TI
Extended
BASIC
FOR THE TI-99/4 HOME COMPUTER

A powerful, high-level programming language that expands the capability of your TI-99/4 Home Computer. Includes these features:

- More than 40 new or expanded commands, statements, functions, and subprograms.
- Multiple-statement lines for speed and efficiency.
- Sprite (moving graphics) capability.
- Subprogram capability that lets you store commonly used subprograms on diskette for use as needed.
- The ability to load and run one program from another.
- Comprehensive program control of errors, warnings, and breakpoints.
- Direct screen control of input and output.
- Support for loading and running TMS9900 Assembly Language programs if the optional Memory Expansion unit (sold separately) is attached to the computer.

CONTENTS: TI Extended BASIC module
(36K bytes of preprogrammed memory)
Owner's reference manual
IMPORTANT NOTICE REGARDING PROGRAMS AND BOOK MATERIALS

The following should be read and understood before purchasing and/or using TI Extended BASIC.

Texas Instruments does not warrant that the programs contained in the TI Extended BASIC module and accompanying book materials will meet the specific requirements of the consumer, or that the programs and book materials will be free from error. The consumer assumes complete responsibility for any decision made or actions taken based on information obtained using these programs and book materials. Any statements made concerning the utility of TI's programs and book materials are not to be construed as express or implied warranties.

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THESE PROGRAMS OR BOOK MATERIALS OR ANY PROGRAMS DERIVED THEREFROM AND MAKES SUCH MATERIALS AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THESE PROGRAMS OR BOOK MATERIALS, AND THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THIS MODULE. MOREOVER, TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER AGAINST THE USER OF THESE PROGRAMS OR BOOK MATERIALS BY ANY OTHER PARTY.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you.

---

TI Extended BASIC

FOR THE TI-99/4 HOME COMPUTER

A powerful, high-level programming language that expands the capability of your TI-99/4 Home Computer. Includes these features:

- More than 40 new or expanded commands, statements, functions, and subprograms.
- Multiple-statement lines for speed and efficiency.
- Sprite (moving graphics) capability.
- Subprogram capability that lets you store commonly used subprograms on diskette for use as needed.
- The ability to load and run one program from another.
- Comprehensive program control of errors, warnings, and breakpoints.
- Direct screen control of input and output.
- Support for loading and running TMS9900 Assembly Language programs if the optional Memory Expansion unit (sold separately) is attached to the computer.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter 1 — INTRODUCTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>7</td>
</tr>
<tr>
<td>Changes from TI BASIC</td>
<td>8</td>
</tr>
<tr>
<td>How to Use this Manual</td>
<td>10</td>
</tr>
<tr>
<td>How to Use the Computer</td>
<td>11</td>
</tr>
<tr>
<td>Operating in TI Extended BASIC</td>
<td>11</td>
</tr>
<tr>
<td>Special Key Functions</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2 — OVERVIEW OF TI EXTENDED BASIC</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>15</td>
</tr>
<tr>
<td>Assignments and Input</td>
<td>16</td>
</tr>
<tr>
<td>Output</td>
<td>18</td>
</tr>
<tr>
<td>Functions, Subroutines, and Subprograms</td>
<td>20</td>
</tr>
<tr>
<td>Built-in Functions</td>
<td>20</td>
</tr>
<tr>
<td>User-Defined Functions</td>
<td>21</td>
</tr>
<tr>
<td>Subroutines</td>
<td>21</td>
</tr>
<tr>
<td>Built-in Subroutines</td>
<td>21</td>
</tr>
<tr>
<td>User-Written Subroutines</td>
<td>23</td>
</tr>
<tr>
<td>Sound, Speech, and Color</td>
<td>25</td>
</tr>
<tr>
<td>Sprites</td>
<td>25</td>
</tr>
<tr>
<td>Debugging</td>
<td>26</td>
</tr>
<tr>
<td>Error Handling</td>
<td>26</td>
</tr>
<tr>
<td>Program Entry Example</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3 — TI EXTENDED BASIC CONVENTIONS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running a Program on Power Up</td>
<td>37</td>
</tr>
<tr>
<td>IRS</td>
<td>37</td>
</tr>
<tr>
<td>Files</td>
<td>38</td>
</tr>
<tr>
<td>Line Numbers</td>
<td>38</td>
</tr>
<tr>
<td>Lines</td>
<td>38</td>
</tr>
<tr>
<td>Special Symbols</td>
<td>38</td>
</tr>
<tr>
<td>Spaces</td>
<td>39</td>
</tr>
<tr>
<td>Numeric Constants</td>
<td>39</td>
</tr>
<tr>
<td>String Constants</td>
<td>39</td>
</tr>
<tr>
<td>Variables</td>
<td>39</td>
</tr>
<tr>
<td>Numeric Expressions</td>
<td>41</td>
</tr>
<tr>
<td>String Expressions</td>
<td>41</td>
</tr>
<tr>
<td>Relational Expressions</td>
<td>41</td>
</tr>
<tr>
<td>Logical Expressions</td>
<td>42</td>
</tr>
<tr>
<td>Chapter 4 — REFERENCE SECTION</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>ABS</td>
<td>45</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>46</td>
</tr>
<tr>
<td>ASC</td>
<td>47</td>
</tr>
<tr>
<td>ATN</td>
<td>50</td>
</tr>
<tr>
<td>BREAK</td>
<td>51</td>
</tr>
<tr>
<td>BYE</td>
<td>52</td>
</tr>
<tr>
<td>CALL</td>
<td>54</td>
</tr>
<tr>
<td>CHAR</td>
<td>55</td>
</tr>
<tr>
<td>CHRPAT</td>
<td>56</td>
</tr>
<tr>
<td>CHRS</td>
<td>59</td>
</tr>
<tr>
<td>CLEAR</td>
<td>60</td>
</tr>
<tr>
<td>CLOSE</td>
<td>60</td>
</tr>
<tr>
<td>COINC</td>
<td>61</td>
</tr>
<tr>
<td>COLOR</td>
<td>62</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>64</td>
</tr>
<tr>
<td>COS</td>
<td>66</td>
</tr>
<tr>
<td>DATA</td>
<td>68</td>
</tr>
<tr>
<td>DEF</td>
<td>69</td>
</tr>
<tr>
<td>DELETES</td>
<td>70</td>
</tr>
<tr>
<td>DELSPRITE</td>
<td>72</td>
</tr>
<tr>
<td>DIM</td>
<td>74</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>75</td>
</tr>
<tr>
<td>DISPLAY USING</td>
<td>76</td>
</tr>
<tr>
<td>DISTANCE</td>
<td>77</td>
</tr>
<tr>
<td>END</td>
<td>79</td>
</tr>
<tr>
<td>EOF</td>
<td>80</td>
</tr>
<tr>
<td>ERR</td>
<td>81</td>
</tr>
<tr>
<td>EXP</td>
<td>82</td>
</tr>
<tr>
<td>FOR-TO-STEP</td>
<td>83</td>
</tr>
<tr>
<td>GCHAR</td>
<td>85</td>
</tr>
<tr>
<td>GOSUB</td>
<td>86</td>
</tr>
<tr>
<td>GOTO</td>
<td>88</td>
</tr>
<tr>
<td>HCHAR</td>
<td>89</td>
</tr>
<tr>
<td>IF-THEN-ELSE</td>
<td>91</td>
</tr>
<tr>
<td>IMAGE</td>
<td>92</td>
</tr>
<tr>
<td>INIT</td>
<td>94</td>
</tr>
<tr>
<td>INPUT</td>
<td>97</td>
</tr>
<tr>
<td>INPUT WITH FILES</td>
<td>101</td>
</tr>
<tr>
<td>INT</td>
<td>102</td>
</tr>
<tr>
<td>JOYST</td>
<td>104</td>
</tr>
<tr>
<td>KEY</td>
<td>107</td>
</tr>
<tr>
<td>LEN</td>
<td>108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LET</td>
<td>111</td>
</tr>
<tr>
<td>LINK</td>
<td>112</td>
</tr>
<tr>
<td>INPUT</td>
<td>113</td>
</tr>
<tr>
<td>LST</td>
<td>114</td>
</tr>
<tr>
<td>LOAD</td>
<td>115</td>
</tr>
<tr>
<td>LOCATE</td>
<td>116</td>
</tr>
<tr>
<td>LOG</td>
<td>117</td>
</tr>
<tr>
<td>MAGNIFY</td>
<td>118</td>
</tr>
<tr>
<td>MAX</td>
<td>121</td>
</tr>
<tr>
<td>MERGE</td>
<td>122</td>
</tr>
<tr>
<td>MN</td>
<td>124</td>
</tr>
<tr>
<td>MOTION</td>
<td>125</td>
</tr>
<tr>
<td>NEW</td>
<td>126</td>
</tr>
<tr>
<td>NEXT</td>
<td>127</td>
</tr>
<tr>
<td>NUMBER</td>
<td>128</td>
</tr>
<tr>
<td>OLD</td>
<td>129</td>
</tr>
<tr>
<td>OBJ BREAK</td>
<td>130</td>
</tr>
<tr>
<td>OBJ ERROR</td>
<td>131</td>
</tr>
<tr>
<td>OBJ GOSUB</td>
<td>133</td>
</tr>
<tr>
<td>OBJ GOTO</td>
<td>135</td>
</tr>
<tr>
<td>OBJ WARNING</td>
<td>137</td>
</tr>
<tr>
<td>OPEN</td>
<td>138</td>
</tr>
<tr>
<td>OPTION BASE</td>
<td>141</td>
</tr>
<tr>
<td>PATTERN</td>
<td>142</td>
</tr>
<tr>
<td>PEEK</td>
<td>143</td>
</tr>
<tr>
<td>PI</td>
<td>144</td>
</tr>
<tr>
<td>PCGS</td>
<td>145</td>
</tr>
<tr>
<td>POSITION</td>
<td>146</td>
</tr>
<tr>
<td>PRINT</td>
<td>147</td>
</tr>
<tr>
<td>PRINT USING</td>
<td>148</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>149</td>
</tr>
<tr>
<td>READ</td>
<td>150</td>
</tr>
<tr>
<td>REC</td>
<td>151</td>
</tr>
<tr>
<td>REM</td>
<td>152</td>
</tr>
<tr>
<td>RESEQUENCE</td>
<td>153</td>
</tr>
<tr>
<td>RESTORE</td>
<td>154</td>
</tr>
<tr>
<td>RETURN WITH GOSUB</td>
<td>155</td>
</tr>
<tr>
<td>RETURN WITH ON ERROR</td>
<td>156</td>
</tr>
<tr>
<td>RND</td>
<td>157</td>
</tr>
<tr>
<td>RPTS</td>
<td>158</td>
</tr>
<tr>
<td>RUN</td>
<td>159</td>
</tr>
<tr>
<td>SAVE</td>
<td>160</td>
</tr>
<tr>
<td>SAY</td>
<td>161</td>
</tr>
<tr>
<td>SCREEN</td>
<td>162</td>
</tr>
</tbody>
</table>

TI Extended BASIC
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG8</td>
<td>166</td>
</tr>
<tr>
<td>SGN</td>
<td>167</td>
</tr>
<tr>
<td>SIN</td>
<td>168</td>
</tr>
<tr>
<td>SIZE</td>
<td>169</td>
</tr>
<tr>
<td>SOUND</td>
<td>170</td>
</tr>
<tr>
<td>SPGET</td>
<td>172</td>
</tr>
<tr>
<td>SPRITE</td>
<td>173</td>
</tr>
<tr>
<td>SQR</td>
<td>178</td>
</tr>
<tr>
<td>STOP</td>
<td>178</td>
</tr>
<tr>
<td>STR8</td>
<td>179</td>
</tr>
<tr>
<td>SUB</td>
<td>180</td>
</tr>
<tr>
<td>SUBEND</td>
<td>184</td>
</tr>
<tr>
<td>SUBEXIT</td>
<td>184</td>
</tr>
<tr>
<td>TAB</td>
<td>185</td>
</tr>
<tr>
<td>TAN</td>
<td>186</td>
</tr>
<tr>
<td>TRACE</td>
<td>186</td>
</tr>
<tr>
<td>UNBREAK</td>
<td>187</td>
</tr>
<tr>
<td>UNTRACE</td>
<td>187</td>
</tr>
<tr>
<td>VAL</td>
<td>188</td>
</tr>
<tr>
<td>VCHAR</td>
<td>188</td>
</tr>
<tr>
<td>VERSION</td>
<td>190</td>
</tr>
</tbody>
</table>

## APPENDICES

- **Appendix A** — List of Illustrative Programs  
- **Appendix B** — List of Commands, Statements, and Functions  
- **Appendix C** — ASCII Codes  
- **Appendix D** — Musical Tone Frequencies  
- **Appendix E** — Character Sets  
- **Appendix F** — Pattern-Identifier Conversion Table  
- **Appendix G** — Color Codes  
- **Appendix H** — Color Combinations  
- **Appendix I** — Split Console Keyboard  
- **Appendix J** — Character Codes for Split Keyboard  
- **Appendix K** — Mathematical Functions  
- **Appendix L** — List of Speech Words  
- **Appendix M** — Adding Suffixes to Speech Words  
- **Appendix N** — Error Messages
INTRODUCTION

FEATURES

Texas Instruments Extended BASIC is a powerful computer programming language for use with the Texas Instruments TI-99/4 Home Computer. It has the features expected from a high level language plus additional features not available in many other languages, including those designed for use with large, expensive computers.

TI Extended BASIC goes beyond Texas Instruments Basic to enhance the capability and flexibility of your computer system by adding these features:

- **Input and Output** — The ACCEPT statement allows the input of data from anywhere on the screen. You may clear the screen, accept only certain characters, and limit the number of characters entered using this statement. The DISPLAY statement has been enhanced to allow putting data anywhere on the screen, and DISPLAY ... LSING, PRINT ... USING, and IMAGE have been added for ease in formatting data on the display screen and peripheral devices.

- **Subprograms** — Subprograms with local variables (effecting only values within the subprogram) can be written in TI Extended BASIC. Commonly used subprograms may be stored on a diskette and added to programs as needed. Statements included are SUB, SUBEND, and SUBEXIT. The MERGE command has been added and the SAVE command modified to allow the merging of programs from diskettes.

- **Sprites** — Sprites are spatially defined graphics with the ability to move smoothly on the screen. To provide the sprite capability, the following subprograms have been included in TI Extended BASIC: COLIC, DELSPRITE, DISTANCE, LOCATE, MAGNIFY, MOUNT, PATTERN, POSITION, and SPRITE. COLOR and CHAIN have been redesigned so they also can affect sprites.

- **Functions** — MAX, returning the larger of two numbers; MIN, returning the smaller of two numbers; and PI, returning the value of π, have been included in TI Extended BASIC.

- **Arrays** — Arrays may have up to seven dimensions instead of three.

- **String Handling** — The RPTS function allows the repetition of a string.

- **Error Handling** — With TI Extended BASIC, you can choose what action is taken if there is a minor error (which in TI BASIC causes a warning message), a major error (which in TI BASIC causes an error message and stops the program), or a breakpoint (which in TI BASIC causes the program to halt). The new statements allowing this control are: ON WARNING, ON ERROR, and ON BREAK. RETURN has been modified for use with error handling. The CALL ERR statement can be used to determine the nature of an error that occurs in a program.

- **RUN as a Statement** — RUN can be used as a statement as well as a command. RUN has also been modified to allow you to specify which program to run. As a result, one program can load and run another program from a diskette. You can therefore, write programs of almost unlimited size by breaking them into pieces and letting each segment run the next.

- **Power-up Program Execution** — When TI Extended BASIC is first chosen, it searches for a program named LOAD on the diskette in drive 1. If that program exists, it is placed in memory and run.

- **Multiple Statement Lines** — TI Extended BASIC allows more than one statement to be on a line. This feature speeds program execution, saves memory, and allows logical units for example FOR-NEXT loops to be on a single line.

- **SAVE and LIST Protection** — You may protect your programs from being saved or listed, preventing unauthorized copies of your programs. This, in conjunction with the copy protection feature of the Disk Manager Module, can completely secure a TI Extended BASIC program.

- **IF-THEN-ELSE** — The IF-THEN-ELSE statement now allows statements as the consequences of the comparison. This expansion permits statements such as "IF X > 4 THEN GOSUB 240 ELSE X = X + 1".

- **Multiple Assignments** — TI Extended BASIC allows you to assign a value to more than one variable in a LET statement, saving statements and permitting more efficient programming.

- **Comments** — In addition to the REM statement, comments can be added to the ends of lines in TI Extended BASIC, allowing detailed internal documentation of programs.

- **Assembly Language Support** — With the optional Memory Expansion unit (available separately), TMS9900 assembly language subprograms may be loaded and run. The subprograms INIT, LOAD, LINK, and PEEK are used to access assembly language subprograms. These are no facilities for writing assembly language programs on the TI99/4 Home Computer.

- **Information** — The SIZE command has been added to tell you how much memory remains unused in your computer. The VERSION subprogram returns a value which indicates the version of BASIC that is in use. The CHARPAT subprogram returns a character string indicating the pattern which defines a character.

- **Memory Expansion** — TI Extended BASIC allows the use of an optional Memory Expansion peripheral which permits much larger programs to be written.
INTRODUCTION

CHANGES FROM TI BASIC
The enhancements described above have made some slight changes necessary in other areas of TI BASIC. Because of these, some programs written in TI-99/4 BASIC may not run in TI Extended BASIC.

- The maximum program size is now 864 bytes smaller than in TI BASIC. If you have the Memory Expansion peripheral, much larger programs may be written.

- The characters in character sets 15 and 16 are no longer available. That memory area is used by TI Extended BASIC to keep track of sprites.

- Most programs written in TI BASIC will also run in TI Extended BASIC without difficulty. Under certain circumstances, however, such as using a TI Extended BASIC keyword as a variable in a TI BASIC program, programs written in TI BASIC may not run in TI Extended BASIC. However, you can always load TI BASIC programs into TI Extended BASIC. Programs using the enhancements of TI Extended BASIC will not run correctly in TI BASIC.

HOW TO USE THIS MANUAL
This manual assumes that you are already experienced in programming with TI BASIC. Statements, commands, and functions that are the same as in TI BASIC are only discussed briefly here. For a complete discussion, see the User’s Reference Guide that came with your TI-99/4 Home Computer.

The additional features of TI Extended BASIC are explained in detail and illustrated with examples and programs. To get the maximum use from TI Extended BASIC, read this manual carefully, entering and running the sample programs to see how they work. Even features that are unchanged from TI BASIC should be reviewed. You may find that you have been neglecting a useful statement or discovering a new way to use statements in different combinations.

The remainder of this chapter reviews the basics of operating with TI Extended BASIC. The second chapter discusses the features of TI Extended BASIC and includes a detailed example of entering a program. The third chapter discusses the conventions of operation with TI Extended BASIC. The fourth chapter is a reference section which discusses, in alphabetical order, all TI Extended BASIC commands, statements, and functions.

The 14 appendices contain much useful information, including ASCII character codes, error codes, color codes, keyboard codes, and instructions on how to add suffixes to speech words.

HOW TO USE THE COMPUTER
Before using the computer with TI Extended BASIC, you must insert the Solid State Software™ Command Module into the computer. If the program is off, slowly slide the module into the slot on the console until it is in place.

Then turn on the computer. (If you have peripherals, turn them on before turning on the computer.) The master title screen appears. If the computer is already on, return to the master title screen. Then slide the module into the slot.

Press any key to make the master selection list appear. The title of the module, TI EXTENDED BASIC, is third on the list. Type 3 to select TI Extended BASIC.

OPERATING IN TI EXTENDED BASIC
There are three main operating modes in TI Extended BASIC: Command Mode, Edit Mode, and Run Mode.

Command Mode is the mode entered when you choose TI Extended BASIC on the master selection list. In the Command Mode you may enter TI Extended BASIC commands, statements that may be used as commands, and program lines.

Edit Mode is used to edit existing lines of a TI Extended BASIC program. To enter Edit Mode, type a line number and press either SHIFT E (UP) or SHIFT X (DOWN). (TI BASIC also allows EDIT followed by a line number, which TI Extended BASIC does not allow.) The line specified is then displayed on the screen. You may change it by typing a new line by typing over part of the old line, or by using the editing keys discussed below. You are also in the Edit Mode when you press SHIFT R (REDO) to repeat a program line or command.

In Run Mode, a TI Extended BASIC program is executed. You can stop a running program only by pressing SHIFT C (CLEAR), which causes a breakpoint, or with SHIFT Q (QUIT). Note: SHIFT Q (QUIT) also erases the entire program, returns you to the master title screen, and may delete information from some of your files. The use of BYE is recommended in place of SHIFT Q (QUIT) to leave TI Extended BASIC.
INTRODUCTION

SPECIAL KEY FUNCTIONS

The following are the keys that have a special function when pressed at the same time as the SHIFT key: E, D, S, X, R, T, G, F, C, Q. Each of these keys is discussed below.

SHIFT E (UP) is used in the Edit Mode. If you are not in the Edit Mode, you may enter it by typing a line number and then pressing SHIFT E (UP). The line specified is then displayed on the screen and may be edited. If you are already in the Edit Mode, pressing SHIFT E (UP) enters the present line as you have changed it and displays the next lower numbered line in the program. Pressing SHIFT E (UP) when you are at the lowest numbered line in the program returns you to the Command Mode. If you are entering a line in the Command Mode, SHIFT E (LP) has the same effect as ENTER.

SHIFT D (RIGHT) moves the cursor one space to the right. The cursor does not erase or change characters as it passes over them. At the end of a line on the screen, the cursor wraps around to the next screen line. When the cursor is at the end of an input line, it does not move.

SHIFT S (LEFT) moves the cursor one space to the left. The cursor does not erase or change characters as it passes over them. If the cursor is at the beginning of a line, the cursor does not move. If the cursor is at the left margin but not at the beginning of an input line, the cursor goes to the right margin of the screen line above it.

SHIFT X (DOWN) is used in the Edit Mode. If you are not in the Edit Mode, you may enter it by typing a line number and then pressing SHIFT X (DOWN). The line specified by the line number is then displayed on the screen and may be edited. If you are in the Edit Mode, pressing SHIFT X (DOWN) enters the present line as you have changed it and displays the next higher numbered line in the program. Pressing SHIFT X (DOWN) when you are at the highest numbered line in the program returns you to the Command Mode. If you are entering a line in the Command Mode, SHIFT X (DOWN) has the same effect as ENTER.

SHIFT R (REDO) causes the characters on the line previously input to reappear on the screen. Thus if you wish to enter a line similar to the most recently entered line, press SHIFT R (REDO). If you enter a line and make a mistake, you can recall the line using SHIFT R (REDO) and correct it using the Edit Mode features. This key lets you avoid retyping a long line.

SHIFT T (ERASE) erases all characters on the current line, but leaves the cursor on that line. If you are in the Command Mode, the cursor returns to the left margin of the screen and you may enter a new line, including the line number. However, if you are editing a line or the computer is providing the line numbers (through the use of NUM), the line number is not erased.

SHIFT G (INSERT) instructs the computer to accept inserted characters. Each subsequent key that you type is inserted at the cursor position and the character at the cursor position and all characters to the right of the cursor are shifted one position to the right. Insertion continues with each character typed until ENTER or one of the other special function keys is pressed. Characters at the end of a long input line may be lost.

SHIFT F (DELETE) deletes the character that the cursor is on and shifts all characters to the right of the cursor one position to the left.

SHIFT C (CLEAR) performs different functions depending on the mode that you are in. If you are in the Edit Mode, any changes that were made to the line are ignored, including SHIFT T (ERASE), and the computer returns to Command Mode. If you are in Run Mode, the program is stopped with a breakpoint. If you are in Command Mode, the characters that you have typed on the current line are deleted. When using SHIFT C (CLEAR) to stop a program, hold the keys down until TI Extended BASIC recognizes the breakpoint.
INTRODUCTION

SHIFT Q (QUIT) returns the computer to the master title screen. When you press SHIFT Q (QUIT), all data and program material are erased from the computer’s memory. If you are using a disk system, some of your data files may be lost. Leave TI Extended BASIC by entering BYE instead of using SHIFT Q (QUIT).

ENTER indicates that you have finished typing the information on the current line and are ready for the computer to process it.

Overview of TI Extended BASIC

This chapter briefly describes the TI Extended BASIC commands, statements, and functions and suggests ways in which you can use them. The first eight sections are Commands; Assignments and Input; Output; Functions, Subroutines, and Subprograms; Sound, Speech, and Color; Sprites; Debugging; and Error Handling. The final section is an example of the entry of a program showing the entry process and the use of some of the TI Extended BASIC elements.
OVERVIEW OF TI EXTENDED BASIC

COMMANDS
Commands tell the computer to perform a task immediately (that is, as soon as you press ENTER), while statements are executed when a program is run. In TI Extended BASIC many commands can be used as statements, and most statements can be used as commands. A list of all the commands, statements, and functions is given in Appendix B, indicating the commands that can be used as statements and the statements that can be used as commands.

NEW
To remove a program from TI Extended BASIC to prepare the computer to accept a new program, use the NEW command. Programs are also removed from memory by the OLD command and the RUN command when used with a file name.

NUMBER and RESEQUENCE
When you are entering a program, the computer assigns line numbers for you if you enter the NUMBER command. If you wish to resequence the line numbers of a program after it is written, use the RESEQUENCE command.

LIST
To review the program that you have entered, use the LIST command. The program can be listed on the screen or to a peripheral device.

RUN
The RUN command instructs the computer to perform, or "execute," a program. The RUN command may be followed by a line number to have it start program execution at a specific line, or by a device and filename to load and execute a program from a diskette.

TRACE, UNTRACE, BREAK, UNBREAK, and CONTINUE
All of these commands are related to "debugging" a program, which is finding a problem that causes an error condition or an incorrect result. These commands are discussed further in the "Debugging and Error Handling" section of this chapter.

SAVE, OLD, MERGE, and DELETE
When you are finished working on a program, you may want to store it on a cassette or diskette for later use. The SAVE command, followed by the name of the storage device and a program name, performs this task for you. Then, when you wish to reuse, list, edit, or change a program, you can load it into memory with the OLD command. If a program has been saved using the merge option, you can combine it with a program already in memory with the MERGE command. When you have no further use for a program that has been saved on diskette, you can remove it with the DELETE command.

SIZE
The SIZE command lets you determine how much memory space is left, so you can decide whether to continue to add program lines or end the program and have a second program run from the first program with RUN used as a statement.

BYE
When you have finished using TI Extended BASIC, use the BYE command to return to the master title screen.

Several of the commands (RUN, BREAK, UNBREAK, TRACE, UNTRACE, and DELETE) can also be used as statements in programs.

ASSIGNMENTS AND INPUT
This section discusses statements in TI Extended BASIC that assign values to variables and enter data into programs.

LET and READ
If you know what values are to be assigned to variables, use LET or READ statements. LET is used when you are assigning a fairly small number of values or are calculating values to be assigned, and READ is used, in conjunction with DATA and RESTORE, when you are assigning numerous values.

INPUT and LINPUT
When you want the user of the program to assign values, it is customary to give a prompt that asks for the necessary information. INPUT allows you to give a prompt and accept input. INPUT only allows the entry of values at the bottom of the screen and cannot check to see that the data entered is the type of information the program expects. The final limitation on INPUT is that commas and quotation marks affect what is entered. With LINPUT, there is no editing of what is input, so commas and quotation marks can be input. Both INPUT and LINPUT may be used to input data from files on cassettes and diskettes.

ACCEPT
ACCEPT allows input from most screen positions. Using ACCEPT eliminates the necessity of entering data at the bottom of the screen and the "scrolling" of the INPUT statement. However, ACCEPT doesn't allow a prompt as the INPUT statement does. Therefore, a PRINT or DISPLAY statement must be included in the program to tell the user the type of entry that is required. ACCEPT can check the input to see that it is numeric, alphabetical, or specific characters. ACCEPT is for screen and keyboard use only.
CALL KEY and CALL JOYST
If pressing a single key is all that the program user is required to do, then
CALL KEY can be used. For example, if a 'y' for "yes" or 'N' for "no" is the
required response, use the CALL KEY statement to accept the entry. CALL
KEY does not display a character on the screen. It scans the keyboard or a
portion of the keyboard to see if a key has been pressed. The major limitation
of CALL KEY is that only a single keystroke is accepted. The data is not
recorded as a character, but rather as the ASCII code for the character or as
some other code (See Appendices C and J for a list of the codes used.) If you
wish to show the key that was pressed, you must use DISPLAY, PRINT,
CALL VCHAR, or CALL FCHAR. The input from a Wired Remote Controller
can be used with CALL JOYST. As with CALL KEY, the data is not
displayed, and no scrolling takes place.

CALL CHARPOT, CALL COINC, CALL DISTANCE, CALL ERR, FOR-TO-
STEP, CALL GCHAR, CALL POSITION, NEXT, CALL SPGET, and CALL
VERSION
Each of these statements assigns one or more values to a variable. CALL
CHARPOT assigns a value that specifies the pattern of a character. CALL
COINC assigns a value to tell if sprites or a sprite and a point on the screen
are at or near the same location on the screen. CALL DISTANCE indicates
the distance between two sprites or a sprite and a point on the screen. CALL
3RR specifies the error that occurred and where it occurred. CALL GCHAR
reads what character is at a given screen location. CALL POSITION reads
where a sprite is on the screen. CALL SPGET assigns the coded value of a
speech phrase to a variable to be used with CALL SAY. CALL VERSION
indicates the version of BASIC in use.

FOR-TO-STEP and NEXT deserve special comment. The FOR-TO-STEP
statement sets the value of a variable so that it can be used to control the
number of times a loop is executed. Each time NEXT is encountered, the
value of the variable is changed. After the loop has been completed, the
variable has a value that is the first value outside the range specified in the
FOR-TO-STEP statement.

OUTPUT
This section discusses the TI Extended BASIC statements which are used to
output data during program execution. Usually, output consists of displaying
information on the screen, printing data on a printer, or saving data on an
external device. However, output can also involve changing the color of the
screen, changing the colors of characters, making noises, speaking, or
sending data to peripheral devices.

PRINT, DISPLAY, PRINT..., USING, DISPLAY..., USING, and IMAGE
The two most frequently used output statements are PRINT and DISPLAY.
The print separators (comma, semicolon, and colon) and the TAB function
are used to control the placement of information as it is output. PRINT
displays items at the bottom of the screen and scrolls them upward. With DISPLAY, you can display data almost anywhere on the
screen without scrolling. DISPLAY can also clear the screen, erase characters
on a line, and cause a beep.

PRINT..., USING, and DISPLAY..., USING, are like PRINT and DISPLAY except
that the format of the printed or displayed characters is determined by the
USING clause, possibly in conjunction with an IMAGE statement. The USING
clause allows exact control of the format. PRINT and PRINT..., USING,
possibly in conjunction with IMAGE, are the only output statements that
can be used to send data to an external device.

CALL HCHAR, CALL VCHAR, and CALL SPRITE
CALL HCHAR and CALL VCHAR place a character at any screen position
and optionally repeat it horizontally or vertically. CALL SPRITE displays
"sprites" on the screen. Sprites are graphics that can be moved smoothly in
any direction and changed in pattern, size, and color. CALL SPRITE and the
other statements related to sprites are discussed later in this chapter.

CALL SCREEN and CALL COLOCR
In addition to displaying characters and data on the screen, you can change
the color of the screen and the colors of the characters. CALL SCREEN sets
the screen color. CALL COLOR specifies the foreground and background
colors of characters or the color of sprites.

CALL SOUND and CALL SAY
CALL SOUND outputs sounds. A wide range of sounds is available. In
addition, CALL SAY (possibly used with CALL SPGET) makes the computer
speak if you have a Solid State Speech™ Synthesizer attached to your
computer.

FUNCTIONS, SUBROUTINES, AND SUBPROGRAMS
TI Extended BASIC provides extensive functions and subprograms for
handling numbers and characters. In addition, you may construct your own
functions and write your own subprograms and subroutines.

Functions are TI Extended BASIC language elements that return a value,
usually based on parameters given to the function. Many functions are
mathematical in nature; others control or affect the result. x output produced
by the statements in which they occur. The TI Extended BASIC functions are
ABS, ASC, ATN, CHR$, COS, EOF, EXP, INT, LEN, LOG, MAX, MIN, PI,
POS, REC, RND, RPT$, SEG$, SIN, SQR, STR$, TAN, and VAL.
You can also define your own functions using DEF. Functions are used within TI Extended BASIC statements.

**Built-in Functions**
The following briefly discusses each built-in function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Value Returned and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Absolute value of a numeric expression.</td>
</tr>
<tr>
<td>ASC</td>
<td>The numeric ASCII code of the first character of a string expression.</td>
</tr>
<tr>
<td>ATN</td>
<td>Trigonometric arctangent of a numeric expression given in radians.</td>
</tr>
<tr>
<td>CHRS</td>
<td>Character that corresponds to an ASCII code.</td>
</tr>
<tr>
<td>COS</td>
<td>Trigonometric cosine of a numeric expression given in radians.</td>
</tr>
<tr>
<td>EOF</td>
<td>End-of-file condition of a file.</td>
</tr>
<tr>
<td>EXP</td>
<td>Exponential value (e) of a numeric expression.</td>
</tr>
<tr>
<td>INT</td>
<td>Integer value of a numeric expression.</td>
</tr>
<tr>
<td>LEN</td>
<td>Number of characters in a string expression.</td>
</tr>
<tr>
<td>LOG</td>
<td>Natural logarithm of a numeric expression.</td>
</tr>
<tr>
<td>MAX</td>
<td>Larger of two numeric expressions.</td>
</tr>
<tr>
<td>MIN</td>
<td>Smaller of two numeric expressions.</td>
</tr>
<tr>
<td>PI</td>
<td>π with a value of 3.141592654.</td>
</tr>
<tr>
<td>POS</td>
<td>Position of the first occurrence of one string expression within another.</td>
</tr>
<tr>
<td>REC</td>
<td>Current record position in a file.</td>
</tr>
<tr>
<td>END</td>
<td>Random number from 0 to 1.</td>
</tr>
<tr>
<td>RPTS</td>
<td>String expression equal to a number of copies of a string expression concatenated together.</td>
</tr>
<tr>
<td>SEGs</td>
<td>Substring of a string expression, starting at a specified point in that string and ending after a certain number of characters.</td>
</tr>
<tr>
<td>SGN</td>
<td>Sign of a numeric expression.</td>
</tr>
<tr>
<td>SIN</td>
<td>Trigonometric sine of a numeric expression given in radians.</td>
</tr>
<tr>
<td>SQRT</td>
<td>Square root of a numeric expression.</td>
</tr>
<tr>
<td>STRS</td>
<td>String equivalent of a numeric expression.</td>
</tr>
<tr>
<td>TAB</td>
<td>Position for the next item in the print-list of PRINT, PRINT...USING, DISPLAY, or DISPLAY...USING.</td>
</tr>
<tr>
<td>TAN</td>
<td>Trigonometric tangent of a numeric expression given in radians.</td>
</tr>
<tr>
<td>VAL</td>
<td>Numeric value of a string expression which represents a number.</td>
</tr>
</tbody>
</table>

**User-Defined Functions**
DEF is used to define your own functions. Functions up to one line in length may be defined, with up to one argument. Longer functions may be constructed by having new definitions refer to previously defined functions. However, long functions might be more efficiently handled with subroutines or subprograms.

**Subroutines**
GOSUB and ON...GOSUB are used to call subroutines. A subroutine is a series of statements designed to perform a task and is normally used in a program when it performs a task several times. By using GOSUB or ON...GOSUB, you do not have to type the same lines of code several times. The subroutine can use the values of any variable in the program and change those values.

**Built-In Subprograms**
Built-in subprograms are TI Extended BASIC elements that perform special functions. They always are accessed with the CALL statement. The built-in subprograms are CHAR, CHARPAT, CHARSET, CLEAR, CCINC, COLOR, DELSPRITE, DISTANCE, ERR, GCHAR, HCHAR, INIT, JOYST, KEY, LINK, LOAD, LOCATE, MAGNIFY, MOTION, PATTERN, PEEK, FCSSION, SAY, SCREEN, SOUND, SPGET, SPRITE, VCHAR, and VERSION.

Built-in subprograms perform many different tasks. Some of the subprograms affect the display and determine what key has been pressed on the keyboard.

<table>
<thead>
<tr>
<th>Built-in Subprogram</th>
<th>Action and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>Clears the screen.</td>
</tr>
<tr>
<td>COLOR</td>
<td>Specifies the colors of characters in character sets or the color of sprites.</td>
</tr>
<tr>
<td>GCHAR</td>
<td>Returns the ASCII code of the character at a screen position.</td>
</tr>
<tr>
<td>HCHAR</td>
<td>Displays a character on the screen and optionally repeats it horizontally.</td>
</tr>
<tr>
<td>JOYST</td>
<td>Returns values indicating the position of the Wired Remote Controllers (optional).</td>
</tr>
<tr>
<td>KEY</td>
<td>Returns a code indicating the key that has been pressed.</td>
</tr>
<tr>
<td>SCREEN</td>
<td>Specifies the color of the screen.</td>
</tr>
<tr>
<td>VCHAR</td>
<td>Displays a character on the screen and optionally repeats it vertically.</td>
</tr>
</tbody>
</table>
OVERVIEW OF TI EXTENDED BASIC

Built-in subprograms can also define and control sprites.

**Built-in Subprogram**  **Action and Comments**
CHAR  Specifies the pattern for a character used for a sprite or a graphic.
COINC Determines if two sprites or a sprite and a point on the screen are at or near the same location on the screen.
COLOR Specifies the color of a sprite or a character set.
DELSRITE Deletes sprites.
DISTANCE Determines the distance between two sprites or a sprite and a location.
LOCATE Specifies the position of a sprite.
MAGNIFY Changes the size of sprites.
MOTION Specifies the motion of a sprite.
PATTERN Specifies the character that defines a sprite.
POSITION Determines the position of a sprite.
SPRITE Defines sprites, specifying the character that defines them, their color, their position, and their motion.

A third category of built-in TI Extended BASIC subprograms involves sound and speech.

**Built-in Subprogram**  **Action and Comments**
SAI Causes the computer to speak words when used in conjunction with the Solid State Speech™ Synthesizer.
SOUND Generates sounds.
SPGET Retrieves the codes that make speech.

Four built-in subprograms are only used with machine language subprograms obtained from Texas Instruments or another source written in TMS9900 machine language on another computer. Machine language subprograms cannot be written on the TI-99/4 Home Computer. Detailed instructions on the use of INIT, LINK, LOAD, and PEEK are provided with machine language subprograms.

Finally there are some miscellaneous built-in subprograms.

**Built-in Subprogram**  **Action and Comments**
CHARPAT Returns a value that identifies the pattern of a character.
CHARSET Resets characters 32 through 95 to their original predefined patterns and colors.
ERR Returns values which give information about an error that has occurred.
VERSION Specifies the version of BASIC that is being used.

**User-Written Subprograms**
You may write your own subprograms. They are a series of statements designed to perform a task. They may be used in a program when you expect to perform the task several times or to perform the same task in several different programs. Using the MERGE option when you save a subprogram allows it to be included in other programs.

When a subprogram is in a program, it must follow the main program. The structure of a program must be as follows:

Start of Main Program
.
.
.
Subprogram Calls
.
.
.
End of Main Program

The program will stop here without a STOP or END statement. Subprograms are optional.

Start of First Subprogram
.
.
.
End of First Subprogram

Nothing may appear between subprograms except remarks and the END statement.

Start of Second Subprogram
.
.
.
End of Second Subprogram

Only remarks and END may appear after the subprograms.

End of Program
OVERVIEW OF TI EXTENDED BASIC

Subprograms are called by the use of CALL followed by the subprogram's name and an optional list of parameters and values. The first line of a subprogram is SUB followed by the name of the subprogram and optionally followed by a list of parameters.

The subprograms you write are not part of the main program. They cannot use the values of variables in the main program, so any values that are needed must be supplied by the parameter list in the CALL statement. Variable names may be duplicates of those in the main program or other subprograms without affecting the values of the variables in the main program or other subprograms. Subprograms may call other subprograms, but may not call themselves, either directly or indirectly.

SUBEND must be the last statement in a subprogram. When that statement is executed, control returns to the statement following the statement that called the subprogram. Control may also be returned by the SUBEXIT statement.

SOUND, SPEECH, AND COLOR

You may highlight important sections of your program's output through the use of sounds, speech, and colors. This "human engineering" makes the program easier and more interesting to use.

CALL SOUND

SOUND outputs sounds. Tones may be output in lengths of from .001 to 4.25 seconds at volumes from 0 (loudest) to 30 (softest). The frequency range is from 110 (A below low C) to 44.733 (above the range of human hearing). In addition, 8 noises are available. Up to three tones and one noise may be produced at the same time. Appendix D lists the frequencies that are used to produce the musical notes.

CALL SAY and CALL SPGET

SAY produces speech when a Texas Instruments Solid State Speech™ Synthesizer (sold separately) is attached to the console. You can choose among 373 letters, numbers, words, and phrases listed in Appendix L. In addition, you can construct new words from old by combining words. For example, SOME + THING produces "something" and THERE + FOUR produces "therefore."

SPGET is used to retrieve the speech codes that produce speech. These patterns can then be used to produce more natural speech and can be used to change words. Because making new words is a complex process, it is not discussed in this manual. However, suffixes can be added rather simply. Appendix M tells how to add the suffixes ING, S, and ED to any word, so that words such as ANSWERING, ANSWERS, ANSWERED, INSTRUCTING, INSTRUCTS, and INSTRUCTED are included in the computer's vocabulary.

CALL COLOR and CALL SCREEN

COLOR changes the colors of character sets and determines sprite colors. SCREEN specifies the color of the screen as one of the sixteen colors available on the TI-99/4 Home Computer.

SPRITES

Sprites are graphics that can be displayed and moved on the screen. One advantage that sprites have over other characters is that they can be at any of 49,152 positions of 192 rows and 256 columns rather than one of the 768 positions of 24 rows and 32 columns used by statements such as CALL VCHAR and CALL HCHAR. Because of this greater resolution, sprites can move more smoothly than characters. Also, once set in motion, sprites can continue to move without further program control.

CALL SPRITE

CALL SPRITE defines sprites. This subprogram specifies the character pattern that sprites use, their color, their position, and, optionally, their motion.

CALL CHAR and CALL MAGNIFY

Although you may use any of the predefined characters, numbers 32 through 95, as a sprite, CALL CHAR is generally used to define a new pattern for a sprite. Up to four 8 by 8 dot characters may be used to form a sprite. The MAGNIFY subprogram controls the resolution and size of sprites.

CALL COLOR, CALL LOCATE, CALL PATTERN, and CALL MOTION

Once a sprite is set up, it can be altered by various subprograms. COLOR changes the color of a sprite. LOCATE moves the sprite to a new position. PATTERN changes the character that defines a sprite. MOTION alters the motion of a sprite.

CALL COINC, CALL DISTANCE, and CALL POSITION

Three subprograms provide information about sprites while a program is running. COINC returns a value that indicates if sprites or a sprite and a point on the screen are at or near the same place on the screen. DISTANCE returns a value that specifies the distance between two sprites or a sprite and a point on the screen. POSITION returns values that indicate the position of a sprite.

CALL DELSPRITE

CALL DELSPRITE allows you to delete sprites. If you prefer, you may "hide" sprites by locating them off the bottom of the screen.
OVERVIEW OF TI EXTENDED BASIC

DEBUGGING

Debugging a program is finding logical or typing errors in a program. BREAK, CONTINUE, TRACE, ON BREAK, UNBREAK, UNTRACE, and SHIFT C (CLEAR) are most often used in debugging.

BREAK, ON BREAK, CONTINUE, and UNBREAK

BREAK causes the computer to stop program execution so that you can print the values of variables or change their values. BREAK also resets characters to their standard colors (black on transparent), restores the standard screen color (cyan), restores the standard characters (32-95) to their standard representation, and deletes sprites.

ON BREAK tells the computer what to do if a break occurs. You can use this statement to tell the computer to ignore breakpoints that you have entered in the program. CONTINUE causes the computer to continue program execution after a breakpoint. UNBREAK cancels any breakpoints set with BREAK. Note: If you have put ON BREAK CONTINUE, the computer will not stop when you press SHIFT C (CLEAR).

TRACE and UNTRACE

TRACE causes the computer to display each line number before the statement(s) on that line is (are) executed. Using this statement allows you to follow the sequence of operation of a program. UNTRACE cancels the operation of TRACE.

ERROR HANDLING

You may include statements in a program to handle errors that occur while the program is running.

CALL ERR, CN ERROR, ON WARNING, and RETURN

CALL ERR returns information indicating where an error has occurred and what the error is. Appendix N lists the error codes that are returned. ON ERROR specifies what the computer does if an error occurs. ON WARNING specifies what the computer does if a condition arises that would normally cause a warning message to be issued. RETURN is used with ON ERROR in addition to its use with GOSUB. It repeats execution of the statement that caused the error, returns to the statement following the one that caused the error, or transfers control to some other part of the program that avoids the error that has occurred.

PROGRAM ENTRY EXAMPLE

Now that you've had a brief overview of the features of TI Extended BASIC, you may enjoy reviewing or even entering and experimenting with a demonstration program. This section demonstrates a number of the useful features of TI Extended BASIC. By following the suggestions in this section, you can learn some useful shortcuts in the entry process.

This program allows you to play a game called Codebreaker. In playing it, you determine the length of a code (1 to 8 digits). Then you decide the range of digits that may be included in the code (up to ten). The computer selects the digits in the code without repeating digits. You then guess what the digits are and their sequence. After each guess, the computer tells you how many digits you guessed correctly and how many are in the correct place. If you repeat a digit in your guess, it is counted as right each time it appears. Using this information, you guess again. You win when you guess all the digits correctly and place them in the proper sequence.

For example, suppose you've chosen to play the game using four digits with each digit being any one of nine numbers (0, 1, 2, 3, 4, 5, 6, 7, or 8). The code the computer chooses might be 0743, which you are trying to break. Here is a possible sequence of guesses.

GUESS  RIGHT  PLACE  EXPLANATION OF THE
0000  4     1     COMPUTER'S RESPONSE
1234  2     0
5678  1     0
2348  2     1
0347  4     2
3047  4     1
0734  4     2
0743  4     4

To begin entering the example, turn on any peripheral devices you have connected to the computer. Insert the TI Extended BASIC Command Module and turn on the computer. Press any key to go to the master selection list. Press 3 to select TI Extended BASIC.

In the following, the characters you type and the keys you press are UNDERLINED.
OVERVIEW OF TI EXTENDED BASIC

CODEBREAKER Program Entry

COMMENTS

Automatically numbers the program lines.

Title and language.

Reserves room for the codes and guesses.

Makes the codes random.

Clears the screen, beeps, and puts the title CODEBREAKER on the 11th row starting in the 9th column.

REDO repeats whatever was done before ENTER was last pressed. Using the edit keys [SHIFT 3 (INSECT)], [SHIFT F (DELETE)], and the arrows, change line 130 to: 140 DISPLAY AT(19,1)BEEP: "NUMBER OF CODES? (1-8)".

Beeps and displays NUMBER OF CODES? (1-8) on the 19th row starting at the first column.

Press SHIFT R (REDO) again. Now change line 140 to: 150 DISPLAY AT(21,6)BEEP: "DIGITS EACH CODE?".

Beeps and displays DIGITS EACH CODE? on the 21st row starting at the 6th column.

Accepts into CODES an entry on the 19th line, 24th column, allowing only digits to be entered.

Change line 160 to: 170 ACCEPT AT(21,24) VALIDATE (DIGIT):DIGITS.

Accepts into DIGITS an entry on the 21st line, 24th column, allowing only digits to be entered.

DISPLAY

* READY *

>NUM

>100 REM CODEBREAKER XBASIC

>110 DIM CODE$(8), GUESS$(8)

>120 RANDOMIZE

>130 DISPLAY AT(11,9)BEEP ERA

SE ALL: "CODEBREAKER"

>140

SHIF T R

140 DISPLAY AT(19,1)BEEP: "NU
MBER OF CODES? (1-8)"

>SHIFT R

150 DISPLAY AT(21,6)BEEP: "DI
GITS EACH CODE?"

>160 ACCEPT AT(19,24) VALIDATE

(DIGIT): CODES

>170 ACCEPT AT(21,24) VALIDATE

(DIGIT): DIGITS

LIST

100 REM CODEBREAKER XBASIC

110 DIM CODE$(8), GUESS$(8)

120 RANDOMIZE

130 DISPLAY AT(11,9)BEEP ERA

SE ALL: "CODEBREAKER"

140 DISPLAY AT(19,1)BEEP: "NU
MBER OF CODES? (1-8)"

150 DISPLAY AT(21,6)BEEP: "DI
GITS EACH CODE?"

160 ACCEPT AT(19,24) VALIDATE

(DIGIT): CODES

170 ACCEPT AT(21,24) VALIDATE

(DIGIT): DIGITS

RUN

CODEBREAKER

NUMBER OF CODES? (1-8) ■

DIGITS EACH CODE?

Enter anything except a digit. The computer beeps and does not accept it.

Enter 4. The cursor moves down to the second prompt. Enter 10. The program ends and you can continue entry.

* READY *

>NUM 180

>180 IF CODES>DIGITS THEN DIS

PLAY AT(24,7)BEEP: "NO MORE CODES THAN DIGITS":COTO 160

Enter

Numbers lines starting with 180.

Checks to see that there will be enough digits for the number of codes. If CODES is less than or equal to DIGITS, control passes to the next line. If CODES is greater than DIGITS, the message NO MORE CODES THAN DIGITS is displayed on the last line of the screen, and control is transferred to line 160 again.
OVERVIEW OF TI EXTENDED BASIC

Starts the loop to choose the codes.
The words after the exclamation point are a comment.
Chooses codes at random.

Starts the loop to prevent duplicate codes.

Checks for duplicates. Chooses a new code if there is a duplicate.
Finishes duplicate check loop.
Finishes code choice loop.
Sets a variable to keep track of where information is displayed on the screen.

Clears the screen and displays a column heading on the top line.
REDO line 260 so it reads: DISPLAY AT(24,3):"ENTER 'X' FOR SOLUTION".
Displays an instruction at the bottom of the screen.
Numbers lines starting at 280.
Accepts the guess at the proper row.
Checks for giving up or resetting.

Begins loop to break up the guess to check it for accuracy.
Separates guess into individual digits.
Completes loop to separate guess.
Sets RIGHT and PLACE to zero.
 Begins outside loop to check the guess against the code.
Begins inside loop to check guess.
If a guess doesn’t match a code, goes to the next line. If a guess matches a code, adds one to the number correct. Then if the guess is in the correct place, adds one to the number in the correct place.

>190 FOR A=1 TO CODES CHOOSE CODES
>200 CODES(A)=STPS(IN$(RAND*DINT(100))
>210 FOR B=0 TO A-1:CHECK FOR Duplicates
>220 IF CODES(A)=CODES(B) THEN N=200
>230 NEXT B
>240 NEXT A
>250 R0=2

>260 DISPLAY AT(1,1) ERASE ALL
:"'GUESS " RIGHT " PLACE:" ENTER

>270 DISPLAY AT(24,3):"ENTER 'X' FOR SOLUTION"
>280 ACCEPT AT(ROW,1):C$" " >290 IF C$="X" THEN 470 :GIVE UP OR IRESET
>300 FOR D=1 TO CODES:IF D=0 THEN 470
>310 GUESS(D)=SEG$(C$,3,1)

>320 NEXT D
>330 RIGHT,PLACE=0
>340 FOR E=1 TO CODES:CHECK GUESS FOR CORRECTNESS
>350 IF E=PLACE THEN
>360 IF CODE$(E)=GUESS$(E) THEN EN RIGHT=RIGHT+1:IF E+F THEN N PLACE=PLACE+1

>370 NEXT F
>380 NEXT E
>390 DISPLAY AT(ROW,14):RIGHT

>400 REDO line 390 to be: DISPLAY AT(ROW,14):RIGHT

>410 DISPLAY AT(ROW,22):PLACE.

>420 Displays the number of digits that are correct.

>430 Numbers lines starting at 410.
Checks to see if the code has been solved. If it has, goes to the next line.
If it hasn’t, adds one to the row.
Then if the row is more than 22, goes to line 470 and gives the solution. Otherwise, returns to line 280 to accept another guess.

>440 Displays the win message with the number of guesses at the 23rd row starting at the first column.
REDO line 420 to be: DISPLAY AT(24,1):BEEP:"PLAY AGAIN? (Y/N)"

>450 Displays the prompt PLAY AGAIN? (Y/N) Y at the 24th row starting at the first column.
REDO line 440 to be: ACCEPT AT(24,19):SIZE(-1)
BEEP: "VALIDATE ("Y/N") :X"

>460 STOP
OVERVIEW OF TI EXTENDED BASIC

Displays the message THE CODE IS at the 23rd row, 1st column.

Begins a loop to display the digits. Displays the digits.

Finishes the loop. Leave the number mode.

Press DOWN ARROW as if to edit line 430 so you can use **SHIFT R** (REDO).

Press REDO. Line 510 is a duplicate of line 430 so change the line number to 510.

Displays the prompt PLAY AGAIN? (YN) Y at the 24th row starting at the 1st column.

Press DOWN ARROW as if to edit line 440 so you can use **SHIFT R** (REDO).

Press REDO. Line 520 is a duplicate of line 440 so change the line number to 520.

Accepts an entry into X$ on the 24th row, 19th column. Does not remove any character that is already displayed (in this case a Y from the DISPLAY statement in line 510). accepts only one character, beeps and accepts only Y or N. Pressing **ENTER** at this point when the program is running confirms the Y that was displayed by line 5.0.

If Y is entered, returns to line 130, allows changing the number of digits in the code and the number of acceptable digits, and starts a new game.

```bas
>470 DISPLAY AT(23,1)"THE CODE IS":"LOSS, GIVE UP, OR RESET
>480 FOR G=1 TO CODES
>490 DISPLAY AT(23,12+C):CODE $G
>500 NEXT G
>510
>430 DOWN ARROW
>430 DISPLAY AT(24,1):BEEP:"PLAY AGAIN? (Y/N) Y" ENTER
>510 DISPLAY AT(24,1):BEEP:"PLAY AGAIN? (Y/N) Y" ENTER
>440 DOWN ARROW
>440 ACCEPT AT(24,19):SIZE(-1)
>520 ACCEPT AT(24,19):SIZE(-1)
>530 IF X$="Y" THEN 130
```

Before running a program, you should proofread it. Here is a list of the entire program for you to check against your program list.

100 REM CODEBREAKER XBASIC
110 REM CODEBREAKER XBASIC
120 DIM CODES$(8), GUESS$(8)
130 RANDOMIZE
140 DISPLAY AT(11,9): BEEP 51A
150 DISPLAY AT(19,1): BEEP: "NUMBER OF CODES? (1-8)"
160 DISPLAY AT(21,6): BEEP: DIGITS EACH CODE?
170 ACCEPT AT(19,24): VALIDATE (DIGIT): CODES
180 IF CODES>DIGITS THEN DISPLAY AT(24,2): BEEP: "NO MORE CODES THAN DIGITS": GOTO 150
190 FOR A=1 TO CODES: ICHOOSE CODES
200 CODES$(A)=STR$(INT(RND*DIGITS))
210 IF B=0 THEN A=1: NO DUPLICATES
220 IF CODES$(A)=CODES$(B) THEN 200
230 NEXT B
240 NEXT A
250 ROW+=2
260 DISPLAY AT(1,1): ERASE ALL : "GUESS RIGHT PLACE"
270 DISPLAY AT(24,3): "ENTER 'I' FOR SOLUTION"
280 ACCEPT AT(ROW,1): G$'
290 IF G$='X' THEN 470: GIVE UP OR RESET
300 FOR D=1 TO CODES: BREAK
310 ENTER
Now run the program by typing RUN and pressing ENTER. Choose 4 codes with 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) possible in each code. Guessing the code in six tries is excellent. Finding it in eight is very good.

If you wish to use the program again, save it on diskette or cassette. To save it on cassette, make sure the cassette player is connected. Then enter SAVE CS1 and follow the instructions that appear on the screen.

To save the program on diskette, enter SAVE DSK1, filename with whatever filename you wish to use to save it, such as CODEBREAK.

After saving the program, or if you do not wish to save the program, enter NEW. The program is removed and you may enter another program.

If you have saved the program, you can easily reload it into the computer's memory for reuse or further editing. Reload the program from a cassette by entering OLD CS1 and then following the instructions that appear on the screen. Reload the program from diskette by entering OLD DSK1, filename using whatever filename you used to save it.

When you have finished using TI Extended BASIC, enter EYE to return to the master title screen.
This chapter discusses the format that TI Extended BASIC programs must take and the ways in which TI Extended BASIC functions.
RUNNING A PROGRAM ON POWERUP

If a program named LOAD is on the diskette in disk drive 1 when TI Extended BASIC is chosen, that program is loaded and run. The effect is the same as if you had entered RUN "DSK1:LOAD". If the program does not exist, there is a momentary delay while TI Extended BASIC looks for it.

FILES

Files are groups of data put on external devices. The most common files are on cassettes or diskettes, but data sent through external devices such as the RS232 Interface and the optional thermal printer are also considered to be files by TI Extended BASIC.

LINE NUMBERS

Line numbers are required in TI Extended BASIC programs. Line numbers specify the order in which lines are executed and are used to identify what lines to execute next when using IF-THEN-ELSE, GOTO, GOSUB, ON ERROR, ON...GOTO, and ON...GOSUB. Line numbers may also be used by BREAK, LIST, NUM RESTORE, RETURN, and RUN. Line numbers may be any integer from 1 through 32767.

The computer automatically generates line numbers if you issue the NUM command. When not followed by a line number, it provides line numbers starting at 100, incrementing each subsequent line by 10. You may resequence line numbers with the RES command.

LINES

Lines may be up to 140 characters long, including the line number and spaces. If you have reached the end of a line, additional characters you enter replace the 140th character. It is possible to make a line longer than 140 characters in the Edit Mode by the use of SHIF G (INSERT).

SPECIAL SYMBOLS

Special symbols separate statements and remarks on the same line. A line of TI Extended BASIC consists of a line number, one or more TI Extended BASIC statements, and an optional remark. For example:

100 FOR A = 1 TO 100: PRINT A: SQR(A): NEXT A: PRINT SQUARE ROOTS

The statement separator symbol, a double colon (::), is used to separate statements on the same line. The tail remark symbol, an exclamation point (!), is used to separate an explanatory remark from the rest of the line. Remarks are not executed when the program is run.

SPACES

Spaces are required in TI Extended BASIC between the elements that make up statements to enable the computer to distinguish variable names from TI Extended BASIC elements. However, spaces are not required before or after relational symbols or before or after the tail remark symbol or the statement separator symbol. You may insert extra spaces when inputting commands and statements, but they are deleted by TI Extended BASIC. When listing programs, TI Extended BASIC may add spaces around the tail remark symbol and statement separator symbol.

NUMERIC CONSTANTS

Numeric constants may be entered with any number of digits. However, they are rounded to 13 or 14 digits by the computer due to the internal storage method used by the computer, and are normally displayed as a maximum of 10 digits. For extremely large or small numbers, it is usually more convenient to use scientific notation to enter numbers. The computer normally uses scientific notation when printing very large or small numbers.

In scientific notation, a number is given as a mantissa (a number with one place to the left of the decimal point) times 10 raised to an integer power. 1.5 is expressed in scientific notation as 1.5 x 10^1. 150 is expressed as 1.5 x 10^2; -1.500 is expressed as -1.5 x 10^1; 156.769,000,000,000 is expressed as 1.56769 x 10^{11}; and 0.153789 is expressed as 1.53789 x 10^{-2}. In TI Extended BASIC, the " x 10^" is represented by "E". Thus 1.5 x 10^2 becomes 1.5E3.

Numeric constants are defined in the range -9.999999999999E127 to -1E-128. 0, and 1E-128 to 9.999999999999E127. If the exponent of a calculated number is greater than 99, then ** is normally printed or displayed as the power. The entire exponent is kept internally and can be displayed with a USING clause in a PRINT or DISPLAY statement.

STRING CONSTANTS

String constants in TI Extended BASIC can be up to one input line long. If the string is enclosed in quotation marks, quotation marks in the string are represented by double quotation marks.

VARIABLES

Variables in TI Extended BASIC may consist of one to 15 characters. The first character of a variable must be a letter of the alphabet, the at symbol (@), or an underscore (_). Subsequent characters may be those symbols plus any of the digits. The last character of a string variable must always be a dollar sign ($). Variables are either scalar or arrays with up to seven dimensions.
TI EXTENDED BASIC CONVENTIONS

Certain words are reserved for use by TI Extended BASIC. They are the commands, statements, functions, and operators that make up the language. These words may not be used as a variable name, but they may make up part of a variable name. The following is a complete list of the words reserved for TI Extended BASIC.

ABS  ELSE  NUMBER  SEQUENTIAL
ACCEPT ERASE NUMERIC SGN
ALL ERROR OLD SIN
AND EXP OPEN SIZE SQR
APPEND FIXED OPTION STEP
ASC FOR OR STOP
ATN GOSUB OUTPUT STRS
BASE GOTO PERMANENT SUB
BEep IF PI SUBEND
BREAK IMAGE POS SUBEXIT
BYE INPUT PRINT TAB
CALL INT RANDOMIZE TAN
CHR$ INTERNAL READ THEN
CLOSE LEN REC TO
CON LET RELATIVE TRACE
CONTINUE LINPUT REM UALPHA
COS LIST RES UNBREAK
DATA LOG RESEQUENCE UNTRACE
DEF MAX RESTORE UPDATE
DELETE MERGE RETURN USING
DIGIT MIN RND VAL
DIM NEW RPT$ VALIDATE
DISPLAY NEXT RUN VARIABLE
ELSE NOT SAVE WARNING
END NUM SEG$ XOR

The following are examples of valid variable names:
Numeric: X A9, ALHAN, BASE__PAY V(3), T(X,Y,Z,Q,A,R,P6), TABLE(Q37,M4)
String: S8, Y28, NAMES, Q$8(X,7,L/2), ADDRESS8(4)

NUMERIC EXPRESSIONS

Numeric expressions are constructed from numeric constants, numeric variables, and functions using the arithmetic operators for addition (+), subtraction (-) multiplication (*), division (/), and exponentiation (^).

The minus sign (-) can be used either to indicate subtraction or as a unary minus. As a unary minus, it reverses the sign of what follows it. For example, -5*2 is equal to -10 as it is taken to mean -5*(2).

The normal hierarchy for evaluating a numeric expression is exponentiation, followed by multiplication and division, and then by addition and subtraction. However, any part of a numeric expression that is enclosed in parentheses is evaluated first. The following shows the effect of parentheses on determining the value of an expression:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Intermediate Results</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 + 2^2 / 2 - 6</td>
<td>4 + 4 / 2 - 6</td>
<td>4 + 2 - 6</td>
</tr>
<tr>
<td>(4 + 2) ^ 2 / 2 - 6</td>
<td>6 ^ 2 / 2 - 6</td>
<td>33 / 2 - 6</td>
</tr>
<tr>
<td>4 + 2^2 / (2 - 6)</td>
<td>4 + 4/(-4)</td>
<td>4 - 1</td>
</tr>
</tbody>
</table>

STRING EXPRESSIONS

String expressions are constructed from string variables, string constants, and function references using the operation for concatenation (&) to combine strings. If a constructed string exceeds a length of 255 characters, the extra characters on the right are truncated and a warning message is issued. The following is an example of concatenation:

100 AS="HI" & " THERE!"

AS=""HI"" & " THERE!" sets AS equal to "HI THERE!".

RELATIONAL EXPRESSIONS

Relational expressions are most often used in the IF-THEN-ELSE statement, but may be used anywhere that numeric expressions are allowed. A relational expression has a value of -1 if it is true and a value of 0 if it is false. Relational operations are performed from left to right, after all arithmetic operations are completed and before string concatenation (the ampersand operator). The relational expressions are:

Equal to (=)  Not equal to (<>)
Less than (<)  Less than or equal to (<=)
Greater than (> ) Greater than or equal to (>=)
TI EXTENDED BASIC CONVENTIONS

The following examples illustrate the use of relational expressions:

\[
\text{IF } X < Y \text{ THEN } 200 \text{ ELSE GOSUB 420 next executes Inc 200 if } X \text{ is less than } Y. \text{ If } X \text{ is greater than or equal to } Y, \text{ then the statement GOSUB 420 is executed.}
\]

\[
\text{IF } L(C) = 12 \text{ THEN } C = S + 1 \text{ ELSE } > 100 \text{ IF } L(C) = 12 \text{ THEN } C = S + 1 \text{ ELSE COUNT = COUNT + 1: GOTO 140 sets LSE COUNT = COUNT + 1: GOTO 140}
\]

\[
\text{C equal to } S \text{ plus 1 if } L(C) \text{ equals } 12. \text{ If } L(C) \text{ is not equal to } 12, \text{ then COUNT is set equal to COUNT plus 1 and line 140 is executed next.}
\]

\[
A - 2 < 5 \text{ sets } A \text{ equal to } -1 \text{ as it is } > 100 \text{ A = 2 < 5 true that } 2 \text{ is less than } 5.
\]

\[
\text{PRINT "THIS" = "THAT" prints } 0 \text{ as } > 100 \text{ PRINT "THIS" = "THAT" it is not true that "THIS" is equal to "THAT".}
\]

\[
A = B - 7 \text{ sets } A \text{ equal to } -1 \text{ if } B \text{ is } > 100 \text{ A = B - 7 equal to } 7, \text{ and to } 0 \text{ if } B \text{ is not equal to } 7. \text{ There is no effect on } B. \text{ Note that this is not the same as the usual arithmetical meaning of } A = B = 7.
\]

LOGICAL EXPRESSIONS

Logical expressions are used with relational expressions. The logical operators are AND, OR, NOT, and XOR. If true, logical expressions are given a value of -1. If false, they are given a value of 0. The order of precedence for logical expressions, from highest to lowest, is NOT, XOR, AND, and OR.

A logical expression using AND is true if both its left and right clauses are true.

A logical expression using OR is true if either its left clause is true, its right clause is true, or both its left and right clauses are true.

A logical expression using NOT is true if the clause following it is not true.

A logical expression using XOR (exclusive or) is true if either its left or its right clause is true, but not both its left and right clauses are true.

The following examples illustrate the use of logical expressions:

\[
\text{IF } 3 < 4 \text{ AND } 5 < 6 \text{ THEN } L = 7 \text{ sets } L = > 100 \text{ IF } 3 < 4 \text{ AND } 5 < 6 \text{ THEN } L = 7 \text{ equal to } 7 \text{ since } 3 \text{ is less than } 4 \text{ and } 5 \text{ is less than } 6.
\]

\[
\text{IF } 3 < 4 \text{ AND } 5 > 6 \text{ THEN } L = 7 \text{ does not set } L \text{ equal to } 7 \text{ because } 3 \text{ is less than } 4, \text{ but } 5 \text{ is not greater than } 6.
\]

\[
\text{IF } 3 < 4 \text{ OR } 5 > 6 \text{ THEN } L = 7 \text{ sets } L = > 100 \text{ IF } 3 < 4 \text{ OR } 5 > 6 \text{ THEN } L = 7 \text{ equal to } 7 \text{ because } 3 \text{ is less than } 4, \text{ but } 5 \text{ is not greater than } 6.
\]

\[
\text{IF } 3 < 4 \text{ XOR } 5 > 6 \text{ THEN } L = 7 \text{ sets } L = > 100 \text{ IF } 3 < 4 \text{ XOR } 5 > 6 \text{ THEN } L = 7 \text{ equal to } 7 \text{ because } 3 \text{ is less than } 4, \text{ but } 5 \text{ is not greater than } 6.
\]

\[
\text{IF } 3 < 4 \text{ AND } 3 = 4 \text{ THEN } L = 7 \text{ sets } L = > 100 \text{ IF } 3 < 4 \text{ NOT } 3 = 4 \text{ THEN } L = 7 \text{ equal to } 7 \text{ because } 3 \text{ is not equal to } 4.
\]

\[
\text{IF } 3 = 4 \text{ AND } (\text{NOT } 6 = 5 \text{ XOR } 2 = 2) \text{ THEN } 200 \text{ does not pass control to line } 200 \text{ because while it is true that } 3 \text{ is not equal to } 4, \text{ it is true that both } 6 \text{ is not equal to } 5 \text{ and } 2 \text{ is equal to } 2, \text{ so the clause in parentheses is not true.}
\]

\[
\text{IF } (A \text{ OR } B) \text{ AND } (C \text{ XOR } D) \text{ THEN } 200 \text{ passes control to line } 200 \text{ if either } A \text{ or } B \text{ or both } A \text{ and } B \text{ are true (equal to -1), and } C \text{ or } D, \text{ but not both } C \text{ and } D \text{ are true (equal to -1).}
\]

The logical operators can also be used directly on numbers. They convert the numbers to binary notation, perform the designated operation on a bit level, and then convert the result back to decimal representation. A more detailed discussion of the use of logical operators with numbers can be found in a mathematics or engineering text dealing with logic.

The numbers must be from -32,768 to 32,767, represented in binary notation as from 10000000000000000 to 0111111111111111, with negative numbers given in 2's complement form signified by a 1 in the most significant bit. In binary notation, each place is an additional power of 2 rather than an additional power of 10 as in decimal notation. The following shows numbers in both decimal and binary notation.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>-32,768</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>32,767</td>
<td>0101111111111111</td>
</tr>
</tbody>
</table>

TI Extended BASIC
TI EXTENDED BASIC CONVENTIONS

The above is the equivalent to

\[ 1_{10} = 00000000000000012 = 1 \]

\[ 25_{10} = 0000000000110012 = 110012 \]

\[ 6_{10} = 00000000000000112 = 110_{2} \]

\[ -13_{10} = 1111111111111111_{2} = \overline{1101}_{2} \]

AND places a 1 in the corresponding binary position if there is a 1 in both the number preceding and following it. Otherwise it places a zero.

OR places a 1 in the corresponding binary position if there is a 1 in either the number preceding or following it or both. Otherwise it places a zero.

XOR places a 1 in the corresponding binary position if there is a 1 in either the number preceding it or following it but not both. Otherwise it places a zero.

NCT places a 1 in the corresponding binary position if there is a zero in the number following it. Otherwise it places a zero.

The following illustrate the result of the logical operators when used on numbers.

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>BINARY</th>
<th>DECIMAL</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td>000000000000001</td>
<td>A: 1</td>
<td>000000000000001</td>
</tr>
<tr>
<td>B: 2</td>
<td>000000000000010</td>
<td>B: 3</td>
<td>000000000000011</td>
</tr>
<tr>
<td>A AND B: 0</td>
<td>000000000000000</td>
<td>A AND B: 1</td>
<td>000000000000001</td>
</tr>
<tr>
<td>A: 6</td>
<td>000000000000110</td>
<td>A: 47</td>
<td>000000000001111</td>
</tr>
<tr>
<td>B: 5</td>
<td>000000000000101</td>
<td>B: 62</td>
<td>000000000011110</td>
</tr>
<tr>
<td>A AND B: 4</td>
<td>000000000000100</td>
<td>A AND B: 46</td>
<td>000000000011110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>BINARY</th>
<th>DECIMAL</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td>000000000000001</td>
<td>A: 1</td>
<td>000000000000001</td>
</tr>
<tr>
<td>B: 2</td>
<td>000000000000010</td>
<td>B: 3</td>
<td>000000000000011</td>
</tr>
<tr>
<td>A OR B: 3</td>
<td>000000000000011</td>
<td>A OR B: 3</td>
<td>000000000000011</td>
</tr>
<tr>
<td>A: 6</td>
<td>000000000000110</td>
<td>A: 47</td>
<td>000000000001111</td>
</tr>
<tr>
<td>B: 5</td>
<td>000000000000101</td>
<td>B: 62</td>
<td>000000000011110</td>
</tr>
<tr>
<td>A OR B: 7</td>
<td>000000000000111</td>
<td>A OR B: 63</td>
<td>000000000111111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>BINARY</th>
<th>DECIMAL</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td>000000000000001</td>
<td>A: 1</td>
<td>000000000000001</td>
</tr>
<tr>
<td>B: 2</td>
<td>000000000000010</td>
<td>B: 3</td>
<td>000000000000011</td>
</tr>
<tr>
<td>A XOR B: 3</td>
<td>000000000000011</td>
<td>A XOR B: 2</td>
<td>000000000000010</td>
</tr>
<tr>
<td>A: 6</td>
<td>000000000000110</td>
<td>A: 47</td>
<td>000000000001111</td>
</tr>
<tr>
<td>B: 5</td>
<td>000000000000101</td>
<td>B: 62</td>
<td>000000000011110</td>
</tr>
<tr>
<td>A XOR B: 3</td>
<td>000000000000011</td>
<td>A XOR B: 17</td>
<td>000000000010001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>BINARY</th>
<th>DECIMAL</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td>000000000000001</td>
<td>A: 1</td>
<td>000000000000001</td>
</tr>
<tr>
<td>NOT A: -2</td>
<td>1111111111111111</td>
<td>NOT A: -1</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>A: 8</td>
<td>000000000000001</td>
<td>A: 47</td>
<td>000000000001111</td>
</tr>
<tr>
<td>NOT A: -7</td>
<td>1111111111111111</td>
<td>NOT A: -48</td>
<td>1111111111111111</td>
</tr>
</tbody>
</table>

This chapter is an alphabetical list of all of the TI Extended BASIC commands, statements, and functions, with a detailed explanation of how each works. Examples and sample programs are included wherever necessary for clarity.

In the format of the elements, key words are CAPITALIZED. Variables are in italics. Optional portions are enclosed in [brackets]. Items that may be repeated are indicated by ellipses (...). Alternative forms are presented one above the other.

Appendix A contains a list of the illustrative programs. The Index gives the pages on which each TI Extended BASIC element is used in an illustrative program.
ABS

Format

ABS(numeric-expression)

Description

The ABS function gives the absolute value of numeric-expression. If
numeric-expression is positive, ABS gives the value of numeric expression. If
numeric-expression is negative, ABS gives its negative (a positive number). If
numeric-expression is zero, ABS returns zero. The result of ABS is always a
non-negative number.

Examples

PRINT ABS(42.3) prints 42.3.  
>100 PRINT ABS(42.3)

V1 = ABS(-6.124) sets V1 equal to
6.124.  
>100 V1 = ABS(-6.124)

ACCEPT

Format

ACCEPT [AT(row.column)] [VALIDATE(data-type ...)] [BEEP]
[ERASE ALL] [SIZE(numeric-expression)] ; variable

Description

The ACCEPT statement suspends program execution until data is entered
from the keyboard. Many options are available with ACCEPT, making it far
more versatile than INPUT. It may accept data at any screen position, make
an audible tone (bEEP) when ready to accept the data, erase all characters on
the screen before accepting data, limit data accepted to a certain number of
characters, and limit the type of characters accepted.

Options

The following options may appear in any order following ACCEPT.

AT(row.column) places the beginning of the input field at the specified row
and column. Rows are numbered 1 through 24. Columns are numbered 1
through 80 with column 1 corresponding to what is called column 3 in the
VCHAR, HCHAR, and GCHAR subprograms.

VALIDATE(data-type ...) allows only certain characters to be entered. Data-
type specifies which characters are acceptable. If more than one data-type is
specified, a character from any of the data-types given is acceptable. The
following are the data-types.

- UALPHA permits all uppercase alphabetic characters
- DIGIT permits 0 through 9
- NUMERIC permits 0 through 9, "", "", "+", "+", and "E"
- String-expression permits the characters contained in string-
  expression.

BEEP sounds a short tone to signal that the computer is ready to accept
input.

ERASE ALL fills the entire screen with the blank character before accepting
input.

SIZE(numeric-expression) allows up to the absolute value of numeric-
expression characters to be input. If numeric-expression is positive, the field
in which the data is entered is cleared before input is accepted. If numeric-
expression is negative, the input field is not blanked. This allows a default
value to be previously placed in the field and entered by just pressing ENTER.
If there is no SIZE clause, the line is blanked from the beginning position to
the end of the line.

If the ACCEPT statement is used without the AT clause, the last two
characters on the screen (at the lower right) are changed to "edge
characters" (ASCII code 31).
ACCEPT

Examples

ACCEPT AT(5,7):Y accepts data at the fifth row, seventh column of the screen into the variable Y.

ACCEPT VALIDATE("YN"):R$ accepts Y or N into the variable R$.

ACCEPT ERASE ALL:B accepts data into the variable B after putting the blank character into all screen positions.

ACCEPT AT(R,C):SIZE(FIELDLEN) BEEP VALIDATE(DIGIT,"AYN"):X$ accepts a digit or the letters A, Y, or N into the variable X$. The length of the input may be up to FIELDLEN characters. The data is accepted at row R, column C, and a beep is sounded before data is accepted.

Program

The program at the right illustrates a typical use of ACCEPT. It allows entry of up to 20 names and addresses, and then displays them all.

>100 ACCEPT AT(5,7):Y
>100 ACCEPT VALIDATE("YN"):R$
>100 ACCEPT ERASE ALL B
>100 ACCEPT AT(R,C):SIZE(FIELD LEN) BEEP VALIDATE(DIGIT,"AYN "):X$
>190 NEXT S
>200 CALL CLEAR
>210 DISPLAY AT(1,1):"NAME"," ADDRESS"
>220 FOR T=1 TO S-1
>230 DISPLAY AT(3+2,T):NAME$(T),ADDR$(T)
>240 NEXT T
>250 GOTO 250
(Press SHIFT C to stop the program.)
ASC

**Format**

`ASC(string-expression)`

**Description**

The ASC function gives the ASCII character code which corresponds to the first character of `string-expression`. A list of the ASCII codes is given in Appendix C. The ASC function is the inverse of the CHR$ function.

**Examples**

```plaintext
PRINT ASC(“A”); prints 65.
B = ASC(“1”); sets B equal to 49
DISPLAY ASC(“HELLO”); displays 72.
```

ATN

**Format**

`ATN(numeric-expression)`

**Description**

The ATN function returns the measure of the angle (in radians) whose tangent is `numeric-expression`. If you want the equivalent angle in degrees, multiply by 180/PI. The value given by the ATN function is always in the range -PI/2 < ATN(X) < PI/2.

**Examples**

```plaintext
PRINT ATN(0); prints 0.
Q = ATN(.44); sets Q equal to .4145068746.
```

>100 PRINT ASC(“A”)

>100 B = ASC(“1”)

>100 DISPLAY ASC(“HELLO”)

>100 PRINT ATN(0)

>100 Q = ATN(.44)
Format
BREAK [line-number-list]

Description
The BREAK command requires a line-number-list. It causes the program to stop immediately before the lines in line-number-list are executed. After a breakpoint is taken because the line is listed in line-number-list, the breakpoint is removed and no more breakpoints occur at that line unless a new BREAK command or statement is given.

The BREAK statement without line-number-list causes the program to stop when it is encountered. The line at which the program stops is called a breakpoint. Every time a BREAK statement without line-number-list is encountered, the program stops even if an ON BREAK NEXT statement has been executed.

You can also cause a breakpoint in a program by pressing SHIFT C (CLEAR) while the program is running, unless breakpoints are being handled in some other way because of the action of ON BREAK.

BREAK is useful in finding out why a program is not running exactly as you expect it to. When the program has stopped you can print values of variables to find out what is happening in the program. You may enter any command or statement that can be used as a command. If you edit the program, however, you cannot resume with CONTINUE.

A way to remove breakpoints set with BREAK followed by line numbers is the UNBREAK command. Also, if a breakpoint is set at a program line and that line is deleted, the breakpoint is removed. Breakpoints are also removed when a program is saved with the SAVE command. See ON BREAK for a way to handle breakpoints.

Whenever a breakpoint occurs, the standard character set is restored. Thus any standard characters that had been redefined by CALL CHAR are restored to the standard characters. A breakpoint also restores the standard colors, deletes sprites, and resets sprite magnification to the default value of 1.

Options
The line-number-list is optional when BREAK is used as a statement, but is required when BREAK is used as a command. When present, it causes the program to stop immediately before the lines in line-number-list are executed. After a breakpoint is taken because the line is listed in line-number-list, the breakpoint is removed and no more breakpoints occur at that line unless a new BREAK command or statement is given.

Examples
BREAK as a statement causes a breakpoint when that statement is executed.

BREAK 120,130 as a statement causes breakpoints before execution of the line numbers listed.

BREAK 200,300,1105 as a command causes breakpoints before execution of the line numbers listed.
**BYE**

**Format**

BYE

**Description**

The BYE command ends TI Extended BASIC and returns the computer to the master title screen. All open files are closed, all program lines are erased, and the computer is reset. Always use the BYE command instead of SHIFT Q (QUIT) to leave TI Extended BASIC. SHIFT Q (QUIT) does not close files, which may result in data being lost from external devices.

---

**CALL**

**Format**

CALL subprogram-name? [(parameter-list)]

**Description**

The CALL statement transfers control to subprogram-name. The subprogram may be either one built into TI Extended BASIC, such as CLEAR, or one you have written. After the subprogram is executed, the next statement after the CALL statement is executed. CALL may be either a statement or a command for calling built-in TI Extended BASIC subprograms, but must be a statement when calling subprograms that you write.

**Options**

The parameter-list is defined according to the subprogram you are calling. Some require no parameters at all, some require parameters, and some have optional parameters. Each built-in subprogram is discussed under its own entry in this manual. The subprograms you can write are discussed in the section in Chapter II on subprograms and under SUB. The following are the subprogram-names of the built-in TI Extended BASIC subprograms:

- CHAR
- CHARFAT
- CHARSET
- CLEAR
- COINC
- COLOR
- DELSPRITE
- DISTANCE
- ERR
- GCHAR
- HCHAR
- PATTERN
- PEEK
- POSITION
- SAY
- SCREEN
- SOUND
- SFGET
- SFRITE
- VCHAR
- VERSION

**Program**

The program at the right illustrates the use of CALL with a supplied subprogram (CLEAR) in line 100 and the use of a written subprogram (TIMES) in line 120.

```
>100 CALL CLEAR
>110 X=4
>120 CALL TIMES(X)
>130 PRINT X
>140 STOP
>200 SUB TIMES(Z);
>210 Z=Z*PI
>220 SUBEND
>RUN
--screen clears
12.56637061
```
CHAR subprogram

Format
CALL CHAR(character-code, pattern-identifier) [... ]

Description
The CHAR subprogram allows you to define special graphics characters. You can redefine the standard set of characters (ASCII codes 32-95) and the undefined characters. ASCII codes 96-143. Note that fewer program defined characters are available in TI Extended BASIC than in TI BASIC, where ASCII codes 96-156 are allowed. The CHAR subprogram is the inverse of the CHARPAT subprogram.

Character-code specifies the character which you wish to define and must be a numeric expression with a value from 32 through 143. Pattern-identifier is a 0 through 64 character string expression which specifies the pattern of the character(s) you are defining. This string expression is a coded representation of the dots which make up a character on the screen.

Each character is made up of 64 dots comprising an 8 by 8 grid as shown below.

Each row is partitioned into two blocks of four dots each:

Any row

| LEFT BLOCKS | RIGHT BLOCKS |
| ROW 1       |             |
| ROW 2       |             |
| ROW 3       |             |
| ROW 4       |             |
| ROW 5       |             |
| ROW 6       |             |
| ROW 7       |             |
| ROW 8       |             |

If the pattern-identifier is less than 16 characters, the computer assumes that the remaining characters are zeros. If the pattern-identifier is 17 to 32 characters, two character-codes are defined, the first with the first through sixteenth characters and the second with the remaining characters, with zeros added as needed. If the pattern-identifier is 33 to 48 characters, three character-codes are defined, the first with the first through sixteenth characters, the second with the seventeenth through thirty-second characters, and the third with the remaining characters, with zeros added as needed. If the pattern-identifier is 49 to 64 characters, four character-codes are defined, the first with the first through sixteenth characters, the second with the seventeenth through thirty-second characters, the third with the thirty-third through forty-eighth characters, and the fourth with the remaining characters, with zeros added as needed. If the pattern-identifier is longer than 64 characters or is long enough to define characters higher than character code 143, the excess is ignored.

<table>
<thead>
<tr>
<th>Binary Code</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

Characters are created by turning some dots "on" and leaving others "off." The space character (ASCII code 32) is a character with all the dots turned "off." Turning all the dots "on" produces a solid block. The color of the on dots is the foreground color. The color of the off dots is the background color.

All the standard characters are set with the appropriate dots on. To create a new character, you specify what dots to turn on and leave off. In the computer a binary code, one number for each of the 64 dots, is used to specify which dots are on and off in a particular block. A more human-readable form of binary is hexadecimal. The following table shows all the possible on/off conditions for the four dots in a given block, and the binary and hexadecimal codes for each condition.

If the pattern-identifier is less than 16 characters, the computer assumes that the remaining characters are zeros. If the pattern-identifier is 17 to 32 characters, two character-codes are defined, the first with the first through sixteenth characters and the second with the remaining characters, with zeros added as needed. If the pattern-identifier is 33 to 48 characters, three character-codes are defined, the first with the first through sixteenth characters, the second with the seventeenth through thirty-second characters, and the third with the remaining characters, with zeros added as needed. If the pattern-identifier is 49 to 64 characters, four character-codes are defined, the first with the first through sixteenth characters, the second with the seventeenth through thirty-second characters, the third with the thirty-third through forty-eighth characters, and the fourth with the remaining characters, with zeros added as needed. If the pattern-identifier is longer than 64 characters or is long enough to define characters higher than character code 143, the excess is ignored.
CHAR SUBPROGRAM

Programs
To describe the dot pattern pictured below, you code this string for CALL CHAR:

"1898FF3D3C3CE404"

<table>
<thead>
<tr>
<th>LEFT BLOCKS</th>
<th>RIGHT BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>98</td>
</tr>
<tr>
<td>FF</td>
<td>3D</td>
</tr>
<tr>
<td>3C</td>
<td>3C</td>
</tr>
<tr>
<td>E4</td>
<td>04</td>
</tr>
</tbody>
</table>

ROW 1
ROW 2
ROW 3
ROW 4
ROW 5
ROW 6
ROW 7
ROW 8

The program at the right uses this and one other string to make a figure "dance."

If a program stops for a breakpoint, the predefined characters (ASCII codes 32 through 95) are reset to their standard pattern. Those with codes 96 through 143 keep their program-defined pattern. When the program ends normally or because of an error, all predefined characters are reset.

>100 CALL CLEAR
>110 A$="1898FF3D3C3CE404"
>120 B$="1819FFBC3C3CZ20"
>130 CALL COLOR(9,7,12)
>140 CALL VCHAR(12,16,96)
>150 CALL CHAR(96,A$)
>160 GSUB 200
>170 CALL CHAR(96,B$)
>180 GSUB 200
>190 GOTO 150
>200 FOR DELAY=1 TO 50
>210 NEXT DELAY
>220 RETURN

-- screen clears
-- character moves
(Press SHIFT C to stop the program.)

>100 CALL CLEAR
>110 CALL CHAR(96,"FFFFFFFFFF"
>120 CALL CHAR(42,"FOFOFOFO"
>130 CALL HCHAR(12,17,42)
>140 CALL VCHAR(14,17,96)
>150 FOR DELAY=1 TO 500
>160 NEXT DELAY
>RUN

CHARPAT subprogram

Format
CALL CHARPAT(character-code,string-variable [,...])

Description
The CHARPAT subprogram returns in string-variable the 16-character pattern identifier that specifies the pattern of character-code. The CHARPAT subprogram is the inverse of the CHAR subprogram. See the CHAR subprogram for an explanation of the value returned in string-variable.

Example
CALL CHARPAT(33,C$) sets C$ equal to "0010101010001000", the pattern identifier for character 33, the exclamation point.
 CHARSET subprogram

**Format**
CALL CHARSET

**Description**
The CHARSET subprogram restores the standard character patterns and standard colors for characters 32 through 95. Normally when a program is run by another program using RUN as a statement, characters 32 through 95 are not reset to their standard patterns and colors. CHARSET is useful when this feature is not desired.

**Example**
CALL CHARSET restores the standard characters and their colors.

>100 CALL CHARSET

CHR$ function

**Format**
CHR$(numeric-expression)

**Description**
The CHR$ function returns the character corresponding to the ASCII character code specified by numeric-expression. The CHR$ function is the inverse of the ASC function. A list of the ASCII character codes for each character in the standard character set is given in Appendix C.

**Examples**
PRINT CHR$(72) prints H.

>100 PRINT CHR$(72)

X8 = CHR$(33) sets X8 equal to !.

>100 X8 = CHR$(33)

**Program**
For a complete list of all ASCII characters and their corresponding ASCII values, run the program on the right.

>100 CALL CLEAR
>110 FOR A = 32 TO 95
>120 PRINT A; "", CHR$(A); ""
>130 NEXT A

CLEAR subprogram

**Format**
CALL CLEAR

**Description**
The CLEAR subprogram is used to clear (erase) the entire screen. When the CLEAR subprogram is called, the space character (ASCII code 32) is placed in all positions on the screen.

**Programs**
When the program at the right is run, the screen is cleared before the PRINT statements are performed.

>100 CALL CLEAR
>110 PRINT "HELLO THERE!"
>120 PRINT "HOW ARE YOU?"
>130 RUN
--screen clears
HELLO THERE!
HOW ARE YOU?

If the space character (ASCII code 32) has been redefined by the CALL CHAR subprogram, the screen is filled with the new character when CALL CLEAR is performed.

>100 CALL CHAR(32,"010307CF1F3F7FF")
>110 CALL CLEAR
>120 GOTO 120
>130 RUN
--screen is filled with 4
(Press SHIFT C to stop the program.)
CLOSE

Format
CLOSE [file-number] [:DELETE]

Description
The CLOSE statement stops a program’s use of the file referenced by file-number. After the CLOSE statement is performed, the file cannot be used by the program unless you OPEN it again. The computer no longer associates the file-number with the closed file, so you can assign that number to another file.

When no program is running, the following actions close all open files:
   Editing the program
   Entering the BYE command
   Entering the RUN command
   Entering the NEW command
   Entering the OLD command
   Entering the SAVE command
   Entering the LIST command to a device

If you use SHIFT Q (QUIT) to leave TI Extended BASIC, the computer does not close any open files, and you may lose data on any files that are open. To avoid this possibility, you should leave TI Extended BASIC with BYE instead of SHIFT Q (QUIT).

Options
You may delete a diskette file at the same time you close it by adding “:DELETE” to the statement. Other devices, such as cassette recorders, do not allow DELETE. The manual for each device discusses the use of DELETE.

Examples
When the computer performs the CLOSE statement for a cassette tape recorder, you receive instructions for operating the recorder.

>100 OPEN #24: "CSI", INTERNAL, INPUT, FIXED
   --program lines
   --program runs

>200 CLOSE #24
>RUN
   --opening instructions
   --program runs

* PRESS CASSETTE STOP
>100 OPEN #24: "DSK1.MYDATA", INTERNAL, INPUT, FIXED
   --program lines
   --program runs

>200 CLOSE #24
>RUN
   --program runs

The CLOSE statement for a diskette requires no further action on your part.
COINC subprogram

Format
CALL COINC(#sprite-number,#sprite-number,tolerance,numeric-variable}
CALL COINC([#sprite-number.dot-row.dot-column.tolerance,numeric-variable]
CALL COINC(ALL,numeric-variable)

Description
The COINC subprogram detects a coincidence between a sprite and another
sprite or a position on the screen. The value returned in numeric-variable is
-1 if there is a coincidence and 0 if there is no coincidence.

If the keyword ALL is given, the coincidence of any two sprites is reported. If
two sprites are identified by #sprite-number, their coincidence is reported. If
#sprite-number and a location are identified, their coincidence is reported.

If the keyword ALL is given, sprites are coincident only if one or more of the
dots which make them up occupy the same position on the screen. If two
sprites or a sprite and a location are given, then tolerance must be specified,
and two sprites are coincident if their upper left hand corners are within the
value specified by tolerance. A sprite and a location are coincident if the
upper left hand corner of the sprite and the position specified by dot-row and
dot-column are within the value specified by tolerance. These coincidents are
reported even if there is no apparent overlap of the sprites or the sprite and
the position.

Dot-row and dot-column are numbered consecutively starting with 1 in the
upper left hand corner of the screen. Thus the dot-row can be from 1 to 192
and the dot-column can be from 1 to 256. (Actually the dot-row can go up to
256, but the positions from 193 through 256 are off the bottom of the
screen.) If any part of the sprite occupies the position given, then there is a
coincidence.

Whether or not a coincidence is detected depends on several variables. If the
sprites are moving very quickly, COINC may not be able to detect their
coincidence. Also, COINC checks for a coincidence only when it is called, so a
program may miss a coincidence that occurs when the program is executing
some other statement.

Program
The program at the right defines two
sprites that consist of a triangle.

Line 160 shows a coincidence
because the sprites are within 10
dots of each other.
Line 160 shows no coincidence
because the shaded areas of the
sprites are not coincident.

>100 CALL CLEAR
>110 $S$="0103070F1F3F7FFFF"
>120 CALL CHAR(96,$S$
>130 CALL CHAR(100,$S$
>140 CALL SPRITE(#1,96,7,8,8
>150 CALL SPRITE(#2,100,5,1,1
>160 CALL COINC(#1,#2,10,C)
>170 PRINT C

>180 CALL COINC(ALL,C)
>190 PRINT C
>RUN
-1
0
COLOR subprogram

CALL COLOR(#sprite-number,foreground-color [,...])
CALL COLOR(character-set,foreground-color,background-color [,...])

Description

The COLOR subprogram allows you to specify either a foreground-color for #sprite-number or a foreground-color and background-color for characters in the character-set. In a given CALL COLOR, you may define sprite color(s) or character set colors, but not both.

Each character has two colors. The color of the dots that make up the character itself is called the foreground-color. The color that occupies the rest of the character position on the screen is called the background-color. In sprites, the background-color is always code 1, transparent, which allows characters and the screen color to show through. To change the screen color, see the SCREEN subprogram. Foreground-color and background-color must have values from 1 through 15. The color codes are shown below:

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Transparent</td>
</tr>
<tr>
<td>2</td>
<td>Black</td>
</tr>
<tr>
<td>3</td>
<td>Medium Green</td>
</tr>
<tr>
<td>4</td>
<td>Light Green</td>
</tr>
<tr>
<td>5</td>
<td>Dark Blue</td>
</tr>
<tr>
<td>6</td>
<td>Light Blue</td>
</tr>
<tr>
<td>7</td>
<td>Dark Red</td>
</tr>
<tr>
<td>8</td>
<td>Cyan</td>
</tr>
<tr>
<td>9</td>
<td>Medium Red</td>
</tr>
<tr>
<td>10</td>
<td>Light Red</td>
</tr>
<tr>
<td>11</td>
<td>Dark Yellow</td>
</tr>
<tr>
<td>12</td>
<td>Light Yellow</td>
</tr>
<tr>
<td>13</td>
<td>Dark Green</td>
</tr>
<tr>
<td>14</td>
<td>Magenta</td>
</tr>
<tr>
<td>15</td>
<td>Gray</td>
</tr>
<tr>
<td>16</td>
<td>White</td>
</tr>
</tbody>
</table>

Example Codes

CALL COLOR(3,5,8) sets the foreground-color of characters 48 through 55 to 5 (dark blue) and the background-color to 8 (cyan).

CALL COLOR(#5,16) sets sprite number 5 to have a foreground-color of 16 (white). The background-color is always 1 (transparent).

CALL COLOR(#7,INT(RND*16 +1)) sets sprite number 7 to have a foreground-color chosen randomly from the 16 colors available. The background color is 1 (transparent).
CONTINUE

Format
CONTINUE
CON

Description
The CONTINUE command restarts a program which has been stopped by a breakpoint. It may be entered whenever a program has stopped running because of a breakpoint caused by the BREAK command or statement or SHIFT C (CLEAR). However, you cannot use the CONTINUE command if you have edited a program line. CONTINUE may be abbreviated as CON.

When a breakpoint occurs, the standard character set and standard colors are restored. Sprites cease to exist. CONTINUE does not restore standard characters that have been reset or any colors. Otherwise, the program continues as if no breakpoint had occurred.

COS

Format
COS(radian-expression)

Description
The cosine function gives the trigonometric cosine of radian-expression. If the angle is in degrees, multiply the number of degrees by π/180 to get the equivalent angle in radians.

Program
The program on the right gives the cosine of several angles.

>100 A=1.04719755196
>110 B=60
>120 C=45*PI/180
>130 PRINT COS(A);COS(B)
>140 PRINT COS(B*PI/180)
>150 PRINT COS(C)
>160 PRINT COS(.5)
>170 PRINT COS(.7071067812)
DATA

Format

DATA data-list

Description

The DATA statement allows you to store data inside your program. The data, which may be numeric or string constants, is listed in data-list separated by commas. During program execution, the READ statement assigns the values in data-list to the variables specified in variable-list in the READ statement.

DATA statements may be located anywhere in a program. However, the order in which they appear is important. Data from several DATA statements is read sequentially, beginning with the first item in the first DATA statement. If a program has more than one DATA statement, the DATA statements are read in the order in which they appear in the program, unless otherwise specified by a RESTORE statement. Thus the order in which data appears in the program normally determines the order in which data is read.

DATA statements cannot be part of multiple statement lines.

Data in data-list must correspond to the type of the variable to which it is assigned in the READ statement. Thus if a numeric variable is specified in the READ statement, a numeric constant must be in the corresponding position in the DATA statement. Similarly, if a string variable is specified, a string constant must be supplied. A number is a valid string, so you may have a numeric constant in a DATA statement where a string is called for in the READ statement. If a DATA statement contains adjacent commas, the computer assumes you want to enter a null string (a string with no characters).

When using string constants in a DATA statement, you may enclose the string in quotes. However, if the string you include contains a comma, leading spaces, or trailing spaces, you must enclose the string in quotes. If the string is enclosed in quotes, quotes in the string are represented by double quotes.

Program

The program at the right reads and prints several numeric and string constants. Lines 100 through 130 read five sets of data and print their values, two to a line.

Lines 190 through 220 read seven data elements and print each on its own line.

First two elements of line 140. 2 4
Second two elements of line 140. 5 7
Last element of line 140 and first of line 150. 8 1
Second and third elements of line 150. 2 3
Fourth and fifth elements of line 150. 4 5
Line 160. "THIS HAS QUOTES"
Line 170. "THIS HAS QUOTES"
Line 180. "NO QUOTES, HERE"
First element of line 230. 1
Second element of line 230. NUMBER
Null string for two commas in line 230.
Last element of line 230. TI

> :00 FOR A = 1 TO 5
> :10 READ C
> :20 PRINT B
> :30 NEXT A
> :140 DATA 2, 4, 6, 7, 8
> :150 DATA 1, 2, 3, 4, 5
> :160 DATA """"THIS HAS QUOTES"
> :170 DATA "NO QUOTES, HERE"
> :180 DATA "NO QUOTES HERE EITHER"
> :190 FOR A = 1 TO 7
> :200 READ B
> :210 PRINT B
> :220 NEXT A
> :230 DATA 1, NUMBER, :TI

>RUN
**Format**

DEF function-name [parameter] = expression

**Description**

The DEF statement allows you to define your own functions. Function-name may be any variable name. If you specify a parameter following function-name, the parameter must be enclosed in parentheses and may be any scalar variable name. If expression is a string, function-name must be a string variable name, i.e. the last character must be a dollar sign.

The DEF statement must occur at a lower numbered line than any reference to the function it defines. However, a DEF statement may not appear in an IF-THEN-ELSE statement. When the computer encounters a DEF statement during program execution, it proceeds to the next statement without taking any action. A function may be used in any string or numeric expression by using function-name followed by an expression enclosed in parentheses if a parameter was specified in the DEF statement.

When a reference to a function is encountered in an expression (by using function-name in a statement), the function is evaluated using the current values of the variables specified in the DEF statement and the value of parameter if there is one. A DEF statement can refer to other defined functions. However, the function you specify may not refer to itself either directly (e.g. DEF B=B*2) or indirectly (e.g. DEF F=G:DEF G=F).

Attempting to print the value of a function with PRINT used as a command does not work if the Memory Expansion is connected to your computer.

**Options**

If you specify a parameter for a function, when a reference to the function is encountered in an expression, its value is assigned to parameter. The value of the function is then determined using the value of parameter and the values of the other variables in the DEF statement. If parameter is given in the DEF statement, an argument value must always be given when referring to the function.

The parameter name used in the DEF statement affects only the DEF statement in which it is used. This means that it is distinct from any other variable with the same name which appears elsewhere in the program.

Parameter may not be used as an array. You can use an array element in a function as long as the array does not have the same name as parameter. For example you may use DEF F(A)=B(Z) but not DEF F(A)=A(Z).
DELETE

**Format**

DELETE device-filename

**Description**

The DELETE command allows you to remove a program or data file from the computer's filing system. Device-filename is a string expression. If a string constant is used, it must be enclosed in quotes. You may also delete data files by using the keyword DELETE in the CLOSE statement.

Some devices (such as diskettes) allow deleting files; others (such as cassettes) do not. Read the manual for the specific device for more information.

**Example**

DELETE "DSK1.MYFILE" deletes >DELETE "DSK1.MYFILE"

the file named MYFILE from the
diskette in disk drive 1.

**Program**

The program on the right illustrates >100 INPUT "FILENAME": X$ a use of DELETE. >110 DELETE X$

DELSPRITE subprogram

**Format**

CALL DELSPRITE(#sprite-number [,....])
CALL DELSPRITE(ALL)

**Description**

The DELSPRITE subprogram removes sprites from further access by a program. You may delete one or more sprites by specifying their numbers preceded by a number sign (#) and separated by commas, or you may delete all sprites by specifying ALL. After being deleted with DELSPRITE, a sprite can be recreated with the SPRITE subprogram.

**Examples**

CALL DELSPRITE(#3) deletes sprite >100 CALL DELSPRITE(#3)

number 3.

CALL DELSPRITE(#4,#3*C) deletes >100 CALL DELSPRITE(#4,#3*C)
sprite number 4 and the sprite whose number is found by multiplying 3 by C.

CALL DELSPRITE(ALL) deletes all >100 CALL DELSPRITE(ALL)
sprites.
**DIM**

**Format**

DIM array-name(integer1 [,integer2] ... [,integer7] [,...])

**Description**

The DIM statement reserves space in the computer's memory for numeric and string arrays. You can dimension an array only once in a program. If you dimension an array, the DIM statement must appear in the program at a lower numbered line than any other reference to the array. If you dimension more than one array in a single DIM statement, array-names are separated by commas. Array-name may be any variable name. A DIM statement may not appear in an IF-THEN-ELSE statement.

You may have up to seven-dimensional arrays in TI Extended BASIC. The number of integers separated by commas following the array name determines how many dimensions the array has. The values of the integers determine the number of elements in each dimension.

Space is allocated for an array after you enter the RUN command but before the first statement is executed. Each element in a string array is a null string and each element in a numeric array is zero until it is replaced with another value.

The values of the integers determine the maximum value of each subscript for that array. If you are using an array not defined in a DIM statement, the maximum value of each: subscript is 10. The first element is zero unless an OPTION BASE statement sets the minimum subscript value to 1. Thus an array defined as DIM A(5) is a one-dimensional array with seven elements unless the zero subscript is eliminated by the OPTION BASE statement.

**Examples**

DIM XS(30) reserves space in the computer's memory for 31 members of the array called XS.

DIM D:100,B(10,9) reserves space in the computer's memory for 101 members of the array called D and 110 (11 times 10) members of the array called B.

>100 DIM XS(30)

>100 DIM D(100),B(10,9)

**DISPLAY**

**Format**

DISPLAY [ [AT(row,column)] [BEEP] [ERASE ALL] [SIZE(numeric-expression)] ] variable-list

**Description**

The DISPLAY statement displays information on the screen. Many options are available with DISPLAY, making it far more versatile than PRINT. It may display data at any screen position, make an audible tone (beep) when displaying data, blank screen positions, and erase all characters on the screen before displaying data.

**Options**

AT(row,column) places the beginning of the display field at the specified row and column. Rows are numbered 1 through 24. Columns are numbered 1 through 28 with column 1 corresponding with what is called column 3 in the VCHAR, HCHAR, and GCHAR subprograms. If the AT option is not present, data is displayed at row 24, column 1, just as it is with the PRINT statement.

BEEP sounds a short tone when the data is displayed.

ERASE ALL fills the entire screen with the blank character before displaying data.

SIZE(numeric-expression) puts numeric-expression blank characters on the screen starting at row and column. If the SIZE option is not present, the rest of the row at which data is to be displayed is blanked. If numeric-expression is larger than the number of positions remaining in the row, only the rest of the row is blanked.

**Examples**

DISPLAY AT(5,7):Y displays the value of Y at the fifth row, seventh column of the screen.

DISPLAY ERASE ALL:B puts the blank character into all screen positions before displaying the value of B.

DISPLAY AT(R,C) SIZE(FIELDLEN) BEEP:X$ displays the value of XS at row R, column C. First it beeps and blanks FIELDLEN characters.
DISPLAY

Program
The program at the right illustrates a use of DISPLAY. It allows you to position blocks at any screen position to draw a figure or design.

```plaintext
>100 CALL CLEAR
>110 CALL COLOR(9,5,5)
>120 DISPLAY AT(23,1):"INTER ROW AND COLUMN."
>130 DISPLAY AT(24,1):"ROW: COLUMN:"
>140 FOR COUNT=1 TO 2
>150 CALL KEY(0,ROW(COUNT),S)
>160 IF S<0 THEN 150
>170 DISPLAY AT(24,5+COUNT*SIZE(1):CHR$(ROW(COUNT)-48)
>180 NEXT COUNT
>190 FOR COUNT=1 TO 2
>200 CALL KEY(0,COLUMN(COUNT),S)
>210 IF S<0 THEN 200
>220 DISPLAY AT(24,16+COLUMN*SIZE(1):CHR$(COLUMN(COUNT)-48)
>230 NEXT COUNT
>240 ROW=10*(ROW(1)-48)+ROW(2)-48
>250 COLUMN1=10*(COLUMN(1)-48)+COLUMN(2)-48
>260 DISPLAY AT(ROW1,COLUMN1)
>270 GOTO 130
(Press SHIFT C to stop the program.)
```

DISPLAY USING

Format
DISPLAY [option-list:] USING string-expression [: variable-list]
DISPLAY [option-list:] USING line-number [: variable-list]

Description
The DISPLAY...USING statement is the same as DISPLAY with the addition of the USING clause, which specifies the format of the data in variable-list. If string-expression is present, it defines the format. If line-number is present, it refers to the line number of an IMAGE statement. See IMAGE for an explanation of how the format is defined.

Examples
DISPLAY AT(10,4):USING "###":N
>100 DISPLAY AT(13,4):USING "###":N

DISPLAY AT(10,4):USING "###":N displays the value of N at the tenth row and fourth column, with the format "###".

DISPLAY USING "###":N displays the value of N at the 24th row and first column, with the format "###".
**DISTANCE subprogram**

**Format**  
CALL DISTANCE(#sprite-number,#sprite-number,numeric-variable)  
CALL DISTANCE(#sprite-number,dot-row,dot-column,numeric-variable)

**Description**  
The DISTANCE subprogram returns the square of the distance between two sprites or between a sprite and a location. The position of each sprite is considered to be its upper left hand corner. Dot-row and dot-column are from 1 to 256. The squared distance is returned in numeric-variable.

The number returned is computed as follows: The difference between the dot-rows of the sprites (or the sprite and the location) is found and squared. Then the difference between the dot-columns of the sprites (or the sprite and the location) is found and squared. Then the two squares are added. If the sum is larger than 32767, then 32767 is returned. The distance between the sprites (or the sprite and the location) is the square root of the value returned.

**Examples**  
CALL DISTANCE(#3,#4,DIST) sets DIST equal to the square of the distance between the upper left hand corners of sprite #3 and sprite #4.  
CALL DISTANCE(#4,13,89,D) sets D equal to the square of the distance between the upper left hand corner of sprite #4 and position 18, 89.

---

**END**

**Format**  
END

**Description**  
The END statement ends your program and may be used interchangeably with the STOP statement. Although the END statement may appear anywhere, it is normally placed as the last line in a program and thus ends the program both physically and logically. The STCP statement is usually used in other places that you want your program to halt. In TI Extended BASIC you are not required to use the END statement. The program automatically stops after it executes the highest numbered line.
**EOF**

**Format**

`EOF(file-number)`

**Description**

The `EOF` function is used to test whether there is another record to be read from a file. The value of `file-number` indicates the file to be tested and must correspond to the number of an open file. The `EOF` function cannot be used with cassettes.

The `EOF` function always assumes that the next record is going to be read sequentially, even if you are using a RELATIVE file.

The value that the `EOF` function provides depends on where you are in the file. If you are not at the last record of the file, the function returns a value of 0. If you are at the last record of the file, the function returns a value of 1. If the diskette or other storage medium is full, you are at the end of the file and there is no more room for any data, the function returns a value of -1.

For more information, see the Disk Memory System manual.

**Examples**

PRINT `EOF(3)` prints a value according to whether you are at the end of the file that was opened as #3.

>100 PRINT `EOF(3)`

IF `EOF(27)<0` THEN 150 transfers control to line 150 if you are at the end of the file that was opened as #27.

>100 IF `EOF(27)<0` THEN 150

IF `EOF(27)` THEN 1150 transfers control to line 1150 if you are at the end of the file that was opened as #27.

>100 IF `EOF(27)` THEN 1150

**ERR subprogram**

**CHAPTER 4**

**Format**

`CALL ERR(error-code, error-type [, error-severity, line-number])`

**Description**

The `ERR` subprogram returns the `error-code` and `error-type` of the most recent uncleared error. An error is cleared when it has been accepted by the `ERR` subprogram, another error has occurred, or the program has ended.

`Error-codes` are two or three digit numbers. The meanings of each of the codes is in Appendix N.

If `error-type` is a negative number, then the error was in the execution of the program. If the `error-code` is 130 (IO ERROR), the `error-type` is a positive number and the number is the number of the file that caused the error.

If no error has occurred, `CALL ERR` returns all values as zeros.

`CALL ERR` is used in conjunction with `ON ERROR`.

**Options**

You may optionally obtain the `error-severity` and `line-number` on which the error occurred. The `error-severity` is always 9. The `line-number` is the number of the line being executed when the error occurred. It is not always the line that is the source of the problem since an error may occur because of values generated or actions taken elsewhere in a program.

**Examples**

`CALL ERR(A,B)` sets `A` equal to the `error-code` and `B` equal to the `error-type` of the most recent error.

>100 `CALL ERR(A,B)`

`CALL ERR(W,X,Y,Z)` sets `W` equal to the `error-code`, `X` equal to the `error-type`, `Y` equal to the `error-severity`, and `Z` equal to the `line-number` of the most recent error.

>100 `CALL ERR(W,X,Y,Z)`
**ERR SUBPROGRAM**

**Program**

The program on the right illustrates the use of CALL ERR. An error is caused in line 110 by calling for an illegal screen color. Because of line 100, control is transferred to line 130. Line 140 prints the values obtained. The 79 indicates that a bad value was provided.

The -1 indicates that the error was in a statement. The 9 is the error-severity. The 110 indicates that the error occurred in line 110.

>100 0: ERROR 130
>110 CALL SCREEN (18)
>120 STOP
>130 CALL ERR(4,X,Y,Z)
>140 PRINT W;X;Y;Z
>RUN
> 79 -1 9 110

---

**EXP**

**Format**

EXP(numeric-expression)

**Description**

The EXP function returns the exponential value \( e^x \) of numeric-expression. The value of \( e \) is 2.718281828459.

**Examples**

\[ Y = EXP(7) \] assigns to \( Y \) the value of \( e \) raised to the seventh power which is 1096.633158429.

\[ L = EXP(4.394960467) \] assigns to \( L \) the value of \( e \) raised to the 4.394960467 power which is 81.0442688868.
FOR TO [STEP]

Format

FOR control-variable = initial-value TO limit [STEP increment]

Description

The FOR-TO-STEP statement repeats execution of the statements between FOR-TO-STEP and NEXT until the control-variable is outside the range of initial-value to limit. The FOR-TO-STEP statement is useful when repeating the same steps in a loop. The FOR-TO-STEP statement cannot be used in an IF-THEN-ELSE statement.

Control-variable may be any un subscripted numeric variable. It acts as a counter for the loop. Initial-value and limit are numeric expressions. The loop starts with control-variable given a value of initial-value. The second time through the loop, the value of control-variable is changed by one or optionally by increment, which may be a positive or negative number. This continues until the value of control-variable is outside the range initial-value to limit. Then the statement after NEXT is executed. The value of control-variable is not changed when the computer leaves the loop.

The value of control-variable can be changed within the loop, but this must be done carefully to avoid unexpected results. Loops may be “nested,” that is one loop may be contained wholly within another. You may leave a loop using GOTO, GOSUB, IF-THEN-ELSE, or NEXT, and then return. However, you may not enter a FOR-NEXT loop at any point except at its start.

If initial-value exceeds limit at the beginning of the FOR-NEXT loop, none of the statements in the loop are executed. Instead execution continues with the first statement after the NEXT statement.

Examples

FOR A = 1 TO 5 STEP 2 executes the statements between this FOR and NEXT A three times, with A having values of 1, 3, and 5. After the loop is finished, A has a value of 7.

FOR J = 7 TO -5 STEP -.5 executes the statements between the FOR and NEXT J 25 times, with J having values of 7, 6.5, 6, ..., -4, -4.5, and -5. After the loop is finished, J has a value of -5.5.

>100 CALL CLEAR
>110 D=0
>120 FOR CHAR=33 TO 63 STEP 3
>130 FOR ROW=1+D TO 21+D STEP 4
>140 FOR COLUMN=1-D TO 29-D STEP 4
>150 CALL VCHAR(ROW,COLUMN,CHAR
>160 NEXT COLUMN
>170 NEXT ROW
>180 D=2
>190 NEXT CHAR
>200 GOTO 200

(Press SHIFT C to stop the program.)
GCHAR subprogram

Format
CALL GCHAR(row, column, numeric-variable)

Description
The GCHAR subprogram reads a character from anywhere on the display screen. The computer returns in numeric-variable the ASCII code for the character in the position described by row and column.

Row and column are numeric expressions. A value of 1 for row indicates the top of the screen. A value of 1 for the column indicates the left side of the screen. The screen can be thought of as a grid as shown below.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
</tbody>
</table>

Examples
CALL GCHAR(12, 16, X) assigns to X the ASCII code of the character that is in row 12, column 16.

CALL GCHAR(R, C, K) puts into K the ASCII code of the character that is in row R, column C.

---

GOSUB

Format
GOSUB line-number
GO SUB line-number

Description
The GOSUB statement allows transfer to a subroutine. When executed, control is transferred to line-number and that statement and any following (which may include any statements, including GOTO statements and other GOSUB statements) are executed. When a RETURN statement is encountered, control is returned to the next statement following the GOSUB statement. Subroutines are most useful when the same action is to be performed in different parts of a program. See also ON...GOSUB. Subroutines in TI Extended BASIC may call themselves.

Example
GOSUB 200 transfers control to statement 200. That statement and the ones up to RETURN are executed, and then control returns to the statement after the calling statement.

>100 CALL GOSUB(12, 16, X)
Program

The program on the right illustrates a use of GOSUB. The subroutine at line 260 figures the factorial of the value of NUMB. The whole program figures the solution to the equation

\[ \text{NUMB} = \frac{X!}{Y! \cdot (X-Y)!} \]

where the exclamation point means factorial. This formula is used to figure certain probabilities. For instance, if you enter X as 52 and Y as 5, you’ll find the number of possible five card poker hands.

>100 CALL CLEAR
>110 INPUT "ENTER X AND Y: "; X,Y
>120 IF X<Y THEN 110
>130 IF X>69 OR Y>69 THEN 110
>140 NUMB=X
>150 GOSUB 260
>160 NUMERATOR=NUMB
>170 NUME=Y
>180 GOSUB 260
>190 DENOMINATOR=NUMB
>200 NUME=X-Y
>210 GOSUB 260
>220 DENOMINATOR=DENOMINATOR*NUMB
>230 NUME=NUMERATOR/DENOMINATOR
>240 PRINT "NUMBER IS":NUME
>250 STOF
>260 REM FIGURE FACTORIAL
>270 IF NUMB<0 THEN PRINT "NEGATIVE" : : GOTO 110
>280 IF NUMB<2 THEN NUMB=1 : : GOTO 330
>290 NUMB=NUMB-1
>300 NUMB=NUMB*MULT
>310 MULT=MULT-1
>320 IF MULT>1 THEN 300
>330 RETURN

GOTO

Format

GOTO line-number
GOTO line-number

Description

The GOTO statement allows you to transfer control unconditionally to another line within a program. When a GOTO statement is executed, control is passed to the first statement on the line specified by line-number.

The GOTO statement should not be used to transfer control into subprograms.

Program

The program at the right shows the use of GOTO in line 160. Anytime that line is reached the program executes line 130 next and proceeds from that new point.

>110 REM ADD 1 THROUGH 100
>120 ANSWER=0
>130 ANSWER=ANSWER+NUMB
>140 NUMB=NUMB+1
>150 IF NUMB>100 THEN 170
>160 GOTO 30
>170 PRINT "THE ANSWER IS":ANSWER
>RUN
THE ANSWER IS 5050
HCHAR subprogram

Format
CALL HCHAR(row, column, character-code [, repetition ])

Description
The HCHAR subprogram displays a character anywhere on the display screen and optionally repeats it horizontally. The character with the ASCII value of character-code is placed in the position described by row and column and is repeated horizontally repetition times.

A value of 1 for row indicates the top of the screen. A value of 24 is the bottom of the screen. A value of 1 for column indicates the left side of the screen. A value of 32 is the right side of the screen. The screen can be thought of as a grid as shown below.

![Screen Grid Diagram]

Examples
CALL HCHAR(12,16,33) places character 33 (an exclamation point) in row 12, column 16.

CALL HCHAR(1,1,ASC('!'),768) places an exclamation point in row 1, column 1, and repeats it 768 times, which fills the screen.

CALL HCHAR(R,C,K,T) places the character with an ASCII code specified by the value of K in row R, column C and repeats it T times.
**IF THEN [ELSE]**

**Format**

IF relational-expression THEN line-number1 [ELSE line-number2]
IF relational-expression THEN statement1 [ELSE statement2]
IF numeric-expression THEN line-number1 [ELSE line-number2]
IF numeric-expression THEN statement1 [ELSE statement2]

**Description**

The IF-THEN-ELSE statement allows you to transfer control to line-number1
or to perform statement1 if relational-expression is true; or if numeric-expression
is not equal to zero. Otherwise control passes to the next
statement, or optionally to line-number2 or statement2.

Statement1 and statement2 may each be several statements long, separated
by the statement separator symbol. They are only executed if the clause
immediately before them is executed. The IF-THEN-ELSE statement cannot
contain DATA, DEF, DIM, FOR, NEXT, OPTIC, BASE, SUB, or SUBEND.

**Examples**

IF X>5 THEN GOSUB 300 ELSE X=X+5
X=X+5 operates as follows: If X is
greater than 5, then GOSUB 300 is
executed. When the subroutine is
ended, control returns to the line
following this line. If X is 5 or less, X
is set equal to X + 5 and control
passes to the next line.

IF G THEN C=C+1:GOTO 500:
ELSE L=L/C:GOTO 300
0: :ELSE L=L/C:GOTO 300

operates as follows: If G is not zero,
then C is set equal to C + 1 and
control is transferred to line 500. If G
is zero then L is set equal to L/C and
control is transferred to line 300.

IF A>3 THEN 300 ELSE A=0:
GOTO 10 operates as follows: If A is
greater than 3, then control is
transferred to line 300. Otherwise, A
is reset to zero and control
is transferred to line 10.

>100 IF A$="Y" THEN COUNT=COUNT+1:DISPLAY AT(24,1):
HERE WE GO AGAIN!":GOTO 300
operates as follows: If A is not equal
to "Y", then control passes to the
next line. If A is equal to "Y", then
COUNT is incremented by 1, a
message is displayed, and control is
transferred to line 300.

IF HOURS <= 40 THEN
PAY = HOURS * WAGE ELSE
PAY = HOURS * WAGE + 0.5 * WAGE*
(HOURS - 40) :: OT = 1 operates as
follows: If HOURS is less than or
equal to 40, then PAY is set equal to
HOURS * WAGE and control passes to
the next line. If HOURS is greater
than 40 then PAY is set equal to
HOURS * WAGE + 0.5 * WAGE * (HO-
OURS - 40). OT is set equal to 1, and
control passes to the next line.

IF A = 1 THEN IF B = 2 THEN C = 3
ELSE D = 4 ELSE E = 5 operates as
follows: If A is not equal to 1, then E
is set equal to 5 and control passes to
the next line. If A is equal to 1 and B
is not equal to 2 then D is set equal
to 4 and control passes to the next
line. If A is equal to 1 and B is equal
to 2, then C is set equal to 3 and
control passes to the next line.
Program

The program on the right illustrates a use of IF-THEN-ELSE. It accepts up to 1000 numbers and then prints them in order from smallest to largest.

>100 CALL CLEAR
>110 DIM VALUE(1000)
>120 PRINT "ENTER VALUES TO BE SORTED.": "ENTER '9999' TO END ENTRY."
>130 FOR COUNT=1 TO 1000
>140 INPUT VALUE(COUNT)
>150 IF VALUE(COUNT)=9999 THEN N=170
>160 NEXT COUNT
>170 COUNT=COUNT+1
>180 PRINT "SORTING..."
>190 FOR SORT1=1 TO COUNT-1
>200 FOR SORT2=SORT1+1 TO COUNT
>210 IF VALUE(SORT1)>VALUE(SORT2) THEN TEMP=VALUE(SORT1):
VALUE(SORT1)=VALUE(SORT2):
VALUE(SORT2)=TEMP
>220 NEXT SORT2
>230 NEXT SORT1
>240 FOR SORT3=1 TO COUNT
>250 PRINT VALUE(SORT3)
>260 NEXT SORTED

Format

IMAGe format-string

Description

The IMAGE statement specifies the format in which numbers are printed or displayed when the USING clause is present in PRINT or DISPLAY. No action is taken when the IMAGE statement is encountered during program execution. The IMAGE statement must be the only statement on a line. The following description of format-string also applies to the use of an explicit image after the USING clause in PRINT...USING and DISPLAY...USING.

Format-string must contain 254 or fewer characters and may be made up of any characters. They are treated as follows:

Pound signs (#) are replaced by the print-list values given in PRINT...USING or DISPLAY...USING. One pound sign must be allowed for each digit of the value and one for the negative sign if it is present, or for each character that is to be printed. If there is not enough room to print the numbers or characters in the space allowed, each pound sign is replaced with an asterisk (*). If more numbers are after the decimal place than are allowed by the number of pound signs after the decimal place in the IMAGE statement, the number is rounded to fit. If there are fewer non-numeric characters than are allowed for in the print string, the value printed will have blanks for the extra characters.

To indicate that a number is to be given in scientific notation, circumflexes (a) must be given for the E and power numbers. There must be four or five circumflexes, and 10 or fewer characters (minus sign, pound signs, and decimal point) when using the E format.

The decimal point separates the whole and fractional portions of numbers, and is printed where it appears in the IMAGE statement.

All other letters, numbers, and characters are printed exactly as they appear in the IMAGE statement.

Format-string may be enclosed in quotation marks. If it is not enclosed in quotation marks, leading and trailing spaces are ignored. However, when used directly in PRINT...USING or DISPLAY...USING, it must be enclosed in quotation marks.

Each IMAGE statement may have space for many images, separated by any character except a decimal point. If more values are given in the PRINT...USING or DISPLAY...USING statement than there are images, then the images are reused, starting at the beginning of the statement.

If you wish, you may put format-string directly in the PRINT...USING or DISPLAY...USING statement immediately following USING. However, if a
format-string is used often, it is more efficient to refer to an IMAGE statement.

Examples

IMAGE $****.## allows printing of any number from -999.999 to 9999.999. The following show how some sample values will be printed or displayed.

<table>
<thead>
<tr>
<th>Value</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>-999.999</td>
<td>-999.999</td>
</tr>
<tr>
<td>-34.5</td>
<td>-34.500</td>
</tr>
<tr>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>12.4565</td>
<td>12.457</td>
</tr>
<tr>
<td>6312.991</td>
<td>6312.999</td>
</tr>
<tr>
<td>9999999</td>
<td>**********</td>
</tr>
</tbody>
</table>

>100 IMAGE $####.##
>110 PRINT USING 100:A

>200 IMAGE THE ANSWERS ARE # AND ###
>210 PRINT USING 200:A;B

IMAGE THE ANSWERS ARE ****
AND #.## allows printing of two numbers. The first may be from -99 to 999 and the second may be from -9.9 to 99.9. The following show how some sample values will be printed or displayed.

<table>
<thead>
<tr>
<th>Values</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>-3.459 THE ANSWERS ARE -7 AND -3.46</td>
</tr>
<tr>
<td>0</td>
<td>0 THE ANSWERS ARE 0 AND 0.00</td>
</tr>
<tr>
<td>14.8</td>
<td>12.75 THE ANSWERS ARE 15 AND 12.75</td>
</tr>
<tr>
<td>795</td>
<td>852 THE ANSWERS ARE 795 AND 852</td>
</tr>
<tr>
<td>-984</td>
<td>64.7 THE ANSWERS ARE *** AND 64.70</td>
</tr>
</tbody>
</table>

>300 IMAGE DEAR ####, DEAR JOHN.
>310 PRINT USING 300:XX

<table>
<thead>
<tr>
<th>Values</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOHN</td>
<td>DEAR JOHN</td>
</tr>
<tr>
<td>TOM</td>
<td>DEAR TOM</td>
</tr>
<tr>
<td>RALPH</td>
<td>DEAR ****</td>
</tr>
</tbody>
</table>

Programs

The program on the right illustrates a use of IMAGE. It reads and prints seven numbers and their total. Lines 110 and 120 set up the images. They are the same except for the dollar sign in line 110. To keep the blank space where the dollar sign was, the format-string in line 120 is enclosed in quotation marks.

Line 180 prints the values using the IMAGE statements.

Line 210 shows that the format can be put directly in the PRINT...USING statement.

The amounts are printed with the decimal points lined up.

>100 CALL CLEAR
>110 IMAGE $####.##
>120 IMAGE "####.##"
>130 DATA 233.45,-147.95,0.4, 37.26,-51.299,35.2,464
>140 TOTAL=0
>150 FOR A=1 TO 7
>160 READ AMOUNT
>170 TOTAL=TOTAL+AMOUNT
>180 IF A=1 THEN PRINT USING 110:AMOUNT ELSE PRINT USING 120:AMOUNT
>190 NEXT A
>200 PRINT " "
>210 PRINT USING "$####.##":TOTAL
>220 RUN

$ 233.45
(-147.95
  37.26
  -51.30
  85.20
  464.00
  --------
$ 629.06
The program at the right shows the effect of using more values in the PRINT...USING statement than there are images in the IMAGE statement.

```
>100 IMAGE *** **,** *,
>110 PRINT USING 100:53.34,50 .34,37.26,37.26
>RUN
50.34, 50.3
37.26, 37.3
```

### INIT subprogram

#### Format

CALL INIT

#### Description

The INIT subprogram is used, along with LINK, LOAD, and PEEK, to access assembly language subprograms. The INIT subprogram checks to see that the Memory Expansion is connected, prepares the computer to run assembly language programs, and loads a set of supporting routines into the Memory Expansion.

The INIT subprogram must be called before LOAD and LINK are called. INIT removes any previously loaded subprograms from the Memory Expansion. The effects of INIT last until the Memory Expansion is turned off and does not need to be called from each program that is using the subprogram involved.

If the Memory Expansion is not attached, a syntax error is given.
INPUT

**Format**

INPUT [input-prompt:] variable-list

(For information on using the INPUT statement with a file, see INPUT with files.)

**Description**

This form of the INPUT statement is used when entering data from the keyboard. The INPUT statement suspends program execution until data is entered from the keyboard. The optional input-prompt may display on the screen what data is expected.

Variable-list contains the variables (scalar or array elements: numeric or string) which are assigned values when the INPUT statement is executed. The variables are separated by commas. If a value in variable-list is input, it may later be used as a subscript in the same INPUT statement.

When inputting string values, they may optionally be enclosed in quotation marks. However, if you wish to have leading or trailing blanks or commas, the entire string must be enclosed in quotation marks. If more than one value is to be input, separate the values by input by commas.

**Options**

The optional input-prompt is a string expression. It must be followed by a colon. It is displayed on the screen when the INPUT statement is executed. If there is no input-prompt, a question mark and space are displayed to indicate that input is expected. If there is an input-prompt, it takes the place of the question mark and space.

**Examples**

INPUT X allows the input of a number.

INPUT X$,Y allows the input of a string and a number.

INPUT "ENTER TWO NUMBERS: ":A,B prints the prompt ENTER TWO NUMBERS and then allows the entry of two numbers.

INPUT A(J),J first evaluates the subscript of A and then accepts data into that subscript of A. Then a value is accepted in A(J).

>100 INPUT J,A(J)

**Program**

The program on the right illustrates the use of INPUT from the keyboard. Lines 110 through 140 allow the person using the program to enter data, as requested with the input-prompts.

Lines 170 through 250 construct a letter based on the input

>100 CALL CLEAR
>110 INPUT "ENTER YOUR FIRST NAME: ":FNAME$
>120 INPUT "ENTER YOUR LAST NAME: ":LNAME$
>130 INPUT "ENTER A THREE DIGIT NUMBER: ":DOLLARS
>140 INPUT "ENTER A TWO DIGIT NUMBER: ":CENTS
>150 IMAGE OF $###.## AND THAT IF YOU
>160 CALL CLEAR
>170 PRINT "DEAR ":FNAME$ ": ":
>180 PRINT "THIS IS TO REMIND YOU"
>190 PRINT "THAT YOU OWE US THE AMOUNT"
>200 PRINT USING 150:DOLLARS,CENTS/100
>210 PRINT "DO NOT PAY US, YOU WILL SOON"
>220 PRINT "RECEIVE A LETTER FROM OUR"
>230 PRINT "ATTORNEY, ADDRESS ED TO"
>240 PRINT FNAME$:LNAME$;
>250 PRINT TAB(15):"SINCERELY"
>260 PRINT TAB(15):"I, DUN YOU"
>260 GOTO 160

(Press SHIFT C to stop the program.)
INPUT (with files)

Format
INPUT #file-number [.REC record-number] :variable-list
(For information on using the INPUT statement to enter data from the keyboard, see INPUT.)

Description
The INPUT statement, when used with files, allows you to read data from files. The INPUT statement can only be used with files opened in INPUT or UPDATE mode. DISPLAY files may not have over 160 characters in each record.

File-number and variable-list must be included in the INPUT statement. Record-number may optionally be included when reading random access (RELATIVE) files from disks.

All statements which refer to files do so with a file-number from 0 through 255. File-number is assigned to a particular file by the OPEN statement. File number 0 is dedicated to the keyboard and screen of the computer. It cannot be used for other files and is always open. File-number is entered as a number sign (#) followed by a numeric expression that, when rounded to the nearest integer, is a number from 0 to 255, and is the number of a file that is open.

Variable-list is the list of variables into which you want the data from the file to be placed. It consists of string or numeric variables separated by commas with an optional trailing comma.

Options
You can optionally specify the number of the record that you want to read as record-number. It can only be specified for diskette files which have been opened as RELATIVE. The first record of a file is number 0.

Examples
INPUT #1:X$ puts into X$ the next value available in the file that was opened as #1.

INPUT #23:X,A,LL$ puts into X, A, and LL$ the next three values from the file that was opened as #23.

INPUT #11:REC 44:tax puts into TAX the first value of record number 44 of the file that was opened as #11.

INPUT #3:A,B,C puts into A, B, and C the next three values from the file that was opened as #3. The comma after C creates a pending input condition. When the next INPUT or LINPUT statement using this file is performed, one of the following actions occurs: If the next INPUT or LINPUT statement has no REC clause, the computer uses the data beginning where the previous INPUT statement stopped. If the next INPUT or LINPUT statement includes a REC clause, the computer terminates the pending input condition and reads the specified record.
INPUT (with files)

Program

The program at the right illustrates a use of the INPUT statement. It opens a file on the cassette recorder and writes 5 records on the file. It then goes back and reads the records and displays them on the screen.

>100 OPEN #1: "CS1", SEQUENTIAL
   , INTERNAL, INPUT, FIXED 64
>110 FOR A=1 TO 5
>120 PRINT #1: "THIS IS RECORD ", A
>130 NEXT A
>140 CLOSE #1
>150 CALL CLEAR
>160 OPEN #1: "CS1", SEQUENTIAL
   , INTERNAL, INPUT, FIXED 64
>170 FOR B=1 TO 5
>180 INPUT #1: A$, C
>190 DISPLAY AT(B,1): A$; C
>200 NEXT B
>210 CLOSE #1
>220 RUN
   *REWIND CASSETTE TAPE CS1
   THEN PRESS ENTER
   *PRESS CASSETTE RECORD CS1
   THEN PRESS ENTER
   *PRESS CASSETTE STOP CS1
   THEN PRESS ENTER
   *REWIND CASSETTE TAPE CS1
   THEN PRESS ENTER
   *PRESS CASSETTE PLAY CS1
   THEN PRESS ENTER
   THIS IS RECORD 1
   THIS IS RECORD 2
   THIS IS RECORD 3
   THIS IS RECORD 4
   THIS IS RECORD 5
   *PRESS CASSETTE STOP CS1
   THEN PRESS ENTER

See the Disk Memory System manual for instructions on using diskettes.

INT

CHAPTER 4

Format

INT(numeric-expression)

Description

The INT function returns the greatest integer less than or equal to numeric-expression.

Examples

PRINT INT(3.4) prints 3.

X = INT(3.9) sets X equal to 3.

P = INT(3.999999999) sets P equal to 3.

DISPLAY AT(3,7):INT(4.0) displays 4 at the third row, seventh column.

N = INT(-3.9) sets N equal to -4.

K = INT(-3.0000001) sets K equal to -4.

>100 PRINT INT(.4)

>100 X = INT(3.90)

>100 P = INT(3.999999999)

>100 DISPLAY AT(3,7):INT(4.0)

>100 N = INT(-3.9)

>100 K = INT(-3.0000001)
JOYST subprogram

Format
CALL JOYST(key-unit,x-return,y-return).

Description
The JOYST subprogram returns data into x-return and y-return based on the position of the joystick in the Wired Remote Controller (available separately) labeled key-unit. Key-unit is a numeric expression with a value of 1 through 4. The values 1 and 2 are joysticks 1 and 2. Values 3 and 4 are reserved for possible future use.

The values returned in x-return and y-return depend on the position of the joystick. The values returned are shown below. The first value in the parentheses is placed in x-return. The second value is placed in y-return.

Example
CALL JOYST(1,X,Y) returns values in X and Y according to the position of joystick number 1.

Program
The program on the right illustrates a use of the JOYST subprogram. It creates a sprite and then moves it around according to the input from a joystick.

>100 CALL CLEAR
>110 CALL SPRITE(#1,33,5,96,1,28)
>120 CALL JOYST(1,X,Y)
>130 CALL MOTION(#1,-Y,X)
>140 GOTO 120
(Press SHIFT C to stop the program.)

KEY subprogram

Format
CALL KEY(key-unit,return-variable,status-variable)

Description
The KEY subprogram assigns the code of the key pressed to return-variable. The value assigned depends on the key-unit specified. If key-unit is 0, input is taken from the entire keyboard, and the value placed in return-variable is the ASCII code of the key pressed. If no key is pressed, return-variable is set equal to -1. See Appendix C for a list of the ASCII codes.

If key-unit is 1, input is taken from the left side of the keyboard. If key-unit is 2, input is taken from the right side of the keyboard. The possible values placed in return-variable are given in Appendix C. Values of 3, 4, and 5 are reserved for possible future uses.

Status-variable indicates whether a key has been pressed. A value of 1 means a new key was pressed since the last CALL KEY was executed. A value of -1 means the same key was pressed as in the previous CALL KEY. A value of 0 means no key was pressed.

Example
CALL KEY(0,K,S) returns in K the ASCII code of any key pressed on the keyboard, and in S a value indicating whether any key was pressed.

Program
The program on the right illustrates a use of the KEY subprogram. It creates a sprite and then moves it around according to the input from the left side of the keyboard. Note that line 130 returns to line 120 if no key has been pressed.

>100 CALL CLEAR
>110 CALL SPRITE(#1,33,5,96,1,28)
>120 CALL KEY(1,K,S)
>130 IF S=0 THEN 12C
>140 IF K=5 THEN X=-4
>150 IF K=0 THEN Y=4
>160 IF K=2 THEN X=4
>170 IF K=3 THEN X=2
>180 IF K=1 THEN X,=0
>190 IF K>5 THEN X,Y=0
>200 CALL MOTION(#1,Y,X)
>210 GOTO 120
(Press SHIFT C to stop the program.)
LEN

Format
LEN(string-expression)

Description
The LEN function returns the number of characters in string-expression. A space counts as a character.

Examples
PRINT LEN("ABCDE") prints 5.  
X = LEN("THIS IS A SENTENCE.") sets X equal to 19.  
DISPLAY LEN("") displays 0.  
DISPLAY LEN(" ") displays 1.

LET

CHAPTER 4

Format
[LET] numeric-variable [,numeric-variable,...] = numeric-expression  
[LET] string-variable [,string-variable,...] = string-expression

Description
The LET statement assigns the value of an expression to the specified variable(s). The computer evaluates the expression on the right and puts its value into the variable(s) on the left. If more than one variable is on the left, they are separated with commas. The LET is optional, and is omitted in the examples in this manual. All subscripts in the variable(s) on the left are evaluated before any assignments are made.

You may use relational and logical operators in numeric-expression. If the relation or logical value is true, numeric-variable is assigned a value of -1. If the relation or logical value is false, numeric-variable is assigned a value of 0.

Examples
T = 4 puts the value 4 into T.  
X,Y,Z = 12.4 puts the value 12.4 into X, Y, and Z.  
A = 3<5 puts -1 into A since it is true that 3 is less than 5.  
B = 12<7 puts 0 into B since it is not true that 12 is less than 7.  
I,A(I) = 2 puts 3 into A(I) with whatever value I had before, and then puts 3 into I.  
L$,D$,B$ = "B" puts "B" into L$, D$, and B$.  
>100 T=4  
>100 X,Y,Z=12.4  
>100 A=3<5  
>100 B=12<7  
>100 I,A(I)=2  
>100 L$,D$,B$="B"
LINK subprogram

Format
CALL LINK(subprogram-name [,argument-list])

Description
The LINK subprogram is used along with INIT, LOAD, and PEEK to access assembly language subprograms. The LINK subprogram passes control and, optionally, a list of parameters from a TI Extended BASIC program to an assembly language subprogram.

Subprogram-name is the name of the subprogram to be called. It must have been previously loaded into the Memory Expansion with the CALL LOAD command or statement. Argument-list is a list of variables and expressions as required by the specific assembly language subprogram being called.

LINPUT

Format
LINPUT [file-number [,REC record-number] :] string-variable
LINPUT [input-prompt] string-variable

Description
The LINPUT statement allows the assignment of an entire line, file record, or (if there is a pending input record) the remaining portion of a file record into string-variable. No editing is performed on what is input, so commas, leading and trailing blanks, semicolons, colons, and quotation marks are placed in string-variable as they are given.

Options
A #file-number may be specified. If the file is in RELATIVE format, a specific record may be specified with REC. The file must be a DISPLAY-type file. If no file is specified, an input-prompt may be displayed prior to accepting input from the keyboard.

Examples
LINPUT LS assigns into LS anything typed before ENTER is pressed.
LINPUT "NAME: " :NMS displays NAME: and assigns into NMS anything typed before ENTER is pressed.
LINPUT #1,REC M:LS(M) assigns into LS(M) the value that was in record M of the file that was opened as #1.

Program
The program on the right illustrates the use of LINPUT. It reads a previously existing file and displays only the lines that contain the word "THE".

100 OPEN #1: "DSK1.TXT", INPU
110 IT EOF(1) THEN CLOSE #1
120 LINPUT #1: A$
130 I:POS(A$, "THE", 1)
140 IT I<>0 THEN PRINT A$
150 GOTO 110
LIST

Format
LIST ["device-name";] [line-number]
LIST ["device-name";] [start-line-number] - [end-line-number]

Description
The LIST command allows you to display program lines. If LIST is entered
with no numbers following it, the entire program in memory is listed. If a
number follows LIST, the line with that number is listed. If a number
followed by a hyphen follows LIST, that line and all lines following it are
listed. If a number preceded by a hyphen follows LIST, all lines preceding
it and that line are listed. If two numbers separated by a hyphen follow LIST,
the indicated lines and all lines between them are listed.

By pressing and holding a key until TI Extended BASIC responds, you may
temporarily halt a listing so that you can look at it on the screen. Press any
key again to restart the listing. Similarly, pressing SHIFT C (CLEAR) stops the
listing.

Options
The listing normally is displayed on the screen. If you wish, you can instead
direct the list to some other device, such as the optional thermal printer or
RS232 interface, by specifying device-name.

Examples
LIST lists the entire program in
memory on the display screen.
LIST 100 lists line 100.
LIST 100- lists line 100 and all lines
after it.
LIST -200 lists all lines up to and
including line 200.
LIST 100-200 lists all lines from 100
through 200.
LIST "TP" lists the entire program
on the optional thermal printer.
LIST "TP": -200 lists all lines up to
and including line 200 on the
optional thermal printer.

LOAD subprogram

Format
CALL LOAD("access-name" [address,byte] [,... file-field,...])

Description
The LOAD subprogram is used, along with INIT, LINK, and PEEK, to access
assembly language subprograms. The LOAD subprogram loads an assembly
language object file or direct data into the Memory Expansion for later
execution using the LINK statement.

The LOAD subprogram can specify one or more files from which to load
object data or lists of direct load data, which consists of an address followed
by data bytes. The address and data bytes are separated by commas. Direct
load data must be separated by file-field, which is a string expression
specifying a file from which to load assembly language object code. File-field
may be a null string when it is used merely to separate direct load data
fields. Use of the LOAD subprogram with incorrect values can cause the
computer to cease to function and require turning it off and back on.

Assembly language subprogram names (see LINK) are included in the file.
LOCATE subprogram

Format
CALL LOCATE(#sprite number,dot-row,dot-column [,....])

Description
The LOCATE subprogram is used to change the location of the given sprite(s) to the given dot-row(s) and dot-column(s). Dot-row and dot-column are numbered consecutively starting with 1 in the upper left hand corner of the screen. Dot-row can be from 1 to 192 and dot-column can be from 1 to 256. (Actually dot-row can go up to 236, but the locations from 193 through 256 are off the bottom of the screen.) The location of the sprite is the upper left hand corner of the character(s) which define it.

Program
The program on the right illustrates the use of the LOCATE subprogram. Line 110 creates a sprite as a fairly quickly moving red exclamation point. Line 140 locates the sprite at a location randomly chosen in lines 120 and 130. Line 150 repeats the process.

Also see the third example of the SPRITE subprogram.

LOG

Format
LOG(numeric-expression)

Description
The LOG function returns the natural logarithm of numeric-expression where numeric-expression is greater than zero. The LOG function is the inverse of the EXP function.

Examples
PRINT LOG(3.4) prints the natural logarithm of 3.4 which is 1.223775431622.

X = LOG(EXP(7.2)) sets X equal to the natural logarithm of e raised to the 7.2 power, which is 7.2.

S = LOG(SQR(T)) sets S equal to the natural logarithm of the square root of the value of T.

Program
The program at the right returns the logarithm of any positive number to any base.

>100 CALL CLEAR
>110 INPUT "BASE: "; B
>120 IF B<=1 THEN 110
>130 INPUT "NUMBER: "; N
>140 IF N<=0 THEN 130
>150 LG=LOG(N)/LOG(B)
>160 PRINT "LOG BASE ";B; "OF "; N; "IS ";LG
>170 GOTO 110
(Press SHIFT C to stop the program.)
**MAGNIFY subprogram**

**Format**

CALL MAGNIFY(magnification factor)

**Description**

The MAGNIFY subprogram allows you to specify the size of sprites and how many characters make up each sprite. All sprites are affected by MAGNIFY. Magnification factors may be 1, 2, 3, or 4. If no CALL MAGNIFY is in a program, the default magnification factor is 1.

A magnification factor of 1 causes all sprites to be single size and unmagnified. This means that each sprite is defined only by the character specified when the sprite was created and takes up just one character position on the screen.

A magnification factor of 2 causes all sprites to be single size and magnified. This means that each sprite is defined only by the character specified when it was created, but takes up four character positions on the screen. Each dot position in the character specified expands to occupy four dot positions on the screen. The expansion from a magnification factor of 1 is down and to the right.

A magnification factor of 3 causes all sprites to be double size and unmagnified. This means that each sprite is defined by four character positions that include the character specified. The first character is the one specified when the sprite was created if its number is evenly divisible by four, or the next smallest number that is evenly divisible by four. That character is the upper left quarter of the sprite. The next character is the lower left quarter of the sprite. The next character is the upper right quarter of the sprite. The final character is the lower right quarter of the sprite. The character specified when the sprite was created is one of the four that makes up the sprite. The sprite occupies four character positions on the screen.

A magnification factor of 4 causes all sprites to be double size and magnified. This means that each sprite is defined by four character positions that include the character specified. The first character is the one specified when the sprite was created if its number is evenly divisible by four, or the next smallest number that is evenly divisible by four. That character is the upper left quarter of the sprite. The next character is the lower left quarter of the sprite. The next character is the upper right quarter of the sprite. The final character is the lower right quarter of the sprite. The character specified when the sprite was created is one of the four that makes up the sprite. The sprite occupies sixteen character positions on the screen. The expansion from a magnification factor of 3 is down and to the right.
MAGNIFY subprogram

Program
The following program illustrates a use of the MAGNIFY subprogram. When it is run, a little figure appears near the center of the screen. In a moment, it gets to be twice as big, covering four character positions. In another moment, it is replaced by the upper left corner of a larger figure, still covering four character positions. Then the full figure appears, covering sixteen character positions. Finally it is reduced in size to four character positions.

Line 110 defines character 96.

Line 120 sets up a sprite using character 96. By default the magnification factor is 1.

Line 140 changes the magnification factor to 2.

Line 160 redefines character 96. Because the definition is 64 characters long, it also defines characters 97, 98, and 99.

Line 180 changes the magnification factor to 4.

Line 200 changes the magnification factor to 3.

>100 CALL CLEAR
>110 CALL CHA(96,’189FF3D3C’)
>120 CALL SPRITE(#1,96,5,92,1
>24)
>130 COSUB 230
>140 CALL MAGNIFY(2)
>150 COSUB 230
>160 CALL CHA(96,’013C3417F’
>3F07C70707C7’E7C40000000C000’
>80F0EE2E3E010E060606070’)
>170 COSUB 230
>180 CALL MAGNIFY(4)
>190 COSUB 230
>200 CALL MAGNIFY(3)
>210 COSUB 230
>220 STOP
>230 FEM DELA*
>240 FOR DELA*=1 TO 500
>250 NEXT DELA
>260 RETURN
**MERGE**

**Format**
MERGE ['"] device-filename ['"]

**Description**
The MERGE command merges lines in filename from the given device into the program lines already in the computer's memory. If a line number in filename duplicates a line number in the program already in memory, the new line replaces the old line. Otherwise the lines are inserted in line number order among the lines already in memory. The MERGE command does not clear breakpoints. Also, MERGE can only be used with diskettes.

NOTE: Files can only be merged into memory if they were saved using the MERGE option. See the SAVE command for more information.

**Example**
MERGE DSK1 SUB merges the program SUB into the program currently in memory.

**Program**
If the program on the right is saved on DSK1 as BOUNCE with the merge option, it can be merged with programs such as the one shown on the next page.

```ti
>MERGE DSK1.SUB

>100 CALL CLEAR
>110 FANOMIZE
>140 LET RND50=INT(RND*50-25)
>150 COSUB 10000
>1000 FOR AA=1 TO 20
>1001K QQ=RND50
>1002C LL=RND50
>1003C CALL MOTION(#1,QQ,LL)
>1004C NEXT AA
>1005C RETURN

>SAVE "DSK1.BOUNCE", MERGE
```

**On the right is a program you can put into the computer's memory:**

```ti
>120 CALL CHAR(76,"18183CFFFF"
>3C1818")
>130 CALL SPRITE(#1,96,7,92,1
>28)
>150 COSUB 500
>160 STOP

**Now merge BOUNCE with the above program.**

**The program that results from merging BOUNCE with the above program is shown on the right:**

```ti
>LIST
>100 CALL CLEAR
>110 FANOMIZE
>120 CALL CHAR(#6,"16183CFFFF"
>3C1818")
>130 CALL SPRITE(#1,96,7,92,1
>28)
>140 DEF RND50=INT(RND*50-25)
>150 COSUB 10000
>160 STCP
>10000 FOR AA=1 TO 20
>1001K QQ=RND50
>1002C LL=RND50
>1003C CALL MOTION(#1,QQ,LL)
>1004C NEXT AA
>1005C RETURN
```

**Note that line 150 is from the program that was merged, not from the program that was in memory.**
MIN

Format
MIN(numeric-expression1, numeric-expression2)

Description
The MIN function returns the smaller of numeric-expression1 and numeric-expression2. If they are equal then their value is returned.

Examples
PRINT MAX 3,8) prints 3.
F = MIN(3E12,1800000) sets F equal to 1800000.
G = MIN(-12,-4) sets G equal to -12.
L = MIN(A,B) sets L equal to -5 if A is 7 and B is -5.

MOTION subprogram

Format
CALL MOTION *sprite-number,row-velocity,column-velocity [...]

Description
The MOTION subprogram is used to specify the row-velocity and column-velocity of a sprite. If both the row- and column-velocities are zero, the sprite is stationary. A positive row-velocity moves the sprite down and a negative value moves it up. A positive column-velocity moves the sprite to the right and a negative value moves it to the left. If both row-velocity and column-velocity are nonzero, the sprite moves smoothly at an angle determined by the actual values.

The row- and column-velocities may be from -128 to 127. A value close to zero is very slow. A value far from zero is very fast. When a sprite comes to the edge of the screen, it disappears and reappears in the corresponding position on the other side of the screen.

Program
The program at the right illustrates a use of the MOTION subprogram.
Line 110 creates a sprite.

Lines 120 and 130 set values for the motion of the sprite.

Line 140 displays the current values of the motion of the sprite.

Line 150 sets the sprite in motion.

Lines 160 and 170 complete the loops that set the values for the motion of the sprite.
**NEW**

**Format**

NEW

**Description**

The NEW command clears the memory and screen and prepares the computer for a new program. All values are reset and all defined characters become undefined. Any open files are closed. Characters 32 through 95 are reset to their standard representations. The TRACE and BREAK commands are canceled.

Be sure to save the program that you have been working on before you enter NEW as it is unrecoverable by any means once NEW has been entered.

**NEXT**

**Format**

NEXT control-variable

**See** ON BREAK, ON WARNING, and RETURN (with ON ERROR) for the use of NEXT clause with those statements.

**Description**

The NEXT statement is always paired with the FOR-TO-STEP statement for construction of a loop. Control-variable must be the same as control-variable in the FOR-TO-STEP statement. The NEXT statement may not appear in an IF-THEN-ELSE statement.

The NEXT statement controls when the loop is repeated. Each time the NEXT statement is executed, control-variable is changed by the value following STEP in the FOR-TO-STEP statement or by 1 if there is no STEP clause. If the value of control-variable is between initial-value and limit, the loop is executed again. If it is not, control passes to the statement after NEXT. Thus the value of contro-variable at the end of the loop is always the first value outside the range of the FOR-TO-STEP statement. See FOR-TO-STEP for more information.

**Program**

The program on the right illustrates a use of the NEXT statement in lines 130 and 140.

```>
100 TOTAL=0
110 FOR COUNT=10 TO 0 STEP -2
120 TOTAL=TOTAL+COUNT
130 NEXT COUNT
140 FOR DELAY=1 TO 100 :NEXT
150 PRINT TOTAL;COUNT;DELAY
RUN
30 -2 101
```
NUMBER

Format
NUMBER [initial-line] [.increment]
NUM [initial-line] .increment]

Description
The NUMBER command generates sequenced line numbers, allowing entry of program lines without typing the line numbers. If initial-line and increment are not specified, the line numbers start at 100 and increase in increments of 10. You may give the command at any time in the Command Mode. If a line already exists, the current line is displayed. You may type over it to replace it, alter it using the edit functions, or press ENTER to confirm it. To leave the NUMBER mode, press ENTER when a line comes up with no statements on it or press SHIFT C (CLEAR) when any line is displayed. NUMBER may be abbreviated as NUM.

Options
You may specify an initial-line and/or increment.

Example
In the following, what you type is UNDERLINED. Press ENTER after each line.
NUM instructs the computer to number starting at 100 with increments of 10.

NUM 110 instructs the computer to number starting at 110 with increments of 10. Change line 110 to Z = 11.

NUM 105.5 instructs the computer to number starting at 105 with increments of 5.
Line 110 already exists.

OLD

Format
OLD ["] device-program-name ["]

Description
The OLD command loads program-name from device into memory. The program must first have been put on device using the SAVE command. CLD closes any open files and removes the program currently in memory before loading program-name. To add program lines from another program to a program in memory, see the MERGE command.

Device can be several different things. If it is CS1 or CS2, designating one of the two possible cassette recorders, then no program-name is given. The program loaded is the program that is on the cassette. Instructions on operating the cassette recorder are displayed on the screen.

See the Disk Memory System Manual for instructions on using OLD with diskettes.

Examples
OLD CS1 loads a program from a cassette recorder into the computer's memory.

OLD "DSK1.MYPROG" loads the program MYPROG into the computer's memory from the diskette in disk drive one.

OLD DSK3.UPDATE80 loads the program UPDATE80 into the computer's memory from the diskette named DISK3.
ON BREAK

Format
ON BREAK STOP
ON BREAK NEXT

Description
The ON BREAK statement determines the action taken if a breakpoint is encountered during the execution of a program. The default action is STOP, which causes program execution to halt and the standard breakpoint message to be printed. The alternative is NEXT, which transfers control to the next line without a breakpoint occurring.

You can use ON BREAK NEXT to have a program ignore breakpoints which you have put in a program for debugging purposes. (NOTE: ON BREAK NEXT does not have any effect on a BREAK statement which is not followed by a program line number. The breakpoint will occur even if the statement ON BREAK NEXT has been executed.) When ON BREAK NEXT is in effect, the external break, SHIFT C (CLEAR), does not stop a program. In that case only SHIFT Q (QUIT) can stop the program. SHIFT Q (QUIT) erases the program and returns you to the main screen and may interfere with the proper operation of some external devices such as disk drives.

Program
The program on the right illustrates the use of ON BREAK. Line 110 sets a breakpoint in line 150. Line 120 sets breakpoint handling to go to the next line. A breakpoint occurs in line 130 in spite of line 120. Enter CCNITINUE. No breakpoint occurs in line 150 because of line 120. SHIFT C (CLEAR) has no effect during the execution of lines 140 through 160 because of line 120. Line 170 restores the normal use of SHIFT C (CLEAR).

>100 CALL CLEAR
>110 BREAK 120
>120 ON BREAK NEXT
>130 BREAK
>140 FOR A=1 TO 50
>150 PRINT "SHIFT C IS DISABLED."
>160 NEXT A
>170 ON BREAK STOP
>180 FOR A=1 TO 50
>190 PRINT "NOW IT WORKS."
>200 NEXT A

ON ERROR

Format
ON ERROR STOF
ON ERROR line-number

Description
The ON ERROR statement determines the action taken if an error occurs during the execution of a program. The default action is STOP, which causes the standard error message to be printed and program execution to halt. The alternative is to give a line-number which transfers control to that line in case of an error.

Once an error has occurred and control has been transferred, error handling reverts to the normal action. STOP. If you wish to have any new errors handled differently, an ON ERROR statement must be executed again.

If a line-number is specified by ON ERROR, the line-number must be the beginning of a subroutine similar to that called by GOSUB. It should end with a RETURN statement. See RETURN (with ON ERROR) for more information.

NOTE: A transfer of control following the execution of an ON ERROR statement acts like the execution of a GOSUB statement. As with GOTO and GOSUB, you must avoid transfers to and from subprograms. The most common result of an illegal transfer into a subprogram is a syntax error on a statement that appears to be correct.
ON ERROR

Program
The program at the right illustrates the use of ON ERROR. Line 110 causes any error to pass control to line 160.
An error occurs in line 130 and control is passed to line 160.
Line 170 causes the next error to pass control to line 230. Line 180 finds out about the error using CALL ERR.
Line 190 transfers control to line 230 if the error isn’t the expected line.
Line 200 transfers control to line 230 if the error isn’t the one expected.
Line 210 changes the value of X$ to an acceptable value. Line 220 returns control to the line in which the error occurred.
Line 240 reports the nature of the unexpected error and the program stops.

>100 CALL CLEAR
>110 ON ERR 160
>120 K$="A"
>130 X$=VAL(K$)
>140 PRINT X$;"SQUARED IS";X$^2
>150 STOP
>160 REM ERRCR SUBROUTINE
>170 ON ERR 230
>180 CALL ERR(CODE,TYPE,SEVERITY,LINE)

>190 IF LINE<>130 THEN RETURN 230
>200 IF CODE=74 THEN RETURN 230
>210 K$="5"
>220 RETURN
>230 REM UNKNOWN ERRCR
>240 PRINT "ERROR";CODE;" IN LINE";LINE
>RUN

5 SQUARED IS 25

ON GOSUB

format
ON numeric-expression GOSUB line-number [...]
ON numeric-expression GO SUB line-number [...]

description
The ON...GOSUB statement transfers control to the subroutine beginning at line-number in the position corresponding to the value of numeric-expression. Other than giving a choice, it acts the same as the GOSUB statement, but it is more efficient in that it may require fewer lines of code than using an IF-THEN-ELSE statement.

Numeric-expression must have a value from 1 through the number of line-numbers.

Examples
ON X GOSUB 1000.2000.300
transfers control to 1000 if X is 1.
2000 if X is 2, and 300 if X is 3.
ON P-4 GOSUB 200.250.300.
800,170 transfers control to 200 if P is 1 (P is 5), 250 if P is 2, 300 if P is 3, 800 if P is 4, and 170 if P is 5.
ON GOSUB

Program
The program on the right illustrates a use of ON...GOSUB. Line 220 determines where to go according to the value of CHOICE.

>100 CALL CLEAR
>110 DISPLAY AT(11,1):"CHOOSE ONE OF THE FOLLOWING:"
>120 DISPLAY AT(13,1):"1 ADD TWO NUMBERS."
>130 DISPLAY AT(14,1):"2 MULTIPLY TWO NUMBERS."
>140 DISPLAY AT(15,1):"3 SUBTRACT TWO NUMBERS."
>150 DISPLAY AT(20,1):"YOUR CHOICE:"
>160 DISPLAY AT(22,2):"FIRST NUMBER:"
>170 DISPLAY AT(23,1):"SECOND NUMBER:"
>180 ACCEPT AT (20,14) VALIDATE
   : (NUMERIC):CHOICE
>190 IF CHOICE<1 OR CHOICE>3 THEN 180
>200 ACCEPT AT (22,16) VALIDATE
   : (NUMERIC):FIRST:
>210 ACCEPT AT (23,16) VALIDATE
   : (NUMERIC):SECOND:
>220 ON CHOICE GOSUB 240,260,280
>230 GOTO 180
>240 DISPLAY AT(3,1):FIRST:"PLUS";SECOND:"EQUALS";FIRST+SECOND
>250 RETURN
>260 DISPLAY AT(3,1):FIRST:"TIMES";SECOND:"EQUALS";FIRST*SECOND
>270 RETURN
>280 DISPLAY AT(3,1):FIRST:"MINUS";SECOND:"EQUALS";FIRST-SECOND
>290 RETURN
(Press 'SHIFT C' to stop the program.)

ON GOTO

Format
ON numeric-expression GOTO line-number; [...] 
ON numeric-expression GO TO line-number; [...]

Description
The ON...GOTO statement transfers control to the line-number in the position corresponding to the value of numeric-expression. Other than giving a choice, it acts the same as the GOTO statement, but it is more efficient in that it may require fewer lines of code than using an IF-THEN-ELSE statement.

Examples
ON X GOTO 1000,2000,3000=>10C ON X GOTO 1000,2000,3000
transfers control to 1000 if X is 1.
2000 if X is 2, and 3000 if X is 3. The equivalent statement using an IF-
THEN-ELSE statement is IF X = 1 THEN 1000 ELSE F X = 2 THEN
2000 ELSE IF X = 3 THEN 300 ELSE PRINT "ERROR!":STOP.

ON P:4 GOTO 200,250,300,800,170=>10C ON P:4 GOTO 200,250,300,800,170
transfers control to 200 if P = 1.
(If P = 5), 250 if P = 4 is 2, 300 if P = 4 is 3, 800 if P = 4 is 4, and 170 if P = 4 is 5.
ON GOTO

Program
The program on the right illustrates a use of ON..GOTO. Line 220 determines where to go according to the value of CHOICE.

>100 CALL CLEAR
>110 DISPLAY AT(11,1):"CHOOSE ONE OF THE FOLLOWING:"
>120 DISPLAY AT(13,1):"1 ADD TWO NUMBERS."
>130 DISPLAY AT(14,1):"2 MULTIPLY TWO NUMBERS."
>140 DISPLAY AT(15,1):"3 SUBTRACT TWO NUMBERS."
>150 DISPLAY AT(20,1):"YOUR CHOICE:"
>160 DISPLAY AT(22,2):"FIRST NUMBER:" 
>170 DISPLAY AT(23,1):"SECOND NUMBER:" 
>180 ACCEPT AT (20,14) VALIDATE (NUMERIC):CHOICE 
>190 IF CHOICE<1 OR CHOICE>3 THEN 180 
>200 ACCEPT AT (22,16) VALIDATE (NUMERIC):FIRST 
>210 ACCEPT AT (23,16) VALIDATE (NUMERIC):SECOND 
>220 ON CHOICE GOTO 230, 250, 270 
>230 DISPLAY AT(3,1):FIRST;"PLUS";SECOND;'EQUALS';FIRST+SECOND 
>240 GOTO 180 
>250 DISPLAY AT(3,1):FIRST;'TIMES';SECOND;'EQUALS';FIRST*SECOND 
>260 GOTO 180 
>270 DISPLAY AT(3,1):FIRST;'MINUS';SECOND;'EQUALS';FIRST-SECOND 
>280 GOTO 180 
(Press SHIFT-3 to stop the program.)

ON WARNING

Format
ON WARNING PRINT
ON WARNING STOP
ON WARNING NEXT

Description
The ON WARNING statement determines the action taken if a warning occurs during the execution of a program. The default action is PRINT, which causes the standard warning message to be printed and the program to continue execution. One alternative is STOP, which causes the standard warning message to be printed and the program to halt execution. The other alternative is NEXT which causes the program to continue execution without printing any message.

Program
The program on the right illustrates the use of ON WARNING. Line 110 sets warning handling to go to the next line. Line 120 therefore prints the result without any message.

Line 130 sets warning handling to the default, printing the message and then continuing execution. Line 140 therefore prints 140, then the warning, and then continues.

Line 150 sets warning handling to print the warning message and then stop execution. Line 160 therefore prints 160 and then the warning message and then stops.

>100 CALL CLEAR
>110 ON WARNING NEXT 
>120 PRINT 120, 5/0 
>130 ON WARNING PRINT
>140 PRINT 140, 5/0 
>150 ON WARNING STOP
>160 PRINT 160, 5/0 
>170 PRINT 170
>180 RUN
120 9.999992***
140 * WARNING NUMERIC OVERFLOW IN 140
160 * WARNING NUMERIC OVERFLOW IN 160
OPEN

**Format**

OPEN *file-number*|device|filename [file-organization] [file-type] [open-mode] [record-type]

**Description**

The OPEN statement prepares a BASIC program to use data files stored on a diskette or cassette by providing a link between file-number and a file. To set up this link, the OPEN statement describes a file's characteristics. If the file already exists, the description that is given in the program must match the actual characteristics of the file. Files on cassettes are not checked, however, so errors may occur if the characteristics do not match.

File-number must be included in the OPEN statement. Statements which refer to files do so with a file-number from 0 through 255. File number 0 is the keyboard and screen of the computer. It cannot be used for other files and is always open. You may assign the other numbers as you wish, with each file having a different number. File-number is entered as a number sign (#) followed by a numeric expression that, when rounded to the nearest integer, is a number from 0 to 255, and is not the number of a file that is already open.

Device must also be included in the OPEN statement. If device is CS1 or CS2, designating one of the two cassette recorders, then no filename is given. Instructions on operating the cassette recorder are displayed on the screen.

If device is DSK1, DSK2, or DSK3, designating one of the three disk drives, then filename is the name of a file on the diskette in the given drive. If device is DSK, diskette-name, where diskette-name is the name of a diskette in one of the drives, then filename is the name of a file on the diskette named diskette-name. The computer searches the drives, starting at DSK1, until it finds the diskette with the given name. Then it looks for filename on the other information may be in any order, or may be omitted. If an item is omitted, the computer assumes certain defaults, which are described below.

File-organization can be either sequential or random. Records in a sequential file are read or written one after the other. Records in random files can be read or written in any order. Random files may also be processed sequentially. To indicate which structure the file has, enter either SEQUENTIAL for sequential files or RELATIVE for random files. You may optionally specify the initial number of records on a file by following the word SEQUENTIAL or RELATIVE with a numeric expression. If you do not specify the file-organization, the default is SEQUENTIAL.

File-type may be either DISPLAY or INTERNAL. Files can be written either in human-readable form, called ASCII (DISPLAY), or in machine-readable form, called binary (INTERNAL). Binary records may take up less space and are processed more quickly by the computer. However, if the information is going to be printed or displayed, ASCII format is usually a better choice.

To specify that you wish the file to be in ASCII format, enter DISPLAY. To specify binary format, enter INTERNAL. If you do not specify a file-type, the default is DISPLAY. Usually INTERNAL is the best choice when using files on cassettes or diskettes. And DISPLAY is the best choice when using files on the thermal printer or RS232 interface.

Open-mode may be UPDATE, INPUT, OUTPUT, or APPEND. The computer may be instructed that the file may be both read and written on, that it may only be read, that it may only be written on, or that it may only be added to. However, if the file is marked as protected, it cannot be written on and may only be opened for input.

To be able both to read from and write to a file, specify UPDATE. To just read from a file, specify INPUT. To just write to a file, specify OUTPUT. To only add to a file, specify APPEND. Append mode can only be specified for VARIABLE length records. If you do not specify an open-mode, the default is UPDATE.

Note that if an unprotected file already exists on a diskette, specifying an open-mode of OUTPUT to the same file name writes over the existing file with the new data. You can prevent this by moving to the end of the file by using the RESTORE statement with the proper record or opening the file in the APPEND mode.

Record-type may be either VARIABLE or FIXED. Files may have records that are all the same length or that vary in length. If all records are the same length, any that are shorter are padded to make up the difference. Any that are longer may be truncated to the proper length. You may specify records of variable length by entering VARIABLE. You specify records of fixed length by entering FIXED.

If you like, you may specify a maximum length of a record by following VARIABLE or FIXED with a numeric expression. The maximum record is dependent on the device used. If you do not specify a record length, the default is 80 for diskettes, 64 for cassettes, 80 for the RS232 interface, and 32 for the thermal printer.

RELATIVE files must have FIXED length records. If you do not specify a record-type for a RELATIVE file, the default is FIXED.
OPEN

SEQUENTIAL files may be either FIXED or VARIABLE. If you do not specify a record-type for a SEQUENTIAL file, the default is VARIABLE. A fixed-length file may be reopened for either SEQUENTIAL or RELATIVE access independent of previous file-organization assignments.

Examples
OPEN #1: "CS1", FIXED, OUTPUT
opens a file on cassette one. The file is SEQUENTIAL, kept in DISPLAY
format, in OUTPUT mode with FIXED length records with a maximum length of 64 bytes.

OPEN #23: "DSK.MYDISK.X."
RELATIVE 100, INTERNAL, UPDATE
FIXED opens a file named "X". The
file is on the diskette named MYDISK
in whichever drive that diskette is
located. The file is RELATIVE, kept
in INTERNAL format with FIXED
length records with a maximum
Length of 80 bytes. The file is
opened in UPDATE mode and starts
with 100 records made available for it.

OPEN #243: AS, INTERNAL, if AS
equals "DSK2.ABC". assumes a file
on the diskette in drive two with a
name of ABC. The file is

format, in UPDATE mode with
VARIABLE length records with a
maximum length of 80 bytes

OPEN #17: "TP", OUTPUT prepares
the thermal printer for printing.

OPTION BASE

Format
OPTION BASE 0
OPTION BASE 1

Description
The OPTION BASE statement sets the lowest allowable subscript of arrays to zero or one. The default is zero. If an OPTION BASE statement is used, it must have a lower line number than any DIM statement or reference to an array. There may only be one OPTION BASE statement in a program, and it applies to all array subscripts. The OPTION BASE statement may not appear in an IF-THEN-ELSE statement.

Example
OPTION BASE 1 sets the lowest allowable subscript of all array/s to one.

>100 OPTION BASE 1

>100 OPEN #1: "CS1", FIXED, OUTPUT
**PATTERN subprogram**

**Format**

CALL PATTERN(#sprite-number,character-value [,...,])

**Description**

The PATTERN subprogram allows you to change the character pattern of a sprite without affecting any other characteristics of the sprite.

*Sprite-number* specifies the sprite you are using. *Character-value* may be any integer from 32 to 143. See the CHAR subprogram for information on defining the pattern for characters. See the MAGNIFY subprogram for more information.

**Program**

The program on the right illustrates the use of the PATTERN subprogram. Lines 110 through 140 build a floor.

Lines 150 through 200 define characters 96 through 107.

>100 CALL CLEAR
>110 CALL COLOR(12,16,16)
>120 FOR A=19 TO 24
>130 CALL HCHAR(A,1,120,32)
>140 NEXT A

>150 1$="01071B21D4.41FF41F12111907000E0C98484828FF"  
FF83484848583000  
>160 3$="01061B20305C46818142"  
4624E18070030601B3426248281  
8162A0C841B30000  
>170 0$="0106182C44642818146"  
5C302018070030601B04C3A6281  
8142E2443B83000  
>180 CALL CHAR(96,A$)
>190 CALL CHAR(100,B$)
>200 CALL CHAR(104,C$)
>210 CALL SPRITE(#1,96,5,130,1,0,8)

Line 210 creates a sprite in the shape of a wheel and starts it moving.

Line 220 makes the sprite double size.

Lines 230 through 270 make the spokes of the wheel appear to move as the character displayed is changed.

Also see the third example of the SPRITE subprogram.

---

**PEEK subprogram**

**Format**

CALL PEEK(address,numeric-variable-list)

**Description**

The PEEK subprogram is used, along with INIT, LINK, and LOAD to access assembly language subprograms. The PEEK subprogram returns values in the variables in numeric-variable-list that correspond with the values in the byte specified by address and the bytes following it. PEEK can be used without assembly language subprograms, but the information obtained is of little use.

Detailed instructions on the use of INIT, LINK, LOAD, and PEEK are included with custom written programs that may be available on diskette or cassette.

Indiscriminate use of PEEK may cause the computer to "lock up" and require it to be turned off and back on before further use.

**Example**

CALL PEEK(8192,X1,X2,X3,X4)  
>100 CALL PEEK(8192,X1,X2,X3,X4)

8193, 8194, and 8195 in X1, X2, X3, and X4, respectively.
Format
PI

Description
The PI function returns the value of \( \pi \) as 3.14159265359.

Example
\[
\text{VOLUME} = \frac{4}{3} \pi r^3 \text{ sets VOLUME equal to four thirds times } \pi \text{ times six cubed, which is the volume of a sphere with a radius of six.}
\]

\[
>100 \text{ VOLUME} = \frac{4}{3} \pi 6^3
\]
POSITION subprogram

Format
CALL POSITION(#sprite-number,.dot-row,.dot-column [,...] )

Description
The POSITION subprogram returns the position of the specific sprite(s) in
the given .dot-row(s) and .dot-column(s) as numbers from 1 to 256. They are
the position of the upper left corner of the sprite. If the sprite is not defined,
dot-row and dot-column are set to zero.

The sprite continues to move after its position is returned, so that must be
allowed for. The distance moved depends on the sprite's speed.

Example
CALL POSITION(#1,.Y,.X) returns the
position of the upper left hand corner
of sprite #1.

Also see the third example of the
SPRITE subprogram

PRINT

Format
PRINT [*file-number ,REC record-number] [:] [print-list]

Description
The PRINT statement allows you to transfer the values of the elements of the
print-list to the display screen or optionally to an external file or
device. Print-list consists of string constants, numeric constants, string
variables, numeric variables, numeric expressions, string expressions, and/or
TAB function. Each element in print-list is separated from the others by
semicolon, comma, or colon. The semicolon, comma, and colon control spacing for the screen or a file
when in DISPLAY format. A semicolon causes the next element to be
printed immediately adjacent to the previous element. A comma causes the
next element of print-list to be put in the next print field. Each print field is
limited to 80 characters long. The number of print fields depends on the record length
of the device being used. On the screen, the print fields are at positions 1 and
If the cursor is past the start of the last print field, the next item is
printed on the next line. A colon causes the next element to be put on the
next line or record. To print several blank lines, you may put several colons
after the PRINT statement. However, they must have spaces between them
or they are not confused with the statement separator symbol (:).

A separator may be placed following the last element of print-list, which
causes the placement of the next element of the next PRINT PRINT...,USING,
DISPLAY (without AT), or DISPLAY USING (without AT) statement written
at the same device. It causes the next output statement to be considered
to be a continuation of the current one unless it is a PRINT statement with a
RECORD clause.

When printing a new line on the screen, everything (except sprites) is
tracked up one line (so the top line is lost) and the new line is printed at the
bottom of the screen.

Options
The #file-number determines the file that is to be printed on. If it is omitted
or #0, the screen is used. Otherwise file-number must be the number of a file
that is already open. See OPEN.

The RECORD clause is used to specify the record on which you wish to print the
elements in print-list. RECORD may only be used with files that were opened as
RELATIVE files. See OPEN.
In printing to INTERNAL format files, the comma and semicolon both place the elements in print-list adjacent to each other. In DISPLAY format files, the comma and semicolon act as described above, with the semicolon placing the element adjacent to the previous element and the comma putting the element in the next print field.

**Examples**

**PRINT causes a blank line to appear on the display screen.**

PRINT "THE ANSWER IS"; ANSWER causes the string constant THE ANSWER IS to be printed on the display screen, followed immediately by the value of ANSWER. If ANSWER is positive, there will be a blank for the positive sign after IS.

PRINT X:Y/2 causes the value of X to be printed on a line and the value of Y/2 to be printed on the next line.

PRINT #12,REC 7:A causes the value of A to be printed on the eighth record of the file that was opened as number 12. (Record number 0 is the first record.)

PRINT #32 A,B,C, causes the values of A, B, and C to be printed on the next record of the file that was opened as number 32. The final comma creates a pending output condition. The next PRINT statement directed to file number 32 will print on the same record as this PRINT statement unless it specifies a record, thereby closing the pending output condition.

**PROGRAM**

The program at the right prints out various values in various positions on the display screen.

- >100 PRINT #1,REC 3:A,B
- >100 PRINT #1,C,D
- >100 PRINT #1,REC 3:A,B
- >100 PRINT #1,C,D
- >100 CALL CLEAR
- >110 PRINT 1;2,3,4;5;6,7,8;9
- >120 PRINT 1,2,3,4,5,6
- >130 PRINT 1:2,3
- >140 PRINT
- >150 PRINT 1;2,3;
- >160 PRINT 4;5,6,4

RUN

```
1 2 3 4 5 6 7 8 9
1 2
3 4
5 6
1 2
2 3
```

>1 2 3 4 5 1.5
PRINT USING

**Format**
PRINT ["file-number",REC record-number]|USING string-expression:print-list
PRINT |"file-number",REC record-number]|USING line-number:print-list

**Description**
The PRINT...USING statement acts the same as PRINT with the addition of the USING clause, which specifies the format to be used. *String-expression* defines the format in the manner described in IMAGE. *Line-number* refers to the line number of an IMAGE statement. See the IMAGE statement for more information on the use of *string-expression*.

**Examples**
PRINT USING "###.###":2.5 prints 32.50.
PRINT USING "THE ANSWER IS ###.###":123.98 prints THE ANSWER IS 123.98
PRINT USING 185.37.4,-86.2 prints the values of 37.4 and -86.2 using the IMAGE statement in line .85.

RANDOMIZE

**Format**
RANDOMIZE [numeric-expression]

**Description**
The RANDOMIZE statement resets the random number generator to an unpredictable sequence. If RANDOMIZE is followed by a numeric-expression, the same sequence of random numbers is produced each time the statement is executed with that value for the expression. Different values give different sequences.

**Program**
The program at the right illustrates a use of the RANDOMIZE statement. It accepts a value for numeric-expression and prints the first 10 values obtained using the RND function.

>100 CALL CLEAR
>110 INPUT "SEED: ":S
>120 RANDOMIZE S
>130 FOR A=1 TO 10:PRINT A;RND
>140 GOTO 110
(Press SHIFT C to stop the program.)
**RESEQUENCE**

**Format**

RESEQUENCE [initial-line] [,increment]
RES [initial-line] [,increment]

**Description**

The RESEQUENCE command changes the line numbers of the program in memory. If no initial-line is given, the line numbering starts with '00. If no increment is given, an increment of 10 is used. RESEQUENCE may be abbreviated as RES.

In addition to renumbering lines, any line references in the statements BREAK, DISPLAY...USING, GOSUB, GOTO, IF-THEN-ELSE, ON ERROR, ON...GOSUB, ON...GOTO, PRINT...USING, RESTORE, RETURN, and RUN are also changed so that they refer to the same lines of code as before resequencing. If a line referred to in a statement does not exist, the line number is replaced with 32767.

If, because of the initial-line and increment chosen, the program requires lines larger than 32767, the resequencing process is halted and the program is unchanged.

**Examples**

RES resequences the lines of the program in memory to start with 100 and number by 10's.

RES ,100 resequences the lines of the program in memory to start with 1000 and number by 10's.

RES 1000,15 resequences the lines of the program in memory to start with 1000 and number by 15's.

RES 15 resequences the lines of the program in memory to start with 100 and number by 15's.

RES >RES

RES >RES 1000

RES >RES 1000,15

RES >RES ,15
**RESTORE**

**Format**
RESTORE [line-number]
RESTORE #file-number [, REC record-number]

**Description**
The RESTORE statement can be used either with DATA statements or with files. When used with DATA statements, RESTORE sets the DATA statement which will be used by the next READ statement. If no line-number is given, the DATA statement with the lowest numbered line is used by the next READ statement. If line-number is given, then the DATA statement with that line number or (if it is not a DATA statement) the next DATA statement following it is used.

When used with files, the RESTORE statement sets the record that is used by the next PRINT, INPUT, or LINPUT statement referring to file-number. If no REC clause is given, the next record is the first record in the file, record number 0. If the REC clause is present, record-number specifies the next record to be used.

If there is pending output because of a previous PRINT, DISPLAY, PRINT...USING, or DISPLAY...USING, then that pending record is written on the file before the RESTORE statement is executed. Pending input data is removed by the RESTORE statement.

**Examples**
RESTORE sets the next DATA statement to be used in the first DATA statement in the program.

RESTORE 120 sets the next DATA statement to be used in the DATA statement at line 120 or, if line 120 is not a DATA statement, to the next DATA statement after line 120.

RESTORE #1 sets the next record to be used by the next PRINT, INPUT, or LINPUT statement using file #1 to be the first record in the file.

RESTORE #4, REC H5 sets the next record to be used by the next PRINT, INPUT, or LINPUT statement using file #4 to be record H5.

---

**RETURN (with GOSUB)**

**Format**
RETURN

**Description**
See also RETURN (with ON ERROR).

RETURN used with GOSUB transfers control back to the statement after the GOSUB or ON...GOSUB which was most recently executed.

**Program**
The program on the right illustrates a use of RETURN as used with GOSUB. The program figures interest on an amount of money put in savings.

```ti-basic
>100 CALL CLEAR
>110 INPUT "AMOUNT DEPOSITED: " : AMOUNT
>120 INPUT "ANNUAL INTEREST RATE: " : RATE
>130 IF RATE<1 THEN RATE=RATE * 100
>140 PRINT "NUMBER OF TIMES COMPOUNDED" 
>150 INPUT "ANNUAL: " : COMP
>160 INPUT "STARTING YEAR: " : Y
>170 INPUT "NUMBER OF YEARS: " : N
>180 CALL CLEAR
>190 FOR A=Y TO Y+N
>200 GOSUB 240
>210 PRINT A, INT(A*100+1)/100
>220 NEXT A
>230 STOP
>240 FOR B=1 TO COMP
>250 AMOUNT+=AMOUNT*AMOUNT*FAT / (COMP*100)
>260 NEXT B
>270 RETURN
```
RETURN (with ON ERROR)

**Format**
RETURN [line-number]
RETURN [NEXT]

**Description**
See also RETURN (with GOSUB).

RETURN is used with ON ERROR. After an ON ERROR statement has been executed, an error causes transfer to the line specified in the ON ERROR statement. That line, or one after it, should be a RETURN statement. If RETURN is given without anything following it, control is returned to the statement on which the error occurred and the program executes it again. If RETURN is followed by line-number, control is transferred to the line specified and execution starts with that line. If RETURN is followed by NEXT, control is transferred to the statement following the one that caused the error.

**Program**
The program on the right illustrates the use of RETURN with ON ERROR. Line 120 causes an error to transfer control to line 170. Line 130 causes an error. Line 140, the next line after the one that causes the error, prints 140. Line 170 checks to see if the error has occurred four times and transfers control to 220 if it has. Line 186 increments the error counter by one. Line 190 prints 190. Line 200 resets the error handling to transfer to line 170. Line 210 returns to the line that caused the error and executes it again. Line 220, which is executed only after the error has occurred four times, prints 220 and returns to the line following the one that caused the error.

Also see the example of the ON ERROR statement.

**RND**

**Format**
RND

**Description**
The RND function returns the next pseudo-random number in the current sequence of pseudo-random numbers. The number returned is greater than or equal to zero and less than one. The sequence of random numbers returned is the same every time a program is run unless the RANDOMIZE statement appears in the program.

**Examples**
COLOR16 = INT(RND*16) + 1 sets >100 COLOR16=INT(RND*16)+1
COLOR16 equal to some number from 1 through 16.
VALUE = INT(RND*16) + 10 sets >100 VALUE=INT(RND*16)+10
VALUE equal to some number from 10 through 25.
LL(8) = INT(RND* (B - A + 1)) + A sets >100 LL(8)=INT(RND*(B-A+1))+A
LL(8) equal to some number from A through B.
**RPT$**

**Format**

RPT$(string-expression, numeric-expression)

**Description**

The RPT$ function returns a string equal to numeric-expression repetitions of string-expression. If RPT$ produces a string longer than 255 characters, the excess characters are discarded and a warning is given.

**Examples**

M$ = RPT$("ABCD", 4) sets M$ equal to "ABCDABCDABCDABCD".

CALL CHAR(96, RPT$("0000FFFF", 8)) defines characters 96 through 99 with the string "0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF".

PRINT USING RPT$("#", 40) prints the value of X$ using an image that consists of 40 number signs.

---

**RUN**

**Format**

RUN "device.program-name"

RUN [line-number]

**Description**

The RUN command, which can also be used as a statement, starts program execution. The program to be run is first loaded into memory from device.program-name if that option is specified. The program is then checked for certain errors, such as FOR NEXT loops that are missing the NEXT statement, and errors in syntax in statements. The values of all numeric variables are set to zero and the values of all string variables are set to null (a string of zero characters). The program is then executed.

**Options**

If device.program-name is specified, the program to be run is based from the specified device. The program and data currently in memory are lost.

If line-number is specified, the program in memory is run starting at line-number.

**Examples**

RUN causes the computer to begin execution of the program in memory.

RUN 200 causes the computer to begin execution of the program in memory starting at line 200.

RUN "DSK1:PRG3" causes the computer to load and begin execution of the program named PRG3 from the diskette in disk drive 1.

RUN "DSK1:PRG3"
Program
The program at the right illustrates the use of the RUN command used as a statement. It creates a "menu" and lets the person using the program choose what other program he wishes to run. The other programs should RUN this program rather than ending in the usual way, so that the menu is given again after they are finished.

>100 CALL CLEAR
>110 PRINT "1 PROGRAM 1."
>120 PRINT "2 PROGRAM 2."
>130 PRINT "3 PROGRAM 3."
>140 PRINT "4 END."
>150 PRINT
>160 INPUT "YOUR CHOICE: ":C
>170 IF C=1 THEN RUN "DSK1.PF.G1"
>180 IF C=2 THEN RUN "DSK1.PF.G2"
>190 IF C=3 THEN RUN "DSK1.PF.G3"
>200 IF C=4 THEN STOP
>210 JOTOC 13C

SAVE

Format
SAVE device:program-name [.PROTECTED]
SAVE device:program-name [.MERGE]

Description
The SAVE command allows you to copy the program in memory to an external device under the name program-name. By using the OLD command, you can later recall the program into memory. The method of saving onto a cassette recorder is given in the User's Reference Guide. The method of saving onto a diskette is given in the Disk Memory System manual. SAVE clears breakpoints that have been put into a program.

Options
Only the PROTECTED option is available with cassette recorders.

By using the keyword PROTECTED, you may optionally specify that a program can only be run or brought into memory with OLD. The program cannot be listed, edited, or saved. This is not the same as using the protection available with the Disk Manager Module. NOTE: Be sure to keep an unprotected copy of any program because the protection feature is not reversible. If you also wish to protect the program from being copied, use the protect feature of the Disk Manager module.

You may optionally specify that the program is to be available for merging with another program by using the keyword MERGE. Only programs saved with the keyword MERGE may be merged with another program.

Examples
SAVE DSK1.PRG1 saves the program in memory on the diskette in disk drive 1 under the name PRG1.

SAVE DSK1.PRG1,PROTECTED saves the program in memory on the diskette in disk drive 1 under the name PRG1. The program may be loaded into memory and run, but it may not be edited, listed, or resaved.

SAVE DSK1.PRG1,MERGE saves the program in memory on the diskette in disk drive 1 under the name PRG1. The program may later be merged with a program in memory by using the MERGE command.
SAY subprogram

Format
CALL SAY(\textit{word-string} [,\textit{direct-string}] [....])

Description
The SAY subprogram causes the computer to speak \textit{word-string} or the value specified by \textit{direct-string} when the \textit{Solid State Speech}™ Synthesizer (sold separately) is connected. For a complete description of SAY, see the manual that comes with the Speech Editor Command Module and Speech Synthesizer (both sold separately).

The value of \textit{word-string} is any string value listed in Appendix L. If it is given as a literal value, it must be enclosed in quotation marks. The value of \textit{direct-string} is a value returned by SFGET. The value of \textit{direct-string} may be altered to add suffixes as described in Appendix M.

\textit{Word-strings} and \textit{direct-strings} must be alternated in the CALL SAY subprogram. If you wish to have two \textit{direct-strings} or \textit{word-strings} spoken consecutively, you may put in an extra comma to indicate the position of the item omitted.

Examples
CALL SAY("HELLO, HOW ARE YOU") causes the computer to say "Hello, how are you."

CALL SAY(\texttt{AS}, \texttt{BS}) causes the computer to say the words indicated by \texttt{AS} and \texttt{BS}, which must have been returned by SFGET.

Program
The program on the right illustrates using CALL SAY with a \textit{word-string} and three \textit{direct-strings}.

>100 CALL SAY("HOW", \texttt{X$})
>110 CALL SFGET("ARE", \texttt{Y$})
>120 CALL SFGET("YOU", \texttt{Z$})
>130 CALL SAY("HELLO", \texttt{X$}, \texttt{Y$}, \texttt{Z$})

SCREEN subprogram

Format
CALL SCREEN(\textit{color-code})

Description
The SCREEN subprogram changes the color of the screen to the color specified by \textit{color-code}. All portions of the screen that do not have characters on them, or have characters or portions of characters that are color 1 (transparent), are shown as the color specified by \textit{color-code}. The standard screen color for TI Extended BASIC is 8 cyan.

The color codes are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Color</th>
<th>Code</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Transparent</td>
<td>9</td>
<td>Medium Red</td>
</tr>
<tr>
<td>2</td>
<td>Black</td>
<td>10</td>
<td>Light Red</td>
</tr>
<tr>
<td>3</td>
<td>Medium Green</td>
<td>11</td>
<td>Dark Yellow</td>
</tr>
<tr>
<td>4</td>
<td>Light Green</td>
<td>12</td>
<td>Light Yellow</td>
</tr>
<tr>
<td>5</td>
<td>Dark Blue</td>
<td>13</td>
<td>Dark Green</td>
</tr>
<tr>
<td>6</td>
<td>Light Blue</td>
<td>14</td>
<td>Magenta</td>
</tr>
<tr>
<td>7</td>
<td>Dark Red</td>
<td>15</td>
<td>Grey</td>
</tr>
<tr>
<td>8</td>
<td>Cyan</td>
<td>16</td>
<td>White</td>
</tr>
</tbody>
</table>

Examples
CALL SCREEN(8) changes the screen to cyan, which is the standard screen color.

CALL SCREEN(2) changes the screen to black.
**SEG$**

**Format**

SEG$(string-expression, position, length)

**Description**

The SEG$ function returns a substring of a string. The string returned starts at position in string-expression and extends for length characters. If position is beyond the end of string-expression, the null string (""") is returned. If length extends beyond the end of string-expression, only the characters to the end are returned.

**Examples**

X$ = SEG$("FIRSTNAME", 1, 5)

Y$ = SEG$("LASTNAME", 1, 9)

Z$ = SEG$("NAME", 1, 10)

PRINT SEG$(X$, Y$, Z$)

100 SEG$("A", B, C)

**SGN**

**Format**

SGN(numeric-expression)

**Description**

The SGN function returns 1 if numeric-expression is positive, 0 if it is zero, and -1 if it is negative.

**Examples**

100 IF SGN(X$) = 1 THEN 300 ELSE 400

100 ON SGN(X$) = 2 GOTO 200, 300, 400
Format
SIN(radian-expression)

Description
The sine function gives the trigonometric sine of radian-expression. If the angle is in degrees, multiply the number of degrees by PI/180 to get the equivalent angle in radians.

Program
The program on the right gives the sine of several angles.

>100 A=.5233987755982
>110 B=30
>120 C=45*PI/180
>130 PRINT SIN(A);SIN(B)
>140 PRINT SIN(B*PI/180)
>150 PRINT SIN(C)
>RUN
.5 .9880316241
.7071067812

Example
SIZE gives the available memory.

>SIZE
13928 BYTES FREE

SIZE gives the available memory. If the Memory Expansion peripheral is attached, stack and program space are given.

>SIZE
13928 BYTES OF STACK FREE
245.1 BYTES OF PROGRAM SPACE FREE
SOUND subprogram

Format
CALL SOUND(duration, frequency1, volume1 [ ...frequency4, volume4])

Description
The SOUND subprogram tells the computer to produce tones or noise. The values given control three aspects of the sound: Duration, frequency, and volume.

<table>
<thead>
<tr>
<th>Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>1 to 4250</td>
<td>The length of the sound in thousands of a second.</td>
</tr>
<tr>
<td></td>
<td>-1 to -4250</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Tone: 110 to 44733</td>
<td>What sound is played.</td>
</tr>
<tr>
<td></td>
<td>Noise: -1 to -8</td>
<td>How loud the sound is.</td>
</tr>
<tr>
<td>Volume</td>
<td>0 to 30</td>
<td></td>
</tr>
</tbody>
</table>

Duration is from .001 to 4.250 seconds, although it may vary up to 1/60th of a second. The computer continues performing program statements while a sound is being played. When you call the SOUND subprogram, the computer waits until the previous sound has been completed before performing the new CALL SOUND. However, if a negative duration is specified, the previous sound is stopped and the new one begins immediately.

Frequency specifies the frequency of the note to be played with a value from 110 to 44733. (Note: This range goes higher than the range of human hearing. People vary in their ability to hear high notes, but generally the highest is approximately a value of 10000.) The actual frequency produced by the computer may vary up to 10 percent. Appendix D lists the frequencies of some common notes.

A value of -1 to -8 specifies one of eight different types of noise:

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Periodic Noise Type 1</td>
</tr>
<tr>
<td>-2</td>
<td>Periodic Noise Type 2</td>
</tr>
<tr>
<td>-3</td>
<td>Periodic Noise Type 3</td>
</tr>
<tr>
<td>-4</td>
<td>Periodic Noise that varies with the frequency of the third tone specified</td>
</tr>
<tr>
<td>-5</td>
<td>White Noise: Type 1</td>
</tr>
<tr>
<td>-6</td>
<td>White Noise: Type 2</td>
</tr>
<tr>
<td>-7</td>
<td>White Noise: Type 3</td>
</tr>
<tr>
<td>-8</td>
<td>White Noise that varies with the frequency of the third tone specified</td>
</tr>
</tbody>
</table>

A maximum of three tones and one noise can be played simultaneously.

Volume specifies the loudness of the note or noise. Zero is loudest and 30 is softest.

Examples
CALL SOUND(1000, 110, 0) plays A below low C loudly for one second.
CALL SOUND(500, 110, 0, 131, 196, 0, 196, 3) plays A below low C and low C loudly, and G below C not as loudly, all for half a second.
CALL SOUND(4250, -8, 0) plays loud white noise for 4250 seconds.
CALL SOUND(DUR, TONE, VOL) plays the tone indicated by TONE for a duration indicated by DUR, at a volume indicated by VOL.

Program
The program on the right plays the Th notes of the first octave that is available on the computer.

```plaintext
>100 CALL SOUND(1000, 110, 0)
>100 CALL SOUND(500, 110, 0, 131, 196, 0, 196, 3)
>100 CALL SOUND(4250, -8, 0)
>100 CALL SOUND(DUR, TONE, VOL)
>100 X=2^((1/12)*A) + A
>100 FOR A=1 TO 13
>120 CALL SOUND(100, 11) * X, A, 0
>130 NEXT A
>130 NEXT A
```
SPGET subprogram

**Format**

CALL SPGET("word-string,.return-string")

**Description**

The SPGET subprogram returns in return-string the speech pattern that corresponds to word-string. For a complete description of SPGET, see the manual that comes with the Speech Editor Command Module and Solid State Speech™ Synthesizer (both sold separately).

The value of word-string is any string value listed in Appendix A. If it is given as a literal value, it must be enclosed in quotation marks. The value of return-string is used with SAY, and may be altered to add suffixes as described in Appendix M.

**Program**

The program on the right illustrates using CALL SPGET.

```ti
>100 CALL SPGET("HOW",X$)
>110 CALL SPGET("ARE",Y$)
>120 CALL SPGET("YOU",Z$)
>130 CALL SAY("HELLO",X$,Y$,Z$)
```

SPRITE subprogram

**Format**

CALL SPRITE("sprite-number,.character-value,.sprite-color,.dot-row,.dot-column,.row-velocity,.column-velocity",...)

**Description**

The SPRITE subprogram creates sprites. Sprites are graphics which have a color and a location anywhere on the screen. They can be set in motion in any direction at a variety of speeds, and continue their motion until it is changed by the program or the program stops. They move more smoothly than the usual character which jumps from one screen position to another.

Sprite-number is a numeric expression from 1 to 28. If the value is that of a sprite that is already defined, the old sprite is deleted and replaced by the new sprite. If the old sprite has a row- or column-velocity, and no new one is specified, the new sprite retains the old velocities.

Sprites pass over fixed characters on the screen. When two or more sprites are coincident, the sprite with the lowest sprite number covers the other sprites. While five or more sprites are on the same screen row, the one(s) with the highest sprite number(s) disappear.

Character-value may be any integer from 32 to 143. See the CHAR subprogram for information on defining characters. The character-value can be changed by the PATTERN subprogram. The sprite is defined as the character given and, in the case of double-sized sprites, the next three characters. See the MAGNIFY subprogram for more information.

Sprite-color may be any numeric expression from 1 to 16. It determines the foreground color of the sprite. The background color of a sprite is always 1, transparent. See the COLOR and SCREEN subprograms for more information.

Dot-row and dot-column are numbered consecutively starting with 1 in the upper left hand corner of the screen. Dot-row can be from 1 to 192 and dot-column can be from 1 to 256. (Actually dot-row can go up to 256, but the positions from 193 through 253 are off the bottom of the screen.) The position of the sprite is the upper left hand corner of the character(s) which define it.

Information about the position of a sprite can be found using the POSITION, COINC, and DISTANCE subprograms. The location of a sprite can be changed using the LOCATE subprogram. COLOR changes the color of a sprite. Sprites can be deleted with the DELSPRITE subprogram.

When a breakpoint occurs or the program stops, sprites cease to exist. They do not reappear with CONTINUE.
SPRITE subprogram

Options

Row-velocity and column-velocity may optionally be specified when the sprite is created. If both row- and column-velocity are zero, the sprite is stationary. A positive row-velocity moves the sprite down and a negative value moves it up. A positive column-velocity moves the sprite to the right; and a negative value moves it to the left. If both row-velocity and column-velocity are non-zero, the sprite moves at an angle in a direction determined by the actual values.

Row- and column-velocity may be from -128 to 127. A value close to zero is very slow. A value far from zero is very fast. When a sprite comes to the edge of the screen, it disappears and reappears in the corresponding position on the other side of the screen. The velocity of a sprite may be changed using the MOTION subprogram.

Programs

The following three programs show some possible uses of sprites. The third one uses all the subprograms that can relate to sprites except for COLOR and DISTANCE.

Line 140 creates a dark blue sprite in the center of the screen and a dark red sprite in the upper left corner of the screen. Line 150 creates a white sprite near the upper right corner of the screen and starts it moving slowly at a 45 degree angle down and to the right. The sprite is an exclamation point.

Line 160 creates a sprite at the upper left corner of the screen and starts it moving very fast at a 45 degree angle up and to the right.

>100 CALL CLEAR
>110 CALL CHAR(96,"FFFFFF"
>120 CALL CHAR(98,"8888888"
>130 CALL CHAR(100,"0000"
>140 CALL SPRITE(#1,96,5,92,1
>24,#2,100,7,1,1)
>150 CALL SPRITE(#28,3),16,12
>160 CALL SPRITE(#15,96,14,1,
>1,127,-126)
>170 GOTO 170
(Press SHIFT C to stop the program.)

>110 CALL CHAR(96,"0008BIC7F"
>120 RANDOMIZE
>130 CALL SCREEN(2)
>140 FOR A=1 TO 28
>150 CALL SPRITE(#A,96,INT(A/3)
>+.5),3,92,124,A*INT(RND*.5)-2
>.25+1/2*SGN(RND-.5),A*INT(RND
>.5)-2.25+1/2*SGN(RND-.5))
>160 NEXT A
>170 GOTO 140

The program on the right makes a rather spectacular use of sprites. Line 110 defines character 96
Line 150 defines the sprites. 28 in all. The sprite-number is the current value of A. The character-value is
96. The sprite-color is INT(A/3)-3.
The starting dot-row and dot-column are 92 and 124, the center of the screen. The row- and column-
velocities are chosen randomly using the value of A*INT(RND*4.5)
-2.25+1/2*SGN(RND-.5). Line
170 causes the sequence to repeat.

The following program uses all he subprograms that can relate to sprites except for COLOR and DISTANCE. They are CHAR, COINC, DELSPRITE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, and SPRITE.

The program creates two double sized magnified sprites in the shape of a person walking along a floor. There is a barrier that one of them passes through and the other jumps through. The one that jumps through goes a little faster after each jump, so eventually it catches the other one. When it does, they each become double size unmagnified sprites and continue walking. When they meet the second time, the one that has been going faster disappears and the other continues walking.

Lines 110, 120, 140, 150, 250, and 260 define the sprites.

Line 130 sets the meeting counter to zero.

Lines 170 through 200 build the floor.
Lines 210 through 246 build the barrier.

>210 CALL COLOR(13,15,15)
>220 CALL VCHR(14,22,128,6)
>230 CALL VCHR(14,23,128,6)
>240 CALL VCHR(14,24,128,6)
>250 CALL SPRITE(#1,96,5,113,129,#2,96,7,139)
>260 CALL MAGNIFY(4)
>270 XDIR=4
>280 PAT=2
>290 CALL MOTION(#1,0,XDIR,#2,0,4)
>300 CALL PATTERN(#1,93+PAT,#2,98-EAT)
>310 PAT=-PAT
>320 CALL COINC(ALL,CC)

Line 270 sets the starting speed of the sprite that will speed up.

Line 290 sets the sprites in motion.

Line 300 creates the illusion of walking.

Line 320 checks to see if the sprites have met.

Line 330 transfers control if the sprites have met. Lines 340 and 350 check to see if the sprite has reached the barrier and transfers control if it has.

Line 360 loops back to continue the walk. Lines 370 through 460 handle the sprites running into each other. Lines 380 and 390 stop them.

Line 400 checks to see if it is the first meeting. Line 410 increments the meeting counter. Line 420 finds their position.

Line 430 makes them smaller.

Line 440 puts them on the floor and moves the fast one slightly ahead.

Line 450 starts them moving again.

>330 IF C00>0 THEN 376
>340 CALL POSITION(#1,YPOS1,XPOS1)
>350 IF XPOS1>36 AND XPOS1<192 THEN 470
>360 CTO 300
>370 REM COINCIDENCE
>380 CALL MOTION(#1,0,#2,0,0)
>390 CALL PATTERN(#1,93,#2,98)
>400 IF CNT<0 THEN 540
>410 CNT=CNT+1
>420 CALL POSITION(#1,YPOS1,XPOS1,#2,YPOS2,XPOS2)
>430 CALL MAGNIFY(3)
>440 CALL LOCATE(#1,YPOS1+16,XPOS1+3,#2,YPOS2+16,XPOS2)
>450 CALL MOTION(#1,0,XDIR,#2,0,4)
>460 GOTO 340

Lines 470 through 530 handle the fast sprite jumping through the barrier. Line 480 stops it. Line 490 finds where it is.

Line 500 puts it at the new location beyond the barrier.

Lines 510 and 520 start it moving again, a little faster.

Lines 540 through 640 handle the second meeting.

Line 560 starts the slow sprite moving, while line 570 deletes the fast sprite. Lines 580 through 630 make the slow sprite walk 20 steps.
**SQR**

**Format**

SQR(numeric-expression)

**Description**

The SQR function returns the positive square root of numeric-expression. SQR(X) is equivalent to $X^{1/2}$. Numeric-expression may not be a negative number.

**Examples**

PRINT SQR(4) prints 2.  
$>100$ PRINT SQR(4)

X = SQR(2.57E5) sets X equal to the square root of 257,000 which is 506.9516742.

**STOP**

**Format**

STOP

**Description**

The STOP statement stops program execution. It can be used interchangeably with the END statement except that it may not be placed after subprograms.

**Program**

The program on the right illustrates the use of the STOP statement. The program adds the numbers from 1 to 100.

$>100$ CLEAR

$>110$ TX=0

$>120$ NUMB=1

$>130$ TX=TX+NUMB

$>140$ NUMB=NUMB+1

$>150$ IF NUMB>100 THEN PRINT T

$>160$ STOP

$>160$ GOTO 130

**STRS**

**Format**

STRS(numeric-expression)

**Description**

The STRS function returns a string equivalent to numeric-expression. This allows the functions, statements, and commands that act on strings to be used on the character representation of numeric-expression. The STRS function is the inverse of the VAL function.

**Examples**

NUMS = STRS(78.6) sets NUMS equal to "78.6".

LLS = STRS(3E15) sets LLS equal to "3.E15".

IS = STRS(A*4) sets IS equal to a string equal to what ever value is obtained when A is multiplied by 4.

For instance, if A is equal to -8.18 IS is set equal to "-32".
SUB

Format

SUB subprogram-name [(parameter-list)]

Description

The SUB statement is the first statement in a subprogram. Subprograms are used when you wish to separate a group of statements from the main program. You may use subprograms to perform an operation several times in a program or in several different programs or to use variables that are specific to the subprogram. The SUB statement may not be in an IF-THEN-ELSE statement.

Subprograms are called with CALL subprogram-name [(parameter-list)]. Subprograms are ended with SUBEND and left when either a SUBEND or a SUBEXIT statement is executed. Control is returned to the statement following the statement that called the subprogram. You must never transfer control out of a subprogram with any statement except SUBEND or SUBEXIT. This includes passing control with CN ERROR.

When a subprogram is in a program, it must follow the main program. The structure of a program must be as follows:

Start of Main Program
.
.
.
Subprogram Calls
.
.
.
End of Main Program

The program will stop here without a STOP or END statement.

Start of First Subprogram
.
.
.
End of First Subprogram

Nothing may appear between subprograms except remarks and the END statement.

Start of Second Subprogram
.
.
.
End of Second Subprogram

Only remarks and END may appear after the subprograms.

End of Program

Options

All variables used in a subprogram other than those in parameter-list are local to that subprogram, so you may use the same variable names that are used in the main program or in other subprograms, and alter their values, without having any effect on other variables. Likewise, the values of variables in the main program or other subprograms have no effect on the values of the variables in the subprogram. (However, DATA statements are available to subprograms.) Communicating values to and from the main program is done with the optional parameter-list. The parameters need not have the same names as in the calling statement, but they must be of the same data type (numeric or string) and in the same order as the items in the CALL. If simple variables passed to subprograms have their values changed in the subprogram, the values of the variables in the main program are also changed. An array element such as A(1) in the parameter list of the calling statement is also changed in value in the main program when control is returned to the main program.

A value that is given in the calling statement as an expression is passed as a value only and changes in the value in the subprogram do not change values in the main program. Entire arrays are passed by reference, so changes in elements in the subprogram also change the values of the elements of the array in the main program. Arrays are indicated by following the parameter name with parentheses. If the array has more than one dimension, a comma must be placed inside the parentheses for each additional dimension.

If you wish, you may pass values only for simple variables by enclosing them in parentheses. Then the value can be used in the subprogram, but it is not changed in the return to the main program. For example, CALL SPRG1((A)) passes the value of A to a subprogram that starts SUB SPJ31(X), and allows that value to be used in X, but does not change the value of A in the main program if the subprogram changes the value of X.

If a subprogram is called more than once, any local variables used in the subprogram retain their values from one call to the next.
Examples

SUB MENU marks the beginning of a subprogram. No parameters are passed or returned.

SUB MENU(COUNT,CHOICE) marks the beginning of a subprogram. The variables COUNT and CHOICE may be used and/or have their values changed in the subprogram and returned to the variables in the same position in the calling statement.

SUB PAYCHECK(DATE,Q,SSN, PAYRATE, TABLE(.)) marks the beginning of a subprogram. The variables DATE, Q, SSN, PAYRATE, and the array TABLE with two dimensions may be used and/or have their values changed in the subprogram and returned to the variables in the same position in the calling statement.

Note that this R is not the same as the R used in lines 100 and 110 in the main program.

Program

The program on the right illustrates the use of SUB. The subprogram MEN@ had been previously saved with the merge option. It prints a menu and requests a choice. The main program tells the subprogram how many choices there are and what the choices are. It then uses the choice made in the subprogram to determine what program to run.

Beginning of subprogram MENU.

>100 CALL MENU(S,R)
>110 ON R GOTO 120,130,140,150,160
>120 RUN "DSKL.PAYABLES"
>130 RUN "DSKL.RECEIVE"
>140 RUN "DSKL.PAYROLL"
>150 RUN "DSKL.INVENTORY"
>160 RUN "DSKL.LEDGER"
>170 DATA ACCOUNTS PAYABLE, ACCOUNTS RECEIVABLE, PAYROLL, INVENTORY, GENERAL LEDGER
>10000 SUB MENU(COUNT,CHOICE);
>10010 CALL CLEAR
>10020 IF COUNT<22 THEN PRINT "TOO MANY ITEMS" ; CHOICE=0 : SUBEXIT
>10030 RESTORE
>10040 FOR R=1 TO COUNT
>10050 READ TEMPS
>10060 TEMPS=SEG(S(TEMPS,1,25))
>10070 DISPLAY AT(R,1) R;TEMPS $;
>10080 NEXT R
>10090 DISPLAY AT(R+1,):"YOU R CHOICE:" 1"
>10100 ACCEPT AT(R+1,1) BEEP
>10110 IF CHOICE<1 OR CHOICE>
COUNT THEN 10100
>10120 SUBEND
**SUBEND**

**Format**

SUBEND

**Description**

The SUBEND statement marks the end of a subprogram. When SUBEND is executed, control is passed to the statement following the statement that called the subprogram. The SUBEND statement must always be the last statement in a subprogram. The SUBEND statement may not be in an IF-THEN ELSE statement. The only statements that may immediately follow a SUBEND statement are REM, END, or the SUB statement for the next subprogram.

**SUBEXIT**

**Format**

SUBEXIT

**Description**

The SUBEXIT statement allows ending a subprogram before the end of the subprogram (indicated with SUBEND). When it is executed, control is passed to the statement following the statement that called the subprogram. The SUBEXIT statement need not be present in a subprogram.

**TAB**

**Format**

TAB(numeric-expression)

**Description**

The TAB function specifies the starting position for the next print-item in a PRINT, PRINT...USING, DISPLAY, or DISPLAY...USING statement. If numeric-expression is greater than the length of a record for the device on which the printing is being done (for example, 28 for the screen, 32 for the thermal printer, the specified value for a file on a diskette or cassette), then it is repeatedly reduced by the record length until it is between 1 and the record length.

If the number of characters already printed on the current record is less than or equal to numeric-expression, the next print item is printed beginning on the position indicated by numeric-expression. If the number of characters already printed on the current record is greater than the position indicated by numeric-expression, the next print-item is printed on the next record beginning in the position indicated by numeric-expression.

The TAB function is treated as a print-item, so it must have a print separator (colon, semicolon, or comma) before and/or after it. The print separator before TAB is evaluated before the TAB function. Normally semicolons are used before and after TAB.

**Examples**

PRINT TAB(12);35 prints the number 35 at the twelfth position.

PRINT 35;TAB(18);"NAME" prints 356 at the beginning of the line and NAME at the eighteenth position of the line.

PRINT "ABCDEFHJIKLMM";TAB(5);"NOP" prints ABCDEFHJIKLMM at the beginning of the line and NOP at the fifth position of the next line.

DISPLAY AT(12,1);"NAME";TAB(15);"ADDRESS" displays NAME at the beginning of the twelfth line on the screen and ADDRESS at the fifteenth position on the twelfth line of the screen.
TAN

Format
TAN(radian-expression)

Description
The tangent function gives the trigonometric tangent of radian-expression. If the angle is in degrees, multiply the number of degrees by PI/180 to get the equivalent angle in radians.

Program
The program on the right gives the tangent of several angles.

```
>100 DEF .78539*1633973
>110 E=26.565*11177
>120 C=45*PI/.80
>130 PRINT TAN(A);TAN(B)
>140 PRINT TAN(B+PI/130)
>150 PRINT TAN(C)
>RUN
  1. 7.17470553
  .5
  1
```

UNBREAK

Format
UNBREAK [line-list]

Description
The UNBREAK command removes all breakpoints. It can optionally be set for only those in line-list. UNERASE can be used as a statement.

Examples
UNBREAK removes all breakpoints.
>UNBREAK
>420 UNERASE

UNBREAK 100,130 removes the breakpoints from lines 100 and 130.
>UNBREAK 100,130
>320 UNERASE 100,130

UNTRACE

Format
UNTRACE

Description
The UNTRACE command removes the effect of the TRACE command. UNTRACE can be used as a statement.

Example
UNTRACE removes the effect of TRACE.
>UNTRACE
>420 UNTRACE

TRACE

Format
TRACE

Description
The TRACE command causes each line number to be displayed on the screen before the statements on that line are executed. This enables you to follow the course of a program as a debugging aid. The TRACE command may be used as a statement. The effect of the TRACE command is canceled when the NEW command or UNTRACE command or statement is performed.

Example
TRACE causes the computer to display a trace of the lines of a program on the screen.

>TRACE
$100 TRACE
VAL

**Format**

VAL(string-expression)

**Description**

The VAL function returns the number equivalent to string-expression. This allows the functions, statements, and commands that act on numbers to be used on string-expression. The VAL function is the inverse of the STRS function.

**Examples**

NUM = VAL("78.6") sets NUM equal to 78.6.

LL = VAL("3E15") sets LL equal to 3.E15.

VCHAR

**Format**

CALL VCHAR(row, column, character-code [, repetition])

**Description**

The VCHAR subprogram places a character anywhere on the display screen and optionally repeats it vertically. The character with the ASCII value of character-code is placed in the position described by row and column and is repeated vertically repetition times.

A value of 1 for row indicates the top of the screen. A value of 25 is the bottom of the screen. A value of 1 for column indicates the left side of the screen. A value of 32 is the right side of the screen. The screen can be thought of as a grid as shown below.

**Examples**

CALL VCHAR(12, 16, 33) places character 33 (an exclamation point) in row 12, column 16.

CALL VCHAR(1, ASC("'"), 1, 768) places an exclamation point in row 1, column 1, and repeats it 768 times, which fills the screen.

CALL VCHAR(R, C, K, T) places the character with an ASCII code of K in row R, column C and repeats it T times.

>100 CALL VCHAR(12, 16, 33):

>100 CALL VCHAR(1, 1, ASC("'"), 768)

>100 CALL VCHAR(R, C, K, T)
VERSION subprogram

Format
CALL VERSION(numeric-variable)

Description
The VERSION subprogram returns a value indicating the version of BASIC
that is being used. TI Extended BASIC returns a value of 100.

Example
CALL VERSION(V) sets V equal to 100.
>100 CALL VERSION(V)

Appendices

The following appendices give useful information concerning TI Extended
BASIC.

Appendix A: List of Illustrative Programs
Appendix B: List of Commands, Statements, and Functions
Appendix C: ASCII Codes
Appendix D: Musical Tone Frequencies
Appendix E: Character Sets
Appendix F: Pattern-Identifier Conversion Table
Appendix G: Color Codes
Appendix H: High Resolution Color Combinations
Appendix I: Split Console Keyboard
Appendix J: Character Codes for Split Keyboard
Appendix K: Mathematical Functions
Appendix L: List of Speech Words
Appendix M: Adding Suffixes to Speech Words
Appendix N: Error Messages
## List of Illustrative Programs

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>ILLUSTRATED</th>
<th>LINES</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>16</td>
<td></td>
<td>Entry of 20 names</td>
<td>48</td>
</tr>
<tr>
<td>CALL</td>
<td>8</td>
<td></td>
<td>CLEAR and user written subroutine</td>
<td>55</td>
</tr>
<tr>
<td>CHAR</td>
<td>12</td>
<td></td>
<td>1. Moving figure</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td>2. Resetting characters</td>
<td>58</td>
</tr>
<tr>
<td>CHRS</td>
<td>4</td>
<td></td>
<td>List of ASCII codes</td>
<td>60</td>
</tr>
<tr>
<td>CLEAR</td>
<td>3</td>
<td></td>
<td>(Simple example)</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>(Simple example)</td>
<td>61</td>
</tr>
<tr>
<td>COINC</td>
<td>10</td>
<td></td>
<td>(Simple example)</td>
<td>65</td>
</tr>
<tr>
<td>COS</td>
<td>6</td>
<td></td>
<td>(Simple example)</td>
<td>69</td>
</tr>
<tr>
<td>DATA</td>
<td>14</td>
<td></td>
<td>(Simple example)</td>
<td>71</td>
</tr>
<tr>
<td>DELETE</td>
<td>2</td>
<td></td>
<td>(Simple example)</td>
<td>74</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>18</td>
<td></td>
<td>Draw on screen</td>
<td>78</td>
</tr>
<tr>
<td>ERR</td>
<td>5</td>
<td></td>
<td>(Simple example)</td>
<td>84</td>
</tr>
<tr>
<td>FOR-TO-STEP</td>
<td>11</td>
<td></td>
<td>Design</td>
<td>87</td>
</tr>
<tr>
<td>GOSUB</td>
<td>24</td>
<td></td>
<td>Probability</td>
<td>90</td>
</tr>
<tr>
<td>GOTO</td>
<td>8</td>
<td></td>
<td>Add 1 through 100</td>
<td>91</td>
</tr>
<tr>
<td>IF-THEN-ELSE</td>
<td>17</td>
<td></td>
<td>Sequence numbers</td>
<td>96</td>
</tr>
<tr>
<td>IMAGE</td>
<td>12</td>
<td></td>
<td>(Simple example)</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>(Simple example)</td>
<td>100</td>
</tr>
<tr>
<td>INPUT</td>
<td>17</td>
<td></td>
<td>Writes letter</td>
<td>103</td>
</tr>
<tr>
<td>INPUT (with files)</td>
<td>12</td>
<td></td>
<td>(Simple example)</td>
<td>106</td>
</tr>
<tr>
<td>JOYST</td>
<td>5</td>
<td></td>
<td>Moves sprite</td>
<td>108</td>
</tr>
<tr>
<td>KEY</td>
<td>12</td>
<td></td>
<td>Moves sprite</td>
<td>109</td>
</tr>
<tr>
<td>LINPUT</td>
<td>6</td>
<td></td>
<td>(Simple example)</td>
<td>113</td>
</tr>
<tr>
<td>LOCATE</td>
<td>6</td>
<td></td>
<td>(Simple example)</td>
<td>116</td>
</tr>
<tr>
<td>LOG</td>
<td>8</td>
<td></td>
<td>Log to any base</td>
<td>117</td>
</tr>
</tbody>
</table>

## LIST OF ILLUSTRATIVE PROGRAMS

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>ILLUSTRATED</th>
<th>LINES</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAGNIFY</td>
<td>17</td>
<td>(Simple example)</td>
<td>Magnifying function</td>
<td>120</td>
</tr>
<tr>
<td>MERGE</td>
<td>13</td>
<td></td>
<td>Moves sprite</td>
<td>122</td>
</tr>
<tr>
<td>MOTION</td>
<td>8</td>
<td></td>
<td>Moves sprite</td>
<td>125</td>
</tr>
<tr>
<td>NEXT</td>
<td>6</td>
<td>(Simple example)</td>
<td>Next function</td>
<td>127</td>
</tr>
<tr>
<td>NUMEER</td>
<td>4</td>
<td>(Simple example)</td>
<td>Number function</td>
<td>128</td>
</tr>
<tr>
<td>ON BREAK</td>
<td>11</td>
<td>(Simple example)</td>
<td>On break function</td>
<td>130</td>
</tr>
<tr>
<td>ON ERROR</td>
<td>15</td>
<td>(Simple example)</td>
<td>On error function</td>
<td>132</td>
</tr>
<tr>
<td>ON...GOSUB</td>
<td>20</td>
<td></td>
<td>Choose with a menu</td>
<td>134</td>
</tr>
<tr>
<td>ON...GOTO</td>
<td>19</td>
<td></td>
<td>Choose with a menu</td>
<td>136</td>
</tr>
<tr>
<td>ON WARNING</td>
<td>8</td>
<td>(Simple example)</td>
<td>On warning function</td>
<td>137</td>
</tr>
<tr>
<td>PATTERN</td>
<td>18</td>
<td></td>
<td>Rolling wheel</td>
<td>142</td>
</tr>
<tr>
<td>POS</td>
<td>8</td>
<td></td>
<td>Breakup sentence</td>
<td>145</td>
</tr>
<tr>
<td>PRINT</td>
<td>7</td>
<td>(Simple example)</td>
<td>Print function</td>
<td>149</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>5</td>
<td>(Simple example)</td>
<td>Randomize function</td>
<td>151</td>
</tr>
<tr>
<td>REC</td>
<td>12</td>
<td>(Simple example)</td>
<td>Recursive function</td>
<td>153</td>
</tr>
<tr>
<td>RETURN (with GOSUB)</td>
<td>18</td>
<td></td>
<td>Figure interest</td>
<td>157</td>
</tr>
<tr>
<td>RETURN (with ON ERROR)</td>
<td>13</td>
<td></td>
<td>Handle error</td>
<td>158</td>
</tr>
<tr>
<td>RUN</td>
<td>12</td>
<td></td>
<td>Choose with a menu</td>
<td>162</td>
</tr>
<tr>
<td>SAY</td>
<td>4</td>
<td>(Simple example)</td>
<td>Say function</td>
<td>164</td>
</tr>
<tr>
<td>SIN</td>
<td>6</td>
<td>(Simple example)</td>
<td>Sin function</td>
<td>168</td>
</tr>
<tr>
<td>SOUND</td>
<td>4</td>
<td></td>
<td>Play first 15 notes</td>
<td>171</td>
</tr>
<tr>
<td>SPGET</td>
<td>4</td>
<td>(Simple example)</td>
<td>Sprites get</td>
<td>172</td>
</tr>
<tr>
<td>SPRITE</td>
<td>8</td>
<td>(Simple example)</td>
<td>Sprite function</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>Creation of stars</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td></td>
<td>Walking sprites</td>
<td>175</td>
</tr>
<tr>
<td>STOP</td>
<td>7</td>
<td></td>
<td>Add 1 through 100</td>
<td>178</td>
</tr>
<tr>
<td>SUB</td>
<td>21</td>
<td></td>
<td>Choose with a menu</td>
<td>183</td>
</tr>
<tr>
<td>TAN</td>
<td>6</td>
<td>(Simple example)</td>
<td>Tan function</td>
<td>186</td>
</tr>
</tbody>
</table>
The following is a list of all TI Extended BASIC commands, statements, and functions. Commands are listed first; if a command can also be used as a statement, the letter "S" is listed to the right of the command. Commands that can be abbreviated have the acceptable abbreviations underlined. Next is a list of all TI Extended BASIC statements; those that can also be used as commands have a "C" after them. Finally, there is a list of all TI Extended BASIC functions.

**TI Extended BASIC Commands**
- BREAK S
- BYE S
- CONTINUE S
- DELETE S
- LIST S
- MERGE S
- NUMBER S
- OLD S
- RESEQUENCE S
- RLNS
- SAVE S
- SIZE S
- TRACE S
- UNBREAK S
- UNTRACE S

**TI Extended BASIC Statements**
- ACCEPT C
- CALL S
- CALL CHAR C
- CALL CHARPAT C
- CALL CHARSSET C
- CALL CLEAR C
- CLOSE C
- CALL COINC C
- CALL COLOR C
- CALL DELSPIRE C
- CALL LINK C
- CALL LOAD C
- CALL LCCATE C
- CALL MAGNIFY C
- CALL MOTION C
- CALL NEXT C
- CALL ON BREAK
- CALL ON ERROR
- CALL ON GOSUB
- CALL ON WARNING
- CALL PHASE C
- CALL PI C
- CALL POS S
- CALL REC
- CALL RND
- CALL RPT S
- CALL SAY C
- CALL SCREEN C
- CALL SOUND C
- CALL SPGET C
- CALL SPRINT C
- CALL S'OP C
- CALL SUB
- CALL SUBEND
- CALL SUBEXIT
- CALL VCHAR C
- CALL VERSION C

**TI Extended BASIC Functions**
- ABS
- ASC
- ATN
- CHRS
- COS
- ERF
- EXP
- INT
- PI
- POS
- REC
- RND
- RPT S
- SEG S
- SGN
- SIN
- SGR
- STR S
- TAN
- VAL
The following predefined characters may be printed or displayed on the screen.

<table>
<thead>
<tr>
<th>ASCII CODE</th>
<th>CHARACTER</th>
<th>ASCII CODE</th>
<th>CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>(cursor)</td>
<td>63</td>
<td>? (question mark)</td>
</tr>
<tr>
<td>31</td>
<td>(edge character)</td>
<td>64</td>
<td>@ (at sign)</td>
</tr>
<tr>
<td>32</td>
<td>(space)</td>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>33</td>
<td>! (exclamation point)</td>
<td>66</td>
<td>B</td>
</tr>
<tr>
<td>34</td>
<td>&quot; (quote)</td>
<td>67</td>
<td>C</td>
</tr>
<tr>
<td>35</td>
<td>* (number or pound sign)</td>
<td>68</td>
<td>D</td>
</tr>
<tr>
<td>36</td>
<td>$ (dollar)</td>
<td>69</td>
<td>E</td>
</tr>
<tr>
<td>37</td>
<td>% (percent)</td>
<td>70</td>
<td>F</td>
</tr>
<tr>
<td>38</td>
<td>&amp; (ampersand)</td>
<td>71</td>
<td>G</td>
</tr>
<tr>
<td>39</td>
<td>' (apostrophe)</td>
<td>72</td>
<td>H</td>
</tr>
<tr>
<td>40</td>
<td>( (open parenthesis)</td>
<td>73</td>
<td>I</td>
</tr>
<tr>
<td>41</td>
<td>) (close parenthesis)</td>
<td>74</td>
<td>J</td>
</tr>
<tr>
<td>42</td>
<td>* (asterisk)</td>
<td>75</td>
<td>K</td>
</tr>
<tr>
<td>43</td>
<td>+ (plus)</td>
<td>76</td>
<td>L</td>
</tr>
<tr>
<td>44</td>
<td>, (comma)</td>
<td>77</td>
<td>M</td>
</tr>
<tr>
<td>45</td>
<td>- (minus)</td>
<td>78</td>
<td>N</td>
</tr>
<tr>
<td>46</td>
<td>. (period)</td>
<td>79</td>
<td>O</td>
</tr>
<tr>
<td>47</td>
<td>/ (slash)</td>
<td>80</td>
<td>P</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>81</td>
<td>Q</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>82</td>
<td>R</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>83</td>
<td>S</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td>84</td>
<td>T</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>85</td>
<td>U</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>86</td>
<td>V</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td>87</td>
<td>W</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
<td>88</td>
<td>X</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>89</td>
<td>Y</td>
</tr>
<tr>
<td>57</td>
<td>9</td>
<td>90</td>
<td>Z</td>
</tr>
<tr>
<td>58</td>
<td>: (colon)</td>
<td>91</td>
<td>] (open bracket)</td>
</tr>
<tr>
<td>59</td>
<td>; (semicolon)</td>
<td>92</td>
<td>\ (reverse slash)</td>
</tr>
<tr>
<td>60</td>
<td>&lt; (less than)</td>
<td>93</td>
<td>] (close bracket)</td>
</tr>
<tr>
<td>61</td>
<td>= (equals)</td>
<td>94</td>
<td>^ (exponentiation)</td>
</tr>
<tr>
<td>62</td>
<td>&gt; (greater than)</td>
<td>95</td>
<td>_ (underscore)</td>
</tr>
</tbody>
</table>

The following key presses may also be detected by CALL KEY.

1  SHIFT A (AID)  3  SHIFT F (DEL)
4  SHIFT G (INS)  6  SHIFT R (REDO)
7  SHIFT T (ERASE)  8  SHIFT S (LEFT ARROW)
9  SHIFT D (RIGHT ARROW)  10  SHIFT X (DOWN ARROW)
11  SHIFT E (UP ARROW)  12  SHIFT V (CMD)
13  ENTER  14  SHIFT W (3EGjn)
15  SHIFT Z (BACK)
## Character Sets

<table>
<thead>
<tr>
<th>SET</th>
<th>ASCII CODES</th>
<th>SET</th>
<th>ASCII CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30-31</td>
<td>8</td>
<td>88-95</td>
</tr>
<tr>
<td>1</td>
<td>32-39</td>
<td>9</td>
<td>96-103</td>
</tr>
<tr>
<td>2</td>
<td>40-47</td>
<td>10</td>
<td>104-111</td>
</tr>
<tr>
<td>3</td>
<td>48-55</td>
<td>11</td>
<td>112-119</td>
</tr>
<tr>
<td>4</td>
<td>56-63</td>
<td>12</td>
<td>120-127</td>
</tr>
<tr>
<td>5</td>
<td>64-71</td>
<td>13</td>
<td>128-135</td>
</tr>
<tr>
<td>6</td>
<td>72-79</td>
<td>14</td>
<td>136-143</td>
</tr>
<tr>
<td>7</td>
<td>80-87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Color Codes

<table>
<thead>
<tr>
<th>COLOR</th>
<th>CODE</th>
<th>COLOR</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent</td>
<td>1</td>
<td>Medium Red</td>
<td>9</td>
</tr>
<tr>
<td>Black</td>
<td>2</td>
<td>Light Red</td>
<td>10</td>
</tr>
<tr>
<td>Medium Green</td>
<td>3</td>
<td>Dark Yellow</td>
<td>11</td>
</tr>
<tr>
<td>Light Green</td>
<td>4</td>
<td>Light Yellow</td>
<td>12</td>
</tr>
<tr>
<td>Dark Blue</td>
<td>5</td>
<td>Dark Green</td>
<td>15</td>
</tr>
<tr>
<td>Light Blue</td>
<td>6</td>
<td>Magenta</td>
<td>14</td>
</tr>
<tr>
<td>Dark Red</td>
<td>7</td>
<td>Gray</td>
<td>15</td>
</tr>
<tr>
<td>Cyan</td>
<td>8</td>
<td>White</td>
<td>16</td>
</tr>
</tbody>
</table>

## Pattern-Identifier Conversion Table

<table>
<thead>
<tr>
<th>BLOCKS</th>
<th>BINARY CODE</th>
<th>HEXADECIMAL CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Color Combinations

The following color combinations produce the sharpest, clearest character resolution.

**BEST**

2. 8  Black on Cyan
2. 7  Black on Dark Red
2. 6  Black on Light Blue
2. 3  Black on Medium Green
5. 8  Dark Blue on Cyan
5. 6  Dark Blue on Light Blue
5. 14 Dark Blue on Magenta
13. 8 Dark Green on Cyan
13. 15 Dark Green on Gray
13. 12 Dark Green on Light Yellow
7. 15  Dark Red on Gray
7. 12  Dark Red on Light Yellow
3. 12  Medium Green on Light Yellow

**SECOND BEST**

2. 5  Black on Dark Blue
2. 4  Black on Light Green
2. 12  Black on Light Yellow
13. 15  Dark Green on White
6. 15  Light Blue on Gray
6. 16  Light Blue on White

**THIRD BEST**

2. 16  Black on White
7. 9  Dark Red on Medium Red
14. 15  Magenta on Gray
3. 11  Medium Green on Dark Yellow
9. 15  Medium Red on Gray
9. 12  Medium Red on Light Yellow
16. 7  White on Dark Red

**FOURTH BEST**

8. 2  Cyan on Black
7. 2  Dark Red on Black
15. 16  Gray on White
4. 2  Light Green on Black
10. 16  Light Red on White
9. 4  Medium Red on Light Green

---

Split Console Keyboard

**Key-unit 1**

<table>
<thead>
<tr>
<th>19</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Key-unit 2**

<table>
<thead>
<tr>
<th>19</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Character Codes for Split Keyboard**

<table>
<thead>
<tr>
<th>CODE</th>
<th>KEYS*</th>
<th>CODE</th>
<th>KEYS*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X, M</td>
<td>10</td>
<td>S, 6</td>
</tr>
<tr>
<td>1</td>
<td>A, H</td>
<td>11</td>
<td>T, P</td>
</tr>
<tr>
<td>2</td>
<td>S, J</td>
<td>12</td>
<td>F, L</td>
</tr>
<tr>
<td>3</td>
<td>D, K</td>
<td>13</td>
<td>V, EMT</td>
</tr>
<tr>
<td>4</td>
<td>W, U</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>E, I</td>
<td>15</td>
<td>Z, N</td>
</tr>
<tr>
<td>6</td>
<td>R, G</td>
<td>16</td>
<td>SPACE, B</td>
</tr>
<tr>
<td>7</td>
<td>2, 7</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3, 8</td>
<td>18</td>
<td>G, Q</td>
</tr>
<tr>
<td>9</td>
<td>4, 9</td>
<td>19</td>
<td>1, 6</td>
</tr>
</tbody>
</table>

*Note that the first key listed is on the left side of the keyboard and the second key listed is on the right side of the keyboard.*

Ti Extended BASIC 201
### Mathematical Functions

The following mathematical functions may be defined with DEF as shown:

<table>
<thead>
<tr>
<th>Function</th>
<th>TI Extended BASIC statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secant</td>
<td>DEF SEC(X)=1/COS(X)</td>
</tr>
<tr>
<td>Cosecant</td>
<td>DEF CSC(X)=1/SEC(X)</td>
</tr>
<tr>
<td>Cotangent</td>
<td>DEF COT(X)=1/TAN(X)</td>
</tr>
<tr>
<td>Inverse Sine</td>
<td>DEF ARCSIN(X)=ATN(X/SQR(1-X*X))</td>
</tr>
<tr>
<td>Inverse Cosine</td>
<td>DEF ARCCOS(X)=ATN(X/SQR(X*X-1))+(SIN(X)-1)*PI/2</td>
</tr>
<tr>
<td>Inverse Secant</td>
<td>DEF ARCCSC(X)=ATN(SQR(X*X-1)+SIN(X)-1)*PI/2</td>
</tr>
<tr>
<td>Inverse Cosecant</td>
<td>DEF ARCCOT(X)=ATN(SQR(X*X-1)-1)/PI/2</td>
</tr>
<tr>
<td>Hyperbolic Sine</td>
<td>DEF SINH(X)=EXP(X)-EXP(-X))/2</td>
</tr>
<tr>
<td>Hyperbolic Cosine</td>
<td>DEF COSH(X)=EXP(X)+EXP(-X))/2</td>
</tr>
<tr>
<td>Hyperbolic Tangent</td>
<td>DEF TANH(X)=EXP(X)-EXP(-X)/EXP(X)+EXP(-X)</td>
</tr>
<tr>
<td>Hyperbolic Secant</td>
<td>DEF SINC(X)=2/(EXP(X)+EXP(-X))</td>
</tr>
<tr>
<td>Hyperbolic Cosecant</td>
<td>DEF CSCSCH(X)=2/(EXP(X)-EXP(-X))</td>
</tr>
<tr>
<td>Hyperbolic Cotangent</td>
<td>DEF CSCHH(X)=EXP(-X)*EXP(-X))</td>
</tr>
<tr>
<td>Inverse Hyperbolic Sine</td>
<td>DEF ARCSINH(X)=LOG(X+SQR(X*X+1))</td>
</tr>
<tr>
<td>Inverse Hyperbolic Cosine</td>
<td>DEF ARCCOSH(X)=LOG(X+SQR(X*X-1))</td>
</tr>
<tr>
<td>Inverse Hyperbolic Tangent</td>
<td>DEF ARCTANH(X)=LOG((1+X)/(1-X))/2</td>
</tr>
<tr>
<td>Inverse Hyperbolic Secant</td>
<td>DEF ARCSCECH(X)=LOG(SQR(X*X-1)+1)/PI/2</td>
</tr>
<tr>
<td>Inverse Hyperbolic Cosecant</td>
<td>DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2</td>
</tr>
</tbody>
</table>

### List of Speech Words

The following is a list of all the letters, numbers, words, and phrases that can be accessed with CALL SAY and CALL SPEG. See Appendix M for instructions on adding suffixes to anything in this list.

<table>
<thead>
<tr>
<th>Word</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AG</td>
</tr>
<tr>
<td>ABOUT</td>
<td>AGAIN</td>
</tr>
<tr>
<td>AM</td>
<td>AN</td>
</tr>
<tr>
<td>AND</td>
<td>ANY</td>
</tr>
<tr>
<td>ARE</td>
<td>AS</td>
</tr>
<tr>
<td>ASSUME</td>
<td>AT</td>
</tr>
<tr>
<td>B</td>
<td>BACK</td>
</tr>
<tr>
<td>BASE</td>
<td>BE</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>BLACK</td>
</tr>
<tr>
<td>BLUE</td>
<td>BOTH</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>BUT</td>
</tr>
<tr>
<td>BUY</td>
<td>BY</td>
</tr>
<tr>
<td>BYE</td>
<td>CAN</td>
</tr>
<tr>
<td>CASSETTE</td>
<td>CENTER</td>
</tr>
<tr>
<td>CHECK</td>
<td>CHECK</td>
</tr>
<tr>
<td>CENTER</td>
<td>CHECK</td>
</tr>
<tr>
<td>CHOICE</td>
<td>CHECK</td>
</tr>
<tr>
<td>CLEAR</td>
<td>CLEAR</td>
</tr>
<tr>
<td>COLOR</td>
<td>CLEAR</td>
</tr>
<tr>
<td>COME</td>
<td>CLEAR</td>
</tr>
<tr>
<td>COMES</td>
<td>CLEAR</td>
</tr>
<tr>
<td>COMMAND</td>
<td>COMES</td>
</tr>
<tr>
<td>COMMA</td>
<td>COMMAND</td>
</tr>
<tr>
<td>COMPLET3</td>
<td>COMMAND</td>
</tr>
<tr>
<td>COMPLETED</td>
<td>COMMAND</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>CONNECTED</td>
<td>COMPUTER</td>
</tr>
<tr>
<td>CONSOLES</td>
<td>CONNECTED</td>
</tr>
<tr>
<td>CORRECT</td>
<td>CONSOLES</td>
</tr>
<tr>
<td>COURSE</td>
<td>CORRECT</td>
</tr>
<tr>
<td>CYAN</td>
<td>COURSE</td>
</tr>
<tr>
<td>D</td>
<td>CYAN</td>
</tr>
<tr>
<td>DATA</td>
<td>D</td>
</tr>
<tr>
<td>DECIDE</td>
<td>DATA</td>
</tr>
<tr>
<td>DEVICE</td>
<td>DECIDE</td>
</tr>
<tr>
<td>DID</td>
<td>DEVICE</td>
</tr>
<tr>
<td>DIFFERENT</td>
<td>DID</td>
</tr>
<tr>
<td>DISKETTE</td>
<td>DIFFERENT</td>
</tr>
<tr>
<td>DOES</td>
<td>DISKETTE</td>
</tr>
<tr>
<td>DOING</td>
<td>DOES</td>
</tr>
<tr>
<td>DONE</td>
<td>DOING</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DONE</td>
</tr>
<tr>
<td>DOWN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DRAW</td>
<td>DOWN</td>
</tr>
<tr>
<td>DRAWING</td>
<td>DRAW</td>
</tr>
<tr>
<td>DIES</td>
<td>DRAWING</td>
</tr>
<tr>
<td>DIES</td>
<td>E</td>
</tr>
<tr>
<td>END</td>
<td>DIES</td>
</tr>
<tr>
<td>ENDS</td>
<td>END</td>
</tr>
<tr>
<td>ENTER</td>
<td>END</td>
</tr>
<tr>
<td>ERROR</td>
<td>ENTER</td>
</tr>
<tr>
<td>EXACTLY</td>
<td>ERROR</td>
</tr>
<tr>
<td>EYE</td>
<td>EXACTLY</td>
</tr>
<tr>
<td>HELLO</td>
<td>EYE</td>
</tr>
<tr>
<td>HELP</td>
<td>HELLO</td>
</tr>
</tbody>
</table>
LIST OF SPEECH WORDS

HERE
HIGHER
HIT
HOME
HUNDRED
HURRY
I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT
J
JOYSTICK
JUST
K
KEY
KEYBOARD
KNOW
L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER
M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST
N
NAME
NEAR
NEED
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER
O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER
P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING
Q
R
RANDOMLY
READ (read)
READ (red)
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND
S
SAID
SAVE
SAY
SAYS
SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY

LIST OF SPEECH WORDS

SPACE
SPACES
SPELL
SQUARE
START
STEP
STOP
SUM
SUPPOSED
SUPPOSED TO
SURE
T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS INCORRECT
THAT IS RIGHT
THE (the)
THE I (tha)
THEIR
THEN
THERE
THESE
THILY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRTY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE
U
UNCH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE
V
VARY
VERY
W
WAIT
WATTS
WA*
WE
WEED
WEEDHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WOW
WORD
WORDS
WORK
WORKING
WRITE
X
Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR
Adding Suffixes to Speech Words

This appendix describes how to add ING, S, and ED to any word available in the SolidState Speech™ Synthesizer resident vocabulary.

The code for a word is first read using SFGET. The code consists of a number of characters, one of which tells the speech unit the length of the word. Then, by means of the subprograms listed here, additional codes can be added to give the sound of a suffix.

Words often have trailing-off data that make the word sound more natural but prevent the easy addition of suffixes. In order to add suffixes this trailing-off data must be removed.

The following program allows you to input a word and, by trying different truncation values, make the suffix sound like a natural part of the word. The subprograms DEFING (lines 1000 through 1130), DEFS1 (lines 2000 through 2100), DEFS2 (lines 3000 through 3090), DEFS3 (lines 4000 through 4120), DEFD1 (lines 5000 through 5070), DEFD2 (lines 6000 through 6110), DEFD3 (lines 7000 through 7130), and MENU (lines 1000 through 10120) should be input separately and saved with the MERGE option. (The subprogram MENU is the same one used in the illustrative program with SUB.) You may wish to use different line numbers. Each of these subprograms (except MENU) defines a suffix.

DEFING defines the ING sound. DEFS1 defines the S sound as it occurs at the end of “cats.” DEFS2 defines the S sound as it occurs at the end of “cads.” DEFS3 defines the S sound as it occurs at the end of “wishes.” DEFD1 defines the ED sound as it occurs at the end of “passed.” DEFD2 defines the ED sound as it occurs at the end of “ceased.” DEFD3 defines the ED sound as it occurs at the end of “heated.”

In running the program enter a J for the truncation value in order to leave the truncation sequence.

```
100 REM ***********************
110 REM REQUIRE MERGE OP:
120 REM MENU (LINES 1000 THROUGH 10120)
130 REM DEFING (LINES 1000 THROUGH 1130)
140 REM DEFS1 (LINES 2000 THROUGH 2000)
150 REM DEFS2 (LINES 3000 THROUGH 3090)
160 REM DEFS3 (LINES 4000 THROUGH 4120)
170 REM DEFD1 (LINES 5000 THROUGH 5070)
180 REM DEFD2 (LINES 6000 THROUGH 6110)
190 REM DEFD3 (LINES 7000 THROUGH 7130)
200 REM ***********************
210 CALL CLEAR
220 PRINT "THIS PROGRAM IS USED TO"
```
ADDITION SUFFIXES TO SPEECH WORDS

The data has been given in short DATA statements to make it as easy as possible to input. It may be consolidated to make the program shorter.

1000 SUB DEFS3(A$)
1010 DATA 60,0,52,27,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,3,202,58
1040 DATA 125,120,117,57,47,248
1050 DATA 135,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,187,118,180
1080 DATA 165,97,161,117,218,25
1090 DATA 115,184,227,222,249,238,1
1100 RESTORE 1C10
1110 A$=""
1120 FOR I=1 TO 55:READ A$:A$=A$&CHR$(A$):NEXT I
1130 SUBEND

2000 SUB DEFS1(A$):CATS
2010 DATA 96,3,26
2020 DATA 14,56,130,204,0
2030 DATA 223,177,26,224,103
2040 DATA 85,3,252,106,106
2050 DATA 128,95,44,4,240
2060 DATA 35,11,2,126,16,121
2070 RESTORE 2010
2080 A$=""
2090 FOR I=1 TO 29:READ A$:A$=A$&CHR$(A$):NEXT I
2100 SUBEND

3000 SUB DEFS2(A$):CAJS
3010 DATA 96,0,17
3020 DATA 161,253,158,217
3030 DATA 168,213,198,86,0
3040 DATA 223,153,75,128,0
3050 DATA 95,139,62
3060 RESTORE 3010
3070 A$=""
3080 FOR I=1 TO 20:READ A$:A$=A$&CHR$(A$):NEXT I
3090 SUBEND

APPENDIX

5000 SUB DEFED1(A$):PASSED
5010 DATA 96,0,10
5020 DATA 0,224,128,37
5030 DATA 204,37,240,0,0,0
5040 RESTORE 5010
5050 A$=""
5060 FOR I=1 TO 13:READ A$:A$=A$&CHR$(A$):NEXT I
5070 SUBEND

6000 SUB DEFED2(A$):CAUSE
6010 DATA 96,0,26
6020 DATA 172,163,214,59,35
6030 DATA 109,170,174,58,21
6040 DATA 22,201,220,250,24
6050 DATA 69,148,162,156,234
6060 DATA 75,84,97,145,204
6070 DATA 15
6080 RESTORE 6010
6090 A$=""
6100 FOR I=1 TO 29:READ A$:A$=A$&CHR$(A$):NEXT I
6110 SUBEND
7000 SUB DEFED3(A$):HEATED
7010 DATA 96,0,36
7020 DATA 173,233,31,84,12
7030 DATA 242,205,166,183
7040 DATA 172,163,234,59,35
7050 DATA 109,170,174,68,21
7060 DATA 22,201,92,250,24
7070 DATA 69,146,162,38,235
7080 DATA 75,84,97,145,204
7090 DATA 178,127
7100 RESTORE 7010
7110 A$=""
7120 FOR I=1 TO 39:READ A$:A$=A$&CHR$(A$):NEXT I
7130 SUBEND
10000 SUB MENG(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT>22 THEN PRINT "TOO MANY ITEMS" : CHOICE=0 : SUBEXIT
10030 RESTORE
10040 FOR I=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEGS(TEMP$,1,25)
10070 DISPLAY AT(I,1):I;TEMP$
10080 NEXT I
10090 DISPLAY AT(I+1,1):"YOUR CHOICE: 1"
10100 ACCEPT AT(I+1,1)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE<1 OR CHOICE>COUNT THEN 10100
10120 SUBEND

You can use the subprograms in any program once you have determined the number of bytes to truncate. The following program uses the subprogram DEFIG in lines 1000 through 1130 to have the computer say the word DRAWING using DRAW plus the suffix NG. Note that it was found that DRAW should be truncated by 41 characters to produce the most natural sounding DRAWING. The subprogram DEFIG in lines 1600 through 1130 is the program you saved with the merge option.

100 CALL DEFING(ING$)
110 CALL SPGET(\"DRAW\",DRAW$)
120 L=LEN(DRAW$)+3-41: 3 BYTES OF SPEECH OVERHEAD, 41 BYTES TRUNCATED
130 DRAW$=SEGS(DRAW$,1,2)&CHR$(1)&SEGS(DRAW$,4,1)
140 CALL SAY(\"WE ARE\",DRAW$&IN$,\"A1 SCREEN\")
150 GOTO 140
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,69
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 155,120,117,57,40,248
1050 DATA 135,173,209,25,39,45
1060 DATA 225,34,75,67,29,77
1070 DATA 106,91,44,57,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,134,227,222,24,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55:READ A$:A$=A$&CHR$(A$):NEXT I
1130 SUBEND
(Press SHIFT C to stop the program.)
Errors

The following lists all the error messages that TI Extended BASIC gives. The first list is alphabetical by the message that is given, and the second list is numeric by the number of the error that is returned by CALL ERR. If the error occurs in the execution of a program, the error message is often followed by IN line-number.

### Sorted by Message

<table>
<thead>
<tr>
<th>#</th>
<th>Message</th>
<th>Descriptions of Possible Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>BAD ARGUMENT</td>
<td>* Bad value given in ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQRT, TAN, or VAL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* An array element specified in a SUB statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Bad first parameter or too many parameters in LINK.</td>
</tr>
<tr>
<td>61</td>
<td>BAD LINE NUMBER</td>
<td>* Line number less than 1 or greater than 32767.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Omitted line number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Line number outside the range 1 through 32767 produced by RES.</td>
</tr>
<tr>
<td>57</td>
<td>BAD SUBSCRIPT</td>
<td>* Use of too large or too small subscript in an array.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Incorrect subscript in DIM.</td>
</tr>
<tr>
<td>79</td>
<td>BAD VALUE</td>
<td>* Incorrect value given in AND, CHAR, CHR$CLOSE, EOF, FOE, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, or XOR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Array subscript value greater than 32767.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* File number greater than 255 or less than zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* More than three times one noise generator specified in SOUND.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* A value passed to a subroutine is not acceptable in the subroutine. For example, a sprite velocity value less than -128 or a character value greater than 143.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Value in ON...GOTO or ON...GOSUB greater than the number of lines given.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Incorrect position given after the AT clause in ACCEPT or DISPLAY.</td>
</tr>
<tr>
<td>67</td>
<td>CAN'T CONTINUE</td>
<td>* Program has been edited after being stopped by a breakpoint.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Program was not stopped by a breakpoint.</td>
</tr>
<tr>
<td>69</td>
<td>COMMAND ILLEGAL IN PROGRAM</td>
<td>* EYE, CON, LIST, MERGE, NEW, NUL, OLD, RES, or SAVE used in a program.</td>
</tr>
</tbody>
</table>

### ERRORS

| 84  | DATA ERROR                     | * READ or RESTORE with data not present or with a string where a numeric value is expected.       |
|     |                                | * Line number after RESTORE is higher than the highest line number in the program.               |
|     |                                | * Error in object file in LOAD.                                                                   |
| 109 | FILE ERROR                     | * Wrong type of data read with a READ statement.                                                 |
|     |                                | * Attempt to use CLOSE, EOF, INPUT, OPEN, PRINT, PRINT USING, REC, or RESTORE with a file that does not exist or does not have the proper attributes. |
|     |                                | * Not enough memory to use a file.                                                                |
| 44  | FOR-NEXT NESTING               | * The FOR and NEXT statements of loops do not align properly.                                    |
|     |                                | * Missing NEXT statement.                                                                        |
| 130 | IO ERROR                       | * An error was detected in trying to execute CLOSE, DELETE, LOAD, MERGE, O.D, OPEX, RUN, or SAVE. |
|     |                                | * Not enough memory to list a program.                                                            |
| 16  | ILLEGAL AFTER SUBPROGRAM       | * Anything but END, REM, or SUB after a SUBEND.                                                    |
| 36  | IMAGE ERROR                    | * An error was detected in the use of DISPLAY USING, IMAGE, or PRINT USING.                      |
|     |                                | * More than 10 (E-format) or 14 (numeric format) significant digits in the format string.       |
|     |                                | * IMAGE string is longer than 254 characters.                                                     |
| 28  | IMPROPERLY USED NAME           | * An illegal variable name was used in CALL, LET, or DIM.                                         |
|     |                                | * Using a TI Extended BASIC reserved word in LET.                                                |
|     |                                | * Using a subscripted variable or a string variable in a FOR.                                   |
|     |                                | * Using an array with the wrong number of dimensions.                                            |
|     |                                | * Using a variable name differently then originally assigned. A variable can be only an array, a numeric or string variable, or a user defined function name. |
|     |                                | * Dimensioning an array twice.                                                                   |
|     |                                | * Putting a user defined function name on the left of the equals sign in an assignment statement. |
|     |                                | * Using the same variable twice in the parameter list of a SUB statement.                      |

212

TI Extended BASIC

213

TI Extended BASIC
ERRORS

81 INCORRECT ARGLMENT LIST
   - CALL and SUB mismatch of arguments.

83 INPUT ERROR
   - An error was detected in an INPUT.

60 LINE NOT FOUND
   - Incorrect line number found in BREAK, GOSUB, GOTO, ON ERROR, RUN, or UNBREAK, or after THEN or ELSE.
   - Line to be edited not found.

62 LINE TOO LONG
   - Line too long to be entered into a program.

39 MEMORY FULL
   - Program too large to execute one of the following: DEF, DELETE, LIM, GOSUB, LET, LOAD, ON...GOSUB, OPEN, or SUB.
   - Program too large to add a new line, insert a line, or evaluate an expression.

49 MISSING SUBEND
   - SUBEND missing in a subprogram.

47 MUST BE IN SUBPROGRAM
   - SUBEND or SUBEXIT not in a subprogram.

19 NAME TOO LONG
   - More than 15 characters in variable or subprogram name.

43 NEXT WITHOUT FOR
   - FOR statement missing, NEXT before FOR, incorrect FOR-NEXT nesting or branching into a FOR-NEXT loop.

78 NO PROGRAM PRESENT
   - No program present when issuing a LIST, RESEQUENCE, RESTORE, RUN, or SAVE command.

10 NUMERIC OVERFLOW
   - A number too large or too small resulting from a * , /, + , - operation or in ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN, or VAL.
   - A number outside the range –32768 to 32767 in PEEK or LOAD.

70 ONLY LEGAL IN A PROGRAM
   - One of the following statements was used as a command: DEF, GOSUB, GOTO, IF, MAGE, INPUT, CN BREAK, ON ERROR, ON...GOSUB, ON...GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND, or SUBEXIT.

25 OPTION BASE ERROR
   - OPTION BASE executed more than once, or with a value other than 1 or zero.

97 PROTECTION VIOLATION
   - Attempt to save, list, or edit a protected program.

48 RECURSIVE SUBPROGRAM CALL
   - Subprogram calls itself, directly or indirectly.

51 RETURN WITHOUT GOSUB
   - RETURN without a GOSUB or an error handled by the previous execution of an ON ERROR statement.

56 SPEECH STRING TOO LONG
   - Speech string returned by SPGET is longer than 255 characters.

40 STACK OVERFLOW
   - Too many sets of parentheses.
   - Not enough memory to evaluate an expression or assign a value.

54 STRING TRUNCATED
   - A string created by RPT$, concatenation ("&" operator), or a user defined function is longer than 255 characters.
   - The length of a string expression in the VALIDATE clause is greater than 254 characters.

24 STRING-NUMBER MISMATCH
   - A string was given where a number was expected or vice versa in a TI Extended BASIC supplied function or subprogram.
   - Assigning a string value to a numeric value or vice versa.
   - Attempting to concatenate ("&" operator) a number.
   - Using a string as a subscript.

334 UNEXPECTED Syntax error
   - Unexpected syntax error.

135 SUBPROGRAM NOT FOUND
   - A subprogram called does not exist or an assembly language subprogram named in LINT has not been loaded.
14 SYNTAX ERROR

- An error such as a missing or extra comma or parenthesis, parameters in the wrong order, missing parameters, missing keyword, misspelled keyword, keyword in the wrong order, or the like was detected in a TI Extended BASIC command, statement, function, or subroutine.
- DATA or IMAGE not first and only statement on a line.
- Items after final ";".
- Missing "#" in SPR.TE.
- Missing ENTER, un-comment symbol (!), or statement separator symbol (;).
- Missing THEN after IF.
- Missing TC after FCR.
- Nothing after CALL, SUB, FOR, THEN, or ELSE.
- Two E's in a numeric constant.
- Wrong parameter list in a TI Extended BASIC supplied subroutine.
- Going into or out of a subroutine with GOTO, GOSUB, ON ERROR, etc.
- Calling INIT without the Memory Expansion peripheral atached.
- Calling LINK or LOAD without first calling INIT.
- Using a constant where a variable is required.
- More than seven dimensions in an array.

17 UNMATCHED QUOTES

- Odd number of quotes in an input line.

20 UNRECOGNIZED CHARACTER

- An unrecognized character such as ? or % is not in a quoted string.
- A bad file in an object file accessed by LOAD.
IN CASE OF DIFFICULTY

If TIE Extended BASIC does not appear to be working properly, check the following:
1. Power — Be sure all devices are plugged in. Then turn on the power to the units in the proper sequence. Peripheral devices first, if you have them, followed by the console and monitor. Insert the TIE Extended BASIC module carefully.
2. Connector Separation — Check for proper alignment of the console and any accessory devices such as the Disk Drive Controller, Speech Synthesizer, and RS232 Interface. Remove and reinsert the TIE Extended BASIC module.
3. If none of the above procedures corrects the difficulty, consult “If You Have Questions or Need Assistance” or see the “Service Information” portion of the User’s Reference Guide that came with your computer.

If you have questions concerning module repair or peripheral, accessory, or software purchase, please call our Consumer Relations Department at (800) 858-4565 (toll free within the contiguous United States except Texas) or (800) 692-4279 within Texas. The operators at these numbers cannot provide technical assistance.

For technical questions about programming, specific applications, etc., you can call (606) 741-2663. Please note that this is not a toll-free number and collect calls cannot be accepted.

As an alternative, you can write to:
Consumer Relations Department
Texas Instruments Incorporated
P.O. Box 53
Lubbock, Texas 79408

Because of the number of suggestions which come to Texas Instruments from many sources containing both new and old ideas, Texas Instruments will consider such suggestions only if they are freely given to Texas Instruments. It is the policy of Texas Instruments to refuse to receive any suggestions in confidence. Therefore, if you wish to share your suggestions language program which you have developed, please include the following statement in your letter:

“All of the information forward hereon is presented to Texas Instruments on a nonconfidential, nonobligatory basis. No relationship, confidential or otherwise, expressed or implied, is established with Texas Instruments by this presentation. Texas Instruments may use, copy, distribute, publish, reproduce, or dispose of the information in any way without compensation to me.”
ADDENDUM

TI Extended BASIC Owner's Manual

The program listing on page 153 in the manual is incorrect. Line 110 should read:

>110 OPEN #1:“DSK1.RNDFILE”,RELATIVE,INTERNAL
IMPORTANT PRODUCT INFORMATION
FOR TI EXTENDED BASIC

TI Extended BASIC has been enhanced and modified for use with both the TI-99/4A and TI-99/4 Computers. Several important product differences should be noted in relation to the type of computer you have. Please read this folder and mark the appropriate changes in your copy of the TI Extended BASIC owner's manual.

Although the TI-99/4A and TI-99/4 Computers are similar, the TI-99/4A is easily recognizable by its standard typewriter keyboard which returns both upper-case (large capital) and lower-case (small capital) alphabetical characters. Depressing the ALPHA LOCK key locks the alphabet keys in upper-case mode. To release ALPHA LOCK press the key again.

When the TI Extended BASIC module is in place, both the TI-99/4A and TI-99/4 Computers share several enhancements. However, each computer also has its own unique features. These features are discussed in the following paragraphs.

AUTO REPEAT FEATURE

When using TI Extended BASIC on either computer, holding down a key for more than one second automatically causes its symbol to be repeated on the display until you release the key.
SPECIAL FUNCTION KEYS

The TI-99/4A Computer has the same special computer functions as the TI-99/4. However, these functions are frequently assigned to different keys on the TI-99/4A Computer. The following chart compares the keystroke sequences for the function keys on the two units.

<table>
<thead>
<tr>
<th>Key Name</th>
<th>TI-99/4 Keys</th>
<th>TI-99/4A Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>AID</td>
<td>SHIFT A</td>
<td>FCTN 7</td>
</tr>
<tr>
<td>CLEAR</td>
<td>SHIFT C</td>
<td>FCTN 4</td>
</tr>
<tr>
<td>DELete</td>
<td>SHIFT F</td>
<td>FCTN 1</td>
</tr>
<tr>
<td>INSert</td>
<td>SHIFT G</td>
<td>FCTN 2</td>
</tr>
<tr>
<td>QUIT</td>
<td>SHIFT Q</td>
<td>FCTN =</td>
</tr>
<tr>
<td>REDO</td>
<td>SHIFT R</td>
<td>FCTN 8</td>
</tr>
<tr>
<td>ERASE</td>
<td>SHIFT T</td>
<td>FCTN 3</td>
</tr>
<tr>
<td>LEFT arrow</td>
<td>SHIFT S</td>
<td>FCTN S</td>
</tr>
<tr>
<td>RIGHT arrow</td>
<td>SHIFT D</td>
<td>FCTN D</td>
</tr>
<tr>
<td>DOWN arrow</td>
<td>SHIFT X</td>
<td>FCTN X</td>
</tr>
<tr>
<td>UP arrow</td>
<td>SHIFT E</td>
<td>FCTN E</td>
</tr>
<tr>
<td>PROC'D</td>
<td>SHIFT V</td>
<td>FCTN 8</td>
</tr>
<tr>
<td>BEGIN</td>
<td>SHIFT W</td>
<td>FCTN 5</td>
</tr>
<tr>
<td>BACK</td>
<td>SHIFT Z</td>
<td>FCTN 9</td>
</tr>
<tr>
<td>ENTER</td>
<td>ENTER</td>
<td>ENTER</td>
</tr>
</tbody>
</table>

In addition to these functions, the TI-99/4A Computer has functions represented as symbols on the fronts of the individual keyfaces. These functions may be accessed by pressing FCTN and the appropriate key simultaneously.

CONTROL KEYS

The TI-99/4A Computer also has control characters which are used primarily for telecommunications. To enter a control character, hold down the CTRL key and press the appropriate letter, number, or symbol key.
EXPANDED CHARACTER SET — TI-99/4A

As explained in your TI Extended BASIC manual, codes 32-95 are the predefined standard ASCII characters on the TI-99/4 Computer. The cursor and edge characters, ASCII codes 30 and 31, are assigned to character set 0. The undefined character codes (128-135 and 136-143) are assigned to sets 13 and 14, respectively.

These codes and the corresponding characters are listed in Appendix C of the manual. TheCALI KEY character codes are also listed in Appendix C. Appendix E in the manual lists the 15 character code sets which may be used for color graphics.

Due to the inclusion of the lowercase character set, the defined characters on the TI-99/4A Computer are the standard ASCII characters for codes 32 through 127. The following chart lists these characters and their codes.

<table>
<thead>
<tr>
<th>ASCII CODE</th>
<th>CHARACTER</th>
<th>ASCII CODE</th>
<th>CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>❱ (cursor)</td>
<td>55</td>
<td>7</td>
</tr>
<tr>
<td>31</td>
<td>❱ (edge character)</td>
<td>56</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>(space)</td>
<td>57</td>
<td>9</td>
</tr>
<tr>
<td>33</td>
<td>❯ (exclamation point)</td>
<td>58</td>
<td>⟨ (less than)</td>
</tr>
<tr>
<td>34</td>
<td>&quot; (quote)</td>
<td>59</td>
<td>⟨ (semicolon)</td>
</tr>
<tr>
<td>35</td>
<td># (number or pound sign)</td>
<td>60</td>
<td>⟨ (less than)</td>
</tr>
<tr>
<td>36</td>
<td>$ (dollar)</td>
<td>61</td>
<td>= (equals)</td>
</tr>
<tr>
<td>37</td>
<td>% (percent)</td>
<td>62</td>
<td>&gt; (greater than)</td>
</tr>
<tr>
<td>38</td>
<td>&amp; (ampersand)</td>
<td>63</td>
<td>? (question mark)</td>
</tr>
<tr>
<td>39</td>
<td>′ (apostrophe)</td>
<td>64</td>
<td>@ (at sign)</td>
</tr>
<tr>
<td>40</td>
<td>⟨ (open parenthesis)</td>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>41</td>
<td>⟨ (close parenthesis)</td>
<td>66</td>
<td>B</td>
</tr>
<tr>
<td>42</td>
<td>* (asterisk)</td>
<td>67</td>
<td>C</td>
</tr>
<tr>
<td>43</td>
<td>+ (plus)</td>
<td>68</td>
<td>D</td>
</tr>
<tr>
<td>44</td>
<td>, (comma)</td>
<td>69</td>
<td>E</td>
</tr>
<tr>
<td>45</td>
<td>− (minus)</td>
<td>70</td>
<td>F</td>
</tr>
<tr>
<td>46</td>
<td>. (period)</td>
<td>71</td>
<td>G</td>
</tr>
<tr>
<td>47</td>
<td>/ (slant)</td>
<td>72</td>
<td>H</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>73</td>
<td>I</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>74</td>
<td>J</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>75</td>
<td>K</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td>76</td>
<td>L</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>77</td>
<td>M</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>78</td>
<td>N</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td>79</td>
<td>O</td>
</tr>
<tr>
<td>ASCII CODE</td>
<td>CHARACTER</td>
<td>ASCII CODE</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>81</td>
<td>Q</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>82</td>
<td>R</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>83</td>
<td>S</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>84</td>
<td>`</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>85</td>
<td>U</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>86</td>
<td>V</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>87</td>
<td>W</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>88</td>
<td>X</td>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>89</td>
<td>Y</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>91</td>
<td>[</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>92</td>
<td>\ (reverse slant)</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>93</td>
<td>] (close bracket)</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>94</td>
<td>^ (exponentiation)</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>95</td>
<td>_ (line)</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>96</td>
<td>` (grave)</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>97</td>
<td>a</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>98</td>
<td>b</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>99</td>
<td>c</td>
<td>123</td>
<td>{ (left brace)</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>e</td>
<td>125</td>
<td>} (right brace)</td>
</tr>
<tr>
<td>102</td>
<td>f</td>
<td>126</td>
<td>~ (tilde)</td>
</tr>
<tr>
<td>103</td>
<td>g</td>
<td>127</td>
<td>DEL (appears on screen as a blank)</td>
</tr>
</tbody>
</table>

**CALL KEY SUBPROGRAM**

The information given on the KEY subprogram in Chapter 4 of the TI Extended BASIC manual is accurate for the TI-99/4 Computer. The values of 3, 4, and 5 are not accessible as key units.

However, the TI-99/4A maps key units 0 through 5 to specific modes of operation. If the key-unit is 0, the keyboard is mapped in whichever mode was specified by the previous CALL KEY program line.

If the key-unit is 1, input is taken from the left side of the keyboard. If the key-unit is 2, input is taken from the right side of the keyboard.

A key-unit of 3 maps the computer into the standard TI-99/4 keyboard mode. Both upper- and lower-case characters are returned as upper-case characters only. Function codes 1 through 15 are active, but no control characters are returned.
A key-unit of 4 places the computer in Pascal mode with both upper- and lower-case characters active. The function codes 129 through 143 and the control character codes 1 through 31 are also active.

The key-unit 5 maps the TI-99/4A Computer in the BASIC mode. Both upper- and lower-case characters are active. The active function codes are 1 through 15, and the active control character codes are 128 through 159 (and 187).

In addition, codes are assigned to the function and control keys so that these can be referenced by the CALL KEY subprogram in TI Extended BASIC. The codes assigned depend on the key-unit value specified in a CALL KEY program statement. The following table shows typical code assignments.

**FUNCTION KEY CODES**

<table>
<thead>
<tr>
<th>Codes</th>
<th>Pascal Mode</th>
<th>Function Name</th>
<th>Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI-99/4 &amp; BASIC Modes</td>
<td>129</td>
<td>Aid</td>
<td>FCTN 7</td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>CLEAR</td>
<td>FCTN 4</td>
</tr>
<tr>
<td>2</td>
<td>131</td>
<td>DELETE</td>
<td>FCTN 1</td>
</tr>
<tr>
<td>3</td>
<td>132</td>
<td>INSERT</td>
<td>FCTN 2</td>
</tr>
<tr>
<td>4</td>
<td>133</td>
<td>QUIT</td>
<td>FCTN =</td>
</tr>
<tr>
<td>5</td>
<td>134</td>
<td>REDO</td>
<td>FCTN 8</td>
</tr>
<tr>
<td>6</td>
<td>135</td>
<td>ERASE</td>
<td>FCTN 3</td>
</tr>
<tr>
<td>7</td>
<td>136</td>
<td>LEFT arrow</td>
<td>FCTN S</td>
</tr>
<tr>
<td>8</td>
<td>137</td>
<td>RIGHT arrow</td>
<td>FCTN D</td>
</tr>
<tr>
<td>9</td>
<td>138</td>
<td>DOWN arrow</td>
<td>FCTN X</td>
</tr>
<tr>
<td>10</td>
<td>139</td>
<td>UP arrow</td>
<td>FCTN E</td>
</tr>
<tr>
<td>11</td>
<td>140</td>
<td>PROC'D</td>
<td>FCTN 6</td>
</tr>
<tr>
<td>12</td>
<td>141</td>
<td>ENTER</td>
<td>ENTER</td>
</tr>
<tr>
<td>13</td>
<td>142</td>
<td>BEGIN</td>
<td>FCTN 5</td>
</tr>
<tr>
<td>14</td>
<td>143</td>
<td>BACK</td>
<td>FCTN 9</td>
</tr>
</tbody>
</table>
## CONTROL KEY CODES

<table>
<thead>
<tr>
<th>Mode</th>
<th>BASIC Mode</th>
<th>Code</th>
<th>Press</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
<td>1</td>
<td>SOH</td>
<td>CONTROL A</td>
<td>Start of heading</td>
</tr>
<tr>
<td>130</td>
<td>2</td>
<td>STX</td>
<td>CONTROL B</td>
<td>Start of text</td>
</tr>
<tr>
<td>131</td>
<td>3</td>
<td>ETX</td>
<td>CONTROL C</td>
<td>End of text</td>
</tr>
<tr>
<td>132</td>
<td>4</td>
<td>EOT</td>
<td>CONTROL D</td>
<td>End of transmission</td>
</tr>
<tr>
<td>133</td>
<td>5</td>
<td>ENQ</td>
<td>CONTROL E</td>
<td>Enquiry</td>
</tr>
<tr>
<td>134</td>
<td>6</td>
<td>ACK</td>
<td>CONTROL F</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>135</td>
<td>7</td>
<td>BEL</td>
<td>CONTROL G</td>
<td>Bell</td>
</tr>
<tr>
<td>136</td>
<td>8</td>
<td>BS</td>
<td>CONTROL H</td>
<td>Backspace</td>
</tr>
<tr>
<td>137</td>
<td>9</td>
<td>HT</td>
<td>CONTROL I</td>
<td>Horizontal tabulation</td>
</tr>
<tr>
<td>138</td>
<td>10</td>
<td>LF</td>
<td>CONTROL J</td>
<td>Line feed</td>
</tr>
<tr>
<td>139</td>
<td>11</td>
<td>VT</td>
<td>CONTROL K</td>
<td>Vertical tabulation</td>
</tr>
<tr>
<td>140</td>
<td>12</td>
<td>FF</td>
<td>CONTROL L</td>
<td>Form feed</td>
</tr>
<tr>
<td>141</td>
<td>13</td>
<td>CR</td>
<td>CONTROL M</td>
<td>Carriage return</td>
</tr>
<tr>
<td>142</td>
<td>14</td>
<td>SO</td>
<td>CONTROL N</td>
<td>Snit out</td>
</tr>
<tr>
<td>143</td>
<td>15</td>
<td>SI</td>
<td>CONTROL O</td>
<td>Shift in</td>
</tr>
<tr>
<td>144</td>
<td>16</td>
<td>DLE</td>
<td>CONTROL P</td>
<td>Data link escape</td>
</tr>
<tr>
<td>145</td>
<td>17</td>
<td>DC1</td>
<td>CONTROL Q</td>
<td>Device control 1 (X-ON)</td>
</tr>
<tr>
<td>146</td>
<td>18</td>
<td>DC2</td>
<td>CONTROL R</td>
<td>Device control 2</td>
</tr>
<tr>
<td>147</td>
<td>19</td>
<td>DC3</td>
<td>CONTROL S</td>
<td>Device control 3 (X-OFF)</td>
</tr>
<tr>
<td>148</td>
<td>20</td>
<td>DC4</td>
<td>CONTROL T</td>
<td>Device control 4</td>
</tr>
<tr>
<td>149</td>
<td>21</td>
<td>NAK</td>
<td>CONTROL U</td>
<td>Negative acknowledge</td>
</tr>
<tr>
<td>150</td>
<td>22</td>
<td>SYN</td>
<td>CONTROL V</td>
<td>Synchronous idle</td>
</tr>
<tr>
<td>151</td>
<td>23</td>
<td>ETB</td>
<td>CONTROL W</td>
<td>End of transmission block</td>
</tr>
<tr>
<td>152</td>
<td>24</td>
<td>CAN</td>
<td>CONTROL X</td>
<td>Cancel</td>
</tr>
<tr>
<td>153</td>
<td>25</td>
<td>EM</td>
<td>CONTROL Y</td>
<td>End of medium</td>
</tr>
<tr>
<td>154</td>
<td>26</td>
<td>SUB</td>
<td>CONTROL Z</td>
<td>Substitute</td>
</tr>
<tr>
<td>155</td>
<td>27</td>
<td>ESC</td>
<td>CONTROL .</td>
<td>Escape</td>
</tr>
<tr>
<td>156</td>
<td>28</td>
<td>FS</td>
<td>CONTROL :</td>
<td>File separator</td>
</tr>
<tr>
<td>157</td>
<td>29</td>
<td>GS</td>
<td>CONTROL =</td>
<td>Group separator</td>
</tr>
<tr>
<td>158</td>
<td>30</td>
<td>RS</td>
<td>CONTROL 8</td>
<td>Record separator</td>
</tr>
<tr>
<td>159</td>
<td>31</td>
<td>US</td>
<td>CONTROL 9</td>
<td>Unit separator</td>
</tr>
</tbody>
</table>

You may also obtain detailed CALL KEY subprogram information, including keyboard diagrams, in your User's Reference Guide for the TI-90/4A Computer.

## CALL VERSION SUBPROGRAM

The VERSION subprogram (discussed in Chapter 4 of your TI Extended BASIC manual) now returns a value of 110 on both computers.
DATA STATEMENT

The computer reads any information entered after a DATA statement as a part of the DATA statement. Therefore, in a multi-statement program line, a DATA statement should not be followed by another statement.

SCIENTIFIC NOTATION

Whenever you use scientific (or exponential) notation, be certain that the "E" is an upper-case (large capital) character. A lowercase "e" may cause your program to function improperly.

PRE-SCAN — l@P - and l@P +

After you enter RUN to start a program, you may notice a pause before the program actually begins. This pause is the time the computer takes to "pre-scan" your program to establish memory space for variables, arrays, and data. Then the computer proceeds through each instruction, performs the appropriate functions, and establishes variable values. Since the time required to pre-scan depends on the length of the program, you may want to decrease the pre-scan pause, particularly if you have a long program.

11 Extended BASIC's new pre-scan commands, l@P - and l@P +, allow you to control which instructions will not be pre-scanned. Because the purpose of the pre-scan is to set memory space for variables, only those instructions which contain the first reference to the variables need to be pre-scanned. Therefore, many other instructions in your program do not require a pre-scan.

Careful program planning is required to minimize the statements that need the pre-scan. When certain types of statements (as explained here) are used in your program, the procedures listed below should be included in the pre-scan.

- Enter your first DATA statement within the pre-scan.
- Include the first use of each variable and/or array. (Also, include the OPTION BASE statement, if used.)
- Include the first reference to each CALL statement of any subprogram.
- Include all DEF statements for user-defined functions.
- Include all SUB statements and SUBEND statements in the pre-scan.
Note that a variable in a user-defined (SUB) subprogram is considered to be unique from any other variable used elsewhere in your program, even though the name and value may be the same. Therefore, each variable used in a user-defined subprogram must be included in the pre-scan.

To use the pre-scan option, first be certain that your completed program runs successfully. Then, at the beginning of a group of function statements, use the !@P - command to "turn off" the pre-scan. The following statements will not be pre-scanned, allowing the execution of your program to begin more quickly. Any statements related to variable names (not previously referenced during pre-scan) return a syntax error if the pre-scan is "off." Note that !@P - cannot be followed by another statement in a multiple statement.

To resume the pre-scan, simply enter the command !@P +. This command causes the pre-scan to "turn on" and memory space for variables may be set. Remember to use the !@P + command before a SUB or SUBEND statement and do not incorporate this command as a part of a multiple statement.

You may choose to use the pre-scan feature several times throughout your program. By turning the pre-scan on and off, your program can begin to execute more efficiently. The effectiveness of the pre-scan is more noticeable in large programs than small programs. Note that when using the TI-99/4A Computer, the commands, !@P - and !@P +, may also be entered with a lower-case "p" character.

The following examples illustrate how to include the pre-scan statements in an existing program. The final example demonstrates the most efficient use of the pre-scan feature by making use of a GOTO statement.
Examples:

Original program:

```
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"OPOPOPOPOPOPOPOPO")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 DATA J
220 .
230 .
```

With pre-scan control added:

```
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"OPOPOPOPOPOPOPOPO")
125 ;@P-
130 .
140 .
150 .
155 ;@P+
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
185 ;@P-
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Notice that the first DATA statement has been moved to the beginning of the program so that it is included in the pre-scan. By including statements 125, 155, and 185, the pre-scan is turned off and on and off again. This causes the program to begin to execute more quickly.
With GOTO added:

You have the added ability to "trick" the computer into establishing memory space for CALL statements, as well as variable related statements, without actually performing those statements. To do this, simply use a GOTO instruction in your program. The following example demonstrates the original program adapted with a pre-scan and a GOTO statement.

```
10 DATA 3
20 GOTO 100::DELAY::CALL CHAR::CALL CLEAR::CALL HCHAR::CALL VCHAR::@P-
100 CALL CLEAR
110 CALL CHAR(50, "FFFFFFFFFF")
120 CALL CHAR(42, "OFPOPOPOPOPOPOP")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Note that the GOTO method causes the necessary memory space to be reserved in line 20. However, the statements in line 20 do not execute until they are encountered further on in the program. Thus, as shown in the preceding and following examples, you can put all of your variable references together and your subprogram calls do not have to be syntactically correct. This can be the most efficient use of the pre-scan option.

```
100 GOTO 180::X,Y,ALPHA,BETA,Z=DELTA::DIM B(10,10)
110 CALL KEY::CALL HCHAR::CALL CLEAR::CALL MYSUB
120 DATA 1,2,STRING
130 DEF F(X)=1-X*SIN(X)
140 .
150 .
160 .
170 IF-
180 .
190 .
200 .
```
PROGRAMMING WITH LOWER-CASE LETTERS

Device names must be entered in upper-case (large capital) letters only. For example, "Dsk1" is a correct device name, but "DskA" is not. Any reference to a device name spelled in lower-case (small capital) letters results in an error message.

File names are also very specific. Not only are they exact as to the correct spelling, but they are also specific as to the use of upper- or lower-case letters. For example, the file name, MYFILE, is not the same file as Myfile (a combination of large and small capital letters). Any file name listed in part or whole by lower-case letters is not accessible by the TI-99/4 Computer. Only the TI-99/4A Computer can access a program named or called in lower-case letters.

Lower-case letters in DATA statements or quoted strings function correctly and offer a wide variety of programming techniques on the TI-99/4A Computer. However, lower-case quoted strings and data are not displayed if you run the program on a TI-99/4 Computer. If you plan to run your program on both the TI-99/4A and TI-99/4 Computers, take special care when using lower-case letters.

To display the lower-case letters in your TI-99/4A Computer program when the program is run on a TI-99/4 Computer, simply include the following statements. Small capital letters are created similar to those of the TI-99/4A. Be sure to allow adequate memory space and execution time.

```
100 FOR I=65 TO 90
110 CALL CHAR$(I,A$)
120 DD$ UNI$ 00$I$(A$,1,4)$&UNI$(A$,7,4)$&UNI$(A$,13,4)
130 CALL CHAR$(I+32,B$)
140 NEXT I
```

Insertion of the above program lines into your TI-99/4A program allows pre-programmed lower-case characters to be displayed by the TI-99/4 computer.

SIZE COMMAND

The SIZE example, using the Memory Expansion unit discussed in Chapter 4 of your TI Extended BASIC manual, now informs you that you have 24488 "BYTES OF PROGRAM SPACE FREE".

TAIL REMARKS

If you previously programmed a TAIL REMark that is identical to the pre-scan instructions (l@P+ or l@P+), your program will no longer function properly. These groups of characters are now considered to be "reserved words" for the operation of the computer.
CORRECTION TO APPENDIX C

ASCII code 12 in Appendix C of your TI Extended BASIC manual should be stated as the "TRIG'D" character rather than as the "CMD" character.

LARGE PROGRAM FILES

Some programs written with TI BASIC may be too large to run with TI Extended BASIC because TI Extended BASIC requires more system overhead than TI BASIC. If you attempt to load such a program, your system will lock up. Before you can continue, you must turn your computer off, wait several seconds, and then turn it on again.

Entering a CALL FILES(1) or CALL FILES(2) command before loading your program may free enough memory to run the program with TI Extended BASIC. (A full explanation of the CALL FILES command can be found in the Disk Memory System manual.)

If a CALL FILES command does not free enough memory, you must shorten your TI BASIC program by deleting statements until the program fits in the memory available with TI Extended BASIC. However, if you have a Memory Expansion unit, you can run the entire program by using the following procedure:

1. As a safety measure, make a backup copy of your TI BASIC program on a cassette tape or diskette.
2. With the Memory Expansion unit attached and turned on, load your program with TI BASIC. Next, delete several statements, and save the shortened program on cassette tape or diskette. Then try to load this shortened program with TI Extended BASIC.
3. Type the deleted statements back into the proper places in your program.
4. Save your program on a diskette only. You are now ready to run your program with TI Extended BASIC and the Memory Expansion unit.

Note: Programs converted in this fashion can only be run with TI Extended BASIC and with the Memory Expansion unit attached and turned on. They are not stored in PROGRAM format.
MEMORY EXPANSION UNIT AND CASSETTE-BASED PROGRAMS

The Memory Expansion unit adds 32K bytes of Random Access Memory (RAM) to the built-in memory of the computer. However, even with the Memory Expansion unit available, the largest TI Extended BASIC program that can be stored on a cassette tape is 12K bytes in size. Note that, although the length of the actual program is limited, utilizing the Memory Expansion unit provides other advantages. For example, with the unit attached and turned on, your program (which can be up to 12K bytes in length) is stored in the expansion RAM. The numeric data generated by the program is stored in the Memory Expansion unit and the string data is stored in the computer's built-in memory. Without the unit, the program must be shorter so that both it and the generated data can be stored in the computer's built-in memory.

CONTINUE COMMAND

A CONTINUE command is used to resume your program when you break by using a BREAK command or by pressing CLEAR. However, if your last command (before the CONTINUE command) results in an error, the program may not continue properly. Your final command to the computer before the CONTINUE command must be correct. If you receive an ERROR message, be sure to enter a correct command, such as a PRINT command, before resuming program execution.

MANUAL ERRORS

Page 30

The second sentence in the third paragraph of the "Numeric Constants" section should be corrected to read "...number is greater than 99 or less than -99, then .....".

Pages 79 and 150

The string used in a string-expression with the DISPLAY ... USING and PRINT ... USING statements may be more general than shown in the examples in the manual. For example, both of the following are valid statements.

    PRINT USING A$:X,Y
    DISPLAY USING BPT$("#",5)&Y$:A(12)

Pages 59, 133, and 135

The GOSUB, ON GOSUB, and ON GOTO statements should not be used to transfer control to and from subprograms.
Page 114
If you press **CLEAR** when using the LIST command, the listing stops and cannot be restarted.

Pages 118 and 119
The graphic figures at the bottom of page 118 and the top of 119 should be reversed.

Page 185
The TAR function cannot be used in the PRINT ... USING or DISPLAY ... USING statements. Also, the second paragraph of this explanation should be corrected to read as follows: "If the number of characters already printed on the current record is less than `numeric-expression`, the next `print-item` is printed beginning on the position indicated by `numeric-expression`. If the number of characters already printed on the current record is greater than or equal to the position indicated by `numeric-expression`, the next `print-item` is printed on the next record beginning in the position indicated by `numeric-expression`.

Page 200
In Appendix H, Color Combinations, the color codes for the last two listings in the "Best" category should be as follows:

- 14, 10  Magenta on Light Red
- 3, 16  Medium Green on White

In the "Fourth Best" category, the third combination in the second column should read:

- 6, 2  Light Blue on Black
Format Lines

The following list gives corrections that should be made to the indicated formats and also shows the present format information.

DIM Statement (page 76)
   Correct Format: (integer1[,integer2][..][,integer7])[..]
   Present Format: (integer1[,integer2][..][,integer7])[..]

DISPLAY Statement (page 77)
   Correct Format: [SIZE (numeric-expression)]::print-list
   Present Format: [SIZE (numeric-expression)]::variable-list

DISPLAY ... USING Statement (page 79)
   Correct Formats: USING string-expression::print-list
                   USING line-number::print-list
   Present Formats: USING string-expression::variable-list
                   USING line-number::variable-list

LIINPUT Statement (page 113)
   Correct Format: *file-number[.REC record-number]:
   Present Format: *[file-number][.REC record number]:

PRINT ... USING Statement (page 150)
   Correct Format: *[file-number][.REC record-number],
   Present Format: *[file-number][.REC record-number]

SPRITE Subprogram (page 173)
   Correct Format: dot-column[,row-velocity,column-velocity][,...])
   Present Format: dot-column[,row-velocity,column-velocity][,...])