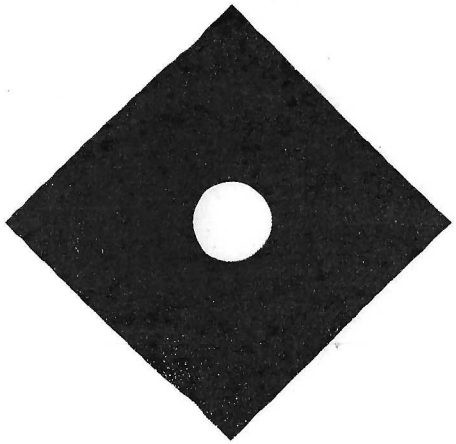

COMPress
A Division of Van Nostrand Reinhold Company, Inc.
P.O. Box 102, Wentworth, NH 03282 (603)764-5831/5225

ISBN 0-933694-83-0

EnBASIC™ AUTHORING SYSTEM

COMPress A Division of Van Nostrand Reinhold Company, Inc.

EnBASIC™
AUTHORING SYSTEM

FOR THE APPLE II PLUS*, //e*
& FRANKLIN ACE 1000**

Paul Tenczar, Stanley Smith and Allen Avner

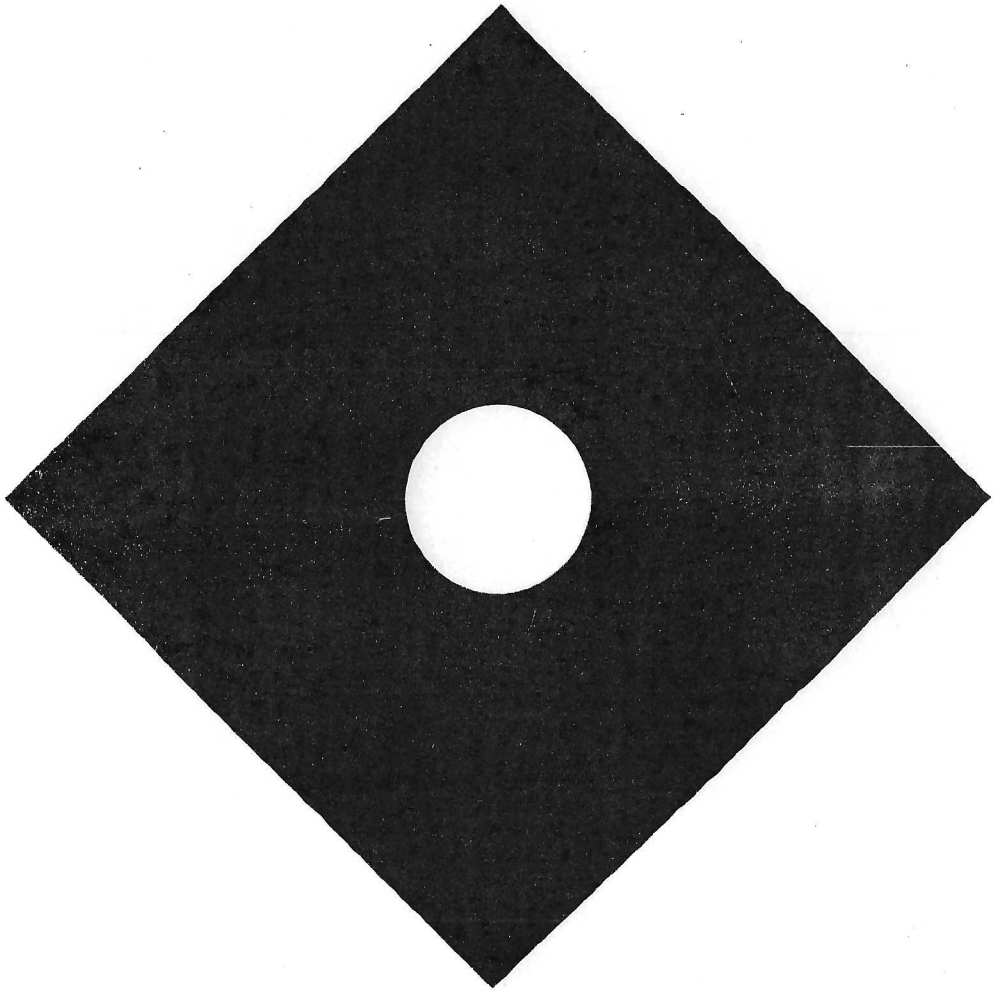

TABLE OF CONTENTS

Tables

Appendices

CHAPTER I

Introduction

This package enhances BASIC for users of the Apple][+ and //e who produce computer-assisted instruction (CAI) and other application packages requiring a "friendly" user interface. Users of the package are assumed to be familiar with BASIC and with general techniques of CAI or software design. The package is designed for Apple //e systems and for Apple][+ systems equipped with ROM Applesoft BASIC, 3.3 DOS, 48K RAM memory, and at least one Apple][disk drive.

The package consists of this manual, a non-copyable, write-protected Master diskette, and a copyable Demo diskette. The Master diskette contains the EnBASIC augmentation routines, standard character and key tables, display-table and key-table editors, interactive demonstrations of features, and a program for setting up diskettes to use the features in your own user-oriented materials. The copyable Demo diskette contains sample programs that allow you to examine working applications of the package.

Each chapter in the main section of this manual provides basic information on a major aspect of the package. Where necessary, this basic information is supplemented by appendices describing advanced methods or details.

In this chapter, with the aid of programs on the Demo and Master diskettes, we will provide an overview of the features of the enhancement package, an outline of what it seeks to do for you, and some practice in using the input procedures provided by the package.

In the next chapter you will be shown how to set up your own test diskette with the enhancement features. You will use this test diskette for writing short sample programs as you go through the descriptions of the EnBASIC commands and editors contained in the rest of the manual.

Let us start the overview now. Place the Master diskette in Drive I of your Apple and boot it by switching the Apple on (or, if it is already on, by using the normal procedures for booting a diskette). Within a few seconds after the diskette has been read, a title display appears.

Follow the instructions on your screen to advance through the title displays until you reach the MAIN INDEX. Select "Demonstration of Features". Here is a summary of the keys that are available during the demo:

RETURN	To move forward
CTRL-B	To move backward one display
ESC	To return to the Main index

When the demonstration program is completed it will give you the choice of returning to the Index display or repeating the demonstration. You may go through the overview more than once if you wish. When you are done, return here. Go ahead and look at the demonstration program now.

Design Goals of the Package

A major goal of this package is to allow high quality CAI materials to be designed for settings (such as public schools) which cannot afford hardware modification of terminals for such features as key-buffering and lower-case characters--capabilities which are necessary for most cost-effective interactive computer applications.

Software designed for use by students or others with minimal computer experience must be forgiving of common human errors and of the ambiguities of communication in human language. Displays and interactions must be in a form acceptable to those who are not familiar with computers and must be programmed efficiently enough to avoid degrading computer performance. Finally, the software that does all of this must usually be produced within severe time limitations.

Unfortunately, the languages most often used in software production (such as BASIC for the Apple) are usually not designed to ease production of user-oriented materials.

In the field of CAI, some of these aims are met by authoring systems which are intended to simplify production of instructional materials that fit a specific teaching approach. These systems allow rapid production of material in a fixed or restricted format that provides effective learning in certain limited situations.

However, the advantages of such systems are often gained by preventing or limiting use of the computer in simulations and other types of advanced applications. Some of these systems also make inefficient use of computer resources, leading to large accessing or processing delays for the user. Processing inefficiencies that lead to large delays in user-machine interactions are rarely acceptable for serious applications.

Serious software designers must find a working compromise between the conflicting goals of efficient production of user-oriented software and access to the full power of the computer. Most find this compromise by using a standard language, such as BASIC, augmented by a library of locally produced editors, subroutines, template programs, and other production tools for increasing production rate and quality.

This package is a ready-made collection of such augmentation tools. Its design is based on direct experience in CAI and user-oriented software over the past 2Ø years for a wide range of systems and user needs. The package was developed by identifying those routines which experience has shown to be on the "critical path" to preparation of high-quality materials.

Features of the Augmentation Package

In many learning situations, users must be able to respond with words, phrases, or sentences during their interaction with the instructional material. This package uses artificial intelligence approaches to meet the range of practical demands encountered in use of constructed, natural-language user responses. It permits discourse within a limited context by automatically handling misspelled words and alternative phrasings witnout requiring that the designer specify every alternative or erroneous form. Further, it performs these tasks without the processing delays often seen in real-time applications of artificial intelligence.

A flexible character-generator facility allows the design of character sets for special applications. Characters may range in height from 6 to 16 dots and in width from 1 to 7 dots (or 1 to 9 dots, if character-separation spaces are included). Standard options of this generator allow for full upper- and lower-case character sets, auto-backspace characters (such as accent marks), superscripts, subscripts, and underlining. Line spacing control is independent of character size. Text may be plotted with fixed-character spacing or with automatic proportional spacing in both normal- and double-sized type. Special modes include rewrite, overstrike, inverse, exclusive-or, and erase mode. These character generator features also allow design of multi-character graphics and effective animations using only a single High Resolution page (HGR2).

Finally, keyboard functions may be altered to meet special needs or to match conventions used by existing materials. Character displays are controlled by specification tables referred to as "display tables". Keyboard effects are controlled by similar specification tables referred to as "key tables". Two particular tables (known as the "Standard Display Table" and the "Standard Key Table") are used by all Master diskette options. Thus, keys pressed while you are using options on the Master diskette obey the Standard Key Table and produce displays made up of characters contained in the Standard Display Table. Appendix D summarizes these effects.

When writing your own software, you will have the option of using these standard tables or of producing your own tables using the table editors. The standard tables provide many more features than would normally be appropriate for user-interaction since they serve both as examples of what features are available and as flexible tools for accessing the editor features.

User Input

Let us now go to the Demo diskette for some direct practice. Boot the Demo diskette and follow the directions to get to its index. Select the sample program titled "Answer Judging". We will use this simple example as a test program to investigate the editing and feedback features available to users. Select "Answer Judging" now, but read the next few paragraphs before starting (we will tell you when to start!).

The program consists only of a single request: "Give another name for the star Polaris." You may respond with a single word or a complete sentence. The "correct" answer is "North (or Pole) Star". Before trying some answers, let us examine the operation of the Erase and Edit function keys:

→	To edit an answer
←	To erase a character

Each time the erase key is pressed while you are typing or editing a line, the last character on the line is erased. The edit key (→) works on word boundaries. The first time it is pressed, the entire line is removed from the screen. On second and subsequent presses of the edit key, a word of the line is replaced until finally all words are returned. Here is a short example of using the edit and erase keys to correct an error in a short sentence:

(sentence is typed)	IT IT THE NORTH STAR	
(press "→" once)		(sentence disappears)
(press "→" again)	IT	(first word returns)
(press "→" again)	IT IT	(next word returns)
(press "←" once)	IT I	("T" is erased)
(type "S")	IT IS	
(press "→" once)	IT IS THE	(next word returns)
(press "→" again)	IT IS THE NORTH	(next word returns)
(press "→" again)	IT IS THE NORTH STAR	

Try the Polaris example now. Do not press RETURN when done. When you are finished exploring the Edit and Erase keys, come back here to learn about other special keys.

The next function we will examine allows us to shift between upper and lower case letters, thus overcoming the lack of a functional letter shift key on the unmodified Apple][:

ESC-ESC or @-@
to Toggle between upper and lower case
(where ESC-ESC means "press ESC twice")

Press the ESC or @ key several times. Notice that the prompt arrow changes with each press. With a single press, it displays a square which indicates that a special two-key sequence has been started (more about this later). With the second press, one of two types of arrows appears:

Prompt	Meaning
■	Waiting for second key of two-key sequence
↱	Next key typed will be in upper case
↳	Next key typed will be in lower case

Try typing now and shift between upper and lower case with ESC-ESC or @-@. Also try CTRL-T (yet another case-toggle).

The ESC and @ keys serve a very special purpose in addition to allowing upper-lower case shifting. They also allow access to many special functions by redefining the function of a key when it is the second key in an ESC-___ or @-___ sequence. The square prompt warns you that you are half-way through such a two-key sequence. One such two-key sequence allows characters to be shifted up or down:

ESC-U	or @-U	Next key typed will be superscripted
ESC-D	or @-D	Next key typed will be subscripted

When pressed alone, keys U and D simply produce a "U" or a "D" on the screen. But when pressed as the second key in a two-key "Access" sequence they produce superscripting or subscripting. The prompt arrow is modified to warn you that superscripting or subscripting is about to take place.

↱↑	Next key will be superscripted upper case
↱↓	Next key will be subscripted upper case
↳↑	Next key will be superscripted lower case
↳↓	Next key will be subscripted lower case

Now type a mixture of upper- and lower-case letters. Mix in some superscripted and subscripted characters.

Later you will encounter situations in which a user-selected alternate character font is used. In this case, the prompt arrow appears in inverse when the alternate character font

is in effect. Thus, in a Russian language lesson where the student must use either a Cyrillic character font or an English character font for responding to questions, the font that is currently available is indicated by the mode of the prompt arrow: inverse (alternate font) or normal (standard font).

The standard characters used by EnBASIC on the Master diskette have many special key effects. These are described in the "Access Key Functions" section of Appendix D.

Let us now return to Polaris. Recall that a correct answer to the question asked in the demo program was "NORTH STAR". Suppose that a student has an imperfect idea of the correct answer. How can we give feedback that will aid him? Try some answers that are close to (but not identical to) the correct answer. Then press RETURN. Notice the markup feedback that is displayed beneath the word when you request that it be judged. Try especially the following types of errors (use the erase and edit keys to modify your answer between tries so you don't have to retype each attempt completely):

Missing word	Star
Incorrect Capitalization	north star
Extra letter	North Starr
Wrong letter	Nortf Star
Inverted letter order	Norht Star
Missing letter	Norh Star
Missing last letter	Nort Star
Wrong word	North Light Star
Out of order	Star North

When you have completed the Polaris exercise, type a very long sentence or series of sentences. Notice that:

- --judging is automatically performed when the response goes beyond a certain length (we will see later how to specify a maximum length for a response);
- --when the response reaches the end of a line, it is automatically continued two lines below (to allow room for spelling and answer markup);
- -- edit and erase key options operate even for responses that are several lines long (test them now).

If you have an Apple //e, toggle the EnBASIC prompt arrow to lower case and verify that the keyboard shift keys can also be used with EnBASIC to produce upper-case letters.

When you have finished, remove the Demo diskette from the disk drive and go on to Chapter II. There we will prepare a test diskette so you can begin producing your own software using the augmentation commands.

CHAPTER II

Preparing Diskettes

From this point on, you will want to write and alter short test programs to try out each new feature. You will need an initialized diskette to hold your test programs. The initialized diskette should contain a copy of DOS and a "greeting" program. Later, you will be able to reuse this diskette to hold your own software. Routines on the Master diskette will help you to prepare diskettes for use of the BASIC augmentation commands.

Place the Master diskette in Drive 1 of your Apple and boot it. Follow the directions that will appear on your Apple screen until you reach the Main Index. For now, we are interested only in the last item on the Main Index: "Set up YOUR disk". Select that option now.

To use the EnBASIC features, your diskette needs a copy of the enhancement program, at least one Display table (for character codes used in text in your software) and at least one Key table (to tell the Apple how keys are interpreted when a user interacts with your material). EnBASIC provides a standard key table and display tables with English alphabets in several sizes as well as Greek and Russian alphabets. You may use one or more of these standard tables in their original form, alter them to suit your own needs by use of the Display table or Key table editors, or use those editors to produce a totally new key/display arrangement. In a later section, you will see how the editors are used. For now, we will use the tables provided.

Follow the directions until you arrive at an option table that asks whether you wish to create a combined EnBASIC package or to transfer individual EnBASIC files. Select Option 1, which automatically provides you with a combined package consisting of the EnBASIC augmentation commands, a key table, and up to two display tables.

In the future, you may wish to use the more general options contained in this setup facility (or the even more advanced options described in Appendix A), but for many uses you will find that the simplified "package" provided by option 1 is all that you will need.

You are asked to provide a name for your combined package. For purposes of illustration, let us suppose that you choose the name "ENPACK". After your package has been transferred to your own diskette, you can use all of the features of EnBASIC with BASIC by citing the package name in a BASIC BRUN statement. After choosing a name, press RETURN.

Next, you are asked whether you wish to select your own key and display tables or use the standard tables. Request the STANDARD Key and Display tables. This insures that the practice programs that you write behave according to the descriptions in this manual.

After selecting the STANDARD key and display tables, the contents of the proposed combined package are shown. It consists of the EnBASIC executer, the standard key table (named "Z.KEY TABLE"), and the standard display table (named "Z.DISPLAY TABLE"). Press RETURN to combine the files into a single package. At this point, the transfer routine asks whether you are using one or two disk drives (so it knows what disk drive to tell you to put source and target diskettes in and when to change them). Respond and then carefully follow the directions you are given.

After you have placed your diskette in the proper drive and have indicated that you are ready for the transfer to take place, the package is copied to your diskette. When the transfer is completed, the following information is shown:

(1) How to execute the package (e.g., "BRUN ENPACK" if you named your package "ENPACK"),
(2) The suggested LOMEM: setting (e.g., 33496), and
(3) The length and starting address of the binary file.

MAKE A NOTE OF THIS INFORMATION NOW!

If you have followed these directions, you now have a diskette that permits access to the augmentation commands and the standard display and key options.

The display on your screen gives directions for what to do if you wish to make more copies, do other transfers, or return to the Main Index. Since you need only one copy for now, you are ready to leave the Master diskette and go to work on your own diskette.

If you are using two disk drives, remove the Master diskette from drive 1 and move your own diskette from drive 2 to drive 1 (if you have only one disk drive, your own diskette will already be in drive 1). Now press CTRL-RESET. If your diskette was properly initialized, DOS will automatically be booted. When the DOS prompt appears, type CATALOG and press RETURN. The resulting list of files should include your greeting program plus the combined augmentation package ("ENPACK" or whatever you have selected as a name for it).

The combined EnBASIC package that has been transferred must now be loaded into your Apple so it can be used along with BASIC. This loading is done when the binary file is BRUN. After the BRUN, BASIC is augmented with EnBASIC commands and the display and key tables are loaded into appropriate memory locations. Finally, before running an enhanced BASIC program, LOMEM: should be set to a value that prevents BASIC variables from overwriting the tables, the augmentation program, and HGR2.

The commands to make the connection to the augmentation package are best placed in the "greeting" program for the diskette, so they are executed automatically whenever the diskette is booted.

Replace your greeting program with the following program. Change the LOMEM: setting from 33496 if a different value was suggested when your own EnBASIC package was produced. [NOTE: This program is also stored on your Demo diskette as program EX II-1 and may be copied from there. Copies of most sample programs are stored on the Demo diskette. You may either type the programs in yourself or copy them from that diskette.]

You may notice some strange commands in the program. These will be explained later.

```
1Ø    HOME: TEXT: REM GREETING
2Ø    HTAB 13: VTAB 5: PRINT "LOADING ENBASIC"
3Ø    PRINT CHR$(13);CHR$(4);"BRUN ENPACK"
4Ø    LOMEM: 33496   (be sure "33496" is correct for you!)
5Ø    &N: HTAB 13: VTAB 14
6Ø    PRINT "@HE@@N@@BASIC @@LOADED"
7Ø    &P: GOTO 5Ø
```

Lines 3Ø and 4Ø are the crucial ones that load the combined augmentation package and set LOMEM: to prevent it from being overwritten. Lines 5Ø through 7Ø provide a test to insure that everything is properly loaded (and would be omitted in an actual greeting program). When you have entered this program, use SAVE to place it on the diskette and then test it by typing RUN.

If all is well, you will see the message "LOADING ENBASIC" while the files are being loaded from diskette. The screen erases when the &N on line 5Ø is reached. Then the message "EnBASIC loaded" appears. Pressing any key should make this last message blink. The fact that this final message contains lower-case letters and blinks when a key is pressed is evidence that the package is present and working. In the following chapters you will learn why lines 5Ø through 7Ø

cause these effects. The PRINT statement on line 6Ø shows how EnBASIC allows you to produce both upper- and lower-case letters using the standard BASIC editor with an unmodified Apple][+. Naturally, you won't need this method if you program your software using an Apple-equivalent terminal that allows you to type both upper- and lower-case letters directly in the BASIC editor. EnBASIC automatically accepts either method of specifying case and displays the appropriate result during run-mode, even on an Apple that normally does not provide lower-case. The examples of EnBASIC in this manual will show both methods of specifying case. You should, of course, use the method dictated by the equipment that you use.

Let us examine line 6Ø now to see how case is specified when your BASIC editor does not provide lower-case. The @

```
6Ø    PRINT "@HE@@N@@BASIC @@LOADED"
```

character is used in the BASIC editor to represent a special Access key. Thus, @ acts just as ESC did during your exploration of Access key options in Chapter I. As you might guess then, @@, like ESC-ESC, toggles between upper- and lower-case letters. The special sequence "@H" at the beginning of the PRINT statement lets EnBASIC know that this particular text is to begin in upper (High) case. An @L would be used if it were to begin in Lower case. Line 6Ø could thus have also been written as,

```
6Ø    PRINT "@HE@LN@HBASIC @LLOADED"
```

without using the @@ case-toggle. The @@ toggle is ignored unless an initial case state has been set by @H or @L.

Since lines 4Ø and 6Ø assume use of the standard key and display tables, use of other display or key tables may not produce the same results.

The LOMEM: command should follow loading of all programs (such as program line editors, etc.) and precede any reference to BASIC variables. Be sure to add the LOMEM: line to the beginning of each of your BASIC programs, since LOMEM: is not retained when shifting between programs.

To provide a "friendly" user program, you will naturally also include a display in the greeting program that explains the delay while the Apple memory is being loaded plus an automatic branch to your own program. Program "SETUP" on the Demo diskette is a sample greeting program.

Appendix A contains details of setting up diskettes for more general situations, but you will probably find that the information you get within the setup program, supplemented by your experience with setting up this trial diskette, is sufficient for most applications.

CHAPTER III

Editing and Positioning Text

A Working Example

One important form of computer-based teaching involves presenting a question or problem to a student, providing a way for the student to respond, and then judging the answer. A typical sequence of operations might be (1) erase the screen, (2) print a question or problem on the screen, (3) accept the student input, (4) judge the answer, and (5) provide feedback on the correctness of the answer. These functions can be performed easily by combining BASIC with the special enhancement commands. Each EnBASIC command is preceded by an &.

[NOTE: The following example, like most in this manual, is designed only to demonstrate the features of EnBASIC. It is not intended to be an example of good CAI design.]

The example below assumes that the augmentation package and the key and display tables have been loaded. A line-by-line explanation follows the example. Note that several lines contain a mixture of upper- and lower-case letters. Read through the explanation now.

```
1ØØ    OK$="right" : NO$="wrong"
11Ø    &N
12Ø    PRINT"Who is buried in Grant's tomb?"
14Ø    &I
15Ø    &A"Grant"
16Ø    &M
17Ø    VTAB 6: HTAB 2
18Ø    IF ZW% THEN PRINT" It is Grant":GOTO 14Ø
19Ø    VTAB 8: HTAB 1: PRINT "DONE!"
```

The OK$="right" and NO$="wrong" specify the words that are shown to the student for a correct or incorrect response.

The command &N (for "New display") erases the screen, sets default display parameters, and initializes default student response judging features.

The command &I (for "Input") allows the terminal to accept a typed student response. Once the response is entered, the student begins judging by pressing the RETURN key (or any other key which the designer wishes to specify). The edit-key option is automatically available to the student while entering text at an &I command.

The command &A (for "Answer"), followed by a word or phrase in double quotes, specifies an acceptable correct response. During judging, the student response is matched with the strings in &A commands and judged right or wrong.

The command &M (for "Markup") indicates that spelling or word-order error symbols are to be shown along with the feedback contained in the BASIC string variable OK$ or NO$.

ZW% is set by EnBASIC to be "true" (greater than Ø) if the student response does not match the expected response. Thus, a wrong response results in the printing of a hint and a return to the &I for another try, while a correct response ends the sequence.

Before going into the details of how these and other commands work, let us look at how such a program would be prepared using the BASIC editor.

Producing Lower-Case Letters

Let us now look more closely at the production of lower-case letters in the BASIC editor. The BASIC program editor uses characters from ROM, so you will not normally be able to see the letters from a display table (or, on an unmodified Apple][+, lower-case letters of any sort) while editing. The unmodified Apple][also has no direct method of producing the lower-case ASCII character codes. Several options are available for those owning an Apple][+ who wish to produce EnBASIC text with both upper- and lower-case letters:

(1) Buy a terminal such as the Apple /// or //e which allows direct production of lower-case key codes and which contains lower-case letters in ROM.
(2) Add a special ROM or card to your Apple][+ which allows lower-case letters to be used during editing.
(3) Buy a software line editor that allows access to lower-case letter codes while in the program editor. For maximum convenience, this option also requires that the Apple][be modified by addition of a special display card (as in the first option above) to permit display of lower-case letters during editing.
(4) Use CHR$(n) specification of lower-case letters. (Not very convenient if many letters are involved!)
(5) Use special EnBASIC access codes which are available during editing and user operation. This approach comes at no added cost or need for added hardware. Use of EnBASIC access codes not only provides lower-case letters for the Apple][+, but also provides special effects which are not standard features of even the more advanced versions of the Apple.

Let us examine some examples of the last approach based on the Standard Key and Display tables. Remember that the effects obtained depend on the tables present during user operation. If your programs have different key or display tables, you may need different function codes.

Not all control characters that are available during user operation are available within the editor. For example, access codes which include the use of the ESC key are not properly stored by the program editor since the program editor ignores the ESC key. Other keys, such as most of the alphabetic keys preceded by CTRL (e.g., CTRL-T), are accepted when entered in the editor, but produce no visible character and are automatically deleted if the line is later edited by the BASIC program editor.

The following control keys used by the Standard Key and Display tables allow you to type lines in the BASIC program editor that can be interpreted with minimum effort even on an unmodified Apple][+. For editing ease, the Standard Key table has defined the @ key as a special ACCESS key. This key signals the first of a pair of keys which together produce a special display effect defined by the key and display table. Here are some examples from the Standard Display and Key tables:

Table III-1. Major Display Control Codes for Editing

Key Sequence	Effect
@H	Display following letters as upper case (even if they have lower-case codes)
@L	Display following letters in lower case (even if they have upper-case codes)
@@	Toggle between upper and lower case (after an initial @H or @L has set case)
@Z	Display following letters in actual case
@+	Lock in Superscript
@;	Lock in Subscript
@U	Superscript next character only
@D	Subscript next character only
@1	Set to regular sized letters
@2	Set to double sized letters
@R	Single-line carriage return

Notice the effects of @H, @L, and @@. Recall that our greeting program in Chapter II produced "EnBASIC loaded" from the command: PRINT "@HE@@N@@BASIC @@LOADED". After @H set upper-case as the starting case, each subsequent @@ switched case interpretation between upper and lower. If we had used PRINT "@LENBASIC @@LOADED", the result would have been "enbasic LOADED".

Here is another example. Place your test diskette in drive 1 and boot it. When everything has been loaded, press CTRL-RESET. This places you in the BASIC program editor with the EnBASIC commands available. Type the following lines, pressing RETURN at the end of each (this example is also on your Demo diskette as program EX III-1):

```
NEW
8Ø &N: HTAB 5: VTAB 5
9Ø PRINT "@HT@@HIS IS AN EXAMPLE OF LOWER CASE"
1ØØ PRINT "@HT@LHE FORMULA FOR WATER IS @@H@D2O  @R@R"
11Ø PRINT "@2 T@@HIS IS @@SIZE 2"
12Ø GOTO 12Ø:  REM  WAIT FOR RESET KEY
RUN
```

Carefully examine the resulting display. Using Table III-1, you should be able to see how each effect was produced. Note that @@, @H, and @L only change the way that following letters are displayed during execution. They do not act as shift keys (@H 4 is shown as "4", not "$"). Note also that line 11Ø uses @@ but has neither @H nor @L. In a series of PRINT statements, only the first needs to have an @H or @L.

An @Z is used mainly to switch off the effects of @H and @L in displays made up of a mixture of both true upper- and lower case letters and letters where case switches are done by @H or @L. If all letters in a program are true upper and lower case, EnBASIC displays them correctly without need for @Z.

Since the &N command clears the screen for you and sets the display to HGR2, code that is in memory can be tested by executing an &N and a GOTO in immediate execution mode. For example, lines 1ØØ and following can be tested quickly from the BASIC editor by typing:

```
&N : GOTO 1ØØ
```

and pressing RETURN. A quick press of RESET returns you to the editor from HGR2 after the test. By shifting between the editor and execution mode in this manner, you can quickly adjust spacing and content of a display containing special text without using special editors or modifications to your Apple and without having to execute the entire program.

Line Spacing by Character Set Design

The following code produces a two-line display because BASIC places a carriage return at the end of each PRINT statement.

```
1ØØ     HOME: TEXT: VTAB 5
11Ø     PRINT "FIRST LINE"
12Ø     PRINT "SECOND LINE"
```


The actual placement of the second line of text on the screen depends on how many dots the carriage return moves the text down. This in turn depends on the design of the carriage return "character" in the display table you are using. The usual method is to use a carriage return character that drops the next line by the same number of dots as the height of your character grid. In the case of a 7 by 8 character grid, this is 8 dots. If you specify a 12-dot carriage return for characters that are 8 dots high, the lines are separated by 4 dots. A 1Ø-dot carriage return provides lines separated by 2 dots, and so forth.

There are two ways of accessing carriage return characters. If you place your character in the standard ASCII CR slot (slot 13), BASIC accesses it for you automatically after every PRINT statement that does not end with a semicolon. If you wish to use a carriage return character that is located in another ASCII display slot, you may simply end your PRINT string with that character and end the PRINT statement with a semicolon (to prevent the regular BASIC access of the slot-13 CR). By placing several carriage return characters with different spacings in your display table this last technique can be used to allow different line spacings without having to position each line.

An example of an alternate carriage return character is the 12-dot carriage return in the Standard Display Table at ASCII slot 3 (which may be accessed either by CHR$(3) or by CTRL-C). In the PRINT statement below, the CHR$(3) and a semicolon were placed at the end of line 12Ø to produce a space-and-a-half carriage return between "THIRD LINE" and "SECOND LINE". Press CTRL-RESET and type the following lines (or run program EX III-2 on the Demo diskette) to see the effect:

```
NEW
1ØØ     &N : VTAB 5:  HTAB 1
11Ø     PRINT "FIRST LINE"
12Ø     PRINT "SECOND LINE" CHR$(3);
13Ø     PRINT "THIRD LINE"
14Ø     GOTO 14Ø: REM   WAIT FOR RESET KEY
RUN
```

&X and &Y The Display-Positioning Commands

Text may be positioned on the High Resolution screen used by EnBASIC by the BASIC HTAB and VTAB commands:

```
1ØØ     &N: HTAB 3: VTAB 1Ø
11Ø     PRINT "TEXT ON LINE 1Ø, BEGINNING AT COLUMN 3"
```

Text may also be positioned on any of the 28Ø horizontal or 192 vertical positions of HGR2. Such character positioning is measured from the upper-left corner of the screen (the zero points for both horizontal and vertical scales) and to the upper-left corner of the character being positioned.

To position the upper-left corner of the letter "M" at 25 dots down from the top of the screen and 5Ø dots over from the left margin, write:

```
1ØØ     &N : &X=5Ø : &Y=25
11Ø     PRINT"M AT X=5Ø AND Y=25"
```

The values used for the &X and &Y commands must be either constants or simple BASIC variables. Calculations are not allowed.

Here is a simple animation using &X and &Y. The word "ZOOM" moves horizontally across the screen each time any key is pressed. Press CTRL-RESET and type the following lines: (or copy program EX III-3 from your Demo diskette)

```
NEW
1ØØ     &N:  PRINT "@$" : REM  SET TO REWRITE MODE
11Ø     &Y=1ØØ
12Ø     FOR I = 1 TO 2ØØ
13Ø     &X=I: PRINT " ZOOM";
14Ø     NEXT
15Ø     &P: GOTO 1ØØ
RUN
```

Line 1ØØ erases the screen with &N and then shifts the screen to rewrite mode with "@$". In rewrite mode, a character written on the screen replaces any character currently at that screen position. Line 11Ø sets writing to a Y position 1ØØ dots down from the top of the screen and line 12Ø sets up a loop to be executed 2ØØ times. Line 13Ø sets the current value of the X position to I and then asks that the word " ZOOM" be printed. Note on line 13Ø that:

(1) the first character in the word is a blank space to erase any parts of the "Z" left from the previous time it was written;
(2) the PRINT is followed by a semicolon (to disable the normal BASIC carriage return following a PRINT).

Line 15Ø establishes a pause (&P = Pause). At line 15Ø, pressing any key causes control to shift to line 1ØØ and repeats the animation.

The equal sign in &X and &Y commands may be omitted, if desired. All of the following statements are equivalent:

```
1ØØ &X=5Ø        1ØØ &X 5Ø        1ØØ &X5Ø
```


Some Special Display Positioning Features

It is often necessary to know the position where output to the screen will next appear. Normally this information is available by a PEEK(36) for horizontal (character) position and a PEEK(37) for vertical (line) position. Since we are using high-resolution coordinates, these locations are inappropriate while using EnBASIC and should not be used. Instead use locations 224, 225, and 226 as follows:

```
X=PEEK(224)+256*PEEK(225)
Y=PEEK(226)
```

where X is the horizontal position on a Ø-279 scale and Y is the vertical position on a Ø-191 scale (with Ø,Ø being the upper left corner of the screen).

To simplify design of animations, X and Y are not updated after every HTAB and VTAB (as BASIC normally does). BASIC also resets HTAB to 1 after each line while EnBASIC does not. This last feature means that, for a sequence of PRINT statements, EnBASIC provides an automatic left margin based on the value of HTAB or &X for the first PRINT statement. This makes it convenient to place short, multi-line explanatory messages at different points on the screen during interactions. Only the position of the upper-left corner of each message needs to be specified. The code below (program EX III-4 on your Demo diskette) provides a two-line message for each corner of the screen by a mixture of techniques.

```
1ØØ   &N:VTAB 1: HTAB 1
11Ø   PRINT "UPPER LEFT"
12Ø   PRINT "CORNER"
13Ø   VTAB 1: &X=19Ø
14Ø   PRINT "UPPER RIGHT": PRINT "CORNER"
15Ø   &Y=176: &X=Ø: PRINT "LOWER LEFT @RCORNER"
16Ø   &Y=176: HTAB 29: PRINT "LOWER RIGHT": PRINT "CORNER"
17Ø   GOTO 1ØØ: REM  FLASH CORNER MESSAGES
```

CHAPTER IV

Handling User Input

We are now ready to approach the problem of accepting typed responses from users of your software. First we will cover commands designed to prepare for an interaction. We saw one such command, &N, in the example on page 11, where it was used to prepare a display for new text and new student input.

Erasing Displays and Preparing for Student Input

&N The NEW Display Command

The &N command combines the functions of two other commands: the &E command, which erases the display and sets default display options, and the &R command, which resets counters and default response-processing options. In addition, &N clears the key buffer of any keys left over from previous interactions. Thus, &N (for "New") is designed to prepare the Apple for a new display and a possible student input. Specifics of the actions of &E and &R (which are also automatically done as a part of &N) are described below.

&E The Full-Screen ERASE Command

The &E (for Erase) sets the display of the Apple to the High Resolution Graphics-page 2 (HGR2), connects the special character generator so that text and graphics may both be displayed at the same time, and erases the screen very rapidly.

Default display values set by the &E command are:

- Display page set to HGR2
- "Color" set to white
- Character spacing set to FIXED (not proportional)
- Automatic carriage return at the end of a line
- Printing done in "Exclusive-or" mode (overwriting with identical text will produce an erase effect)
- HTAB and VTAB both set to 1

Exclusive-or mode has a number of advantages for production of animations and for efficient selective erasure of parts of a screen display. We will see some of these advantages demonstrated later.

Here is a short program which illustrates the use of &E for HGR2 operation. If your test diskette is not already booted, do so now and then press CTRL-RESET and enter the commands below. Each time a key is pressed, the program erases the screen, rewrites the message, displays the key pressed, and surrounds this display with a box. Press keys as rapidly as you can and note the speed of full-screen erasing and the way the key buffer prevents loss of rapidly entered keys. (This program also appears on the Demo diskette as EX IV-1.)

```
NEW
1ØØ &E
11Ø HTAB 12: VTAB 5 : PRINT "PRESS SOME KEYS"
12Ø HTAB 19: VTAB 9 : PRINT CHR$(ZK%)
13Ø HPLOT 11Ø,5Ø TO 15Ø,5Ø TO 15Ø,85 TO 11Ø,85 TO 11Ø,5Ø
14Ø &P: GOTO 1ØØ
RUN
```

The &P (for "Pause") command, which will be explained more completely later, waits until a key is pressed, stores the numeric ASCII code for the key in ZK%, and passes control on to the next statement in the program.

Because the &E command erases the screen considerably faster than the HGR2 command, it becomes possible to do animations by drawing something on the screen, erasing the screen, and then redrawing the object in a slightly different position. The following program (EX IV-2) gives an example of such an animation. Press CTRL-RESET and try it.

```
NEW
1ØØ FOR I=1 TO 4Ø
11Ø &E: HTAB I: VTAB 1Ø: PRINT ">" :NEXT
12Ø GOTO 1ØØ
RUN
```

For each of the 4Ø horizontal screen positions on line 1Ø, the screen is erased and the ">" character is plotted with the result that arrows appear to be flying to the right. Both HTAB and VTAB are set within the loop since both are reset to 1 by &E each time that &E is executed.

Earlier we mentioned the editing trick of using:

```
&N : GOTO ___
```

in immediate execution mode as a convenient way to test how augmented code will appear to the user. The &E may also be used in this fashion since it shifts the display to HGR2 at the same time it erases the screen.

&E=[n] and &E=[n_1,n_2] Selective ERASE Commands

This form of the &E command allows selective erase of part of a display. Erasure starts at the current screen location (set either by prior display operations or by executing HTAB, VTAB or &X, &Y). Size of the area erased is defined by the character size of the current display table. Line spacing is determined by the character in ASCII slot 13 of that display table (which should be a one-line CR to avoid skipping between-lines dots during erasure). Some examples:

&E=12	Erase 12 characters starting at the current screen position. The character size of the current display table determines the size of the area erased.
&E=2Ø,5	Erase 2Ø characters on 5 successive Lines starting at the current screen position. The carriage return character in ASCII slot 13 is used for line spacing.

As with &X and &Y, the = sign may be omitted if desired. All of the following commands erase 15 characters.

```
21Ø   &E=15        21Ø   &E 15        21Ø   &E15
```

The x, y screen position values are not altered by the selective erase commands. This is done to simplify the case where something is to be rewritten immediately at the location just erased.

&R The RESET Input Initialization Command

Execution of the command &R (for "Reset input") provides a convenient default specification for operation of the input command, &I. Table IV-1 summarizes these conditions. The designer may, of course, set these values individually.

Table IV-1. Default initialization by &R

BASIC Variable	Value	Function
ZL%	25Ø	Maximum length of answer (in characters)
ZC%	Ø	Number of tries before getting a correct answer
ZS	Ø	Time in seconds before forcing judgment (see below for more)
ZA$	null	Input buffer contents
ZF$	null	Forced-key input

The &R is useful for initializing new inputs without erasing the current display--as when several inputs are on the same display. The value assigned to ZL% sets the maximum length (in characters) of a student response. Responses are judged automatically if they reach that length. Similarly, ZS sets the maximum time (in seconds) before automatic judging is begun. The student may initiate judging before that time by pressing the RETURN key. If ZS has the value Ø, timing is turned off and the student may take as much time as desired to respond. After judging, ZT contains the time (in seconds) between encountering the &I command and judging the response. Also after judging, the BASIC variable ZC% contains the count of the number of tries the student took at this input since the last &N or &R was executed.

The ZA$ and ZF$ Input Buffers

The BASIC string variable ZA$ contains the response typed by the student at the &I. ZA$ may also be set in the program. This last approach is convenient for showing a student his previous answer after returning from a remedial sequence or during a review. The student can then edit or retype the response, just as if it had been entered at that moment.

It is also sometimes useful to be able to "force" input of selected characters into the student response but not allow the student to edit them. Such forced characters are placed in the force buffer, ZF$, by the lesson designer and are automatically produced whenever the &I command is executed, just as if the student had pressed the keys. Some examples of use of the force buffer are:

- to initiate control key functions automatically so a student response is guaranteed to begin in upper case or a specific character font.
- to provide prefixes or partial answers in "completion" drills (e.g., stems of regular French verbs for a verb drill)

Keys which normally result in letters being plotted on the screen also do so if they are contained in the force buffer, ZF$. These keys are automatically typed as soon as the &I is reached, just as if the student had typed them. Keys typed by the student are added following the forced keys and the combined response is be judged as if the student had typed it all. The only distinction between force-typed and student-typed letters is that the student is unable to erase force-typed letters from the screen. One special effect of forced case keys (@H and @L), is that students can no longer toggle between upper and lower case voluntarily.

Note that when ZF$ is used to force keys into the student response, the characters in ZF$ are counted as part of the student response. This means that ZL% must take into consideration both the characters in ZF$ and the letters typed by the student. If ZA$ contains letters placed there under program control (e.g. a copy of a prior student response to allow the student to easily edit an old answer during review), those letters are also counted. Thus, ZL% contains the count of all characters in ZA$ regardless of how the characters were stored.

In a later section, we will examine use of another string variable, ZI$, that allows insertion of character strings into the student response string as it is being typed.

Summary of Erase and Initialization Options

Except where special control is needed, &N handles most preparation for display and interactions. &N combines the features of &E and &R and also clears prior responses from student input buffers. ZA$ and ZF$, the input and force buffers, allow flexible control of student responses.

For partial erasure of a display, two variants of the &E command allow erasing of specified areas of the display.

Accepting User Responses

&I The INPUT Command

The command, &I (for "Input"), allows the following:

- permits the student to enter and edit text
- permits the designer to limit the student response to a specified number of characters
- measures the time required to answer a question
- allows judging to be begun automatically after a specified time ("forced" judging)
- provides a count of the number of responses a student enters while interacting at an &I
- allows a specific character font to be selected and allows the case to be set to upper, lower, or a mixture of upper and lower case. The default entry mode is lower case.

These features are initialized by execution of the &N or &R commands, or by setting each individual BASIC parameter. Table IV-1 lists the parameters used by &R and their default settings.

To experiment with these features, press CTRL-RESET and type in the following program (EX IV-3 on your Demo diskette).

```
NEW
12Ø  &N
13Ø  VTAB 5: PRINT"INPUT TEST ";
14Ø  ZL% = 1Ø:  ZS = 5ØØ
15Ø  &I
16Ø  VTAB 8: HTAB 12: PRINT "@$" ZC% " TRIES"
17Ø  GOTO 15Ø
RUN
```

Line 14Ø alters the default Input parameters so that judging begins automatically after 1Ø keys are entered (ZL%=1Ø) or 5ØØ seconds elapse (ZS=5ØØ). Line 16Ø shows a way to give students information on the number of attempts made to date at an input. After each judgment, the code on line 16Ø displays the count of tries (ZC%) in rewrite mode (produced by executing the key sequence "@$").

The &I on line 15Ø places the input prompt on the display and waits for keys. After typing something on the screen, editing can be done using the ← key to erase one character at a time and the → key to erase the entire input. Later presses of the → key replot the input, one word at a time, allowing error correction without requiring that the entire response be retyped. Reaching the 1Ø-character limit, exceeding the 5ØØ second limit, or pressing RETURN stores the input in the BASIC string variable ZA$ and moves control to line 16Ø.

Notice that lower case letters appear when alphabetic keys are pressed (the default entry mode). Also note that attempting to enter more than 1Ø characters or pressing the RETURN key repeatedly causes control to pass to the lines after 15Ø. On lines 16Ø and 17Ø, ZC% is updated and control is returned to the &I. Repeated updating of ZC% in this fashion is, of course, not always desirable. Also there may be other types of responses that you do not wish to count. All of this is under your control. If, for example, you want to ignore blank inputs then include this line of code in the above example:

```
155 &A"": IF ZA% THEN  ZC%=ZC%-1: ZA%=Ø :GOTO 15Ø
```

The &A"" recognizes blank responses. If a blank input is made, ZA% is set to "1". The operation ZC%=ZC%-1 then decrements the count of answers given (ZC% was incremented when the blank response was judged). Next, ZA% is set to zero, eliminating all evidence that a blank response was given, and the student is returned to the &I by "GOTO 15Ø".

All of the parameters set by &N (or &R) can also be set directly by the programmer. One may alter these parameters immediately after executing &N or &R; or, as was done in line 155 above, the parameters may be altered during the course of the student interaction. The &N or &R commands in fact need not be used at all if the BASIC variable ZC% is set to zero and other variables are set individually. Table IV-2 shows variables used by the &I command. Whether these values are set by &N, &R, or manually, the setting should normally be done before the &I is encountered.

Table IV-2. Parameters used by &I

VARIABLE	Use	Initial value
ZC%	response count	Ø
ZA%	"correct" flag	Ø = FALSE
ZW%	"wrong" flag	1 = TRUE
ZL%	answer length	set as desired (25Ø maximum)
ZS	maximum time	set as desired (Ø if unlimited)
ZA$	input string	usually null
ZM$	markup string	null
ZF$	force-key string	null or set
ZI$	insert-key string	null or set

In addition to the variables listed in Table IV-2, the variable ZK% is set to the ASCII value of the last key entered at the &I.

Now let us examine the command that permits us to specify an expected student input.

&A The ANSWER Command

The &A (Answer) command provides the way to specify what the author of the lesson will accept as a possible answer. The &A command automatically takes care of extra spaces in the answer and allows you to indicate the words which must be present; extra words which may be present, but which are not required; and words which cannot be present. In addition, you can specify that word order is not important and that punctuation is either required or ignored. If you provide a way for students to type both upper and lower case, the &A command automatically checks to see that words have the correct capitalization. Superscripts, subscripts, and accents are also properly handled by &A.

The &I command places the student's response in the BASIC string variable "ZA$". The &A command then compares the contents of ZA$ with the string specified by &A.

Suppose we show a picture of a sailboat on the screen and ask the question "What is this?". We need an &N to prepare for new input, an &I to accept and store the input in ZA$, and a way to specify that "sailboat" is the expected input.

```
1ØØ  &N              (clear screen and variables for &I)
1ØØ  &I              (accept user input)
12Ø  &A"sailboat"    (specify expected input at prior &I)
```

The "&A" tells the Apple that what follows in the quotation marks is to be treated as an acceptable student response.

This &A command sets the BASIC integer variable "ZA%" to the value 1 if "ZA$" contains only the word "sailboat". If that word is absent, "ZA%" is set to Ø. The variable "ZW%" shows how "wrong" an answer is. For a completely wrong answer "ZW%" is 1. For a partial match, "ZW%" is set to a positive value, N. The larger N is, the closer the answer is to matching the listed answer. Values of ZA% and ZW% are the standard BASIC true/false values when answers are either completely right or wrong.

Table IV-3. Values of ZA% and ZW%

Answer	ZA%	ZW%
Right	1	Ø
Wrong	Ø	1
Partial Match	Ø	>1

Values of ZW% and ZA% can be used for presenting feedback. Press CTRL-RESET and type the following (EX IV-4):

```
NEW
8Ø   &N
9Ø   VTAB 2: HTAB 2: PRINT "SAILBOAT TEST"
1ØØ  VTAB 6: HTAB 15
11Ø  &I
12Ø  &A "@LSAILBOAT"
13Ø  IF ZA% THEN PRINT " OK"
14Ø  IF ZW%=1 THEN PRINT " NO"
15Ø  IF ZW%>1 THEN PRINT " ALMOST"
16Ø  &P: GOTO 8Ø
RUN
```

Type various forms of "sailboat" to see how they are judged. Pressing any key after a judgment gives you another try.

Note the use of @L to set the answer to lower case. Once either @L or @H has been used in an &A, you may also use @@ to toggle case (as with PRINT statements, @@ is ignored unless @H or @L has been used to establish an unambiguous beginning case).

Let us now examine a better way to provide information about errors in student responses.

&M The MARKUP Command

The &M (Markup) command provides student feedback on typing and spelling errors. The &M is placed immediately after the &A command.

If the answer to a question is, for example, "sailboat" and the student types "saleboot", the automatic answer markup feature of &M can tell the student that there is a letter missing between the "a" and the "l", that no "e" should appear between "l" and "b", and that the second "o" is not correct.

```
Right Answer:   sailboat
Student Answer: saleboot
Markup:           tx  =
```

The program below provides automatic markup of the student's answer along with "RIGHT!" or "WRONG!". Press CTRL-RESET and type it in (or use program EX IV-5 from your Demo disk).

```
NEW
11Ø OK$ = "RIGHT!": NO$ = "WRONG!"
12Ø &N
13Ø PRINT "WHAT IS THIS?"
14Ø ZF$="@L"
15Ø &I
16Ø &A"@LSAILBOAT"
17Ø &M
18Ø GOTO 14Ø
RUN
```

Line 14Ø sets ZF$ to @L, forcing students to use only lower case.

The &M command triggers display of the content of the BASIC variable "OK$" or "NO$" depending on whether the student's answer matches the &A string. The string in OK$ or NO$ is displayed following the student response. The &M also provides markups showing how the student response fails to match the desired answer. This markup is stored in the variable "ZM$" and is displayed under the student response.

Table IV-4 lists all response markups along with their ASCII character codes. These symbols are part of the "Standard" character set. Note that these markups work only on HGR2.

Table IV-4. Markup Symbols and ASCII Character Codes

Error	Markup	Decimal	Hex
Extra Character	x	12Ø	$78
Wrong Character	=	61	$3D
Letters inverted	∽	5,4	$5,$4
Missing Character BEFORE here	⌊	19	$13
Missing Character AFTER here	⌋	2Ø	$14
Should be Lower Case	↓	24	$18
Should be Upper Case	↑	23	$17
Should be Subscript	∨	17	$11
Should be Superscript	^	18	$12
Bad Accent	≃	15	$F
Unanticipated Word	X	88	$58
Word Missing Here	▲	16	$1Ø
Word should be moved Left	◀	22	$16

You can, of course, provide other types of responses to the student based on the presence of these ASCII character codes in "ZM$", or you can alter the symbols by use of the Display Table editor on the set of characters used for your own material.

Following entry of a wrong or partially wrong response, the student can use the edit keys to alter the answer. In some cases it is useful to permit quick re-entry of different responses without use of the edit keys. Being able to press the key used to judge the response (e.g. RETURN) a second time and have the entire previous response erased is sometimes convenient. The &M command provides a flag which is used by the &I command to provide such a feature if the program branches back to the &I after an &M is encountered. After branching back to the &I (assuming none of the flags are otherwise altered by manual resetting or execution of &R), the old response is still displayed and can be edited. However, because the &I "knows" that this response has already been judged once, it treats the judge key as a request to erase the old response and prepare for a new one. Once the judge key is used to erase an entire response after such a branch-back to an &I, the &I reverts to its normal behavior. Note that our previous example makes use of this branch-back trick. If you did not notice the effect of pressing RETURN again after a "NO" judgment (and of the availability of the word in the edit buffer), test the example for this now.

When it is desirable to display only the limited feedback contained in OK$ or NO$, the &M must be used (to allow access to those messages), but the effect of the character markup can be eliminated simply by clearing the contents of the markup buffer, ZM$, as shown in the example below:


```
1Ø   OK$="ok": NO$="no"
2Ø   &N
:
9Ø   &I
1ØØ  &A"cat and dog"
11Ø  ZM$=""
12Ø  &M
```

With ZM$ set to null, judgments of "ok" or "no" are given and response editing is possible, but spelling or word-order markups for almost-correct responses are <u>not</u> given.

You may also access information in ZA$, the student-response string. Use caution if you alter information in ZA$ prior to execution of the &M markup string. The &M command uses ZA$ in producing ZM$, so if ZA$ is altered by your BASIC program after the student has initiated judging but before the ZM$ markup is generated, the markup shown the student may no longer be appropriate for the response. If possible, make alterations only to a copy of ZA$ rather to ZA$ itself. Note that, since ZA$ can be set within your program, &A can be used even without input by &I in special applications.

As with &I, &A uses BASIC variables to pass information back to the program. Table IV-5 lists these variables and their normal initial values before use by &A. Initialization and updating of all of these values are normally done for you automatically, but all may also be changed directly to meet special programming needs. The &O mentioned in Table IV-5 is an added Answer-type command and will be explained later.

Table IV-5. Parameters used by &A

VARIABLE	Use	Initial value
ZM$	markup string	null
ZA%	"correct" flag	Ø
ZW%	"wrong" flag	Ø
ZN%	index of &A or &O closest matched	Ø

Response Options

The response-handling commands, such as &A, can be modified by a number of useful options that permit application of the most commonly-needed alternative approaches to handling student responses. These options allow the instructional designer to indicate which synonomous alternative answers are acceptable, whether unspecified words are to be ignored or included during judging, whether certain specified words are to be rejected in responses, and how punctuation and capitalization are to be handled.

S for SYNONYMS

The &A command

```
15Ø  &A"sailboat"
```

accepts only the single student response "sailboat" as being correct. However, it is often desirable to accept synonyms. The "S" (for "synonyms") option allows sailboat and ship to be treated as synonyms.

```
15Ø  &A"<S,ship,sailboat>"
```

In the above command, the "S" tells the computer that the words between "<S" and ">" are synonyms. Actually, either "S" or "s" specifies the option. Also, either commas or spaces may be used as separators. Thus, all of the following commands are equivalent:

```
&A"<S,sail,sailing>"      &A"<s sail sailing>"
&A"<S sailing,sail>"      &A"<s,sail sailing>"
```

Synonyms are not limited to single-word answers. The &A command:

```
15Ø &A"<S,old,antique><S,car,auto,vehicle>"
```

judges any combination of old or antique with car, auto, or vehicle as correct. All of the following phrases would be judged acceptable:

old car	old auto	old vehicle
antique car	antique auto	antique vehicle

I for IGNORE

To make programs easier to use, it is often desirable to allow, but not require, some other words in a response. For example, if the answer to a question is "sailboat" you might want to accept "it is a sailboat".

The "I" (for "ignore") option allows you to specify extra words which may be present but are not required. The following answer command specifies that "it", "is", "a", and "an" are extra words which may be present, but which are not required:

```
15Ø   &A"<I,it,is,a,an><S,old,antique><S,vehicle,car,auto>
```

If the student uses a word not included in the I list, that word is rejected during judging. If we use the &M command

to provide automatic markup of the answer, the response "it is a nice old car" has the word "nice" underlined with XXXX after judging to indicate that "nice" is not an accepted part of the expected answer.

The obvious difficulty of anticipating every possible added word may be met by simply ignoring all words other than those crucial to interpreting the student response. The "X" option permits this approach.

X for EXCLUDE

Instead of listing the words which may be present you can list only the words that must be excluded and allow all others. This answer command:

```
15Ø  &A"<X,not,isn't><S,old,antique><S,car,auto>"
```

indicates that a correct answer cannot include the words "not" or "isn't"; must include either "old" or "antique" and either "car" or "auto"; and may include any other words. This option provides a convenient way to limit responses to positive statements (which tend to be less ambiguous and easier to judge correctly). For the above example,

```
"it is an exemplary antique auto"
```

would be judged correct, but

```
"that isn't such an old car"
      XXXXX
```

or

```
"that isn't bad for an old car"
      XXXXX
```

would have the word "isn't" marked as not belonging. Since capitalization is considered, "NOT" and "not" are treated as different words--allowing "it is NOT an old car" to be treated as a correct answer. But fear not, another option will soon be described which takes care of this problem.

If you want to accept all extra words, simply use <X> alone. This "excludes no words" (i.e., ignores any words other than those listed in other parts of the &A command). In the following &A command, any added words are accepted along with "traffic signal", "traffic light", etc.

```
&A"<X><S,traffic,stop><S,light,signal>"
```

Thus, "it is another darned stop light!" would be judged correct.

P for PUNCTUATION

Punctuation is normally ignored. However, if commas, periods, colons, semicolons, and question marks are an important part of the answer, then the P (for "punctuation") option can be used. This causes punctuation to be treated as if it were a letter. The P option should normally be considered only for short, one-word answers. Here is an example from a chemistry lesson:

```
15Ø &A "<P><S theobromine 3,7-dimethylxanthine>"
```

In the above example, theobromine is a synonym for 3,7-dimethylxanthine. Since a comma is a part of one of the synonyms, it must be present for the response to be considered correct. <P> (and similar options) must precede all response words in the answer command.

Note that we have made use of the fact that spaces can serve as separators between the "S" and between the synonyms since use of the P option prevents us from using commas both as separator and as part of a synonym.

W for Unordered WORDS

Normally, it is assumed that the author intends multiple-word responses to be given by the student in the order specified in the &A command. However, if order does not matter, the W (for "words in any order") option can be used.

For example, for the task "Name the three primary colors", you may not care whether the student says "BLUE, GREEN, RED" or "GREEN, RED, BLUE" or any of the other four possible orderings. The following &A command accepts "RED", "GREEN", and "BLUE" in any order.

```
15Ø  &A"<W> RED GREEN BLUE"
```

Note: <W> must precede all response words in the answer command.

L to judge on LETTERS only

As we have seen, if an answer has letters capitalized which are not capitalized in the answer command, the markup puts a downward arrow (↓) under them. Similarly, if there are lower-case letters in the answer where the author has specified upper case, the markup indicates this with an up arrow (↑).

If you do not care what case is used, then include <L> in your &A command to indicate that only the letter itself is to be considered (and not whether or not it is capitalized).

```
15Ø  &A"<L><X,not,isn't><S,old,antique><S,car,auto>"
```

The <L> in line 15Ø eliminates the problem noted earlier in which "it is NOT an old car" was accepted as correct because "NOT" and "not" were considered as different words.

Note: <L> must precede all response words in an answer command. If you use several such options, they may appear in any order before the first response words, e.g.,

```
&A "<L><W> GREEN BLUE RED"
```

C for CAPITALIZATION

If the expected response contains letters which must be capitalized in addition to letters which could be either upper or lower case, then add <C> to your answer command. The <C> does not allow the student to have a small letter where the author has a capital letter. If the answer to a question is Lincoln then you could use:

```
&A"<C>Lincoln"  or  &A"<C>@HL@LINCOLN"
```

then, if the student types "lincoln", the markup command, &M, plots ↑ under the lower-case "l" to indicate that it should be "L". Any response that is spelled correctly and which capitalizes the first L is accepted. Thus, LiNCOLn, L̃iNcOlN, LINCOLN, etc., are all acceptable.

Note: The <C> must precede all response words in an answer command.

V for Embedded String VARIABLES

BASIC string variables may be incorporated into an answer command by use of the directive "V" (for "Variable"). In

```
1ØØ   B$="<I,it,is,a>"
11Ø   &A"<V,B$><S,traffic,stop><S,signal,light>"
```

line 11Ø is equivalent to:

```
&A"<I,it,is,a><S,traffic,stop><S,signal,light>"
```

because <V,B$> inserts the contents of the BASIC string "B$" into the answer command. This is very useful where you

have groups of "extra" or special vocabulary words that are frequently used in answer commands.

The V feature may be used to construct words by the addition of strings. Consider this example:

```
1Ø    C$="IN"
2Ø    D$="FLAMMABLE"
15Ø   &A"<S,<V,C$><V,D$>,<V,D$>>"
```

The string variable C$ contains "IN" and D$ contains "FLAMMABLE" so the answer command shown above is the same as:

```
155   &A"<S,INFLAMMABLE,FLAMMABLE>"
```

Inserted variables can be treated as single words or as separate words, depending on placement of commas. Note in line 15Ø that the first two inserted variables are treated as a single word, while the third inserted variable is considered a separate word.

The <V operator is ideal for string substitution prior to execution of the &A command. Recall that <W> allows words to be accepted in any order. If we set W$ as follows:

W$ = "<W>" or W$ = "" (i.e., null)

then, &A"<V,W$> RED GREEN BLUE"

ignores word order if W$="<W>" and requires it if W$="".

The <V operator is also extremely useful in constructing drills. A single &A can be used with <V for the entire drill. A BASIC routine is used to load responses from a list into a simple BASIC string variable that is referenced by the &A command. See the first "State Drill" program on the Demo diskette for an example of this approach. CAUTION: In the current version of EnBASIC, variables referenced in answer commands cannot be arrays, nor can the <V operator be embedded within another <V string. Only single-level substitution is permitted.

M for MISSPELLED words accepted

If you wish to accept probable misspellings as equivalent to the correct spelling of a word in a response, include the <M> option in your ANSWER command.

```
15Ø &A"<M>Mississippi"
```

<M> must precede all response words in an answer command. When <M> is in effect, &M does not give spelling markups.

The criterion used by <M> to distinguish between possible misspellings and wrong answers differs from that used for providing &M spelling markups. Differing criteria arise from the fact that the &M spelling markup gives specific information about mismatched letters, while <M> does not. As a rough guide, &M may accept a word with about half of its letters in error as a potential misspelling, while <M> may accept a word with no more than about one third of its letters in error as a potential misspelling. The actual algorithm used to detect misspellings is rather complex, so these figures only suggest the relative differences in judging stringency imposed by these different situations.

Multiple Alternative Responses

&O The Multiple Answer Command

More than one answer may be correct in some situations. If we display a picture of a sailboat and ask "What is this?", some form of "sailboat" is a correct answer, but not the only one. Other correct answers might be "ocean scene" or "computer graphics". Thus, we need a provision for multiple answers. The computer must be able to indicate which of several alternative answers most closely matches a student response and provide an appropriate markup for near-misses. To do this we use the "&O" command ("O" stands for "or") after the first "&A" command. For our sailboat example, we might use the following commands:

```
15Ø &A"<S,sailboat, yacht>"
152 &O"<S,ocean,sea><S,scene,scape>"
154 &O"computer<S,graphic,graphics,drawing>"
```

The first response command in the group is &A, followed by &O commands. This order lets the computer know that it must check if the student matched the first OR the second OR the third response command. Other BASIC statements may be placed between response commands. All of the &A options are also available for the &O command.

The BASIC Integer Variable "ZN%" identifies which &A or &O in the series most closely matches a student response. If a perfect match occurs, ZN% is set to that command number and following &O commands are skipped. If a near-match occurs, ZN% is also set but &M automatically prepares a markup. If no match or near-match occurs, ZN% is set to "Ø".

In the above example, if the student types "sea scape" (the second alternative answer), then ZN% is set to "2". Any near-match like "sea salt" would also result in ZN%=2.

All parameters used by &A (Table IV-5) are also used by &O.

Use of Multiple &A commands

If several &A commands occur after an &I, each is treated as if the others did not exist. This behavior provides a convenient tool for identifying specific words in advanced applications. Suppose we wish to flag the common error of using the word "iodine" where "iodide" is appropriate. Here is one method:

```
100  IO%=0: REM  Flag for using word iodine
110  &I
120  &A "<X><L> iodine": IF ZA%=1 THEN IO%=1"
130  &A "<L> sodium iodide"
140  &M
150  IF IO%=1 THEN PRINT " Remember -- IODIDE!"
160  IF ZA%=0 THEN GOTO 100
170  &P: GOTO 400
```

If the word "iodine" appears anywhere in the response, the &A on line 120 is matched and IO% is set to 1. The program then moves on to the next &A and judges the response again. If all is correct, ZA% is 1 and IO% is 0; the student is routed to the next item. If ZA% for the second &A is 0, the student is routed back to the &I by line 160. If he used the word "iodine", the message on line 150 is shown along with the normal NO$ message controlled by &M.

Technical Note:
In natural-language interactions, students may produce large numbers of alternative values for the string variables ZA$ and ZM$. BASIC stores each new version of a string variable in its memory. Normally it is necessary in BASIC programs to clean up these extra values periodically. Use of methods such as X=FRE(0) are <u>not</u> needed in EnBASIC, however, since memory cleanup is done automatically each time &I is executed. In addition, each time a response is judged, an automatic cleanup of variables such as ZM$ is done.

ERROR Detection in &A and &O

If an error occurs in an &A or &O command, the program goes on to the next line. However, the BASIC integer variable "ZE%" is set to a number from 1 to 9, allowing the error to be identified. Table IV-6 summarizes the error codes.

Prudent design practice suggests checking the value of ZE% during prototype testing of instructional programs to insure both that command formats are correct and that the design

Table IV-6. Error Codes for &A and &O Commands

Value of ZE%	Error
0	No errors
1	Student input string too long (>250 characters)
2	Author input string too long (>250 characters)
3	Quote mark missing at start of command
4	No comma or space between V and BASIC variable: e.g. <VC$> should be <V,C$>
5	No > at end of command: e.g. <V,C$
6	Bad BASIC string variable name
7	Too many words in student response (>32)
8	Too many total student and author words (student words x author words > 224)
9	<C>,<M>,<P>,<U> not before author's words

prevents the student from accidentally producing an error situation. If your program itself generates contents of &A or &O commands (e.g., you are designing authoring tools or a program that allows students to generate their own drills), you should also provide checks of ZE% with appropriate user feedback. An example of such a use of error checks (from the user's viewpoint) can be found in the illustrations of answer judging on the Master diskette. The section that allows you to enter any target answer uses this error checking to provide you with information on improper format of trial answers.

Advanced Application Example:
Providing Help and Retrying an Answer

If a student makes an error in answering a question, it is often desirable to provide some additional help, pause so the help message can be read, and then erase the help and allow the student to correct the wrong response. Experience has shown that after reading the feedback comments, students frequently just start typing their correction without erasing the previous response. Therefore, it is desirable to be able to return immediately to the input with whatever keys the student types after reading the help message.

The type of help which is provided may depend on how many times the student has attempted to answer the question. The BASIC variable ZC% automatically contains a count of the number of times the question has been answered and may be used to control the message.

Here is a working example. Press CTRL-RESET and type it in (or copy program EX IV-6 from your Demo diskette).

```
Program                          Comments
NEW
11Ø  &N                          Prepare for New interaction
12Ø  PRINT"WHAT GOES 'MEOW'?"
13Ø  &I                          Input  response
14Ø  &A"<S,@LCAT,FELINE>"        Answer command with synonyms
15Ø  &M                          Mark up the answer
16Ø  IF ZA% THEN 3ØØ             If OK, go to 3ØØ
17Ø  ZK%=Ø                       Zero ZK% as a flag
18Ø  HTAB 1Ø: VTAB 1Ø            Location of help messages
19Ø  IF ZC%=1 THEN PRINT"@LBEGINS WITH 'C'"
2ØØ  IF ZC%>1 THEN PRINT"@LTRY 'CAT'"
21Ø  IF ZK% THEN 13Ø             Return to input on second pass
22Ø  &T: IF NOT ZK% THEN 22Ø     Loop until keypress
23Ø  GOTO 18Ø                    Keypress--erase help message
3ØØ  HTAB 1Ø: VTAB 1Ø: PRINT "GOOD!"
RUN
```

Lines 11Ø to 15Ø set up the question, allow input, specify the answer, and provide the markup. Line 16Ø tests to see if the answer is right (ZA%= 1) and, if it is, gives the message "GOOD!" and moves on beyond the question.

If the answer is not right, ZK% is zeroed as a flag on line 17Ø. Line 18Ø positions help messages on the screen. On the first attempt to answer the question, the BASIC variable ZC% is set to 1 by the answer command; thus, the hint on line 19Ø is printed on the screen. More than one try gives the hint on line 2ØØ. Next, line 21Ø checks the value of ZK%. Since ZK% was set to zero on line 17Ø, the test fails and the program goes on to 22Ø.

The command, &T, on line 22Ø Tests the keyboard to see if the student has typed a key. If no key has been pressed, ZK% is not altered from the value of Ø set on line 17Ø. As long as no key is pressed, looping continues at line 22Ø. Once any key is pressed, ZK% is set to a non-zero value, and control falls through to line 23Ø and thence to line 18Ø and once again through the help message. Since EnBASIC, by default, writes in exclusive-or mode, rewriting the help message over itself results in automatic erasing of the old message (a very handy feature of exclusive-or mode!). We now have a non-zero value in ZK%, so this time when we come to line 21Ø, we branch back to the &I on line 13Ø where the key pressed at &T is acted on. Thus, a student can type "felin", press RETURN, receive feedback about the missing "e", type that letter, and immediately see the corrected word, ready for rejudging. Such approaches do much to make interaction with the computer a friendly experience.

Key-Buffering and Single-Key Sensing

Student responses made up of natural-language strings in a well-defined context are one important aspect of advanced CAI. The &I, &A, and &O commands provide tools for handling such reponses in BASIC.

Another vital aspect of advanced CAI consists of real-time user interaction with a computer that is busy updating simulations or graphics. Designing such CAI environments requires methods of controlling access to the keyset and sensing individual keys both reliably (keys must not be lost or misinterpreted) and efficiently (key processing should not degrade simulations). The commands described in this section provide a variety of methods for meeting these needs. Among these commands are the &P and &T commands which have already been used in some of our examples.

Key Buffering

EnBASIC makes use of a multi-key input buffer to prevent loss of keys entered during simulations or graphics processing. The various pause, delay, and input commands treat keys in this buffer differently. All but &T remove the oldest key from the buffer. &T tests for keys but does not alter the buffer -- any keys in the buffer remain there. All EnBASIC commands place keys from the keyset into this multi-key buffer. However, keys entered during long, complex non-EnBASIC computations do risk being lost. To prevent such loss, &T commands can be placed in these parts of the program. Keyset entries are then stored in the key buffer for later detection by the program. Execution of the &N command clears the multi-key buffer to prevent unintended interaction between sections of the lesson.

&P The PAUSE Command

The pause command, &P, causes a program to pause until a key is pressed. The ASCII code of the key used to break the pause is placed in the BASIC integer variable ZK% and removed from the input buffer. At the same time, the number of seconds spent between encountering the &P and pressing a key is placed in the BASIC real variable ZT.

The &P provides an easy way to check for specific single keys. For example, to allow only RETURN to function, use:

```
1ØØ  &P: IF ZK% <> 13 THEN 1ØØ
```

Only if RETURN is pressed does ZK% contain the ASCII code 13. Any other key results in the program looping back to the beginning of line 1ØØ. Since ZT is zeroed each time &P

(or &I, &D, &K, &T) is encountered, only the delay for the last cycle through line 1ØØ is stored in ZT.

The &P pause can also be broken by the "timeup" key if ZS has been set previously. Thus,

```
ZS=6.8: &P
```

pauses for 6.8 seconds OR until a key is pressed.

&K The KEY grabber

When &K is encountered, the oldest key waiting in the key buffer is removed and its keycode is stored in ZK%. If no key is present in the key buffer, ZK% is set to Ø.

Here is an example which causes a message to move across the screen until the RETURN key is pressed. Type it in and try it (or select program EX IV-7 from your Demo diskette).

```
NEW
1ØØ     &N:  X% = 1: &Y=6Ø
11Ø      PRINT CHR$(64) CHR$(36) CHR$(64) CHR$(17)
12Ø      X% = X% + 2: &X = X%
13Ø      PRINT " PRESS RETURN TO PAUSE";
14Ø      IF X% > 28Ø THEN X% = 1
15Ø      &K:  IF ZK% <> 13 THEN 12Ø
16Ø      VTAB 15: HTAB 1Ø: PRINT "PRESS ANY KEY TO CONTINUE"
17Ø      &P: GOTO 1ØØ
RUN
```

Line 11Ø puts the character generator in rewrite mode, by producing the ASCII codes for @ and $, and then specifies that text is to wrap around at the right margin (rather than produce an automatic carriage return) by producing the ASCII codes for @ and CTRL-Q. These codes assume use of the Standard Key and Display Tables (summarized in Appendix E).

Lines 12Ø to 14Ø produce a moving message on a line 6Ø dots down from the top of the screen. Note the space before the "P" in "PRESS" in the PRINT statement on line 13Ø. This space overwrites (and thus erases) the "P" left over from each previous positioning of the message.

&K on line 15Ø sets ZK% to the value of the oldest key in the key buffer and removes that key from the buffer. If no key is present, ZK% is set to Ø. ZK% is then tested to see if ASCII code 13 (the RETURN key) was the last key removed. If the RETURN key was not pressed, control passes to line 12Ø and the message continues. If the key removed was the RETURN key, control passes to line 16Ø. Line 16Ø produces a

new message. The &P on line 17Ø causes a pause until a real keypress occurs. At a keypress, control returns to line 1ØØ and the moving message resumes.

Since &K removes keys from the key buffer, it can be used as necessary to clear out the key buffer without altering other parameters (as would happen if &N were used). One key is removed from the multi-key buffer each time &K is encountered. To insure that all keys are removed, simply execute a loop like this:

```
1ØØ &K:IF ZK% <> Ø THEN 1ØØ
```

This line loops until all keys are cleared from the key buffer even if keys are being entered while the looping is in progress.

&T The TEST-for-keypress command

In many types of student interactions it is desirable to have a program pause until a key is pressed, and then go on to an input such as &I. Often the key that is pressed needs to be passed to the input command.

The command &T is provided to simplify construction of a pause which does <u>not</u> <u>remove</u> the keypress from the input buffer. In the example below, the multi-key buffer is checked to see if a key has been pressed. If a key has been pressed, its ASCII code is placed in ZK%. The key is <u>not</u> removed from the key buffer. This same type of test is used in the sample program on page 38 to test for a key after a help message has been shown to a student.

```
1ØØ  ZK%=Ø
11Ø  &T: IF NOT ZK% THEN 11Ø
```

This fast &T test may be used in animations to check for keypresses. If several keys are in the key buffer, ZK% contains only the value of the <u>first</u> key waiting to be processed. Thus, &T should be used only where every key is to be reacted to as it arrives and where unwanted keys are cleared from the multi-key input buffer by some command that acts on that buffer (e.g., &N, &I, &P, or &K).

&D The Timed DELAY Command

Where you wish to produce a timed <u>delay</u> on a display, &D allows you to specify a delay in seconds which no key (except the "timeup" key pressed by the computer) can break

through. Specify the delay by setting BASIC variable "ZS" to the number of seconds desired. The normal value of ZS is Ø, so unless a positive, non-zero value is placed in ZS prior to execution of &D, no delay is seen. Keys pressed while &D is in its delay state do not appear in the key buffer. At completion of timing, ZK% has the value 128, the key code for the EnBASIC "timeup key".

ZS may be set to an initial value of up to about 72Ø,ØØØ seconds. When timing begins, the value of ZS is reset to zero and the actual time spent is shown in ZT (after timing is complete). In the case of &D, ZT at completion is equal to whatever ZS was set to. Remember, you specify ZS, and the computer sets ZT.

ZI$ For Insertion of keys at &I

By using the simple BASIC string variable ZI$, we can make a single keypress produce one or more special keys during input at an &I. Unlike ZF$ or ZA$, which allow forcing of character strings only at the beginning of user input, ZI$ can insert keys at any point in the input process.

Consider this situation: We are writing an instructional program in which we know that numbers appear only as subscripts (e.g., like H_2O) and in which we do not want to require that the student learn to use a special subscripting control key. We want to tell the computer to convert automatically every number key to a two-character string in which the first character is the non-locking subscript control and the second character is the numeric character that is typed. ZI$ permits us to do this by inserting the contents of ZI$ automatically into the input string. First let us examine how &I treats ZI$.

Each time judging occurs at an &I, ZI$ is set to null. Now, suppose we route control back to the &I by means of a GOTO. Recognizing that this is not a new &I setup, the &I checks to see if ZI$ is still null. If ZI$ now contains real characters, the &I places these characters in the key buffer, just as if they had been typed by the student, and then sets ZI$ to null once more.

Thus, in our present example, each time a number key is pressed, we want to halt the input process at the &I, fill ZI$ with the non-locking subscript control character followed by the numeric character belonging to the number key that was pressed, and reroute control back to the &I. How do we do this?

Recall that the function of a "judge" key (like RETURN) is to halt input at an &I and pass control to BASIC commands following the &I. Suppose keys 1, 2, ..., Ø were defined as "judge" keys. Every time one of them was pressed, the input at the &I would halt and control would be passed on to the BASIC statements following the &I.

In statements following the &I, the last key pressed can be sensed by examining the value of ZK%. If ZK% is RETURN, the student has finished responding and the entire answer is ready for judging-- the normal reason for dropping to the lines of BASIC code following the &I. However if ZK% is one of the number keys, the student is unknowingly requesting some special handling of the last key. To provide this special handling, we simply place the subscript control character and the appropriate numeric character in ZI$ and execute a GOTO to get back to the &I. The &I then does its job of placing the contents of ZI$ into the input string, plotting them on the display, clearing ZI$, and then waiting for additional keypresses.

If all numeric keys are defined as judge keys in the key table used by the program, the following code produces the effect just described:

```
1ØØ &I
11Ø IF(ZK%>=48)AND(ZK%<=57) THEN ZI$="@D"+CHR$(ZK%):GOTO 1ØØ
12Ø &A "H2O"
```

Keycodes 48 through 57 are the numeric keys 1 through Ø. If one of these keys causes termination of &I input, the "THEN" function of line 11Ø is executed. This concatenates the character for non-locking subscripts, @D, with the character for the key last pressed, CHR$(ZK%). Control is then returned to the &I on line 1ØØ to continue input. If the RETURN key is pressed, control drops to line 11Ø, fails the test for ZK% being in the range 48 through 57, drops to line 12Ø, and then results in normal judging of the total student input against the criterion response of "H_2O".

Note that the above example works only if the numeric keys have been redefined to be "judge" keys. Key effects are redefined by use of the key table editor. In a later chapter we will explore use of the key table editor and use it to alter effects of keys. For now, however, let us try an example without defining new judge keys.

To demonstrate ZI$ without redefining the Standard key table, try this short program (or use EX IV-8 from the Demo diskette). It uses CTRL-A and CTRL-B, which are already defined as judge keys.

```
NEW
1ØØ &N
11Ø &X=1Ø: &Y=4Ø
12Ø PRINT "CTRL-A FOR H2SO4"
13Ø PRINT "CTRL-B FOR 3,7-dimethylxanthine"
14Ø &I
15Ø IF ZK%=1 THEN ZI$= "@HH@D2SO@D4@L": GOTO 14Ø
16Ø IF ZK%=2 THEN ZI$= "@L3,7-DIMETHYLXANTHINE": GOTO 14Ø
17Ø GOTO 1ØØ
RUN
```

Typing at the &I produces letters as usual until either CTRL-A or CTRL-B is pressed. At that time, input ends and control goes to lines 15Ø and 16Ø, where the value of the last key (in ZK%) can be tested. If CTRL-A is pressed, ZK% is set to 1 (the ASCII value for CTRL-A) and ZI$ is set to "@HH@D2SO@D4@L", the key sequence that produces "H_2SO_4" and returns the keyboard to lower case. If CTRL-B is pressed, ZK% is set to 2 and ZI$ is set to a string that produces "3,7-dimethylxanthine". After ZI$ has been set to a string as a result of CTRL-A or CTRL-B being pressed, control is returned to the &I and the string is instantly displayed on the screen where the user is typing. If input is terminated by another judge key (such as the RETURN key), ZI$ is not altered and we start again.

Note that this approach updates ZC%, the count of "tries" at the input, each time a special key is executed and we return to the &I to continue input. If ZC% should not be altered in a particular situation where ZI$ is employed, ZC% should be decremented in the same line where ZI$ is set. For example:

```
15Ø IF ZK%=1 THEN ZI$= "@HH@D2SO@4": ZC%=ZC%-1: GOTO 14Ø
```

Needless to say, ZI$ is cleared of its current contents each time &I adds it to the input.

This completes our coverage of the augmentation commands that deal with user responses. You may now wish to do a quick review and test of some of the major aspects of these commands by trying the "Illustrations of Features" option available from the Master diskette Main Index. When you have completed that option, you will be ready to go on to the more advanced material in the remaining chapters.

CHAPTER V

Bit Manipulation

(NOTE: This chapter may be skipped if you have no need for the specialized techniques that it describes.)

Limited computer memory can cause problems if a good deal of status information must be kept. Consider a vocabulary drill. Efficient learning requires that items not be repeated after they have been mastered. Unknown or often-missed items must, on the other hand, be repeated in carefully controlled contexts to assure that the student is able to distinguish between similar items and that the student is actually learning the content rather than simply learning how to get through a drill with minimum effort.

In such situations, many items of information must be efficiently stored (to avoid running out of memory) and must be accessible with minimum delay (to avoid making student patience a requirement for learning!). "Efficient storage" dictates packing several items of information into each 8-bit Apple byte. Unfortunately, the mechanisms for packing and unpacking bytes available in BASIC are rather inefficient. For this reason, the &S and &U commands have been provided to permit efficient Setting and Unsetting of individual bits within a simple BASIC integer variable.

&S The SET Bit Command

The &S command works with two reserved integer variables, ZO% and ZB%. ZO% contains a copy of the the object byte and ZB% contains the number of the bit within ZO% to be set. Since &S works only on bytes, bits Ø to 7 are the only bits that can be altered. Larger values of ZB% are interpreted modulo 7--so "8" is treated as Ø and "1Ø" as 2. By convention, bit Ø is the least-significant bit.

To use &S, place a copy of the variable to be altered in ZO%, specify the bit to be set to 1 in ZB%, and then execute &S. Then replace the original variable with the altered version of ZO%. Since &S (and &U) work only on 8-bit bytes, a double-precision variable has only its lower byte altered by this process.

&U The UNSET Bit Command

&U works exactly like &S except that it "unsets" bit ZB% in ZO%--that is, it sets that bit to the value Ø. Again, after the operation, the original variable must be replaced by the altered version of ZO%.

Let us consider an example using both &S and &U. Suppose we wish to store information on which of the 5Ø U.S. state capitals have been learned by a student. These may be stored in 5Ø bits of a string variable. By use of the BASIC MID$ command, we can pull out a selected 8-bit byte, use the &S and &U commands to alter a selected bit, and then replace the altered byte in the string. Let X% represent the selected byte and suppose that bit 5 of that byte is to be set, indicating that the student has learned the name of one particular capital.

```
(after using MID$ to place the desired byte in X%)
1ØØ ZO% = X%
11Ø ZB% = 5
12Ø &S
13Ø X% = ZO%
(which is then returned to the string by concatenation)
```

The most efficient way of testing if a specific bit is set may seem a bit round-about, but follows a similar procedure. To test for the fifth bit of a given byte (which has been placed in X%):

```
1ØØ ZO% = X%
11Ø ZB% = 5
12Ø &U
13Ø IF ZO% <> X% THEN (action to be done if bit 5 is set)
14Ø IF ZO% = X% THEN (action to be done if bit 5 is clear)
```

Thus, to check if a given bit is set, we unset that bit in a copy of the byte and check to see if the two versions of the byte are equal. If not, the bit in the original must have been set. If the copies are equal, then the bit must have already been unset in the original.

CHAPTER VI

The Display Table and Key Table Editors

EnBASIC provides 128 addressable display slots for user-designed characters. The content of these display slots is determined by "display tables" which you may either select from among those provided with EnBASIC or which you may construct using the EnBASIC "Display Table Editor".

Access to the characters contained in the display slots is determined by a list of key effects (a "key table") which specifies what will happen when a student presses each key. As with display tables, you may either use a standard key table provided by EnBASIC or design your own using the EnBASIC "Key Table Editor". Designing your own key table allows your material to meet special needs (e.g., providing key effects that conform with those of existing materials).

The display tables and key tables used with your software are stored as binary files on the diskette containing your software. Though most needs can be met with only a single display table and a single key table, any number can be used. Appendix A tells how multiple display and key tables can be implemented.

Since the Master diskette provides several ready-made display tables and a key table, you need to use the Display Table or Key Table Editors only if your material has special needs not met by these existing tables.

The easiest and best way to explore the capabilites of the Display and Key table editors is simply to use them. Since any alterations you might make to a display or key table are not saved on diskette unless you request it, exploration is quite safe. Each of the editors has provision for testing the effects of changes before they are saved to diskette. As in all computer applications, it is wise to save backup copies of current material as well as copies of previous versions in case of accident (or subtle mistakes in new coding).

Although the editors allow you to assign keys and display slots in any way you wish, you should be aware of some potential problems. In general, it is useful to adhere to ASCII conventions wherever possible since BASIC often assumes that certain display effects are contained in specific display slots. For example, slot 13 is the display slot addressed whenever BASIC performs a carriage return, and slot Ø is addressed whenever a null character is to be plotted. In addition to the standard ASCII functions, some

slots are reserved for special EnBASIC functions, such as the input prompt arrows and spelling markup characters.

To prepare for our exploration of the editors, place the Master diskette in drive 1 and boot it. Go to the Main Index and select Option 6 ("Set up YOUR disk"). Within that option, ask to transfer individual EnBASIC files. Then follow the directions for copying the Standard Key Table and Standard Display Table to your test diskette. This transfer places copies of these tables on your diskette in a form that can be edited. You may recall that these same tables are a part of the combined EnBASIC package that you put on your test diskette in Chapter II; however, tables in such combined packages cannot be addressed for editing. After individual tables have been edited, you may use them from the "Set up YOUR disk" option to create either a new EnBASIC combined package or special package (see Appendix A).

After you have transferred copies of the Standard Display table and Standard Key table to your test diskette, return to the Main Index of the Master diskette.

Display Table Editing

From the Main Index, select "Display Table Editing", then follow the directions for entering it and for mounting your own diskette. Use your test diskette when it refers to "your disk" for now. After mounting your diskette and following the directions, you will arrive at an index of options. The two major options of the Display Table Editor are:

1. Edit an existing Display Table, or
2. Create a new Display Table

Other options, such as renaming a display table, are available as suboptions. In case you forget the name of a table to be edited, the Editor also provides the option of inspecting the catalog of the diskette being used.

Editing is basic to all display table operations and can be done to new or existing tables. Select the Edit option now and specify "Z.DISPLAY TABLE" when it asks what table is desired (this is the name of the Standard Display table that you just placed on your test diskette). The length (in bytes) and character size (in dot units) of the display table currently being edited is shown on the index of edit options. The total amount of space required for a display table is determined by the maximum character height (in dot units--chosen when the table is created). Thus, the size of a display table is given by the formula:

Display Table Size (bytes) = 4 + 128*(character height)

Option 1, "SEE Table Directory", allows you to examine the current contents of each of the 128 display slots for your display table and to compare these to the contents of the Standard Display table. Since you are using a copy of this Standard table, both your display and the standard display are identical (we will change that shortly though!).

Select the directory option now and read the introductory message. Then move to the first display. Note that each slot contains either a character of some sort (including a blank if it is unused) or a special display-control code. The display-control codes are represented in the displays by the following two- or three-letter abbreviations:

BKS = Backspace
CR = Carriage Return
DN = Subscript
DNL = Locking subscript
UP = Superscript
UPL = Locking superscript

Also note that some of the characters are enclosed in parentheses. These are language diacritical marks (or "accents"). They are automatically backspaced to place them over the letters that they are used with and are written in overstrike mode (or erase mode if writing is being done in inverse). You will also recognize some of the special spelling markup characters. Go through the directory now.

For the Standard Display table, the Display Table Directory listing is identical to that appearing in the column labeled "Display" in Appendix E. Each slot can be accessed either by referring to its decimal slot number (Ø to 127) or by a keyset entry that corresponds to that slot (if such exists). Slot number and keyset entry codes for each slot are also shown in Appendix E. Let us try to use those methods to access some individual characters now.

From the index display of the Display Table Editor, select option 2, "EDIT Characters". You are now asked to indicate the character to be edited. You may type the key that corresponds to the desired character or press RETURN to allow specification by slot number (necessary where no key corresponds to the slot number). For example, you may press key "A" to reach the slot that contains that symbol in the Standard Display table or press RETURN and type "65", the slot associated with upper-case A. We will select by character for now. When accessing slots by character, note that:

(1) Upper-case characters are reached by using ESC-ESC to toggle between upper and lower case (try it!).
(2) Some slots cannot be reached on the Apple][since no corresponding key exists (e.g., slot 28). However, access by character is still the most convenient way to edit the most common characters.
(3) You can always toggle to allow entry of key slot numbers simply by pressing the RETURN key while at the selection display (press RETURN a few times and see what happens).

Let us now access a display slot and try some editing. Select the slot containing "A". If you press A and get slot 97 (which contains "a"), simply press RETURN and try again--but press ESC ESC this time to toggle to upper case. Using the table from Appendix E, access several other slots both by slot number and by key until you feel sure that you are able to reach any slot you desire. Do this now.

When you feel comfortable accessing particular display slots, go to slot 46 (which contains the period) and let us examine some of the editing options available. Each character appears greatly enlarged on a matrix of dots representing the space available for design (in the case of the Standard Table, a space 8 dots high and 7 dots wide). The character is also shown in regular size and in double size so you can immediately see the effect of any changes you make. To the right of the display is a summary list of the editing options available and the keys that they use. The first of these options controls position and function of the editing cursor (a + sign located initially near the center of the character design matrix).

The cursor is positioned by pressing the 8 keys surrounding the S key on the keyboard. Pressing the W (the key directly above S) moves the cursor directly up; pressing the C (the key below and to the right of S) moves the cursor down and to the right; and so forth. The cursor cannot be moved off the grid or past the column marked with an arrow and the letter P. This column marks the proportional-spacing bound. Since a character as small as the period needs only a small part of the design area, we can specify that some of the unused space is to be ignored when the character is shown in proportionally-spaced mode. The minus (-) and plus (+) keys are used to move this boundary around. Normally it is set to allow a two-dot space after the right-most column used by the character. Some characters (like the period) may look better with somewhat more space. Each letter design includes the minimum between-character spacing desired, so

at least one free column usually appears on the right edge of the design matrix.

The cursor has three functional states. Initially it is in a "floating" state and can be moved around the design matrix with no effect on the display stored there. Pressing key S places the cursor in the STORE state so it "switches on" each square it moves across. Pressing key R places the cursor in the REMOVE state so it "switches off" each square it moves across. Key F sets the cursor back in the floating state.

Go ahead and alter the character design now using the F, R, S and cursor control keys and the proportional-bound keys. Note the effect on the regular and double-sized displays as you change the character. EnBASIC automatically fills in corners for the double-size characters to produce smoother shapes. Look closely at the differences between the regular and double-size versions of the character you are altering to see the effects of this automatic smoothing on the double-size version.

It is useful to see a character in the context it appears in when used. The CONTEXT feature allows this. Pressing the $ key (SHIFT 4) permits typing a two-line context for the edited character. For example, in designing a new numeral "Ø", you would want to make sure it is distinguishable from characters with similar shapes. You might type "0Øo8" as one line and "eØ89" as the other line (up to four characters may appear on each line). The context presentation appears just below the double-sized letter display as soon as its second line has been entered. Once a context has been entered, it remains until another one is entered or until you leave the Display Table editor.

Use the context feature now to form a context including the character that you are now editing and several others. Then modify the character and note the effect on versions of the character that appear in all of the displays.

A major use of the context feature is for building large characters out of several small characters that fit together in the program. The context feature is used to display the individual component characters while each segment is designed. An example of such a composite display is the running figure in the animation demo on the Demo diskette. Each view of the running figure is actually made of of four regular sized characters fit together to make a figure that is twice as high and twice as wide as the regular characters. See the program listing in Appendix H for the technique used in displaying these multi-character figures.

Special Characters

Within the "EDIT Character" option, we may also specify the special display control characters that we noted in our excursion through the directory display for the Standard Display table. A display slot can be used either to contain information that defines a graphic character (like a period or the letter "a") or it can be used to store information about controlling displays (like carriage returns or superscript directives). Normally it cannot be used for both of these functions at the same time (the exception is diacritical marks which have special automatic backspace-overstrike as well as graphic information).

To use a particular display slot to hold a special control function, press the * key while on the EDIT Character display for that slot. You are given the choice of defining a Carriage Return, Superscript, Subscript, Backspace, or Accent (Backspace-Overstrike). Except for the last, when one of these is selected, it deletes any graphic information that may have been stored in that slot before. Since we are not going to return our changes to the diskette, we are not going to worry about that right now. To get an idea of what each option looks like, we will make the following changes:

(1) Change "c" to a carriage return that descends 6 dots.
(2) Change "e" to a 2-dot raised, non-locking superscript.
(3) Change "b" to a backspace.
(4) Change "f" to a backspace-overstrike character. Then edit it to replace the "f" with a line along the top row of the character grid.

Let us go through the first change in our list step-by-step:

(1-a) Go to the slot for the lower-case "c" (either type "c" or choose display slot 99 from the slot selection display of the "EDIT Characters" option).
(1-b) Select the Special Character option by pressing key * (the asterisk key).
(1-c) Select option 1, "Carriage Return". You will be warned of the fact that you are about to overwrite the existing character in the slot. Press RETURN to continue anyway!
(1-d) You will be asked how many dots the carriage return is to descend. Normally, a carriage return will descend as many dots as the character height (recall the discussion in Chapter III on line spacing by character set design). However, we will use only 6-dots. Type "6" and press RETURN. The design will be stored and you will be returned to the slot selection display.

Now go ahead and do the remaining changes yourself.

Let us continue by selecting Option 1, "Edit an Existing Key Table". When you are asked for the name of your key table, enter "Z.KEY TABLE" (the name of the Standard Key table that you placed on your diskette at the beginning of this chapter). When you are asked for the name of your display table, enter "Z.DISPLAY TABLE" (again, the name of the Standard Display table placed on your diskette at the beginning of this chapter).

A display table must be specified while editing the key table because the effect of a given keypress is largely determined by the display table that is then in effect. When you have specified the key and display table to be used, their current versions will be loaded into RAM from your diskette. You are then offered the following choices:

(1) See Key Table Directory by Function
(2) See Key Table Directory Codes
(3) EDIT Key Table
(4) TEST Key/Display Table
(5) STORE Key Table on Disk
(6) CHANGE NAME of Key Table

Examination of the key table directories provides a useful overview of the joint effect of key table and display table. Select choice 1, the Functional Directory now.

Functional Key Directory

Use the RETURN and CTRL-B keys to page forward and backward through the Functional Directory. The displays show the actual effect of the 128 available key codes (Ø through 127). The ASCII keycode numbers appear in the left column of each display. Beside each keycode number, the Functional Directory shows (1) the related keypress, (2) the display effect resulting from pressing that key by itself, and (3) the display effect of pressing that key after an Access key has been pressed.

Access keys allow us to access more options that would be available if we were limited to one option per available keycode. In fact, there are some 15Ø options available using the Standard Key and Display tables. Many of these extra effects are produced by pressing an Access key and then pressing a regular key which has been defined in the key table to have a special "Access effect". If present, this effect is shown in the Functional Directory of the Key Table Editor in the "Access+Key" column for the given key. For example, look at key slot 66. Slot 66 is related to the B key, which produces "B" when pressed by itself but a "backspace" when pressed after an Access key.

Access keys have an additional special effect on upper-case and lower-case letters. By pressing any Access key twice in a row, a "case toggle" is performed and any letter typed at that input after the toggle appears in the other case. This effect is represented in the Functional Directory by the word "toggle" in the "Access+Key" column. Key number 15, CTRL-O, is an example of an Access key. In the "Key Alone" column, CTRL-O is shown to be an ACCESS key while in the column for "Access+Key" effect it is shown as having the "toggle" effect. The toggle can be produced by any sequence of two Access keys (e.g. ESC followed by CTRL-O); it need not be the same key pressed twice.

Key slots 96 through 127 are directly accessible only on the Apple //e and similar computers. On the unmodified Apple][+ slots 97 to 122 (which contain the lower-case letters) can be accessed using EnBASIC with the computer in "lower-case" mode. Case of displayed letters is determined by use of case toggle keys or forced-case functions which are defined in the key table.

In addition to the special display effects which were set in the display table (e.g., carriage return and backspace), you will notice other special effects such as "judge", "set margin", and "erase mode". These special effects are set in the key table itself. Appendix F contains a list of the special functions that can be assigned to keys and their numeric codes.

Special Function Code Directory

The direct application of the special codes listed in Appendix F may be seen by looking at the Key Table Directory that shows codes alone. Go to the Key Table index now and select the second option-- "See Key Table Directory Codes". You now see all 128 key codes listed (Ø to 127) along with any special codes in effect for them.

If no code number appears next to a key slot number in this table, that key simply has its "regular" function of producing the display table effect from the same-numbered display slot. For example, key slot 71 belongs to the "G" key and has no code associated with it. When the "G" key is pressed, the display from display slot 71 (the capital letter "G") is produced. Pressing an Access key and the "G" key produces no special effect. The Standard Key and Display tables use the same effects for both upper- and lower-case keys (slot 1Ø3, the letter "g", also has no special Access effect) so that Access effects on the letter

After making these changes, go back and examine each of the changed characters via the single-character edit option. Knowing the specifications followed in producing them, you should easily be able to interpret their new descriptions.

Now exit the EDIT Character option and select Option 3, "Test Your Table". Notice that the input prompt arrow on the Test display appears in inverse mode. This indicates that an "alternate" display table (the one we have been editing) is in effect. Type the letters "abcdef". Your edits should produce the following result (do you see why?):

$$\frac{a}{d}$$

Although you have the power to redefine keys in this manner, you will clearly want to take great care in doing so! The safest procedure is to make sure that you know why and how a particular display or function is used before deciding to use its slot for another purpose. In particular it is advisable to keep as close to ASCII functions that might be used by BASIC as possible. That includes making sure that slot Ø (NULL) is not used and slot 13 (CR) is in fact a carriage return. Keys that have special effects within BASIC PRINT statements (such as CTRL-D and ") should also be assigned with great caution to avoid possible problems.

If you are not using spelling feedback, accents, or some of the special math symbols in your material, slots used for those symbols are available to you. You can also safely redesign the shape of these characters if they do not suit you. The lower-case letters (if used) should be kept in their standard ASCII location, since the toggle functions assume this is where they are.

Finally, note that unless a copy of the changed display table is returned to your diskette when you have finished, the changes are lost. If you try to leave the editor after making changes, you will see a flashing prompt warning you about this. Normally you request this storage either by selecting Option 4 "STORE Table on Disk" or by selecting that option at the time you receive a warning when leaving the editor. You may have also noticed that changes cause a warning note to be shown beside the option 4 entry on the index. Since undesirable changes were made to Z.DISPLAY TABLE as a part of our exploration, we do not want to replace the good version of Z.DISPLAY TABLE on our diskette with the altered version. Even if the changes were more desirable, it is good programming practice to keep "backup" copies of old code rather than overwriting it. Option 5, "CHANGE Name of Table being Edited" of the Display Table editor allows an alternative. Select that option now.

Type in the name "TABLE TEST" when you are asked to give a new name for the table. As explained in the option, the name of the version on your diskette is not changed by this action. Nor is a copy of the renamed table automatically returned to your diskette. Press RETURN when you have finished typing the new name, and then RETURN again to return to the Display Table Edit index.

Now select option 4, "STORE Table on Disk", and follow the directions for storing the newly named copy on your diskette. Then press CTRL-B to exit to the "Edit Display Table" index. There select Option 3 "See the CATALOG..." for your diskette. The catalog should now show that you have two display tables stored on your diskette, "Z.DISPLAY TABLE" and "TABLE TEST". This edit-rename-store sequence is a prudent approach whenever major changes are made in files.

We will leave creation of a new display table for your own exploration now that you know how to edit an existing table. One note should suffice. When you create a new display table, you will be asked to indicate a desired character height and width (measured in screen dots). Choose a size that allows your largest character plus space for between-character spacing. Widths greater than 7 can use columns 8 and 9 only for this character spacing. If you use language diacritic marks, you will find that character heights of 1Ø or more are best to insure sufficient space for accent marks and well-formed character shapes.

We are now ready to examine the other major way in which we can control the Apple for friendly user interaction.

Key Table Editing

Place the Master Diskette in drive 1 and boot it (if it is not still there following your exploration of the Display Table Editor). This time select Option 5 on the Main Index, "Key Table Editing". Follow the directions given to you about where your own diskette is to be placed. As before, use your test diskette during this initial exploration.

The Key Table Editor provides the option of editing an existing key table or creating a new key table. Creating a new key table is much like editing an existing one (you are given a copy of the Standard Key table as a beginning). As with the Display Table Editor, editing is basic to use of the Key Table Editor and is the area that we will spend most of our time on. Since changes are not permanent until they have been returned to your diskette, and since you can test the effects of your changes from within the editor, learning by direct practice with the Key Table Editor is both safe and efficient.

keys will be the same no matter what case the keyset is in. While not required, this arrangment prevents user confusion. A slightly more interesting case is represented by key slot 73, the "I" key. Note that key slot 73 has the special code "23" associated with it. Special key codes numbered between 1 and 127 refer to the display slots 1 through 127 and allow us to access display effects in those slots. Slot 23 contains an upward pointing arrow. By assigning a display slot to another key, we are telling the computer to:

(1) produce the "normal" effect if the key alone is pressed,
(2) produce the assigned effect if the key is pressed immediately after an Access key has been pressed.

In the case of key slot 73, this means that the I key produces the letter I if the key is pressed alone and produces the upward arrow of display slot 23 if the I is pressed immediately after an Access key is pressed. Since the @ and ESC keys have been defined as Access keys, the sequence "@I" or "ESC I" produces an upward-pointing arrow.

Special function code 128 is very special in that it is the code that defines an Access key. As we noted in the discussion of the Functional Directory, the Access key is also special in that it serves as a toggle between upper and lower case when pressed twice.

Special function codes from 129 to 146 provide Access-key effects other than those available from the display table. Most of these functions have to do with redefining the way displays look or perform. Key slot 38, the & key, is one example. Key slot 38 is assigned the special function code 133, which produces the effect of shifting the display to inverse writing (see Appendix F). Like the codes from 1-127, the codes from 129 to 146 allow a key to have a special Access-key effect. In the case of the & key, an & symbol is produced if the key is pressed alone. However, if a sequence like @& or ESC-& is typed, subsequent characters are shown in inverse mode. Inverse mode continues until a new display is initiated or until the display mode is switched again (by, for example @%, which returns the display to Exclusive-or mode).

Finally, codes between 251 and 255 redefine a key to have only a non-display control function. An example is 251, which disables a key from making any input at all. More commonly used is code 255, which defines a key as a judging key. A judging key ends input at an &I.

A judging key can be used anywhere there is a need to allow immediate execution of some procedure while the program user is at an &I input. Usually the desired procedure is execution of &A or &O commands following an &I. However, response analysis is not the only reason for termination of user input. An example is the use of CTRL-B in the EnBASIC editors as a means of allowing a user to "back up" to previous displays. Key slot 2, CTRL-B is assigned the control function code 255. When this key is pressed at an &I, input is ended and these programs check for the keycode of the key which caused the termination. If the keycode was 2, we know that the user does not wish to have the input judged, but rather wants to leave the current display and return to an earlier one. Note again that because CTRL-B is assigned a control function code of 255, it no longer is connected to display slot 2 in any way. If we wish to plot the character in display slot 2, we must access it by assigning it to a special two-key Access sequence. In this case, key 44 (",") has been assigned to show the contents of display slot 2 when pressed immediately after an Access key.

Now that we have seen the structure of a working key table, let us see how we can alter the function of a key table.

Assigning Key Functions

Knowing the functional key codes listed in Appendix F and the key slot codes listed in Appendix E, we can assign new functions to keys directly. There is, in fact, one Editor option that allows such direct assignment. However, it is usually more convenient to make assignments from a "menu" that does not require that we remember the code numbers of keys and special functions, or the contents of our display table. The Key Table Editor provides several such menu approaches.

Let us select option 3, "EDIT Key Table", now from the main index of the Key Table Editor. Five options are available:

(1) Assign Control Functions
(2) Assign Display Functions
(3) Assign ACCESS-Display Characters
(4) Assign by Key Number and Code
(5) See effects of revised Key Table

Control functions are the single-key assignments such as defining a key to be an Access, edit, or erase key. Such keys lose their "normal" display function. Select Option "Control Functions" now and follow the directions necessary to redefine the "A" key to be a "NO-OP" (ignored) key.

Redefine the function of "A" now. Note that you are given the option of redefining either "A", "a", or both. When you have reassigned "A" alone, exit to the Assign Key Functions display.

Next let us select Option 2, "Display Functions". These are special Access-key options that assign a second function to keys if they are pressed immediately after an Access key has been pressed. Select this option and then follow the directions necessary to assign the "inverse" mode to key "b" (lower case B). Make this reassignment now. When you have finished, exit to the Assign Key Functions display.

Next examine Option 3, "ACCESS-Display Characters". This option allows us to define a special Access-key sequence that produces any character or display-table effect contained in the current display table. Select Option 3 now and follow the directions to assign the Carriage Return effect contained in display slot 13 to the letter C. After that, follow the directions needed to display the character in display slot 122 (the letter "z") and assign that character to the keys "D" and "d". When you have made all of these assignments, exit to the Assign Key Functions display.

Finally, select Option 4, "Assign by Key Number and Code". Press RETURN after reading the introductory message for this option. Then assign to key slot number 69 the special option code 132. This assignment gives the E key the special Access function of shifting the display into "exclusive-or" writing mode. In this case, we could have made the same assignment in a less mysterious fashion by using the Display Function option since slot 69 can be directly reached by pressing key E. However, if you are using an Apple][+, there are some key slots that cannot be specified by simply pressing a key on the keyboard. Option 4 circumvents that problem. For example, no key exists on the Apple][+ that accesses slot 95. For those programming with the Apple][+, the only way to assign that slot is by use of the key-code and function-code editor.

After making the assignment of function 132 to key 69, exit to the Assign Key Functions display.

Observing Revisions

Before returning changes to disk (which we will not want to do with the test changes we have made so far!), you should always verify that your changes were made correctly. The Key Table Editor provides two methods of verification.

First, we may examine the revised Key-Effect Table. Option 5 on the Assign Key Functions display allows us to check the changes that we have made. Select that option now and move through the table until you reach displays that show the key slots involved (slots 65 through 69 and 97 through 1Ø1). Compare what the Effect display shows for these slots with what the table below shows. If your changes were correctly made, the table and the display should show the same effects for each of the slots. Also check Appendix E to see the "normal" effects for these slots.

Key Code	Key	Key Alone	Access+Key
65	A	NO-OP	
66	B	B	Backspace (unchanged)
67	C	C	CR (Carriage return)
68	D	D	z
69	E	E	Exclusive-or
97	a	a	≈ (unchanged)
98	b	b	Inverse
99	c	c	(unchanged)
1ØØ	d	d	z
1Ø1	e	e	´ (unchanged)

If these effects are not present, return to the instructions above and try again or correct any errors.

A second way to check the effect of changes is to actually press the keys and see what happens. The Key Table Editor provides a test option for this as well. Exit from the revised function display and return to the main Key Table Editor display. Option 4, "Test Key/Display Table" is the option desired. Select it now. Since there must be some way to escape from the test version of the key table, the editor verifies that you have at least one key assigned to a "judge" function before allowing you to actually enter the option (the judge key is used as the method of telling the editor that you have finished testing).

Using ESC-ESC to toggle case at the input arrow, verify that:

(1) In lower case, letters "a" through "e" can be produced by keys "A" through "E",
(2) In upper case, only letters "B" through "E" can be produced by keys "A" through "E" (key "A" is ignored),
(3) Pressing @B backspaces writing one space,
(4) Pressing @b shifts writing to inverse writing mode,
(5) Pressing @c has no effect,
(6) Pressing @C and typing shows a carriage return effect,
(7) Pressing @D or @d produces the letter "z",
(8) Pressing @E shifts writing to normal (exclusive-or) mode,
(9) Pressing @e has no effect.

For the last seven cases, type other keys before and after testing the key effect to make any effects easier to detect.

Now exit from the test option and return to the Key Table Editor main index.

You should notice that, as with the Display Table Editor, there is an option for changing the name of the key table just edited and an option for storing the revised copy on your diskette. Since you have made changes in the key table you are now editing, a message reminding you of this fact appears beside the "STORE Key Table on Disk" option. If you attempt to leave the editor without storing a changed key table, you will also be warned and given another chance to store the updated copy. Since you tested both the name-changing option and the table-storing option in the Display Table Editor, there is no need to try them again here. The key table changes that we made are <u>not</u> ones that will be useful, so we will not return them to the diskette.

Exit from the Key Table Editor now. When you are reminded that your changes have not been saved, indicate that you do <u>not</u> wish to have an updated copy saved.

Changing Standard Tables

After you have had experience in using EnBASIC, you may wish to alter the standard key and display tables on the Master Diskette to meet your special needs. EnBASIC permits you to do this. Simply leave the Master Diskette in drive 1 when the editors tell you to place your own diskette there. Since there is no particular advantage to changing the Master Diskette versions (and some danger, since the editors themselves use these standard tables), <u>extreme</u> caution is advised before taking such a step. If you have any doubts, it is far safer to leave the Master Diskette tables alone and keep special tables on other diskettes.

Your exploration of the Key Table Editor and the main features of EnBASIC is now complete. You may wish to refer back to portions of this manual for details and examples; however, most information needed for use of the package is present in summary form in the various Appendices that follow this chapter.

Illustrations of the effects of many of the Key and Display Table options (such as proportional spacing; erase, inverse, overstrike, rewrite, and exclusive-or display modes) are contained in the "Illustration of Features" option on your Master Diskette.

NOTES

APPENDIX A

Display Table and Key Table Operations

Within the limits of available space, you may have as many display tables and key tables as desired in a given CAI program. If you use no more than two display tables and only one key table in your software, you may use the "combined EnBASIC package" option within the disk setup routine on the Master diskette. A combined package requires no added bookkeeping on your part. If, however, you need a greater number of display tables or key tables within a given program, you will need to know more about management of these tables. This Appendix provides that information.

If a "combined EnBASIC package" is not used, you must specify the location of all tables used so they can be found during operation of the program. Loading of the tables and augmentation program is best done within the "greeting" program of each diskette to allow automatic operation. Let us examine some of the memory considerations involved.

EnBASIC itself is located in Apple memory starting just above HGR2 at 24577. Immediately above it is the main key table, followed by the main display table and any other tables. LOMEM: is best set just above the last memory location used by the last table. APPENDIX B contains an outline showing table and program locations in relation to other occupants of RAM.

Since future editions of the package may differ in length from the current one, the augmentation program automatically computes reference addresses for you to use in specifying addresses of the various tables. By using these reference addresses rather than fixed memory locations, you will be able to incorporate future versions on your diskettes without having to revise the loading addresses of your key and display tables.

Each reference address is divided into two parts since 8 bits does not allow expression of a number as large as the addresses involved. The actual address is determined by multiplying the "high" part by 256 and adding the "low" part. The reference address of the beginning of space for all tables may be found by PEEKing the following location after the augmentation program is loaded:

	low part	high part
Pointers to start of table space	24588	24589

And the actual address would be found by :

```
PEEK(24588) + 256*PEEK(24589)
```

All key tables are exactly 132 bytes long, but display tables vary in length, depending on the maximum height (measured in "dot" units) of characters in the table. The length of a given display table is shown in the Display Table Editor or can be calculated from the formula:

Display Table Length (bytes) = 4 + 128*(vertical dots)

Thus, a display table which allows design of characters that can be up to 12 dots high would be
4 + 128*(12) = 154Ø bytes long.
The "standard" display table has 8-dot-high characters and is thus 4 + 128*(8) = 1Ø28 bytes long.

General LOMEM: specification

With the above information, it is now easy to write a LOMEM: command which will be guaranteed to work for a diskette even if you use a different version of the augmentation program in the future (assuming, of course, that you keep the same key and display tables). LOMEM: must be set to a point above the end of the last table. Since we know where to find the address of the first space available for tables, we need only PEEK that location and add to it the combined lengths of all of the tables used.

Suppose that your diskette will need three display tables and two key tables and that these tables have the following lengths:

Display Table "NORM"	1Ø28 Bytes
Display Table "TINY"	9ØØ Bytes
Display Table "C"	9ØØ Bytes
Key Table "KEYS"	132 Bytes
Key Table "KEY2"	132 Bytes
TOTAL	3Ø92 Bytes

LOMEM: is then set by:

```
PRINT CHR$(13);CHR$(4);"BRUN ENBASIC"
LOMEM: PEEK(24588)+256*PEEK(24589) + 3Ø92
```

The CHR$(13) and CHR$(4) in the PRINT statement above produces a RETURN and a CTRL D to indicate to BASIC that a disk operation is to be done.

General Table Setup Example

We will now continue with our general example to show how display tables would be addressed. For convenience, we begin by defining a character string,

D$ = CHR$(13) + CHR$(4)

and storing the beginning address of all tables (i.e., the first address after the end of the augmentation program),

LO% = PEEK(24588) + 256*PEEK(24589)

(Note that since LOMEM: reassigns the location for Apple variables, we cannot do the above steps until after LOMEM: has been set or the computer will "forget" where LO% and D% are located.)

The string D$ is the equivalent of pressing keys RETURN and CTRL-D, thus clearing any prior inputs and indicating that a disk operation is to be performed. Suppose that Key Table "KEY2" is to be our current key table and that Display Table "NORM" is to be our main display table. Using the knowledge that all key tables are 132 bytes long, we may now use LO% to specify storage locations of these two tables:

```
PRINT D$"BLOAD KEY2, A"(LO%)
PRINT D$"BLOAD NORM, A"(LO% + 132)
```

We would then have 132+1Ø28 = 116Ø bytes of memory used beyond the end of EnBASIC. We may place the remaining tables in any order following the current key and display table. Let us place them in the order KEYS, TINY, and C. This means that the addresses of all tables, relative to LO%, are:

Table	Length	Starting address
KEY2	132	LO%
NORM	1Ø28	LO% + 132
KEYS	132	LO% + 116Ø
TINY	9ØØ	LO% + 1292
C	9ØØ	LO% + 2192

Thus, the commands used to load them would simply be:

```
PRINT D$"BLOAD KEYS, A"(LO% + 116Ø)
PRINT D$"BLOAD TINY, A"(LO% + 1292)
PRINT D$"BLOAD C, A"(LO% + 2192)
```

Specifying an Alternate Font Display

The locations LO% and LO%+132 are special cases for the augmentation program. The key table at LO% and the display table at LO%+132 will automatically be used for all EnBASIC operations unless we alter pointers contained in 24580 and 24581 (for the display table) or 24584 and 24585 (for the key table). These locations contain the address of the start of each of these tables and are used by EnBASIC to determine which table is in effect. POKEing the address of a new table into these locations is thus all that is necessary to instantly switch the effect of student key presses. The only complication is the necessity for splitting the address of a table into two parts.

As an example of the computation of a two-part address, observe how we would specify the location of an "alternate" display table (a display table that may be accessed during program execution by pressing a key defined to be the alternate-font-access key). The following pointer locations are used for the address of the alternate font table:

	low part	high part
Alternate Display Table	24582	24583

Suppose that we wish to use display table "TINY" as the alternate display table. We have placed TINY in memory at LO% + 1292, so we must convert (LO% + 1292) into a two-part address that will fit into 8-bit bytes and then POKE those partial addresses into 25482 and 25483. The following lines will do this:

```
H = INT((LO%+1292)/256): REM HI PART OF POKE ADDRESS
L = (LO%+1292)-256*H : REM LO PART OF POKE ADDRESS
POKE 24582,L: POKE 24583,H : REM  SPECIFY ALTERNATE FONT
```

Changing Tables in Mid-Program

The procedure outlined above may be used to switch one of the current tables within the instructional program. Suppose, for example, that we wish to switch the key table being used from KEY2 to KEYS. All we need to do is determine the two-part address of KEYS and then POKE that into the pointer addresses for the current key table (24584 and 24585). We placed KEYS at LO% + 1160, so the lines needed would be:

```
H = INT((LO%+1160)/256): L = (LO%+1160)-256*H: REM KEYS
POKE 24584,L: POKE 24585,H : REM  MAKE KEY TABLE=KEYS
```


Other table changes are made in exactly the same fashion. Simply determine where you placed the table in memory, convert that address to a two-part address and POKE the two parts into the appropriate pointer addresses. If you wish to "turn off" a table rather than replace it, just POKE 0 into each of the two pointer addresses for that type of table. Here is a summary of the pointer locations. Remember, these are where the table addresses are stored, not the addresses themselves!

	Pointer Address	
	low part	high part
Beginning of Tables	24588	24589
Main Display Table	24580	24581
Alternate Display Table	24582	24583
Key Table	24584	24585

Multiple Programs

If your diskette will use several programs, each must contain its own LOMEM: command since BASIC does not retain this information (or settings of variables such as LO%) between programs. If tables are switched within a program, it is good practice to always return them to a standard "beginning" state before leaving a program. That way, you will be sure of having appropriate tables in effect for beginning titles.

Sample Greeting Setup

To end, let us combine all of the lines of BASIC needed in a greeting program to prepare your diskette to use the list of key and display tables in our example:

```
10   PRINT CHR$(13);CHR$(4);"BRUN ENBASIC" : REM ADD AUGMENT
20   LOMEM: PEEK(24588)+256*PEEK(24589) + 3090
30   D$ = CHR$(13) + CHR$(4): REM  RETURN PLUS CTRL D
40   LO% = PEEK(24588) + 256*PEEK(24589): REM TABLE BASE
50   PRINT D$"BLOAD KEY2, A"(LO%): REM  LOAD KEY TABLE
60   PRINT D$"BLOAD NORM, A"(LO% + 132): REM  MAIN DISPLAY
70   PRINT D$"BLOAD KEYS, A"(LO% + 1160): REM EXTRA KEY T
80   PRINT D$"BLOAD TINY, A"(LO% + 1292): REM EXTRA DISPLAY
90   PRINT D$"BLOAD C, A"(LO% + 2192): REM  EXTRA DISPLAY
100  H = INT((LO%+1292)/256): REM HI PART OF TINY ADDRESS
110  L = (LO%+1292)-256*H : REM LO PART OF TINY ADDRESS
120  POKE 24582,L: POKE 24583,H : REM  TINY=ALTERNATE FONT
```

APPENDIX B

Use of Apple][Memory

49151 ($BFFF) Highest RAM address on 48K system

Disk Operating System

HIMEM 384ØØ ($96ØØ)

Applesoft variables. Strings build down from HIMEM.

LOMEM (set here by user's program)

Alternate display and key tables Main Display Table (address pointers in 2458Ø and 24581) Main Key Table (address pointers in 24584 and 24585) Start of space for key and display tables (address pointers in 24588 and 24589) EnBASIC program (machine language)

24576 ($6ØØØ)

High-Resolution Graphics Page 2 (HGR2)

16384 ($4ØØØ)

BASIC programs (build up from 2Ø49)

2Ø48 ($8ØØ)

BASIC and other system routines

Ø

Note: A total of 14,336 bytes are available for your own BASIC programs (the space between 2Ø48 and 16384). To determine the amount of space still free after some programs are present, you may examine the end-of-program pointers at 175 and 176 (hex AF and BØ). Space remaining is:

```
Bytes free = 16384 - PEEK(175) -256*PEEK(176)
```


Useful Address Pointers and Addresses

	hex	decimal
Beginning of Tables	$6ØØC,D	(24588,9)
Main Display Table	$6ØØ4,5	(2458Ø,1)
Alternate Display Table	$6ØØ6,7	(24582,3)
Current Key Table	$6ØØ8,9	(24584,5)
User-defined "&" options	$6ØØA,B	(24586,7)
EnBASIC fast erase routine	$6Ø1Ø,1	(24592,3)
EnBASIC plot routine	$6Ø12,3	(24594,5)
Shift key flag	$6ØØF	(24591)

Address pointers consist of two locations containing, in order, the low and high part of the address of the named feature. Calling the EnBASIC plot routine causes the character in Register A of the 65Ø2 to be plotted. The "shift key flag" location is Ø if the computer has a non-functional shift key (Apple][+) and 1 if the shift key is functional (Apple //e and others).

Example 1:
The address of the start of the current Key Table is

```
PEEK(24584) + 256*PEEK(24585)
```

Example 2:
To place the alternate-font Display Table at 35ØØØ

```
H = INT((35ØØØ)/256): REM HI PART OF POKE ADDRESS
L = (35ØØØ)-256*H : REM LO PART OF POKE ADDRESS
POKE 24582,L: POKE 24583,H : REM  SPECIFY ALTERNATE FONT
```

Connecting and Disconnecting Augmentation

If it is not removed from memory, EnBASIC can be rapidly disconnected from and reconnected to BASIC. Disconnection is done by placing pointers to your routines at location $3F5 to replace those pointing to EnBASIC. The program can be reconnected simply by executing CALL 24576 which will replace the pointers to it at $3F5 and return EnBASIC to the same state it was in when it was disconnected.

If your routines use a display page other than HGR2, you must shift to that display after disconnecting EnBASIC. When augmentation is reconnected, it will automatically shift back to HGR2 when it executes the first &N command.

Special display and key functions are altered by:

POKE 2459Ø,Ø	EnBASIC Key and Display tables enabled.
POKE 2459Ø,1	Standard BASIC display and key functions enabled.

Low Memory Usage

In addition to the low-memory cells used by BASIC, this package uses several locations during operation of the augmentation process. Use is of two types:

(1) Temporary: used only during execution of a specific augmentation function and available for similar temporary use at other times.

(2) Long-term: used for purposes that require that values not be altered during use of the augmentation and BASIC commands. If these values are altered, the augmentation routines will not function properly.

Locations used are as follows:

Location (hexadecimal)	Availability
$6 through $9	Alterable
$1E through $1F	Alterable
$21	Do not alter
$24 through $25	Do not alter
$26	Alterable
$36 through $37	Do not alter
$4Ø through $49	Alterable
$4E through $4F	Do not alter
$69 through $7Ø	Do not alter
$79 through $7A	Do not alter
$81 through $84	Do not alter
$9D through $A1	Alterable
$B8 through $B9	Do not alter
$CE through $CF	Do not alter
$EØ through $E2	Do not alter
$E4	Do not alter
$E6	Do not alter
$EB through $ED	Alterable
$EE through $EF	Do not alter
$FA through $FB	Alterable
$FC through $FF	Alterable
$2ØØ through $2FF	Do not alter

APPENDIX C

Summary of Commands, Option Codes, Variables, and Error Codes

Command	Use
&A "[str]"	Specify an acceptable answer
&D	Delay for ZS seconds
&E	Erase full screen and set display options
&E=[n]	Erase n characters starting at current screen position
&E=[n_1,n_2]	Erase n_2 lines of n_1 characters starting at current screen position
&I	Accept student input
&K	Put key in ZK% (Ø if none), remove from key buffer
&M	Markup unacceptable student input
&N	Prepare for new display (&E+&R+initialize key buffer)
&O "[str]"	Specify alternative (or) answers following an &A
&P	Pause for single-key input and put ASCII value in ZK%
&R	Reset counters prior to &I
&S	Set bit ZB% in ZO%
&T	Test key buffer, set ZK% to first key in buffer (Ø if none)
&U	Unset bit ZB% in ZO%
&X=[n]	Set x screen coordinate (Ø-279), Ø = left
&Y=[n]	Set y screen coordinate (Ø-191), Ø = top

Options (for &A or &O)

<C>	Require capitalization where included
<I, w1,...>	Ignore listed words w1, w2, ...
<L>	Judge on letters only (ignore capitalization)
<M>	Accept words that are probably misspelled
<P>	Judge punctuation like regular characters
<S, w1,...>	Treat listed words as synonyms
<V, n$ >	Embed contents of variable n$ in command
<W>	Accept words in any order
<X, w1,...>	Reject responses containing excluded words w_1, w_2,...
<X>	Exclude no words (all extra words are acceptable)

NOTES:
[str]= a list of acceptable words and command options
[n]= any legal BASIC constant or simple variable; integer or real
n$ = any BASIC simple variable string name
w1, w2, etc. = words expected in student responses

BASIC String Variables

NO$	String shown to student for Wrong response by &M
OK$	String shown to student for Correct response by &M
ZA$	Student answer string
ZF$	Forced string of keys input at &I
ZI$	Inserted string in input at &I
ZM$	String with markups of student response

BASIC Real Variables

ZS	Seconds delay at &I, &D, or &P before time-up key is pressed
ZT	Time in seconds used by student to respond at &I or &P

BASIC Integer Variables

ZA%	Match to correct answer (Ø=Wrong, 1=OK)
ZB%	Bit to be set or unset by &S or &U (Ø to 7)
ZC%	Count of student responses at &I
ZE%	&A error return (Ø=OK, 1-9 error)
ZK%	ASCII value of key detected by &I, &P, &K, or &T
ZL%	Maximum length of student response string
ZN%	Numerical position of closest matching &A or &O
ZO%	Object byte that is altered by &S or &U
ZW%	Match to wrong response (Ø=OK, ≥1=Wrong)

Error Codes for &A and &O Commands

Value of ZE%	Error
Ø	No errors
1	Student input string too long (>25Ø characters)
2	Author input string too long (>25Ø characters)
3	Quote mark missing at start of command
4	No comma or space between V and BASIC variable: e.g. <VC$> should be <V,C$>
5	No > at end of command: e.g. <V,C$
6	Bad BASIC string variable name
7	Too many words in student response (>32)
8	Too many total student and author words (student words x author words>224)
9	<C>,<M>,<P>,<U> not before author's words

APPENDIX D

Standard Key Table Functions

This Appendix summarizes the effects of the Standard Key Table when used with the Standard Display Table. These tables are used for the editors and may also be used in your own software. ASCII codes resulting from a single key press at run time may be detected by examining the value of ZK%. Some special key effects may also be produced within a BASIC PRINT statement either by typing the key or by use of its ASCII code as the argument of CHR$(n).

Normal Keyboard Functions

--All alphanumeric keys plot the character shown on the keycap when pressed.
--If a keycap has a shifted key marked, it plots that character if pressed with the SHIFT key (except @, which serves as a special ACCESS key).

Special Run-Time Functions

The following functions work only when used at run time when the user is at an &I input. Under "Keys Pressed", an entry such as CTRL-A means that both the CTRL and A keys are pressed simultaneously. "Function code" is the special code used within the Key Table editor to assign a given function. Function codes with values of 128 or more are special EnBASIC functions (see page F-2). Function codes of 1 through 127 are assignments of standard display functions to new keys.

Function	Keys Pressed	Function Code	ASCII
Edit a response at an &I input			
Erase single character	←	254	8
Edit line of text	→	253	21
Subscript next character (//e only)	↓	1Ø	1Ø
Superscript next character (//e only)	↑	11	11
Toggle upper/lower case	CTRL-T	252	2Ø
Judge a response at an &I input			
Standard judge	RETURN	255	13
Alternate Judge ("Answer")	CTRL-A	255	1
Alternate Judge ("Back")	CTRL-B	255	2
Alternate Judge (//e only)	open apple	255	none
Alternate Judge (//e only)	filled apple	255	none

ACCESS Key Functions

The following functions and special displays are produced by a two-key sequence which may be used either at run time or within a BASIC PRINT statement. The first key is a key defined to be an ACCESS key. Note that, of the available ACCESS keys, only @ can normally be used within a PRINT statement. Each two-key sequence must begin with an ACCESS key, even if several such two-key functions are selected immediately after each other.

Function	Keys Pressed	Function Code	ASCII
ACCESS for two-key sequences			
Standard ACC key (&I only)	ESC	128	27
Alternate ACC key (&I only)	CTRL-O	128	15
Alternate ACC key (PRINT or &I)	@	128	64

All of the following special displays or functions require more than one key for access. Most are two-key ACCESS sequences. Where "ACC" is specified as one of the keys in the "Keys Pressed" column, any of the above ACCESS keys may be used (except within PRINT statements, where only @ should be used). If CHR$(n) is used to access these effects in PRINT statements, each key shown in the ASCII column must be used. Where an ACCESS key is required, the ASCII code of any of the ACCESS keys may be used in place of 64 (the ASCII code for the @ key). In the case of some display options, the ASCII code for the display slot itself can be given instead of the multi-key means of accessing that slot. In those cases, only that single ASCII code is shown.

Function	Keys Pressed	Function Code	ASCII
Select font or spacing			
Standard-size letters	ACC-1	146	64-49
Double-size letters	ACC-2	145	64-5Ø
Alternate Font	ACC-F	142	64-7Ø
Standard Font	ACC-S	143	64-83
Fixed character spacing	ACC-Ø	136	64-48
Proportional character spacing	ACC-P	137	64-8Ø
Backspace	ACC-B	8	64-66
Carriage Return (single space)	ACC-R	13	64-82
Carriage Return (1.5 space)	CTRL-C	3	3

Function	Keys Pressed	Function Code	ASCII
Upper/lower case letter control			
Display all letters as upper case	ACC-H	134	64-72
Display all letters as lower case	ACC-L	135	64-76
Display letters in actual case	ACC-Z	144	64-9Ø
Toggle upper/lower case	ACC-ACC	128-128	64-64
Superscripts and subscripts			
Superscript	ACC-U	11	11
Subscript	ACC-D	1Ø	1Ø
Locking superscript	ACC-+	7	7
Locking subscript	ACC-;	12	12
Margin specification			
Set margin	ACC-CTRL-P	141	64-16
automatic CR at right margin	ACC-CTL-R	138	64-18
Wrap-around after right margin	ACC-CTL-Q	139	64-17
Set Special Display Modes			
Set display to ERASE mode	ACC-!	129	64-33
Set display to OVERSTRIKE mode	ACC-#	13Ø	64-35
Set display to REWRITE mode	ACC-$	131	64-36
Set display to EXCLUSIVE-OR mode	ACC-%	132	64-37
Set display to INVERSE mode	ACC-&	133	64-38
Set display to NON-PLOT mode	ACC-*	14Ø	64-42
Language Diacritic Marks			
^ (circumflex)	^	94	94
´ (acute accent-auto backspace)	ACC-E	6	6
` (grave accent-auto backspace)	ACC-Q	127	64-81
¨ (umlaut-auto backspace)	ACC-W	27	64-87
~ (tilde) (~ key on //e)	ACC-N	126	126
Math and scientific symbols			
@ ("at" sign)	ACC-6	64	64-54
≠ (not equal)	ACC-=	3Ø	3Ø
≤ (less than or equal)	ACC-<	25	25
≥ (greater than or equal)	ACC->	26	26
' (minute sign)	ACC-'	96	96
° (degree sign)	ACC-O	28	28
(The Apple //e also allows direct access of the following)			
[(left bracket)	ACC-8	91	91
] (right bracket)	ACC-9	93	93
{ (left brace)	ACC-(	123	123
} (right brace)	ACC-)	125	125
| (absolute value bracket)	ACC-CTRL-I	124	124
\ (back slash)	ACC-/	92	92

Function	Keys Pressed	Function Code	ASCII
Special graphic symbols			
↑ (north arrow)	ACC-I	23	23
→ (east arrow)	ACC-K	29	29
↓ (south arrow)	ACC-M	24	24
← (west arrow)	ACC-J	31	31
■ (7x7 solid square)	ACC-Y	21	64-89
NO-OP (key completely ignored)	CTL-@	251	Ø
Edit and response markups			
↱ (input prompt-upper case)	ACC-,	2	64-44
↳ (input prompt-lower case)	ACC-.	1	64-46
X (extra-word mark)	X	88	88
x (extra-character mark)	low-X	12Ø	12Ø
= (wrong-character mark)	=	61	61
~ (accent-error mark)	ACC-A	15	64-65
▲ (insert-word mark)	CTRL-P	16	16
ˇ (subscript-error mark)	CTRL-Q	17	17
^ (superscript-error mark)	CTRL-R	18	18
⌊ (insert-letter-before mark)	CTRL-S	19	19
⌋ (insert-letter-after mark)	ACC-T	2Ø	64-84
◀ (move-word-left mark)	CTRL-V	22	22
↑ (upper-case error mark)	CTRL-W	23	23
↓ (lower-case error mark)	CTRL-X	24	24
∿ (exchanged-letters mark)	CTRL-E,CTRL-D	5,4	5,4

Notes:

"ACC" may be any of the special Access keys (@, ESC, or CTRL-O) defined on page D-2.

"low" indicates that the keyboard has been placed in lower case mode by use of the EnBASIC case toggle key, by the EnBASIC locking lower-case key, or (Apple //e or modified Apple][+ only) by direct keyboard entry of a lower-case letter.

APPENDIX E

Standard Key/Display Table Functions
Ordered by ASCII Code

Slot Number	ASCII Code	Display	Keys pressed][+	Keys pressed //e	One-key effect	ACCESS+Key effect
00 ($00)	NULL		CTRL-@			
01 ($01)	SOH	↳	CTRL-A		judge	
02 ($02)	STX	↱	CTRL-B		judge	
03 ($03)	ETX	CR	CTRL-C		1.5 line CR	
04 ($04)	EOT	⌋	CTRL-D		⌋	
05 ($05)	ENQ	⌉	CTRL-E		⌉	
06 ($06)	ACK	´	CTRL-F		´	
07 ($07)	BELL	lock super	CTRL-G		lock super	
08 ($08)	BS	backspace	←		erase	
09 ($09)	HT	space	CTRL-I	TAB	space	|
10 ($0A)	LF	subscript	CTRL-J	↓	subscript	
11 ($0B)	VT	superscript	CTRL-K	↑	superscript	
12 ($0C)	FF	lock sub	CTRL-L		lock sub	
13 ($0D)	CR	CR	RETURN		judge	
14 ($0E)	SO	superscript	CTRL-N		superscript	
15 ($0F)	SI	~	CTRL-O		ACCESS	toggle caps
16 ($10)	DLE	▲	CTRL-P		▲	set margin
17 ($11)	DC1	ˇ	CTRL-Q		ˇ	wrap at rt margin
18 ($12)	DC2	^	CTRL-R		^	CR at rt margin
19 ($13)	DC3	⌊	CTRL-S		⌊	
20 ($14)	DC4	⌋	CTRL-T		toggle caps	
21 ($15)	NAK	■	→		edit	
22 ($16)	SYN	◀	CTRL-V		◀	
23 ($17)	ETB	↑	CTRL-W		↑	
24 ($18)	CAN	↓	CTRL-X		↓	
25 ($19)	EM	≤	CTRL-Y		≤	
26 ($1A)	SUB	≥	CTRL-Z		≥	
27 ($1B)	ESC	÷	ESC		ACCESS	toggle caps
28 ($1C)	FS	°	(none)	CTRL-\	degree sign	
29 ($1D)	GS	→	CTR-SHF-M	CTRL-]	→	
30 ($1E)	RS	≠	CTRL-^		≠	
31 ($1F)	US	←	(none)	CTRL--	←	

Notes:

"ASCII Code" is the standard ASCII function of a key and display slot.

"Display" is the information stored in the Standard Display Table for slot number "n". CHR$(n) in a PRINT statement normally produces this display.

"Keys pressed" shows what key produces key-code numbers for the Apple][+ and //e. The //e has all][+ keys, plus those shown in the //e column.

"One-key effect" shows the result of pressing the named key or executing a CHR$(n) in a PRINT statement. Unless altered by the Key Table, the effect is that shown under "Display".

"Access+Key effect" is the effect of pressing an ACCESS key followed by pressing the key for this slot.

Slot Number	ASCII Code	Display	Keys pressed][+	Keys pressed //e	One-key effect	ACCESS+Key effect
32 ($2Ø)	SP	space	SPACE		space	
33 ($21)	!	!	!		!	erase mode
34 ($22)	"	"	"		"	
35 ($23)	#	#	#		#	overstrike mode
36 ($24)	$	$	$		$	rewrite mode
37 ($25)	%	%	%		%	exclusive-or mode
38 ($26)	&	&	&		&	inverse mode
39 ($27)	'	'	'		'	minute sign
4Ø ($28)	(	(	(		(	{
41 ($29)	)	)	)		)	}
42 ($2A)	*	*	*		*	non-plot
43 ($2B)	+	+	+		+	lock superscript
44 ($2C)	,	,	,		,	
45 ($2D)	-	-	-		-	underline
46 ($2E)	.	.	.		.	
47 ($2F)	/	/	/		/	\
48 ($3Ø)	Ø	Ø	Ø		Ø	fixed spacing
49 ($31)	1	1	1		1	regular size
5Ø ($32)	2	2	2		2	double size
51 ($33)	3	3	3		3	
52 ($34)	4	4	4		4	
53 ($35)	5	5	5		5	
54 ($36)	6	6	6		6	@
55 ($37)	7	7	7		7	
56 ($38)	8	8	8		8	[
57 ($39)	9	9	9		9	]
58 ($3A)	:	:	:		:	
59 ($3B)	;	;	;		;	lock subscript
6Ø ($3C)	<	<	<		<	≤
61 ($3D)	=	=	=		=	≠
62 ($3E)	>	>	>		>	≥
63 ($3F)	?	?	?		?	

Slot Number	ASCII Code	Display	Keys pressed][+	Keys pressed //e	One-key effect	ACCESS+Key effect
64 ($4Ø)	@	@	@		ACCESS	toggle caps
65 ($41)	A	A	A		A	~
66 ($42)	B	B	B		B	backspace
67 ($43)	C	C	C		C	
68 ($44)	D	D	D		D	subscript
69 ($45)	E	E	E		E	˘
7Ø ($46)	F	F	F		F	alternate font
71 ($47)	G	G	G		G	
72 ($48)	H	H	H		H	show upper case
73 ($49)	I	I	I		I	↑
74 ($4A)	J	J	J		J	←
75 ($4B)	K	K	K		K	→
76 ($4C)	L	L	L		L	show lower case
77 ($4D)	M	M	M		M	↓
78 ($4E)	N	N	N		N	
79 ($4F)	O	O	O		O	degree sign
8Ø ($5Ø)	P	P	P		P	proportional spacing
81 ($51)	Q	Q	Q		Q	`
82 ($52)	R	R	R		R	single line CR
83 ($53)	S	S	S		S	standard font
84 ($54)	T	T	T		T	⌟
85 ($55)	U	U	U		U	superscript
86 ($56)	V	V	V		V	
87 ($57)	W	W	W		W	~
88 ($58)	X	X	X		X	^
89 ($59)	Y	Y	Y		Y	■
9Ø ($5A)	Z	Z	Z		Z	show cases
91 ($5B)	[	[	(none)	[	[	
92 ($5C)	\	\	(none)	\	\	
93 ($5D)	]	]	SHIFT-M	]	]	
94 ($5E)	^	^	^		^	
95 ($5F)	_	_	(none)	_	underline	

Note:
Slot number 93 can be accessed on the Apple //e only by the] key.

Slot Number	ASCII Code	Display	Keys pressed][+	Keys pressed //e	One-key effect	ACCESS+Key effect
96 ($6Ø)	`	`		`	`	
97 ($61)	a	a		a	a	≈
98 ($62)	b	b		b	b	backspace
99 ($63)	c	c		c	c	
1ØØ ($64)	d	d		d	d	subscript
1Ø1 ($65)	e	e		e	e	´
1Ø2 ($66)	f	f		f	f	alternate font
1Ø3 ($67)	g	g		g	g	
1Ø4 ($68)	h	h		h	h	show upper case
1Ø5 ($69)	i	i		i	i	↑
1Ø6 ($6A)	j	j		j	j	←
1Ø7 ($6B)	k	k		k	k	→
1Ø8 ($6C)	l	l		l	l	show lower case
1Ø9 ($6D)	m	m		m	m	↓
11Ø ($6E)	n	n		n	n	~
111 ($6F)	o	o		o	o	degree sign
112 ($7Ø)	p	p		p	p	proportional spacing
113 ($71)	q	q		q	q	`
114 ($72)	r	r		r	r	single line CR
115 ($73)	s	s		s	s	standard font
116 ($74)	t	t		t	t	⌡
117 ($75)	u	u		u	u	superscript
118 ($76)	v	v		v	v	
119 ($77)	w	w		w	w	¨
12Ø ($78)	x	x		x	x	^
121 ($79)	y	y		y	y	■
122 ($7A)	z	z		z	z	show cases
123 ($7B)	{	{		{	{	
124 ($7C)	|	|		|	|	
125 ($7D)	}	}		}	}	
126 ($7E)	~	~		~	~	
127 ($7F)	DEL	`		DELETE	erase	
128 ($8Ø)	(none)	(none)		(timeup)	judge	
129 ($81)	(none)	(none)		(ó apple)	judge	
13Ø ($82)	(none)	(none)		(é apple)	judge	

Notes:

Key slots 96 through 127 are directly accessible only on the Apple //e. Lower-case alphabetic characters are accessible on the Apple][+ using EnBASIC with the upper/lower case toggle set to "low" and on the Apple //e when the keyboard is in lower case.

Key code 128 is produced by the EnBASIC timeup function.

Key codes 129 and 13Ø are EnBASIC representations of the Apple //e open- and closed-apple keys (i.e., game inputs Ø and 1).

APPENDIX F

Key Function Codes

Keys normally "map" to the the display slot which has the same ASCII slot number as the key. That is, when the key that produces ASCII code 13 is pressed, it normally produces the effect specified in character slot 13. The Key Table Editor allows such functions to be altered. Each of the 128 key slots (some of which may not actually be represented by a key) maps to the corresponding character slot if the Key Table contains the code "Ø" in that slot. If other values are entered, the effect of the key can be changed or augmented. This augmentation scheme allows less than 96 keys on the Apple][+ to access about 15Ø different characters and special functions.

Alternative Display-Access Codes--

1-127 If a key slot is set to a value between 1 and 127, it produces its regular function when pressed as a single key, but produces the function specified by the named display slot if pressed as a two-key sequence: the "ACCESS + Key" sequence. E.g., key 73 (the I key) produces the character in slot 73 (the letter I) if pressed alone, but since it is assigned the value 23 in the key table, it produces an upward-pointing arrow if pressed immediately after an ACCESS key has been pressed. The upward arrow is contained in display slot 23 of the Standard Display table.

Access Key Definition Code--

The code 128 has the special effect of defining a key slot to be an "Access" key. This key slot no longer produces the display effect contained in its related display slot. Instead, it serves as a means of signaling the start of a special two-key sequence.

128- Key becomes an ACCESS key for use in two-key sequences.

An ACCESS key also serves as an upper-case/lower-case letter toggle if pressed twice, thus allowing the Apple][+ to use both upper- and lower-case letters. Single-keypress case-toggling is also available by use of Control Function code 252 (described later in this Appendix).

Display Function Codes--

The codes below define the effect of a key pressed after an Access key has been pressed. The key has its normal effect if pressed alone, but has a special two-key-sequence effect on displays when pressed immediately after an Access key has been pressed. The specific effect of assigning a key slot to one of these special codes within the Key Table editor is:

129- Shift display to ERASE mode
13Ø- Shift display to OVERSTRIKE mode
131- Shift display to REWRITE mode
132- Shift display to EXCLUSIVE-OR mode
133- Shift display to INVERSE mode
134- Display all letters as upper-case
135- Display all letters as lower-case
136- Set display to fixed character spacing
137- Set display to proportional character spacing
138- Provide automatic carriage return at right margin
139- Wrap writing around on same line beyond right margin
14Ø- Set display to non-plotting mode
141- Set margin to current horizontal position
142- Select alternate display table ("alternate font")
143- Select standard display table ("standard font")
144- Display letters in true case (upper/lower)
145- Set display to double size characters
146- Set display to regular size characters

Control Function Codes--

The codes below redefine the effect of a key. The key no longer has any connection with its corresponding display slot. Instead, the key assigned one of these codes in the Key Table editor takes on one of the following input-control or editing functions:

251- Disable key from making any input
252- Toggle upper/lower case letters at &I input
253- Produce edit-key function at &I
254- Produce erase-key function at &I
255- Produce judge-key function at &I

APPENDIX G

Index to Example Programs

Your Demo Diskette contains copies of all programs that you are asked to type in for test purposes. If you wish to see these programs but do not wish to type them in yourself, boot the Demo diskette, press CTRL-RESET, and load, list and run them by the following names. Page numbers are the pages in the manual where the program is described.

APPENDIX H

Sample Program Listings and Displays

The following pages give listings of some of the programs included on your Demo diskette.


```
10  REM  : NORTH STAR
30  LOMEM:  PEEK (24588) + 256 *  PEEK (24589) + 132 +
    1028
40 D$ =  CHR$ (13) +  CHR$ (4)
50 OK$ = "OK":NO$ = "NO"
100  & N: VTAB 4: HTAB 4: PRINT "@PT@LHIS PROGRAM ILLU
    STRATES THE USEOF THE INPUT AND ANSWER COMMANDS."

110  VTAB 10: PRINT "@HT@LRY VARIOUS MISSPELLINGS OF T
    HE ANSWERTO THE QUESTION TO SEE HOW THE PROGRAMRE
    SPONDS."
120  VTAB 20: HTAB 9: PRINT "@HP@LRESS@H RETURN@L TO C
    ONTINUE."
130  & P: IF ZK% = 2 THEN 1000: REM  PRESSED CTRL B?
140  IF ZK% <  > 13 THEN 130: REM  PRESSED RETURN?
200  & N: VTAB 1: HTAB 8: PRINT "@2SPECIAL KEYS@1"
210  VTAB 6: HTAB 2: PRINT "ESC ESC -  Toggles UPPER/1
    ower case": VTAB 8: PRINT "RETURN  -  Judge Answe
    r": VTAB 10: PRINT "  @J      -  Erase Character":
     VTAB 12: PRINT "  @K      -  Edit Answer"
220  VTAB 14: PRINT "CTRL A  -  See the Answer": VTAB
    16: PRINT "CTRL B  -  BACK to exit"
230  VTAB 20: HTAB 8: PRINT "Press RETURN to continue.
    "
240  & P: IF ZK% = 2 THEN 1000
250  IF ZK% <  > 13 THEN 230
500  & N: VTAB 2: HTAB 2: PRINT "@HG@LIVE ANOTHER NAME
     FOR THE STAR @HP@LOLARIS."
510  HPLOT 0,118 TO 279,118
520  VTAB 16: HTAB 14: PRINT "@Z@PRemember"
530  VTAB 18: HTAB 4: PRINT "@ZESC-ESC -  Toggle UPPER
    /lower caseRETURN  -  Judges the answer     @J
     -  Erase last character     @K      -  Edits answe
    r"
540 ZL% = 80: REM   MAX LENGTH OF ANSWER
550 ZF$ = "@@": REM : TOGGLE UPPER CASE
560  VTAB 6: HTAB 10: & I
570  IF ZK% = 2 THEN 1000: REM :CTRL B?
580  IF ZK% = 1 THEN ZC% = 2: REM  CTRL A?
590  & A"<C><I,@LIT,IS,THE,NAME,A><S,@HN@LORTH,@HP@LOL
    E>@HS@LTAR"
600  & M
```

```
610  IF ZA% THEN 690
620 ZK% = 0: REM  ZERO SO CAN USE AS FLAG FOR KEY PRES
    S AT PAUSE
630  VTAB 12: HTAB 4: REM     POSITION OF WRONG COMMEN
    TS
640  IF ZC% = 1 THEN  PRINT "No, please try again.": REM
     1ST TRY WRONG COMMENT
650  IF ZC% > 1 THEN  PRINT "@HT@Lry: @HN@LORTH @HS@LT
    AR"
660  IF ZK% THEN 560
670  & T: IF  NOT ZK% THEN 670
680  GOTO 630: REM :LOOP BACK THROUGH ON WRONG COMMENT
    S TO ERASE
690  VTAB 12: HTAB 1: PRINT "@PT@LHAT IS CORRECT. @H P
    @LRESS @HRETURN @LTO GO ON."
700  & P: REM  PAUSE ANY KEY
1000  & N:  VTAB 10: HTAB 4: PRINT "Press RETURN for th
    e index": VTAB 12: HTAB 10: PRINT "CTRL B to repe
    at"
1010  & P:  IF ZK% = 2 THEN 500
1020  IF ZK% <   > 13 THEN 1010
1030  & N:  VTAB 10: HTAB 4: PRINT "ONE MOMENT PLEASE..
    ."
1040  PRINT D$;"RUN INDEX"
```



```
10  REM  : ANIMATION
20  LOMEM:  PEEK (24588) + 256 *  PEEK (24589) + 132 +
     1028 + 1028
30 D$ =  CHR$ (13) +  CHR$ (4)
60  PRINT D$;"BLOAD MAN,A"( PEEK (24580) + 256 *  PEEK
    (24581) + 1028): REM  LOAD DISPLAY TABLE
65 A = ( PEEK (24580) + 256 *  PEEK (24581) + 1028):H =
     INT (A / 256):L = A - 256 * H: REM  CALC HI LOW
    BYTE POKE
70  POKE 24582,L: POKE 24583,H: REM   ADDRESS OF DISPL
    AY TABLE
100  & N: VTAB 4: HTAB 4: PRINT "2PSAMPLE ANIMATION1"
110  VTAB 9: HTAB 1: PRINT "PThis program contains a s
    ample animationwhich illustrates the use of the r
    ewritemode to erase a character as it is movedand
     the use of &X and &Y for positioningcharacters t
    o the nearest dot."
120  VTAB 24: HTAB 8: PRINT "Press RETURN to continue.
    "
130  & P: IF ZK% <   > 13 THEN 130
140 Y = 100:Y1 = 108: REM   SET Y LOCATION
150 D = 2: REM FIGURE SPACING
160 T = .02: REM FIGURE TIMING
170  GOTO 260
180  REM   KEY CHECKING SUBROUTINE
190 X = X + D: IF X > 279 THEN X = 1
200  & X = X: & Y = Y:ZS = T
210  & P: REM   PAUSE FOR TIMEUP OR KEY
220  IF ZK% = 13 THEN 510
230  IF ZK% = 8 THEN T = T + T
240  IF ZK% = 21 THEN T = .001 + .5 * T
250  RETURN
260  & N: VTAB 4: PRINT "Watch the man run..."
270  VTAB 22: HTAB 8
280  PRINT "P@@RESS @J TO SLOW HIM DOWN"
290  PRINT "@@P@@RESS @K TO SPEED HIM UP
```

```
300  HTAB 1
310  PRINT "@@P@@RESS @@RETURN @@WHEN BOTH OF YOU ARE
     TIRED
320  HPLOT 1,116 TO 279,116: REM  ROAD
330  & X = 1: REM  INITIAL VALUE OF X
340  PRINT  CHR$ (15);"$": REM  REWRITE MODE
350  PRINT  CHR$ (15); CHR$ (17): REM   WRAP  AT RIGHT
      MARGIN
360  PRINT  CHR$ (15);"F": REM  ALT FONT
370  GOSUB 190
380  PRINT " @LAB@R CD"
390  GOSUB 190
400  PRINT " @LEF@R GH"
410  GOSUB 190
420  PRINT " @LIJ@R KL"
430  GOSUB 190
440  PRINT " @LMN@R OP"
450  GOSUB 190
460  PRINT " @LQR@R ST"
470  GOSUB 190
480  PRINT " @LUV@R WX"
490  GOSUB 190
500  GOTO 380
510  & N: VTAB 10: HTAB 8: PRINT "Press RETURN for the
      Index"
520  VTAB 12: HTAB 14: PRINT "CTRL B to repeat"
530  & P: IF  ZK% = 2 THEN 260
540  IF ZK% <  > 13 THEN 530
550  & N: VTAB 10: PRINT "One moment please..."
560  PRINT D$;"RUN INDEX"
```



```
10  REM : STATES
20  LOMEM:  PEEK (24588) + 256 *  PEEK (24589) + 132 +
     1028
30  DIM S$(50),C$(50),P(50)
40  GOSUB 1000: REM  LOAD ARRAYS
50 OK$ = "right":NO$ = "no"
90  REM :****** SOME PRINT STATEMENTS ILLUSTRATE HOW T
    O USE @, P, L, H TO CONTROL FORMAT AND CTRL C FOR
     12 DOT LINE FEED
100  & N: VTAB 4: PRINT "@PT@LHIS PROGRAM ILLUSTRATES
    ONE WAY TO PICKQUESTIONS RANDOMLY FROM A POOL OF
    ITEMSAND TO REPEAT ANY QUESTION NOT ANSWEREDCORRE
    CTLY WITHOUT HELP."
110  VTAB 12: PRINT "HILT ALSO SHOWS HOW TO USE WORDS
    IN A STRINGARRAY IN AN ANSWER COMMAND."
120  VTAB 22: HTAB 8: PRINT "@HP@LRESS@H RETURN@L TO T
    RY IT."
130  & P: IF ZK% = 2 THEN 600: REM  :********** CTRL B
    PRESSED?
140  IF ZK% <  > 13 THEN 130: REM  :********** RETURN
    PRESSED?
150  & N: VTAB 4: HTAB 14: PRINT "SPECIAL KEYS"
160  VTAB 8: HTAB 4: PRINT "ESC ESC - @LTOGGLES @HUPPE
    R@L LOWER CASE": VTAB 10: PRINT "@HCTRL A  - @LGI
    VES THE ANSWER": VTAB 12: PRINT "@HCTRL B  - @LTA
    KES YOU OUT"
170  VTAB 20: HTAB 8: PRINT "P@LRESS @HRETURN @LTO CON
    TINUE."
180  & P: IF ZK% <  > 13 THEN 180: REM  ******* RETURN
     PRESSED?
200 NP = 50: REM ******** SET NUMBER PROBLEMS
210  FOR I = 1 TO NP:P(I) = I: NEXT : REM  ***** SET U
    P ARRAY OF PROBLEM NUMBERS
220  IF ZC% = 1 THEN P(RN) = P(NP):NP = NP - 1: REM  *
    **** IF RIGHT 1 TRY REMOVE FROM ARRAY BY SUBSTIT
    UTING UNWORKED PROBLEM NUMBER.  SHORTEN ARRAY BY
    1
230  IF NP = 0 THEN 100: REM  ******  ALL DONE?
240 RN =  INT (NP * ( RND (1))) + 1: REM  ******PICK N
    EW PROBLEM NUMBER
300  & N: REM : ERASE SCREEN, INITIALLIZE
310  VTAB 4: PRINT "W@LHAT IS THE CAPITAL OF @@";S$(P(
    RN));"?"
```

```
320 AA$ = C$(P(RN)): REM  ******** GET ANS FROM ARRAY
330 ZL% = 25: REM  ****** SET MAX NUMBER CHARS
340 ZF$ = "@@": REM   ***** TOGGLE INPUT TO CAPS
350  HTAB 10: VTAB 10: & I: IF ZK% = 2 THEN 600: REM
     ******** PRESSED  CTRL B  TO EXIT ?
360  IF ZK% = 1 THEN ZC% = 2: GOTO 400: REM   : CTRL A

370  & A"": IF ZA% THEN ZC% = ZC% - 1: GOTO 350: REM
     ******* NO BLANK ANSWERS
380  & A"<C><V,AA$>": & M
390  IF ZA% THEN 460: REM  IF RIGHT GOTO 460
400 ZK% = 0: REM  ******* ZERO FOR KEY PRESS FLAG
410  VTAB 15: IF ZC% = 1 THEN  HTAB 8: PRINT "No, plea
     se try again.": REM  **** 1ST WRONG ANSWER COMMEN
     T
420  IF ZC% > 1 THEN  HTAB 14 -  LEN (AA$) / 2: PRINT
     "Try: ";AA$: REM  **** COMMENT IF MORE THAN 1 WRO
     NG ANSWER
430  IF ZK% THEN 350: REM  ****** HAS KEY BEEN PRESSED
     ?
440  & T: IF  NOT ZK% THEN 440: REM  ****** LOOK FOR K
     EY PRESS
450  GOTO 410: REM  ******* ERASE WRONG COMMENT AND PA
     SS KEYPRES BACK TO INPUT
460  VTAB 24: HTAB 6: PRINT 50 - NP + (ZC% = 1);" righ
     t        ";NP - (ZC% = 1);" to go"
470  & P: GOTO 220: REM  ******* PAUSE
600  & N: VTAB 10: HTAB 8: PRINT "Press RETURN for the
      index"
610  VTAB 12: HTAB 14: PRINT "CTRL B to repeat"
620  & P: IF ZK% = 2 THEN 100
630  IF ZK% <  > 13 THEN 620
640  & N: VTAB 10: PRINT "One moment please..."
650  PRINT  CHR$ (13); CHR$ (4);"RUN INDEX"
1000 S$(1) = "Alabama":S$(2) = "Alaska":S$(3) = "Arizo
     na":S$(4) = "Arkansas":S$(5) = "California":S$(6)
      = "Colorado":S$(7) = "Connecticut":S$(8) = "Dela
     ware":S$(9) = "Florida":S$(10) = "Georgia"
```



```
1010 S$(11) = "Hawaii":S$(12) = "Idaho":S$(13) = "Illi
     nois":S$(14) = "Indiana":S$(15) = "Iowa":S$(16) =
     "Kansas":S$(17) = "Kentucky":S$(18) = "Louisiana"
     :S$(19) = "Maine":S$(20) = "Maryland"
1020 S$(21) = "Massachusetts":S$(22) = "Michigan":S$(2
     3) = "Minnesota":S$(24) = "Mississippi":S$(25) =
     "Missouri":S$(26) = "Montana":S$(27) = "Nebraska"
     :S$(28) = "Nevada":S$(29) = "New Hampshire":S$(30
     ) = "New Jersey"
1030 S$(31) = "New Mexico":S$(32) = "New York":S$(33) =
     "North Carolina":S$(34) = "North Dakota":S$(35) =
     "Ohio":S$(36) = "Oklahoma":S$(37) = "Oregon":S$(3
     8) = "Pennsylvania":S$(39) = "Rhode Island":S$(40
     ) = "South Carolina"
1040 S$(41) = "South Dakota":S$(42) = "Tennessee":S$(4
     3) = "Texas":S$(44) = "Utah":S$(45) = "Vermont":S
     $(46) = "Virginia":S$(47) = "Washington":S$(48) =
     "West Virginia":S$(49) = "Wisconsin":S$(50) = "Wy
     oming"
1050 C$(1) = "Montgomery":C$(2) = "Juneau":C$(3) = "Ph
     oenix":C$(4) = "Little Rock":C$(5) = "Sacramento"
     :C$(6) = "Denver":C$(7) = "Hartford":C$(8) = "Dov
     er":C$(9) = "Tallahassee":C$(10) = "Atlanta"
1060 C$(11) = "Honolulu":C$(12) = "Boise":C$(13) = "Sp
     ringfield":C$(14) = "Indianapolis":C$(15) = "Des
     Moines":C$(16) = "Topeka":C$(17) = "Frankfort":C$
     (18) = "Baton Rouge":C$(19) = "Augusta":C$(20) =
     "Annapolis":
1070 C$(21) = "Boston":C$(22) = "Lansing":C$(23) = "St
      Paul":C$(24) = "Jackson":C$(25) = "Jefferson Cit
     y":C$(26) = "Helena":C$(27) = "Lincoln":C$(28) =
     "Carson City":C$(29) = "Concord":C$(30) = "Trento
     n"
1080 C$(31) = "Santa Fe":C$(32) = "Albany":C$(33) = "R
     aleigh":C$(34) = "Bismark":C$(35) = "Columbus":C$
     (36) = "Oklahoma City":C$(37) = "Salem":C$(38) =
     "Harrisburg":C$(39) = "Providence":C$(40) = "Colu
     mbia"
1090 C$(41) = "Pierre":C$(42) = "Nashville":C$(43) = "
     Austin":C$(44) = "Salt Lake City":C$(45) = "Montp
     elier":C$(46) = "Richmond":C$(47) = "Olympia":C$(
     48) = "Charleston":C$(49) = "Madison":C$(50) = "C
     heyenne": RETURN
```

Main descriptions of items are located on underlined pages.
Pages with letters (e.g. "B2") are in appendices.