of graphics block
SET				X	X	turns on a graphics block
SGN	X		X		X	find sign of number
SHIFT@ key	X	X				clear screen
SHIFT@ key				X	X	halt listing or execution of a program
SHIFT HOME key			X			clear screen and cursor to top left
SHIFT INST/DEL key			X			insert spaces
SHIFT← key				X	X	deletes entire line
SHIFT→ key				X	X	convert to 32 character-per-line format
SHLOAD	X					load shape table
SIN	X		X		X	sine
SPC	X		X			spaces before PRINT
SPEED	X					control output on screen
SQR	X		X		X	square root
ST.				X		stop
STEP	X	X	X		X	step

Terms	A	I	P	T-I	T-II	Meaning
STOP	X			X		stop
STORE	X					store data on tape
STR\$	X		X	X		numeric to string
STRING\$				X		prints ASCII character n times
SYS			X			control to machine language program
SYSTEM				X		load machine language program
T.				X		tab
TAB		X		X		tab
TAN	X		X	X		tangent
TEXT	X	X				return to monitor mode
THEN	X	X	X	X		then
TI			X			TIME (elapsed)
TI\$			X			TIME\$
TIME			X			time function (elapsed)
TIME\$			X			time of day
TRACE	X	X				display program lines as they are executed
TROFF				X		turn off TRACE
TRON				X		display program lines as they are executed
USR	X		X	X		transfer value to machine language
VAL	X		X	X		string to numeric
VARPTR				X		address of variable
VERIFY			X			check save
VLIN AT	X	X				draw vertical line
VTAB	X	X				line to start PRINT
WAIT	X		X			cause program to wait
XDRAW AT	X					draw shape
XPLOT	X					reserved word, no meaning

Operators	Applesoft BASIC	Integer BASIC	PET BASIC	TRS-80 BASIC (Level I)	TRS-80 BASIC (Level II)	Meaning
AND	X		X		X	and
*				X		and
*	X	X	X	X	X	multiplication
@					X	start PRINT location
AT	X			X		start PRINT location
$\lambda$	X	X				exponentiation
!			X	X	X	exponentiation
:	X	X	X	X	X	separate statements in program
,	X	X	X	X	X	separate elements in PRINT statements
/	X	X	X	X	X	division
\$	X		X	X	X	dollar and string sign
=	X	X	X	X	X	equal sign
!					X	double precision to single precision
>	X	X	X	X	X	greater than
>=	X	X	X	X	X	greater than or equal to
<	X	X	X	X	X	less than
<=	X	X	X	X	X	less than or equal to
<>	X		X	X	X	not equal
#		X				not equal
NOT	X	X	X		X	logical not
OR	X	X	X		X	logical or
+				X		logical or



<b>Operators</b>	<b>A</b>	<b>I</b>	<b>P</b>	<b>T-I</b>	<b>T-II</b>	<b>Meaning</b>
( )	X	X	X	X	X	surround arguments
%	X		X		X	define as integer
.	X	X	X	X	X	decimal
+	X	X	X	X	X	addition
?	X	X	X	X	X	abbreviation for PRINT
"	X	X	X	X	X	indicate what is to be printed in PRINT statements
:	X	X	X	X	X	print values without intervening spaces
-	X	X	X	X	X	subtraction

# Appendix A

## ASCII Codes

### What Are ASCII Codes?

Since computers do not understand English, a code was invented to represent the letters, numbers, graphic characters, special functions and special characters that are commonly used by programmers to make programs for the computer. This code is called ASCII and it stands for the American Standard Code for Information Interchange.

At one time the ASCII codes were uniform. Now each computer has a different set of ASCII codes, which sometimes creates difficulty in translating programs. The following chart shows some of the differences.

Code	Apple II	PET	TRS-80	Function
8	CTRL H	-	-	backspace and delete one character
9	CTRL I	-	->	place cursor in the next tab position
13	CTRL M RETURN	RETURN	ENTER	enter that current line into memory
18	CTRL R	RVS key	-	place video into reverse field mode
19	CTRL S	CLR key	CLEAR key	clear screen and/or home the cursor
20	CTRL T	DEL key	-	delete a character during the run of the program

Code	Apple II	PET	TRS-80	Function
27	ESC	SHIFT	SET command	ESC on the Apple places the computer in graphics mode
131	CTRL C	SHIFT RUN to load and run a program (2.0 ROM's)	graphic	function varies
145	CTRL Q	cursor up	graphic	function varies
146	CTRL R	reverse field off	graphic	function varies
147	CTRL S	clear screen	graphic	function varies
148	CTRL T	insert a space	graphic	function varies
222	^	pi $\pi$	TAB 30 spaces	function varies

The ASCII codes 32 to 95 represent characters that are very similar on all three computers.

#### ASCII Character Codes 32-95

Code	Apple II	PET	TRS-80
32	space	space	space
33	!	!	!
34	"	"	"
35	#	#	#
36	\$	\$	\$
37	%	%	%
38	&	&	&
39	'	'	'
40	(	(	(
41	)	)	)
42	*	*	*
43	+	+	+
44	,	,	,
45	-	-	-
46	.	.	.
47	/	/	/
48	0	0	0

Code	Apple II	PET	TRS-80
49	1	1	1
50	2	2	2
51	3	3	3
52	4	4	4
53	5	5	5
54	6	6	6
55	7	7	7
56	8	8	8
57	9	9	9
58	:	:	:
59	;	;	;
60	<	<	<
61	=	=	=
62	ò	ò	ò
63	?	?	?
64	@	@	@
65	A	A	A
66	B	B	B
67	C	C	C
68	D	D	D
69	E	E	E
70	F	F	F
71	G	G	G
72	H	H	H
73	I	I	I
74	J	J	J
75	K	K	K
76	L	L	L
77	M	M	M
78	N	N	N
79	O	O	O
80	P	P	P
81	Q	Q	Q
82	R	R	R
83	S	S	S
84	T	T	T
85	U	U	U
86	V	V	V
87	W	W	W
88	X	X	X
89	Y	Y	Y
90	Z	Z	Z
91	[	[	↑
92	\	\	↓
93	]	]	←
94	^	^	→
95	_	_	-

On the Apple and PET, codes 96 to 127 are duplicates of the characters represented by codes 32 to 63. These codes give lower case letters on the TRS-80 when the keys are SHIFTEd. Most of the video monitors supplied with the TRS-80 computer can display only upper case letters, which can lead to problems in debugging a program.

Example: If one letter in a reserved word like PRINT is SHIFTEd by accident, the computer interprets the word as PRiNT or PrINT, etc. This causes a syntax error when the program is run. If the monitor prints only upper case letters, then the word looks perfectly normal in a listing. Finding the syntax error is therefore very difficult. The solution is to re-type carefully the whole line that is indicated as having the syntax error.

### ASCII Character Codes 96-127

Code	Apple II	PET	TRS-80
96	space	space	shift@
97	!	!	a
98	"	"	b
99	#	#	c
100	\$	\$	d
101	%	%	e
102	&	&	f
103	'	'	g
104	(	(	h
105	)	)	i
106	*	*	j
107	+	+	k
108	,	,	l
109	-	-	m
110	.	.	n
111	/	/	o
112	0	0	p
113	1	1	q
114	2	2	r
115	3	3	s
116	4	4	t
117	5	5	u
118	6	6	v
119	7	7	w
120	8	8	x
121	9	9	y
122	:	:	z
123	;	;	shift ↑
124	<	<	shift ↓
125	=	=	shift ←
126	>	>	shift →
127	?	?	-

POKE 59468,14 puts the PET in lower case mode. SHIFTed characters are then upper case letters. POKE 59468,12 returns the computer to normal.

### PET Graphics and Lower Case Chart

ASCII Code	POKE 59468,12 Graphics	POKE 59468,14 Lower Case
193	⊕	a
194	!	b
195	-	c
196	-	d
197	-	e
198	-	f
199	∩	g
200	∪	h
201	∩	i
202	∪	j
203	∩	k
204	∪	l
205	∩	m
206	∪	n
207	∩	o
208	∪	p
209	⊕	q
210	-	r
211	⊕	s
212	!	t
213	∩	u
214	⊗	v
215	⊙	w
216	⊕	x
217	!	y
218	⊕	z

The codes 128 to 255 are special functions, have no purpose, or are repeats of other codes.

Codes 129-191 are graphics characters on the TRS-80.

Codes 192-255 are TABs for codes 0-63 on the TRS-80.

Codes 161-255 are graphics characters on the PET.

### Repeated ASCII Character Codes on the PET 161-255

Code	Repeated Code	Character
161	255	⊕
162	226	⊕

Code	Repeated Code	Character
163	227	—
164	228	—
165	229	—
166	230	⊠
167	231	⊡
168	232	⊢
169	233	⊣
170	234	⊤
171	235	⊥
172	236	⊦
173	237	⊧
174	238	⊨
175	239	⊩
176	240	⊪
177	241	⊫
178	242	⊬
179	243	⊭
180	244	⊮
181	245	⊯
182	246	⊰
183	247	⊱
184	248	⊲
185	249	⊳
186	250	⊴
187	251	⊵
188	252	⊶
189	253	⊷
190	254	⊸
222	255	⊹

To see all the characters, run the following simple program.

```

10 X = 32
20 PRINTX;:PRINTCHR$(X)
30 FOR A = 1 TO 1000: NEXT A
40 X = X + 1
50 GOTO 20

```

Line 10 initializes X at 32, which is the first non-function code.

Line 20 prints the code and the character associated with it.

Line 30 is a timer.

Line 40 increments X.

Line 50 returns the execution of the program to Line 20.

# Appendix B

## Abbreviations of BASIC Words for the PET and TRS-80

### Abbreviations on the PET

For any BASIC word, the user types the first letter of the word (e.g., L for LIST), and while holding down the SHIFT key, types the second letter. If the computer is in the graphics mode, the second letter appears as a graphic character (e.g., \ for I). Some users like to go into the lower case mode so that the letters are read easily (See POKE for instructions).

In some cases this two-letter method is not adequate because more than one word starts with the same two letters. In these cases, the first two letters are typed, and then the SHIFTed third letter is entered.

All abbreviations are converted to complete words upon the LIST command.

The following is a list of the PET abbreviations.

BASIC	Abbreviation	BASIC	Abbreviation
ABS	Ab	EXP	Ex
AND	An	FOR	Fo
ASC	As	FRE	Fr
ATN	At	GET	Ge
CHR\$	Ch	GOSUB	GOs
CLOSE	CLo	GOTO	Go
CLR	Cl	INPUT#	In
CMD	Cm	LEFT\$	LEf
CONT	Co	LET	Le
DATA	Da	LIST	Li
DEF	De	LOAD	Lo
DIM	Di	MID\$	Mi
END	En	NEXT	Ne



BASIC	Abbreviation	BASIC	Abbreviation
NOT	No	SGN	Sg
OPEN	Op	SPC{	Sp
PEEK	Pe	SQR	Sq
POKE	Po	STEP	STe
PRINT	?	STOP	St
PRINT#	Pr	STR\$	STr
READ	Re	SYS	Sy
RESTORE	REs	TAB{	Ta
RETURN	REt	THEN	Th
RIGHT\$	Ri	USR	Us
RND	Rn	VAL	Va
RUN	Ru	VERIFY	Ve
SAVE	Sa	WAIT	Wa
SIN	Si		

### Abbreviations on the TRS-80

#### Level I

BASIC	Abbreviation	BASIC	Abbreviation
ABS	A.	NEXT	N.
CLOAD	CL.	POINT	P.
CONT	C.	PRINT	P.
CSAVE	CS.	PRINT AT	P.A.
DATA	D.	READ	REA.
END	E.	RESET	R.
FOR	F.	RESTORE	REST.
GOSUB	GOS.	RETURN	RET.
GOTO	G.	RND	R.
INPUT	IN.	RUN	R.
INT	I.	SET	S.
LIST	L.	STEP(after FOR)	S.
MEM	M.	STOP	ST.
NEW	N.	TAB(after PRINT)	T.

#### Level II

BASIC	Abbreviation
PRINT	? (question mark)
REM	' (apostrophe)

# Appendix C

## Apple, PET, and TRS-80 Graphics

### Graphics

	Apple	PET	TRS-80
Screen size	40×40	40×25	40×25
Color	Yes	No	No
Screen Memory	POKE	POKE	POKE

PET has 94 separate characters which can be displayed in normal or reverse. This creates a total of 188 different characters. The commands PRINT and PRINT CHR\$ can be used for graphics. Cursor movements can also be programmed.

The TRS-80 can light up graphic blocks by using the commands POKE, PRINT@, STRING\$, PRINT, PRINT@, PRINT@CHR\$ + CHR\$, PRINT CHR\$, and SET/RESET.

### Graphics on the Apple

The Apple's low resolution graphics screen is made up of a 40 by 40 array, numbered from 0 to 39. See the *Apple Low Resolution Graphics Worksheet*.

Low resolution graphics commands are:

GR sets the computer to the graphics mode.

COLOR sets the color (0 to 15).

PLOT places a colored dot on the screen.

HLIN draws a horizontal line.

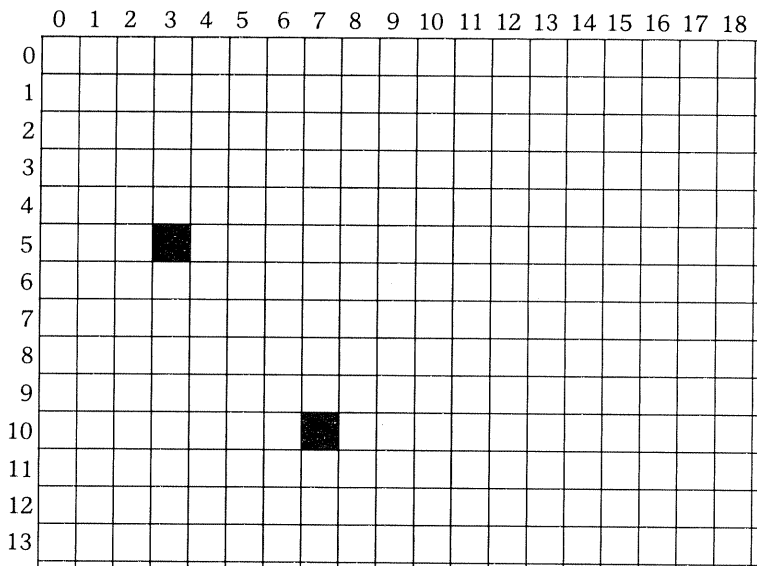
VLIN draws a vertical line.

SCRN returns color on the screen at a particular point.

Each of these commands is explained in more detail in Part I of this book.

If you want to color the square at position (5,3) then use the command PLOT 5,3.

## Column



To color positions (5,3) and (10,7) green use the following program.

```

10 GR
20 COLOR=12
30 PLOT 5,3
40 PLOT 10,7
50 END

```

Line 10 sets mode to low resolution graphics. Line 20 sets color: 12 is the number for green. In lines 30 and 40, the column number comes first, followed by the row number.






### Graphics on the PET

The PET graphic character set consists of 64 graphic symbols, which can be displayed on the screen by using the PRINT statement.

To print a rectangle on the screen, type the lines below. (Keys inside quotation marks are shifted.)

100 PRINT "  " - clear screen

200 PRINT "   five times  " - print top of rectangle

300 PRINT "   " - print side of rectangle

400 PRINT "   "

500 PRINT "   five times  " - print bottom of rectangle

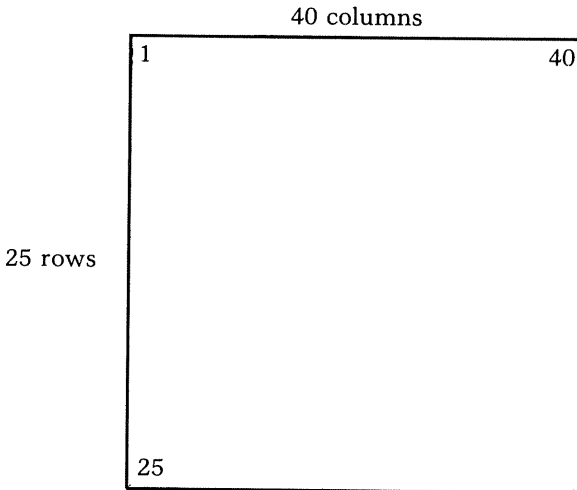
The program appears on the screen as shown below.

```

10 PRINT"[]"
20 PRINT"[]"
30 PRINT"|"
40 PRINT"|"
50 PRINT"[]"
60 GOTO 60
READY.

```

The PET screen display has 25 lines or rows with 40 characters per line.





This program displays all the graphic characters and their numeric value one at a time.

```
10 PRINT " CLR SCREEN HOME " (This key is shifted.)
20 FOR X=161 TO 255
30 PRINT CHR$( X), X
40 GETA$:IFA$=" " THEN 40
50 NEXT X
```

If you want to see all the PET characters, change line 20 to

```
20 FOR X=0 TO 255
```

The TAB and SPC functions can be used with the PRINT statement.

1. Graphics characters can be placed in different locations on the screen.
2. The TAB ( ) function moves the cursor right to a specified column number. The numbers in the brackets must be between 0 and 255 inclusive.

Try this example: 10 PRINT TAB (9), " "

The SPC ( ) function moves the cursor.

3. Write a specified number of spaces or skips. The number in the brackets can be from 0 to 255 inclusive.

Try this example:

```
10 PRINT SPC (50), " - - * - - "
20 PRINT " < < < < < "; SPC (50), " - - * - - "
```

PRINTing, which we have already looked at, is one way to write on the screen of a PET. Another way is by POKING into screen memory, which is much easier than it sounds.

The PET screen has 40 columns and 25 rows for a total of 1,000 blocks or squares that can be used for graphics. Unfortunately, these blocks are not numbered from 1 to 1,000 because the computer uses the memory addresses of the blocks. (See the *PET POKe Screen Memory Graphics Worksheet*.)

The first screen memory location is 32768, and the last screen memory location is 33767. By using the worksheet you can find the memory address you need.

Example: The fifth square in the first row would be 32768 (first address) + (5 - 1) (one less than the number of blocks from the left) = 32772 (memory address of the desired location).

In general, add to the first address in the row a number which is one less than the number of blocks over from the left side.

**Further Examples:**

32768								starting address	→ 32848
32808			7					block number	→ 7
32848			■					one less	→ 6
32888								calculation	→ 32848 + 6 = 32854
32928									
32968									
33008									
33048									

POKE32854,90 prints a diamond in that location.

33128								starting address	→ 33368
33168								block number	→ 14
33208								one less	→ 13
33248								calculation	→ 33368 + 13 = 33381
33288									
33328						14			
33368						■			
33408									
33448									

33488									
33528									
33568									
33608									
33648								21	
33688								■	
33728									

starting address	→ 33688
block number	→ 21
one less	→ 20
calculation	→ 33688 + 20 = 33708

POKE places a character anywhere on the screen by POKEing the numeric value of that character into the memory location. Use the following format: POKE address, value.

Numeric values for the PET graphic characters and the reverse field of the graphic characters are listed on the next three pages.

Use the following program to experiment with these values. The graphic character is POKEd near the middle of the screen.



```

10 PRINT"▯"
20 INPUT"WHAT IS THE NUMERIC VALUE OF THE
CHARACTER";X
30 POKE33268,X
40 GOTO 20

```

A horizontal line can be displayed on the screen by using the following program. The line may be drawn at different positions by changing the value of R (R is the first memory address in a row). The graphic symbol making up the line can be changed by giving different values to the variable X in line 20 (X is the numeric value of the graphic character).

```

10 PRINT"▯" - clears the screen
20 R=33208:X=211 -sets values
30 FOR A=0 TO 39 -loop used to add 0 to 39 to R in line 40
40 POKE R+A, X -POKE into screen memory
50 NEXT A -completes the loop started in line 30
60 GOTO 60 -endless loop to prevent the READY prompt
from destroying the display

```

The STOP key must be used to stop the display in this program; this prints a line of hearts across the middle of the screen. The following program prints a vertical line.

```

10 PRINT"▯"
20 C=32775: X=218 -C is the first column address; X is the
value for a diamond
30 FOR A=0 TO 1000 STEP 40-STEP means that the com-
puter skips to the 40th space past the cursor. This turns on
every 40th graphic block
40 POKE C+A, X-POKEs the screen memory
50 NEXT A-completes the loop started in line 30
60 GOTO 60-sets up an endless loop

```

Varying the STEP value in line 30 prints different lines and scatterings of diamonds.
















































The BASIC function ASC converts a character to a number. This function can be used with the POKE command as follows:

```

10 PRINT"▯"
20 POKE 32928,ASC(" *")

```

This displays the character \* in the fifth row, first column. (See the *PET Graphics Worksheets*.) By changing the character in the quotation marks, other letters, numbers or graphic characters can be poked into screen memory.

Graphic Character	ASCII Code	POKE Number	Reverse POKE Number
	161	97	225
	162	98	226
	163	99	227
	164	100	228
	165	101	229
	166	102	230
	167	103	231
	168	104	232
	169	105	233
	170	106	234
	171	107	235
	172	108	236
	173	109	237
	174	110	238
	175	111	239
	176	112	240
	177	113	241
	178	114	242
	179	115	243
	180	116	244
	181	117	245
	182	118	246
	183	119	247
	184	120	248
	185	121	249
	186	122	250
	187	123	251
	188	124	252
	189	125	253
	190	126	254
	191	127	255
	192	64	192
	193	65	193
	194	66	194
	195	67	195
	196	68	196
	197	69	197
	198	70	198
	199	71	199
	200	72	200
	201	73	201
	202	74	202
	203	75	203
	204	76	204
	205	77	205
	206	78	206
	207	79	207

Graphic Character	ASCII Code	POKE Number	Reverse POKE Number
┌	208	80	208
▣	209	81	209
├	210	82	210
▤	211	83	211
┤	212	84	212
└	213	85	213
⊠	214	86	214
⊡	215	87	215
⊢	216	88	216
⊣	217	89	217
⊤	218	90	218
⊥	219	91	219
⊦	220	92	220
⊧	221	93	221
⊨	222	94	222
⊩	223	95	223
⊪	224	96	224
⊫	225	97	225
⊬	226	98	226
⊭	227	99	227
┌	228	100	228
└	229	101	229
├	230	102	230
┤	231	103	231
└	232	104	232
├	233	105	233
└	234	106	234
├	235	107	235
└	236	108	236
├	237	109	237
└	238	110	238
├	239	111	239
└	240	112	240
├	241	113	241
└	242	114	242
├	243	115	243
└	244	116	244
├	245	117	245
└	246	118	246
├	247	119	247
└	248	120	248
├	249	121	249
└	250	122	250
├	251	123	251
└	252	124	252
├	253	125	253
└	254	126	254
├	255	94	222



### Graphics on the TRS-80

Graphics can be printed on the TRS-80 by using the SET and RESET statements. SET turns a graphics block on; RESET turns a graphics block off.

There are 128 horizontal graphic block addresses numbered from 0 to 127, and 48 vertical graphic block addresses numbered from 0 to 47. (See the *TRS-80 SET and RESET Graphics Worksheet*.) By placing the X and Y co-ordinates inside the brackets after SET or RESET, a total of 6,144 graphic block locations can be turned on or off.

Example: SET(0,0) lights up the block in the top left corner.

RESET(0,0) turns off that block.

SET(62,24) turns on the block that is 62 spaces in from the left (X co-ordinate) and 24 lines down (Y co-ordinate). This is near the center of the screen.

RESET(62,24) turns off that graphic block.

The whole screen can be turned white by using the following program.

```
10 CLS
20 FOR Y=0 TO 47
30 FOR X=0 TO 127
40 SET (X,Y)
50 NEXT X
60 NEXT Y
70 FOR A=1 TO 10000: NEXT A
```

To draw a horizontal line near the middle of the screen, use this program.

```
10 CLS
20 FOR X=0 TO 127
30 Y=24
40 SET(X,Y)
50 NEXT X
60 FOR A=1 TO 10000: NEXT A
```

To draw a vertical line near the middle of the screen, use this program.

```
10 CLS
20 X=64
30 FOR Y=0 TO 47
40 SET(X,Y)
50 NEXT Y
60 FOR A=1 TO 10000: NEXT A
```

The PRINT@ (PRINT AT in Level I) can be used in graphics. (See the *PRINT AT and TAB Graphics Worksheet*.)


















































```
10 FOR X=1 TO64
20 POKE 15872+X,136
30 NEXT X
This speeds a dot across the video screen.
```

See the *TRS-80 POKE Graphics Worksheet* for the exact locations of all the POKE addresses.

## TRS-80 Graphics

Codes can be used with the CHR\$ function or with POKE statements.

 128	 143	 158
 129	 144	 159
 130	 145	 160
 131	 146	 161
 132	 147	 162
 133	 148	 163
 134	 149	 164
 135	 150	 165
 136	 151	 166
 137	 152	 167
 138	 153	 168
 139	 154	 169
 140	 155	 170
 141	 156	 171
 142	 157	 172



 173

 174

 175

 176

 177

 178

 179

 180

 181

 182

 183

 184

 185

 186

 187

 188

 189

 190

 191





# Appendix D

## Reserved Words

Words which have special meaning to the computer are called reserved words. Most computers make it possible for reserved words to take up only one byte of program memory. All other characters, in words or by themselves, use up one byte each of program memory.

Reserved words may not be used as variable names because each one is reserved by the computer for a special purpose. If a reserved word is used as a variable, or part of a variable, the computer recognizes it as the function, statement or command that it represents.

Example: If you use `LISTS` as a variable name for some lists of articles, the computer normally responds with a syntax error message because `LISTS` contains the reserved word `LIST`.

Reserved words for the three computers are listed on the next three pages. To avoid syntax errors and to avoid crashing your program, make sure none of the reserved words are used as variables or as parts of variables.

### APPLESOFT RESERVED WORDS

<code>&amp;</code>	<code>CONT</code>	<code>FN</code>
<code>ABS</code>	<code>COS</code>	<code>FOR</code>
<code>AND</code>	<code>DATA</code>	<code>FRE</code>
<code>ASC</code>	<code>DEF</code>	<code>GET</code>
<code>AT</code>	<code>DEL</code>	<code>GOSUB</code>
<code>ATN</code>	<code>DIM</code>	<code>GOTO</code>
<code>CALL</code>	<code>DRAW</code>	<code>GR</code>
<code>CHR\$</code>	<code>END</code>	<code>HCOLOR=</code>
<code>CLEAR</code>	<code>EXP</code>	<code>HGR</code>
<code>COLOR=</code>	<code>FLASH</code>	<code>HGR2</code>

**APPLESOFT RESERVED WORDS (cont.)**

HIMEM:	ONERR	SGN
HLIN	OR	SHLOAD
HOME	PDL	SIN
HPLOT	PEEK	SPC{
HTAB	PLOT	SPEED =
IF	POKE	SQR
IN#	POP	STEP
INPUT	POS	STOP
INT	PRINT	STORE
INVERSE	PR#	STR\$
LEFT\$	READ	TAB{
LEN	RECALL	TAN
LET	REM	TEXT
LIST	RESTORE	THEN
LOAD	RESUME	TO
LOG	RETURN	TRACE
LOMEM:	RIGHT\$	USR
MID\$	RND	VAL
NEW	ROT =	VLIN
NEXT	RUN	VTAB
NORMAL	SAVE	WAIT
NOT	SCALE =	XPLOT
NOTRACE	SCRN{	XDRAW
ON		

**INTEGER BASIC RESERVED WORDS**

&	IF	POP
ABS	IN#	PRINT
AND	INPUT	PR#
ASC	LEN	REM
AT	LET	RETURN
AUTO	LIST	RND
CALL	LOAD	RUN
CLR	MAN	SAVE
COLOR	MOD	SCRN
CON	NEW	STEP
DEL	NEXT	TAB
DIM	NOT	TEXT
DSP	NOTRACE	THEN
END	OR	TO
FOR	PDL	TRACE
GOSUB	PEEK	VLIN
GOTO	PLOT	VTAB
GR	POKE	XPLOT
HLIN		

**PET RESERVED WORDS**

ABS	INT	RIGHT\$
AND	LEFT\$	RND
ASC	LEN	RUN
ATN	LET	SAVE
CHR\$	LIST	SGN
CLOSE	LOAD	SIN
CLR	LOG	SPC
CMD	MID\$	SQR
CONT	NEW	ST
COS	NEXT	STEP
DATA	NOT	STOP
DEF	ON	STR\$
DIM	OPEN	SYS
END	OR	TAB
EXP	PEEK	TAN
FN	POKE	THEN
FOR	POS	TI
FRE	PRINT	TI\$
GET	PRINT#	TO
GET#	READ	USR
GOSUB	READ#	VAL
GOTO	REM	VERIFY
IF	RESTORE	WAIT
INPUT	RETURN	

**TRS-80 LEVEL I RESERVED WORDS**

ABS	INPUT	READ
AND	INT	RESET
AT	LET	RESTORE
CLS	LIST	RETURN
CONT	MEM	RND
DATA	NEW	SET
END	NEXT	STEP
FOR	ON	STOP
GOSUB	POINT	TAB
GOTO	PRINT	THEN
IF		

**TRS-80 LEVEL II RESERVED WORDS**

@	CHR\$	CONT
ABS	CINT	COS
AND	CLEAR	CSNG
ASC	CLOSE	CVD
ATN	CLS	CVI
CDBL	CMD	CVS

**TRS-80 LEVEL II RESERVED WORDS (cont.)**

DATA	KILL	PUT
DEFDBL	LEFT\$	RANDOM
DEFFN	LET	READ
DEFINT	LSET	REM
DEFSNG	LEN	RESET
DEFUSR	LINE	RESTORE
DEFSTR	LIST	RESUME
DELETE	LOAD	RETURN
DIM	LOC	RIGHT\$
EDIT	LOF	RND
ELSE	LOG	SAVE
END	MEM	SET
ERL	MERGE	SGN
ERR	MID\$	SIN
ERROR	MKD\$	SQR
EXP	MKI\$	STEP
FIELD	MKS\$	STOP
FIX	NAME	STRING\$
FOR	NEW	STR\$
FRE	NEXT	TAB
GET	NOT	TAN
GOSUB	ON	THEN
GOTO	OPEN	TIME\$
IF	OUT	TROFF
INKEY\$	PEEK	TRON
INP	POINT	USING
INPUT	POKE	USR
INSTR	POS	VAL
INT	PRINT	VARPTR

All these words are Level II BASIC reserved words. However, some of them have no function unless a disk operating system is connected to the computer.

### **APPLE, PET AND TRS-80 A COMPARISON**

#### **Reserved Words**

These reserved words are recognized by the computer as having special meaning and, therefore, must not be used in programs for any purpose other than what they were intended. Reserved words usually take up only one byte of program memory.

Reserved words common to the Apple, PET and TRS-80

ABS	AND	ASC	ATN	CHR\$
CONT	COS	DATA	DEF*	DIM
END	EXP	FOR	FRE	GOSUB

GOTO	IF	INPUT	INT	LEFT\$
LEN	LET	LIST	LOG	MID\$
NEW	NEXT	NOT	ON	OR
PEEK	POKE	POS	PRINT	READ
REM	RESTORE	RETURN	RIGHT\$	RND
RUN	SGN	SIN	SQR	STEP
STOP	STR\$	TAB	TAN	THEN
TO	USR	VAL		

\* Not used in the same way.

### Reserved words of similar meaning

Apple	PET	TRS-80
AT		@, AT
AUTO		AUTO
CLEAR	CLR	CLEAR
LOAD	LOAD	CLOAD
SAVE	SAVE	CSAVE
DEL		DELETE
ESC		EDIT
FN	FN	
GET	GET	INKEY\$
HOME	PRINT"␣"*	CLS
INVERSE	RVS(key)*	
NORMAL	shift RVS(key)*	
TRACE		TRON
NOTRACE		TROFF
ONERR		ON ERROR
PLOT	Graphics Keys*	SET, RESET
SCRN	PEEK	POINT
STORE	PRINT#	PRINT#
RESUME		RESUME
SPC	SPC	
VERIFY	VERIFY	CLOAD?
WAIT	WAIT	

\* Not a reserved word, but has the same result.

### Special Reserved Words

Apple			
COLOR	DRAWAT	FLASH	GR HCOLOR
HGR	HGR2	HIMEM	HLIN-AT HPLOT
H PLOT TO	LOMEN	ROT	PDL RECALL
SCALE	SHLOAD	SPEED	TEXT ULIN AT
VTAB	XDRAW AT	XPLOT	

### PET

$\pi$	TI	TI\$	TIME	TIME\$
ST				



**TRS-80**

CDBL  
ERASE  
PRINT USING  
VARPTR

CINT  
ERL

CSNG  
ERR  
STRING\$

OUT  
ERROR  
INP

ELSE  
FIX  
MEM

# Appendix E

## Boolean Operators

### BOOLEAN OPERATORS

Boolean operators can be one of the most difficult computer concepts to grasp.

The most commonly used Boolean Operators are AND, NOT and OR. They are often found in IF-THEN statements.

If we regard the Boolean Operators as testers of the truthfulness of statements, we may shed some light on their meaning and usefulness.

```
Examples:  5 IF A=5 AND B=6 THEN 1000
           50 IF A=51 OR B=19 THEN 1000
           200 IF NOT X THEN 1000
```

What do these statements mean?

The AND operator results in a True, if *both* parts of the conditional IF-THEN statement are True. If the condition posed in the IF...AND half of the statement is met, the instruction following THEN is executed.

```
IF True AND True THEN True, and therefore execute the instruction after THEN
IF True AND False THEN False
IF False AND True THEN False
IF False AND False THEN False
```

In the last three lines the condition is not satisfied, and execution continues on the next line of the program.

The following lines are an example of actual use.

```
100 INPUT"DO YOU WISH TO PLAY A GAME IF YOU ARE
      SUCCESSFUL(Y/N)"A$
110
```

```
. Educational program
.
```

```

400 PRINT"YOUR AVERAGE IS";N
410 IF A$="Y" AND N>79 THEN 1000
420 END

```

```
1000 REM***GAME***
```

These lines could be used in an educational program where a student must attain a proficiency of 80% or more on a number of questions in order to play a game.

Line 100 asks the student if he/she wants to play the game at the end of their work. Lines 110 to 390 contain the educational program.

Line 400 gives the student's average. Line 410 checks to see if the student wanted a game AND if the average is 80% or more. If both these conditions are True, then execution jumps to line 1000 where the game starts. If either of these conditions is False, that is, the student did not want a game or attained an average of less than 80%, the execution continues at line 420.

Line 420 ends the program.

The OR operator results in a True if *either* part of the conditional statement is true.

```

IF True OR True THEN True
IF True OR False THEN True
If False OR True THEN True
If False OR False THEN False

```

Only in the last line is the condition not satisfied.

The following program is an example of actual use.

```

100 INPUT"DO YOU WISH INSTRUCTIONS";A$
110 IF A$="YES" OR A$="Y" THEN 1000
120 IF A$="NO" OR A$="N" THEN 140
130 GOTO 100
140 (rest of program)
1000 PRINT"INSTRUCTIONS"

```

In this program line 100 asks the user if he/she want instructions.

Line 110 checks to see if A\$ equals "YES" or "Y". If either of these are True, then execution jumps to line 1000 for the instructions.

If A\$ does not equal "YES" OR "Y", then execution proceeds to line 120.

Line 120 checks to see if A\$ equals "NO" OR "N". If either is true, then execution jumps to line 140 and the rest of the program. If A\$ does not equal "NO" OR "N", then none of the acceptable answers to our question in line 100 have been input, and execution proceeds to line 130. This causes an unconditional jump to line 100, and the question is repeated.

The NOT operator is the least used of the Boolean operators. Placing NOT in front of a variable either changes the variable to its logical complement or checks to see if the variable is false.

Example: 10 IF NOT X THEN 100

Here, if X is false ( $X=0$ ), then execution continues at line 100. If X is true, then execution continues at the next numbered line.

The following chart summarizes the Boolean Operators. The number 1 represents a True, and 0 represents a False.

<b>AND Operator</b>	<b>OR Operator</b>	<b>NOT Operator</b>
1 AND 1=1	1 OR 1=1	NOT 0=1
1 AND 0=0	1 OR 0=1	NOT 1=0
0 AND 1=0	0 OR 1=1	
0 AND 0=0	0 OR 0=0	



## **ABOUT THE AUTHOR**

Larry Noonan is a resource teacher with the Separate School Board in Ontario, Canada. He helps teachers use microcomputers in the classroom and teaches courses in programming. Larry has written articles for various computer periodicals and is presently taking his Masters of Education in Computers in Education at the Ontario Institute of Studies in Education.







# MORE HELPFUL WORDS FOR YOU

## Computers for Everybody

Jerry Willis and Merl Miller

This fun-to-read book covers all the things you should know about computers. If you're anxious to buy one, use one or just want to find out about them, read this book first.

ISBN 0-918398-49-5

\$5.95



## Peanut Butter and Jelly Guide to Computers

Jerry Willis

This entertaining book is a simple, easy-to-digest source of information on personal computing. It leads you through all the essential knowledge of "computer literacy."

ISBN 0-918398-13-4

\$9.95



## Nailing Jelly to a Tree

Jerry Willis and William Dantley, Jr.

This is a book about software. The emphasis is on learning to use the thousands of available programs that have already been written, and adapting them to your machine.

ISBN 0-918398-42-8

\$15.95



## Small Computers for the Small Businessman

Nicholas Rosa and Sharon Rosa

If you've ever considered a computer for your business but didn't know where to turn, this is the book that will arm you with all the information you'll need to make an intelligent, cost-effective decision.

ISBN 0-918398-31-2

\$16.95



dilithium Press, P.O. Box 606, Beaverton, OR 97075

### Send to: dilithium Press, P.O. Box 606, Beaverton, OR 97075

Please send me the book(s) I have checked. I understand that if I'm not fully satisfied, I can return the book(s) within 10 days for full and prompt refund.

Computers for Everybody

Peanut Butter and Jelly Guide to Computers

Nailing Jelly to a Tree

Small Computers for the Small Businessman

Check enclosed \$ \_\_\_\_\_  
Payable to dilithium Press

Please charge my  
 VISA  Mastercharge

Send me your catalog of books.

# \_\_\_\_\_ Exp. Date \_\_\_\_\_

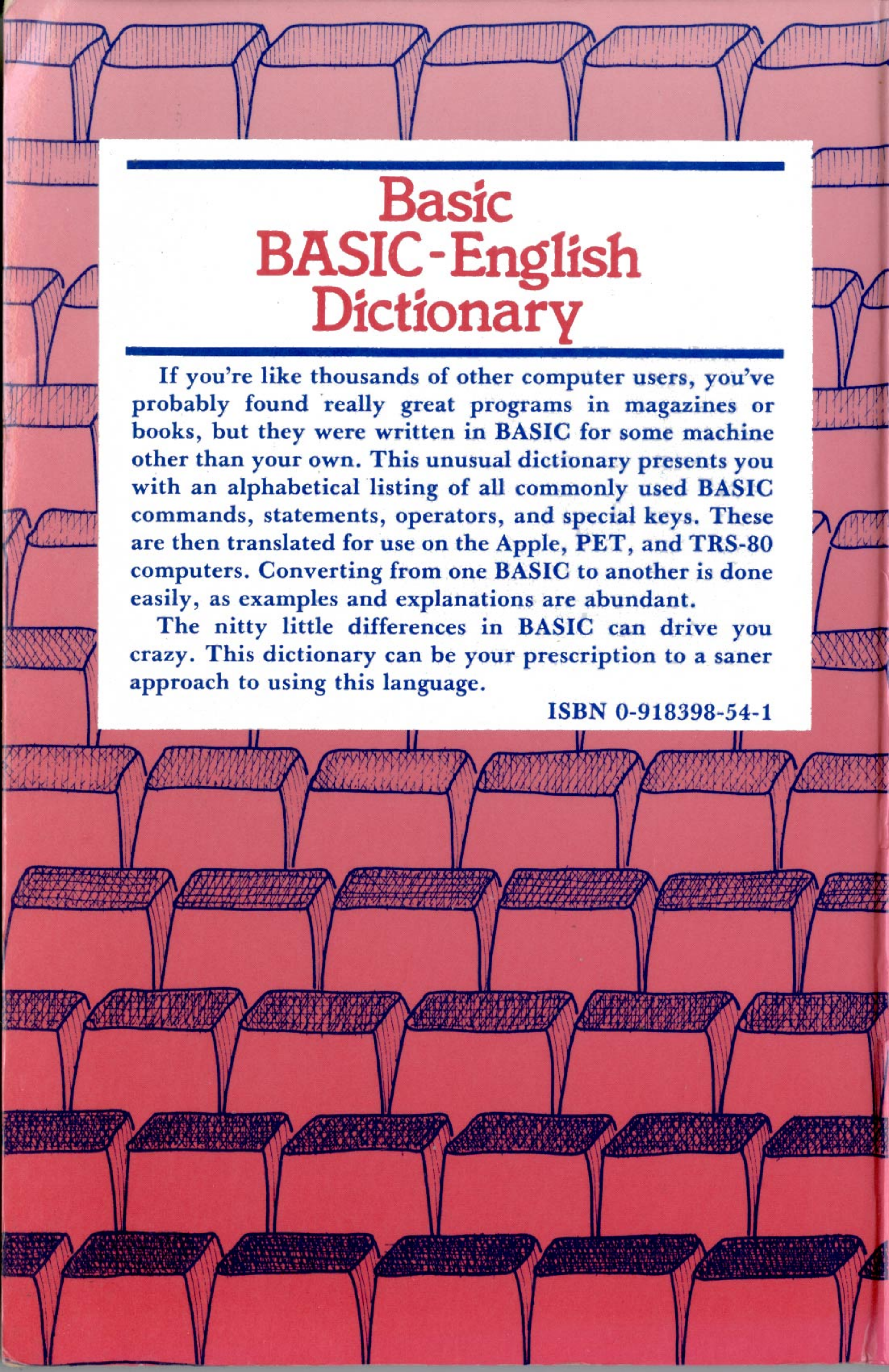
Signature \_\_\_\_\_

Name \_\_\_\_\_

Address \_\_\_\_\_

City, State, Zip \_\_\_\_\_

BB



---

# Basic BASIC-English Dictionary

---

If you're like thousands of other computer users, you've probably found really great programs in magazines or books, but they were written in BASIC for some machine other than your own. This unusual dictionary presents you with an alphabetical listing of all commonly used BASIC commands, statements, operators, and special keys. These are then translated for use on the Apple, PET, and TRS-80 computers. Converting from one BASIC to another is done easily, as examples and explanations are abundant.

The nitty little differences in BASIC can drive you crazy. This dictionary can be your prescription to a saner approach to using this language.

ISBN 0-918398-54-1