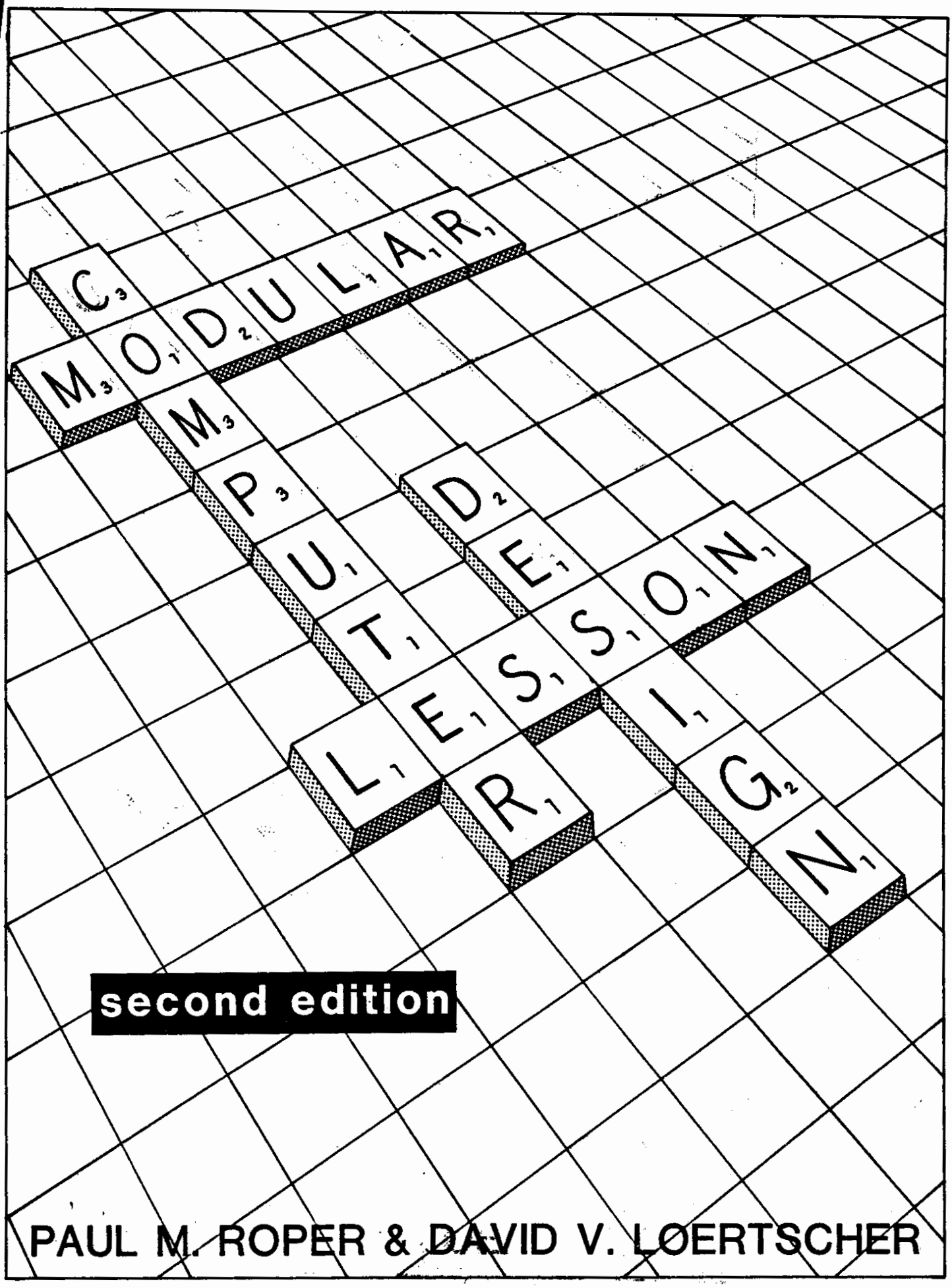


PAUL M. ROPER & DAVID V. LOERTSCHER

MODULAR COMPUTER LESSON DESIGN

APPLE VERSION

SECOND EDITION

by

Paul M. Roper

&

David V. Loertscher

Hi Willow Research and Publishing

1985

c. 1985 by Paul M. Roper and David V. Loertscher

Hi Willow Research and Publishing
P.O. Box 1801
Fayetteville, Arkansas 72702-1801

ISBN: 0-931510-12-0

TABLE OF CONTENTS

INTRODUCTION

Many educators, whether in formal, informal, or corporate education, are becoming computer literate and know the rudiments of programming. These persons know the functions of computer commands like LOAD, LIST, PRINT...etc. What they may not know is how to use these commands to create a lesson, i.e., they have the tools but do not know how to proceed systematically. The authors have seen a number of beginners try to write lessons. They struggle - not because they don't know how to get the computer to respond, but because they get bogged down in hundreds of line numbers and lose their place. They may have used flowcharting techniques but, like other programmers, desire a better way of structuring their programs.

This book provides a simple structure for a computerized lesson. It breaks a large task or lesson down into a number of pieces, each of which can be programmed or coded separately and then pieced together into a whole. It is something like putting a puzzle or a patchwork quilt together.

Chapter one discusses the prerequisites to writing quality computer lessons. Chapter two teaches the modular technique with a detailed example to follow. Chapter three provides a number of techniques that can be used in lessons to make them more professional. Chapter four provides a number of programming tips not readily available in the literature. Chapter five provides useful graphic tools and chapter six introduces Apple Writer and Bank Street Writer to the novice.

There are a number of commercially available authoring programs such as Genius, Super Apple Pilot, Blocks, etc. All these have their strong points. They also have limitations. This book presents an alternative to those

authoring systems and provides the creative teacher and programmer with another way of building lessons for the computer using BASIC. Although the text has been written with Applesoft BASIC in mind, the technique taught here is useful no matter what computer or computer language is employed.

This book will not make you a computer programmer nor does it stress the internal functioning of the computer, but if you already know the rudiments of programming, it will help you improve your methods. It is very easy to spend countless hours at the computer terminal writing code and debugging it. If you will follow the suggestions presented here for structuring your program, precious hours can be saved.

The book has been authored by a systems analyst and an educator. While the systems analyst has been interested in the finer points of the computer's capacitites, the educator has been interested in translating computerese into the practical world of education. The educator in the field who can use this book and understand the structure which it recommends for computer programs is certainly prepared to work with a professional programmer in the creation of commercial quality educational programs. Both the educator and the computer programmer must have some common ground - some basis of communication. Hopefully, the educator who uses this book will be drawn toward the world of computer programming and structure. The programmer who understands the techniques presented here should be drawn toward the needs in education.

The second edition of the work has been expanded and corrected. An addition was made to the modular design process and chapters were added on graphics and word processing. In addition, many new hints and tips have been added. The program Apple Demo is available from the publisher on disk for $5.00.

CHAPTER ONE

THE EDUCATIONAL COMPUTER LESSON

Every time a new technology is created that has potential for educational use, there is a great deal of stress because there is usually a lack of software to accompany the new medium. Usually, very few corporations have extensive staffs working on new software for a new medium, and often most of the developmental work is handled with government grants. Such is the state of the art with computer technology. There is a wealth of hardware on the market but software is still scarce in many subject areas.

Using computers as a teaching technology is not really new but using microcomputers to teach is new. Today, there is a concerted effort to create software for students of all ages and for all curricula. Commercial development seems agonizingly slow, and often a commercial product will not meet a local need. At some period in time, there should be an ample amount of useful computer material available commercially, but at this point, there is not.

Should the local teacher create computer programs for students when these materials are not available commercially? Many teachers have learned to create their own transparencies, slides, tape recordings, games, and other media with varying success. Should they attempt computerized lesson creation? The answer is neither a straight yes nor no.

A number of authors suggest that teachers will never have the time or the talent needed to create effective and qualitative computer course materials. While this generalization may be true for the teaching force as a whole, there are a number of factors to consider which provide encouragement for teachers who

wish to create their own computer media.

Teachers who are knowledgeable of computer programming and operation are in a better position to critique the commercial products created for their classrooms. They are better critics because they have some notion of the potential of the computer, and of programming principles that differentiate between good programming and bad. They know the graphic and computational capabilities of the machines and whether these capabilities are being utilized by a particular program. It is in this sense that knowledge of the technology would contribute to the proper use and selection of computer materials in teaching.

Teachers who have some knowledge of computer programming and operation can also take advantage of programs which can be altered for particular applications. For example, there are a number of programs available which allow the teacher to insert daily spelling lists, terms and definitions, formulas, or problems. If a teacher understands programming, this type of lesson can fit a local need directly designed by the creator.

There are a number of local networks developing in which members share computer programs. For teachers in the network who have programming skills, a number of free programs can be acquired, modified, and used to advantage. Besides the trading partnerships, numerous magazines display programs which can be typed into the computer for use. These can also be modified and adjusted for local use by the teacher.

Another possibility for teachers is to collect bits and pieces of programs which have useful functions and store them on a diskette often called a utility disk. These short programs can then be called up and combined with other bits and pieces to form a useful larger program. For example, an alphabetical s ort

program, some graphic tricks, short games that can be inserted in another program as a reward, a program to provide varying positive reinforcements, and many other sort programs can be collected. If the teacher has such a bank of programs available, then larger programs that fit local needs can be constructed without too much time and effort. In fact, students may be involved in the programming effort to produce their own useful lessons.

This book provides a number of techniques which can be used by the teacher as parts of computerized lessons. But more importantly, the book provides a method of linking parts of a program or modules together into an organized whole. This technique provides structure. It brings a manageable dimension to programming.

But before programming can begin, there are other prerequisites which the teacher must bring to the task of creating a computerized lesson.

The role of experience.

Perhaps the best preparation for creating computerized lessons for a specific group of students is to have experience teaching that group. An elementary school teacher who knows how children think, how they respond to instruction, and how they are motivated is prepared to plan a computerized lesson. Some of the best commercial products to date have been created by teachers who have teamed with programmers to create software. Much of the software available from MECC (The Minnesota Educational Computing Consortium) has been produced in this manner.

The MECC staff has worked two ways. First, a teacher may submit an idea to the MECC staff which assigns a programmer to translate that concept into a computerized lesson. Secondly, the teacher may submit an already-programmed

lesson which the MECC staff "cleans up" and prepares for state and national distribution. Both systems have worked very well in producing a variety of quality software for many different curricular areas.

There are some fundamental questions that experienced teachers automatically ask prior to constructing any lesson. These questions must be asked in the same detail as computerized lessons are designed.
The questions include:

1. What types of topics and subject areas could be computerized?
2. What topics will fit into the curriculum at a certain grade level?
3. What is the normal reading level of the students who will use this lesson?
4. What types of topics interest students?
5. How should a lesson be structured for a student at a certain ability level?
6. What types of concepts can the students understand?
7. Do I know enough about the subject of the lesson and have enough reference materials to provide sound content?
8. What types of unmotivated or mischievous behavior should be anticipated in structuring a lesson for the student?
9. How long is the attention span of the student?

If the above questions cannot be answered automatically, then extra preparation time will be required. A good technique is to test a program idea with several teachers before you begin the design process.

Using an instructional design model.

There are many models which have been developed to provide a systematic method of designing instructional modules. A number of books dealing with instructional design are listed at the end of this chapter. While instructional design models differ, there are a few elements common to the process that will be reviewed here. These important steps include:

1. Analyze the audience. In this step, a thorough analysis of the types of students who will be using the lesson is performed. Who are they? What are their backgrounds? What are their abilities?
2. Create measurable objectives. Decide exactly what students will be required to learn. Write the objectives specifically enough so that you will know when students have mastered the intended content.
3. Define what content will be covered. Know what segment of a topic can be comfortably covered in a computerized lesson. How much will be too much? Too little?
4. Know what media will be used. Will the concept be taught totally by computer? What other media will be used? Will the computerized lesson be a foundation for the full lesson or will it be supplemental?
5. Create the computerized lesson. Use the modular technique described in this book. Be sure to create a set of clear instructions (documentation) for the teacher and the student.

6. Pilot test the lesson. Try out the computerized lesson on the students or classes for which it was designed. Is the lesson as successful as you had hoped? What changes need to be made? Is the documentation adequate?
7. Use the lesson. Put the lesson into use by the group for which it was intended.
8. Evaluate the result.

Does the program live up to your expectations?

Revision and updating may be needed from time to time.

Knowing Computer Characteristics.

Until the Apple computer was developed in 1978, using computerized lessons was very expensive. Now there are many microcomputers available whose costs are not prohibitive. These computers all have differing characteristics which must be considered by the instructional designer. Many commercial companies are creating numerous versions of their lessons so they can be run on the various brands of microcomputers.

Should you write a lesson with multiple versions in mind? The answer to that question will depend on your intentions. If you plan to market your software widely, then you would be wise to know the similarities and differences among the various machines. You can design with conversion to other programming languages in mind. For example, if color graphics are essential to the understanding of a concept, you will not be able to have a TRS80 Model III version available, but you could have a TRS80 color computer version. If the graphics are simple line drawings, then they can be converted rather easily for the various computers, but complex graphics make conversions much more difficult.

But should we always design for the lowest common denominator characteristics of the computer? If you avoid some of the capabilities of your machine that other computers don't have, you will not be using the full teaching potential of your computer. This can make your lessons much less powerful than they are capable of being.

What are the characteristics of computers which are valuable in education? Which of these characteristics are common to other media and which are specific only to computers? Computers have a number of characteristics which are common to other media:

1. Printed information may be presented (book, computers, filmstrips).
2. Color graphics (pictures and animation) or line drawings may be used (film media, computers).
3. Some sound is available (audio media, computers, most film media).
4. Text and graphics can be mixed in a presentation (most film media, computers).
5. Linear lesson design, i.e., all students progress through the lesson in the same sequence (textbooks, all film media, computers).

Computers also have characteristics which are unique. These include:

1. Immediate and tireless feedback (computers can be interactive).

2. The potential to analyze student responses at any point in the lesson and tailor instruction accordingly (branching).
3. The power to solve complex equations or formulas instantly, making simulation of the real world possible.
4. The power to store student responses instantly for access by the instructor at any time.
5. The power to generate problems on demand and at varying degrees of difficulty - as few or as many as needed.
6. The power to sort through material, text, bibliographies at a fantastic rate of speed for desired items.
7. The unique potential to teach logical thinking through debugging skills.

It is not wise to try to design a computerized lesson that can be taught better in another medium and in a different way. Using the computer to instruct just because it is available or is the fashionable way to be teaching, is no reason to design for it. A better plan would be to choose computer instruction when at least one of its characteristics would be beneficial in teaching a given concept If, for example, a student would benefit from numerous math problems being presented with immediate feedback (and with more patience and long suffering than a human is usually capable of delivering), then a computer becomes a proper medium for consideration.

It therefore becomes a matter of careful choice of what to computerize as well as devising a plan to do it well. Often, a task may be done more interestingly in another medium. If that is the case, use other media. There are plenty of lessons which would benefit from computer characteristics without wasting time programming lessons that are better taught by other methods.

BIBLIOGRAPHY

1. Bruce, Phillip, and Pederson, Sam M., The Software Development Project. New York: John Wiley, 1982.

2. Designing Instructional Computing Materials. Minneapolis, Minn.: MECC, 1981.

3. Dick, Walter, and Carey, Lou, The Systematic Design of Instruction. Glenview, Ill.: Scott Foreseman, 1978.

4. Gagne, Robert M., and Briggs, Leslie J., Principles of Instructional Design. New York: Holt, 1979.

5. Peters, Harold J., and Johnson, James W., Author's Guide. Iowa City, Iowa: Conduit, 1978.

6. Russell, James D., The Audio-Tutorial System. Englewood Cliffs, N.J.: Educational Technology Publications, 1978.

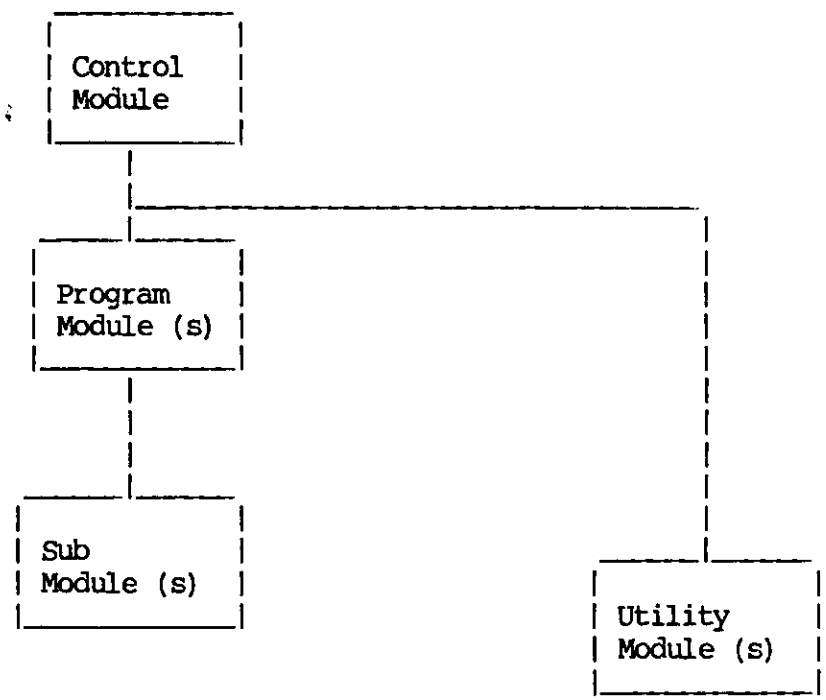
7. Writing Support Materials for Instructional Computer Programs. Minneapolis, Minn.: MECC, 1981.

CHAPTER TWO

MODULAR COMPUTER LESSON DESIGN

Modular computer lesson design is a systematic way of creating computerized lessons. Its concept is to divide a programming task down into small segments which can be programmed independently and then pieced together to create an educational lesson. It is similar to the cut and paste technique in graphic arts where bits and pieces of this and that are combined to create a pleasing handout, poster, etc. Each piece (module) of the computerized lesson can stand alone, is programmed separately, and will run independently of the other modules.

Visually, a computerized lesson might have the following modules:



The components of each of the modules in the model might include:

1. Control module:
   a. Document (explain) what the program does.
   b. Open any files needed.
   c. Initialize any variables used.
   d. Dimension any arrays used.
   e. Present the main menu (if one is used).
   f. Terminate the program with an END statement.

2. Program modules(s) (are subroutines).
   a. Present actual lesson content.
   b. Text and graphics used only once in the program go here. Text or graphics that are used over and over should be in a utility module to be called as needed.

3. Sub-modules (none, one, or more) (are subroutines).
   a. If a program module is too large to be easily maintained or is found to be performing many unrelated tasks, it can be broken down into smaller segments.

4. Utility modules(s) (are subroutines).
   a. Present title graphic or other introductory material.
   b. Contain any graphics called more than once in the lesson.
   c. Store any sound routines called more than once.
   d. Sort data.

e. Provide time delays.

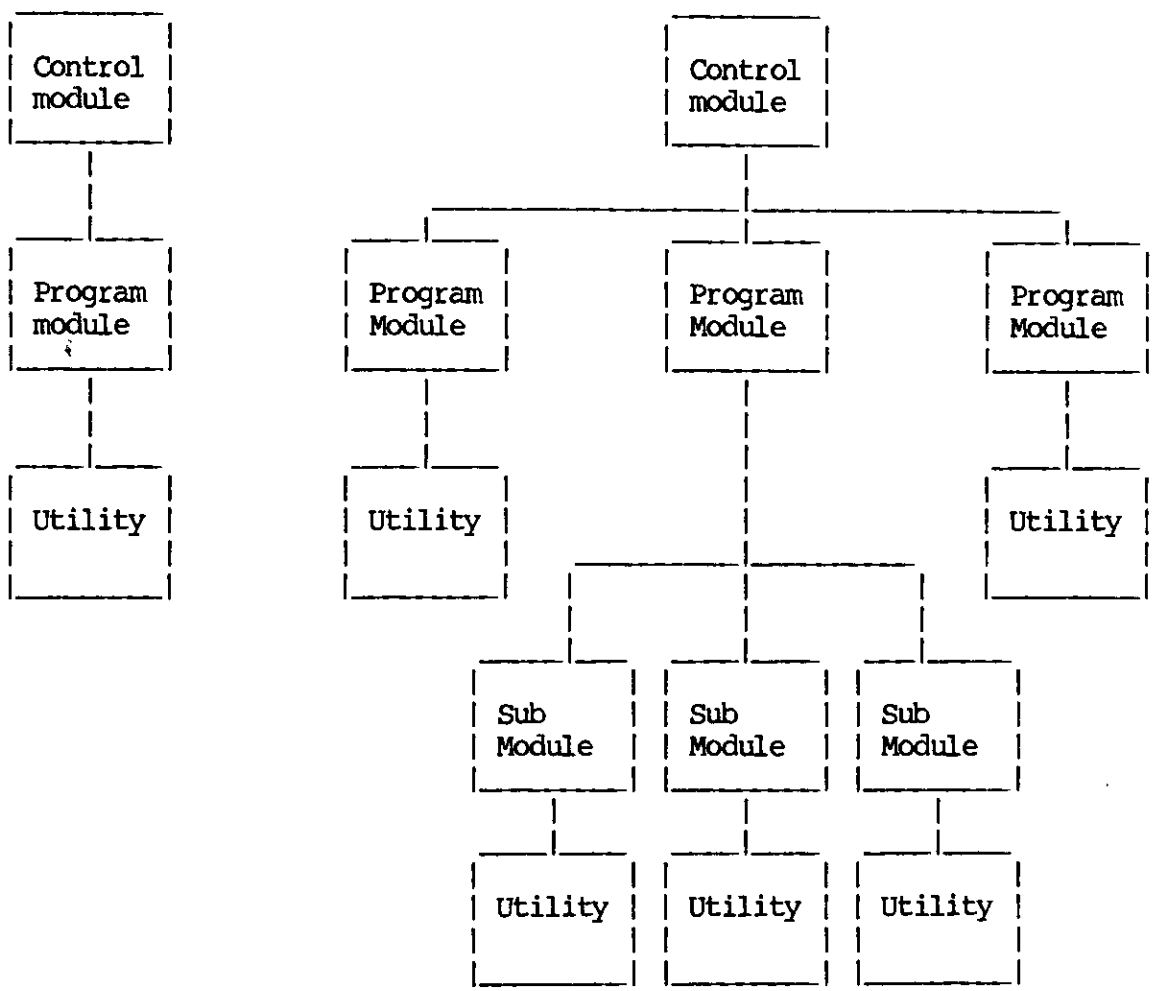
f. Control forward or backward paging instructions.

g. Contain keyboard input controls.

h. Allow error trapping.

i. Contain any other utilities used.

The modularized picture of a lesson can be simple or complex depending on the length and detail of the lesson. For example, a short lesson might only have one program module like the drawing on the left or it might have numerous modules like the drawing on the right.



GETTING STARTED

It should be clear from the preceeding discussion, that the design and programming of a computerized lesson is accomplished in chunks or modules which are independent entities that can then be joined together in a single program. Knowing what the overall design will look like, the task can begin.

STEP ONE

The first step of modular computer lesson design is to define the lesson problem. Here, the steps of instructional design discussed in chapter one should be taken into account. This includes an analysis of the intended audience, the objectives of the lesson, the content to be covered, and the strategy that will be employed. The computerized lesson can be independent of other learning materials or it can be one component in a multi-media unit of instruction.

Sample Lesson Problem Development

Title of sample lesson: Apple Demo

Audience: a student who has mastered the fundamental commands of a programming language and is ready to use those skills to write computerized tutorials.

Objectives:

1. The student will be able to use the sample lesson as a model to follow in the construction of a computerized lesson.
2. The program used in the example will be simple enough that students will be able to follow through the various modules without becoming confused.

3. Enough programming techniques will be demonstrated in the sample lesson so that students can copy, select, add to, and delete ideas as they program their own lessons.
4. A secondary objective is to create a lesson which demonstrates some of the features of the Apple computer.

STEP TWO

The next step in modular design is to break down the main problem into smaller problems. Each of the features of the desired program should be listed. The features are then studied and prioritized according to any constraints that might be present. Constraints might include time, money, personnel and programming expertise. These constraints may limit the length, the content, and the complexity of the lesson. In our sample lesson, we have listed all of the features that we originally wanted in our program and have starred those features that we actually programmed. Our main constraint was time. We also realized that the program would be too long if it contained everything we wanted. We finally decided that we had been too ambitious in our original plan.

Sample Lesson Detailed Features

Progamming features to demonstrate:

1. use of a menu*
2. use of subroutines*
3. documentation within a program*
4. control of input from the keyboard*
5. control of program flow*
6. communication with the user*
7. testing responses from the user*
8. handling errors*
9. management of computer lessons

Apple features to demonstrate in the lesson:

1. computers can count*
2. computers can compute and compare*
3. computers can do graphics*
4. computers can create sound
5. tutorial type computer lessons*
6. simulation type computer lessons
7. gaming type computer lessons
8. drill type computer lessons*

*features chosen for final product

STEP THREE

The next step is to create a VTOC (visual table of contents) of the lesson features. This will be comparable to creating a table of contents for a book which will list module titles (chapter titles) and will give possible beginning program line numbers of each module (page numbers for the chapters). The beginning numbers should be added only if they can be easily forecast in advance. Sub-parts of modules (parts of a book chapter) are drawn underneath main modules. Any subroutines referenced or "called" by a module are also listed under that module.

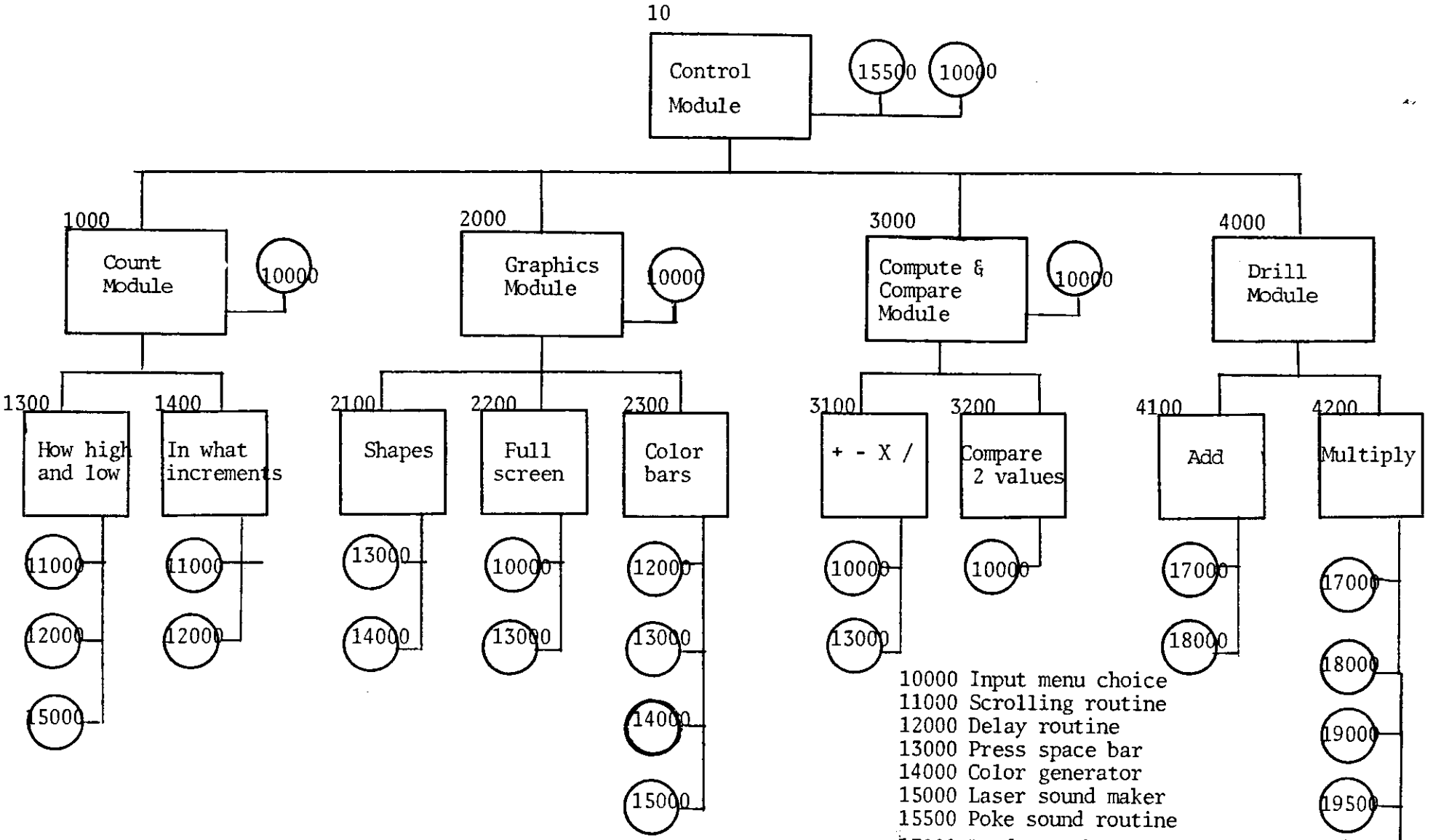
The VTOC presents the programmer with a visual map of the program flow. This map is much easier to follow than a flowchart due to its simplicity. The detail of a flowchart, with all of its program logic, is often a maze of instructions. By contrast, the VTOC presents a clear picture of the program as a whole rather than detailed code.

Using the VTOC as a guide, pieces or parts of the program can be created and debugged separately before the whole program is put together. Such an approach is much less tiring than trying to develop the program logic for the entire program. Program bugs (errors) are easier to find in small programs than in large programs. Each module or piece can be tested and modified before it becomes a part of the whole. Programming is always begun after the VTOC has been created - never the other way around. It is much too easy to get lost in pages of code. The VTOC will provide an index to the entire project.

The VTOC for the sample program follows (see p. 2-8). Note that we selected our main modules from the features that we had outlined. Some of our featues were to be exhibited in the programming style itself, so they did not become a part of the VTOC. We estimated that our line numbers for each main module would start at 1000,2000...etc. As it turned out, we could have used a little more space in some of the modules. Perhaps 1000,3000,5000, etc. would have been a better choice. This can easily be changed as the programing progresses, but we have left our program in its original form to demonstrate this and other problems that can be encountered. When we first drew our VTOC, we did not know all of the utility modules that we would be using. We decided to start all the utilities at 10,000 and keep the start of each new utility at an interval of 1000 or 500 or some other easily remembered number. We kept our VTOC handy at all times with its list of utilities so that we could incorporate the utility as a subroutine into any module we were building.

Be sure that the VTOC lists every subroutine that is called by the program. At the planning stage, the programmer can't forecast every detail of the program so the VTOC may only include the major sections.

As thinking and planning progress, the VTOC will become more detailed. Ideally, however, you should make the VTOC as complete as possible before you begin programming. At some point, the VTOC will be a complete table of contents to the program, like our example.



10000 Input menu choice
11000 Scrolling routine
12000 Delay routine
13000 Press space bar
14000 Color generator
15000 Laser sound maker
15500 Poke sound routine
17000 Random number generator
18000 Generic problem printer
19000 Get math response
19500 Addition problems
19800 Multiplication problems

V T O C (VISUAL TABLE OF CONTENTS)

STEP FOUR

After you have created the VTOC, you are ready to program each of the modules as a separate piece and then join them together to form an entire program. Many programmers do very detailed planning of each module by using a technique called flow charting. We recommend that before you try to flowchart a module, you should do some intermediate planning. If you were writing a research paper, you would outline a section of the paper in a fairly detailed manner so that as you write, your thoughts would be organized. You may also outline fairly detailed steps that are needed to create a program module.

Here is a sample outline of the control module for the Apple Demo program:

Control Module

1. Program title in REM statements.
2. Describe the program in REM statements.
3. Initialize the question and answer variables for the drill module.
4. Present the main menu.
5. Accept student input and check for out-of-range responses. If out of range, reprint the menu.
6. GOSUB to the appropriate student choice or end if the student requests it.
7. End the program.

As in an outline for a research paper, your outline may grow and develop as you actually write the paper. Our item three above, for example, might not be added until we actually begin to program the drill module. At that point, we would realize that we need to initialize our variables and so decide that the best place to do this is back in the control module. We add that feature to the control module (Thank heavens we left space between our line numbers).

After your outline, you are now ready to flow chart a piece of the module if that process will help you. Flow charting will not always be necessary as you gain experice in programming, but it is rarely a waste of time.

STEP FIVE

The actual programming task is now ready to begin. Each module on the VTOC should be programmed or coded separately. The programmer begins at the top (the control module) and continues in order until the last module is completed. This is called top-down programming. Within each module, top-down programming is done. This means that there should be a minimum of jumping around through the module (few GOTO commands). Top-down programming makes it easy to follow through the code in order to find errors and follow the logic.

Each module is an independent piece of the program and should be written and tested before going on to the next module. This can be done by just typing in RUN starting line # of the module or GOSUB starting line # of the subroutine.

Often, programs contain a menu in the control module (example: Which lesson would you like? 1. Addition; 2. Subtraction; 3. Multiplication; 4. Division; 5. Quit). In this case, get the menu working first, then as each of the modules is created, it is tested and then hooked into the program so that the menu calls it.

It is important to find and clean up problems early in the program rather than having them stack up. Solving one problem early will often eliminate later problems. Psychologically, this helps create many small successes rather than building a mountainous number of problems to tackle all at one time. One nice thing about programming in modules is that several persons can be assigned to

write various modules of the program. Also, some of the modules may already be available from other programs that have been written or can be borrowed from other programmers. The idea here is to keep a library of utilities and other useful programs that can be pulled into any lesson being written. This could be called a cut and paste methodology - get the modules you need anywhere you can get them and put them together.

The Apple System Master diskette has a very useful utility program entitled RENUMBER. Using this program, it is very easy to merge one sequence of line numbers into another. If the two programs to be merged have duplicate line numbers, the RENUMBER program can supply a new sequence of line numbers so that the two programs can merge easily.

If you have a library of useful programs or ultilities handy, it is very important that each of these have descriptive titles as well as clear descriptions of exactly what they do. A good catalog of what you have is very helpful. This can even be a file on the diskette that simply contains a list of all your subroutines.

Be sure to keep the VTOC - even after the program is written. It will always be useful as documentation for that program.

In the following pages, the actual program "Apple Demo" is printed out. The program is listed on the left of the page and comments have been added in the right hand column to help the reader follow the logic.

```
1   REM ****************
2   REM *CONTROL MODULE*
3   REM ****************
5   REM  APPLE DEMO
6   REM  BY MIKE ROPER
7   REM  C. 1982
8   REM
9   REM
10  D$ =  CHR$ (4): PRINT D$;"OPEN TEXT DEMO": PRINT D$;"WRITE TEXT DEMO": POKE
    33,30: LIST : PRINT D$;"CLOSE TEXT DEMO": TEXT : END
11  REM  THIS PROGRAM HAS TWO
12  REM  PURPOSES:
13  REM
14  REM  THE FIRST IS TO
15  REM  SHOW HOW A PROGRAM
16  REM  IS PUT IN MODULAR FORM.
17  REM
18  REM  THE SECOND IS TO
19  REM  SHOW SOME OF THE
20  REM  FEATURES OF THE APPLE
21  REM  COMPUTER
22  REM
23  REM
46  REM
50  BEEP$ =  CHR$ (7): REM  STORE BEEP
100  GOSUB 15500: REM  POKE SOUND
110  FOR I = 0 TO 3: READ ADD$(I): NEXT I
120  FOR I = 0 TO 3: READ AANS$(I): NEXT I
130  FOR I = 0 TO 3: READ MULT$(I): NEXT I
135  FOR I = 0 TO 3: READ MANS$(I): NEXT I
325  REM
490  REM  ***************
500  REM  ** MAIN MENU **
505  REM  ***************
510  REM
550  HOME : VTAB 8: HTAB 5
560  PRINT "---- MAIN MENU ----"
570  PRINT : PRINT "   1. I CAN COUNT"
580  PRINT : PRINT "   2. I CAN PRODUCE GRAPHICS"
590  PRINT : PRINT "   3. I CAN COMPUTE AND COMPARE"
600  PRINT : PRINT "   4. I CAN DRILL"
610  PRINT : PRINT "   5. END"
630  GOSUB 10000: REM  GET ANSWER
640  ON ANS GOSUB 1000,2000,3000,4000
650  IF ANS <  > 5 THEN 550
660  END
```

The control module begins.

Line 110-135 initializes question and answer variables for the drill module.

Main menu is presented.

The END statement comes at the end of the control module.

Note that remark statements placed on line 100 and 630 clarify what will happen at that point in the program; this is the subroutine function description. The VTOC also provides this information.

Note that the menu allows the student to get out of the program. This is a very important control characteristic that should be included in almost all computerized lessons.

```
989  REM
990  REM  ******************
992  REM  *  COUNT MODULE  *
996  REM  ******************
997  REM
1000  TIME = 20
1050  REM
1100  HOME : VTAB 10
1200  PRINT "-----   MENU  -----"
1210  PRINT : PRINT
1220  PRINT : PRINT "1. HOW HIGH AND LOW"
1230  PRINT : PRINT "2. IN WHAT INCREMENTS"
1240  PRINT : PRINT "3. RETURN TO MAIN MENU"
1250  GOSUB 10000
1255  IF ANS = 3 THEN  RETURN
1260  ON ANS GOSUB 1300,1400
1270  GOTO 1100
1275  REM
1290  REM  ***************************
1292  REM  *HOW HIGH & LOW SUB MODULE*
1294  REM  ***************************
1295  REM
1300  HOME : VTAB 10
1310  PRINT "I CAN COUNT FROM :": PRINT : PRINT
1315  PRINT "ONE MILLION BELOW ZERO": PRINT
1318  PRINT "TO ONE MILLION ABOVE ZERO": PRINT
1320  PRINT : PRINT "WOULD YOU LIKE ME TO COUNT FOR YOU?"
1321  PRINT : INPUT "(YES OR NO)";ANS$
1330  IF  LEFT$ (ANS$,1) = "Y" THEN 1380
1340  HOME : GR : TEXT : GOSUB 15000
1341  FOR I = 1 TO 50: NEXT I
1345  HOME : VTAB 10: HTAB 5
1346  PRINT "OK, BUT I REALLY DID WANT TO...."
1350  GOSUB 11000: HOME : GOTO 1100
1380  HOME : VTAB 5: PRINT "SO YOU WON'T BE HERE ALL DAY WAITING"
1381  PRINT : PRINT "FOR ME TO COUNT, KEEP THE NUMBERS SMALL"
1382  PRINT : PRINT "FOR EXAMPLE, FROM 1000 TO 5000.
1383  PRINT : PRINT
1385  INPUT "WHERE SHOULD I START COUNTING?   ";BEG
1386  PRINT : PRINT
1388  INPUT "WHERE SHOULD I STOP COUNTING?    ";QUIT
1390  FOR I = BEG TO QUIT
1392  ONERR  GOTO 16000
1393  GOSUB 12000
1395  PRINT I;" ";
1397  NEXT I
1398  GOSUB 11000: HOME : GOTO 1100
```

Note that although the REM line numbers appear in the listed program, they are never actually "seen" by the control module. REM lines are "between" two modules. This reduces execution time.

Note that a sub-menu has been created and that an exit to the main menu has been provided.

Notice again how the REM statements are never "seen" by the program as it runs.

Line 1330 checks to see if the leftmost character in ANS$ is a "y". This means that the student can mistype the "yes" so long as the first letter is a "y".

The ON command in line 1260 is a very easy way to perform various sub-modules without having to use three IF statements.

```
1399  REM  **********************
1400  REM  * INCREMENT S-MODULE *
1405  REM  **********************
1406  REM
1410  HOME : VTAB 5
1415  PRINT "I CAN COUNT BY ANY INCREMENT YOU LIKE.": PRINT
1420  PRINT "I CAN COUNT FORWARD OR BACKWARD"
1430  PRINT : PRINT "(TO MAKE ME PRINT BACKWARD, ENTER"
1431  PRINT : PRINT "A NEGATIVE NUMBER FOR THE INCREMENT)"
1435  VTAB 15: INPUT "BY WHAT INCREMENT SHOULD I COUNT ?";J: PRINT
1440  PRINT : INPUT "WHERE SHOULD I START COUNTING ?";BEG: PRINT
1450  PRINT : INPUT "WHERE SHOULD I STOP COUNTING ?";QUIT
1465  HOME
1470  FOR I = BEG TO QUIT STEP J
1475  ONERR  GOTO 16000
1480  PRINT I;" ";
1485  GOSUB 12000
1490  NEXT I
1495  GOSUB 11000
1500  RETURN
1600  REM
1990  REM  ********************
2000  REM  * GRAPHICS MODULE *
2005  REM  ********************
2010  REM
2050  TEXT : HOME : VTAB 10: HTAB 10
2055  REM
2060  PRINT "--I CAN PRODUCE GRAPHICS--"
2070  PRINT : PRINT "  1. SHAPES"
2075  PRINT : PRINT "  2. FULL SCREEN ONE COLOR"
2080  PRINT : PRINT "  3. BARS OF DIFFERENT COLOR"
2090  PRINT : PRINT "  4. RETURN TO MAIN MENU"
2091  REM
2092  GOSUB 10000
2093  IF ANS = 4 THEN  RETURN
2094  ON ANS GOSUB 2100,2200,2300
2095  REM
```

The VTAB and HTAB commands help position the menus and the text on the screen.

The variables are named so that they are indicative of their function.

Sometimes. REM is used just to create a space in the programming itself to set off sections of the program.

```
2098  REM  **********************
2100  REM  * SHAPES SUB-MODULE *
2101  REM  **********************
2102  FOR N = 1 TO 5
2103  HOME : GR
2105  GOSUB 14000
2106  REM          TRIANGLE
2107  I = 29:J = 29
2109  FOR K = 4 TO 11
2111  HLIN I,J AT K
2113  I = I - 1:J = J + 1
2115  NEXT K
2118  GOSUB 14000
2119  REM       UPPER LEFT FIGURE
2120  FOR I = 1 TO 10
2125  HLIN 1,5 AT I
2130  NEXT I
2135  FOR I = 1 TO 10
2140  VLIN 1,5 AT I
2145  NEXT I
2147  GOSUB 14000
2148  REM   SMALL SQ. BOX AT TOP
2150  FOR I = 13 TO 18
2155  VLIN 0,8 AT I
2160  NEXT I
2165  REM
2170  GOSUB 14000
2171  REM      BOTTOM REC. BOX
2175  FOR I = 10 TO 20
2180  HLIN 10,20 AT I
2185  NEXT I
2187  GOSUB 14000
2188  REM      BOTTOM LONG BAR
2190  FOR I = 21 TO 23
2195  HLIN 0,39 AT I
2196  NEXT I
2197  GOSUB 13000: NEXT N
2198  TEXT : RETURN
```

Note that variables are used to set the drawing positions of the horizontal and vertical lines. This cuts down on the number of programming lines that must be written and entered into the computer. Execution time is also decreased by reducing the number of lines.

This part of the program is a good example of a nested loop. The outer loop N (lines 2102 and 2197) dictate that the entire drawing will be repeated five times. The inner loops actually draw the figures.

Occasionally one of the shapes will disappear on the screen. This is caused by the computer selecting the color black.

```
2199  REM  ********************
2200  REM  * FULL SCREEN COLOR *
2210  REM  ********************
2215  REM
2250  HOME : VTAB 3
2255  PRINT "HERE ARE THE COLORS :": PRINT : PRINT
2260  PRINT "1. MAGENTA","8. BROWN": PRINT
2265  PRINT "2. DARK BLUE","9. ORANGE": PRINT
2270  PRINT "3. PURPLE","10 GRAY": PRINT
2275  PRINT "4. DARK GREEN","11. PINK": PRINT
2280  PRINT "5. GRAY 1","12. LIGHT GREEN": PRINT
2285  PRINT "6. MED. BLUE","13. YELLOW": PRINT
2290  PRINT "7. LIGHT BLUE","14. AQUA": PRINT
2291  PRINT "","15. WHITE"
2292  GOSUB 10000
2293  IF ANS <  = 0 OR ANS > 15 THEN  PRINT BEEP$;: GOTO 2292
2294  HOME : GR : COLOR= ANS
2296  FOR I = 0 TO 39: FOR J = 0 TO 39
2297  PLOT I.J
2298  NEXT J,I: GOSUB 13000: RETURN
2299  REM
2300  REM  **************
2301  REM  * COLOR BARS *
2302  REM  * SUB-MODULE *
2303  REM  **************
2310  REM
2340  TIME = 50
2350  GR : HOME
2355  FOR J = 1 TO 3
2356  GOSUB 15000
2360  FOR I = 0 TO 39
2370  GOSUB 14000: REM  RND COLOR
2380  VLIN 0,39 AT I
2385  NEXT I
2390  GOSUB 12000: NEXT J
2392  GOSUB 13000
2395  RETURN
```

Line 2293 checks to see that the number is 1-15. (Recall that we stored the ASCII representation of a beep in BEEP$ back in the control module). The semicolon keeps the cursor on the same line when subroutine 10000 is called again from line 2292.

Line 2340 sets the variable TIME that will be used by subroutine 12000.

The error trapping that is done on line 2293 is a very important part of any program. If the student pushes any other key than that asked for, the computer knows it and responds with some correctional instructions. Sometimes, the question will be repeated. Other times, the student might be reprimanded. By controlling the loop, counters could be established that limit the number of incorrect entries, or, specific error messages could be printed after certain sequences of attempts.

```
2500  REM
2995  REM  *********************
3000  REM  * COMPUTE & COMPARE *
3005  REM  *       MODULE      *
3008  REM  *********************
3009  REM
3010  HOME : VTAB 10
3015  PRINT : PRINT "I CAN COMPUTE AND COMPARE...."
3020  VTAB 14: PRINT "1. ADD,SUB,MULT,DIV"
3025  PRINT : PRINT "2. COMPARE TWO VALUES"
3030  PRINT : PRINT "3. RETURN TO MAIN MENU"
3040  GOSUB 10000
3042  IF ANS = 3 THEN  RETURN
3043  ON ANS GOSUB 3100,3200
3045  GOTO 3010
3048  REM
3050  REM  ********************
3100  REM  * ADD SUB MULT DIV *
3105  REM  *    SUB MODULE    *
3108  REM  ********************
3110  REM
3130  HOME : VTAB 5
3135  PRINT "I CAN PERFORM THE FOLLOWING :"
3138  VTAB 8
3140  PRINT : PRINT "1. ADD NUMBERS"
3142  PRINT : PRINT "2. SUBTRACT NUMBERS"
3145  PRINT : PRINT "3. MULTIPLY NUMBERS"
3147  PRINT : PRINT "4. DIVIDE NUMBERS"
3148  PRINT : PRINT "5. CHOOSE ANOTHER SUBJECT"
3150  GOSUB 10000
3151  IF ANS = 5 THEN  RETURN
3152  IF ANS > 3 THEN 3172
3155  HOME : PRINT "HOW MANY NUMBERS DO YOU WANT TO ENTER"
3158  PRINT : INPUT "(MUST BE 10 OR LESS )";NN
3161  IF NN > 10 THEN 3155
```

Note that line 3161 sends control back to line 3155 if the input is greater than 10.

ADD SUB MULT DIV SUB MODULE, Cont.

```
3162  FOR J = 1 TO NN
3163  PRINT "ENTER NUMBER ";J
3164  INPUT NUM(J): NEXT J
3165  IF ANS = 5 THEN  RETURN
3166  ON ANS GOSUB 3168,3173,3180,3185,3198
3167  GOTO 3130
3168  HOME : VTAB 5: FOR J = 1 TO NN
3169  SUM = SUM + NUM(J): PRINT  SPC( 10);NUM(J)
3170  NEXT J
3171  PRINT  SPC( 8);"+": PRINT  SPC( 8);"--------"
3172  PRINT  SPC( 10);SUM: GOSUB 13000: RETURN
3173  HOME : VTAB 5:SUM = NUM(1): FOR J = 1 TO NN
3174  SUM = SUM - NUM(J + 1): PRINT  SPC( 10);NUM(J)
3175  NEXT J: PRINT  SPC( 8);"-"
3176  PRINT  SPC( 8);"--------": PRINT  SPC( 10);SUM
3179  GOSUB 13000: RETURN
3180  SUM = 1: FOR J = 1 TO NN:SUM = SUM * NUM(J)
3181  PRINT  SPC( 10);NUM(J): NEXT J
3182  PRINT  SPC( 8);"X": PRINT  SPC( 8);"--------"
3183  PRINT  SPC( 10);SUM
3184  GOSUB 13000: RETURN
3185  HOME : INPUT "ENTER NUMBER TO BE DIVIDED";DVND
3186  PRINT : INPUT "ENTER NUMBER TO DIVIDE BY ";DIVSR
3187  PRINT : PRINT : PRINT
3188  PRINT  TAB( 10);DVND;" / ";DIVSR;" = ";DVND
3189  GOSUB 13000: RETURN
```

A considerable amount of effort should be made in this part of the program to format the screen exactly the way it is wanted. Readability is usually enhanced if you use generous spacing.

The SPC command used above is a little different than TAB. SPC moves the printing over the number of spaces indicated from the present cursor position. This contrasts to the TAB command which always moves over from the left margin.

```
3190  HOME : INPUT "ENTER THE NUMBER YOU WANT TO DIVIDE";DVND
3191  PRINT : INPUT "ENTER THE NUMBER TO DIVIDE BY";DIVSR
3192  PRINT : PRINT : PRINT  TAB( 10);".": PRINT  SPC( 3);DVND; TAB( 10);"/";
      SPC( 5);DIVR;"  = ";DVND / DIVSR: PRINT  TAB( 10);"."
3193  GOSUB 13000: GOTO 3130
3197  RETURN
3198  REM
3199  REM  **********************
3200  REM  * COMPARE TWO VALUES *
3205  REM  *     SUB MODULE     *
3208  REM  **********************
3209  REM
3210  HOME : VTAB 5
3215  PRINT "COMPARE MODULE": PRINT
3220  HTAB 5: PRINT "1. NUMBERS"
3225  HTAB 5: PRINT "2. LETTERS"
3230  HTAB 5: PRINT "3. BOTH NUMBERS AND LETTERS"
3235  HTAB 5: PRINT "4. CHOOSE ANOTHER OPTION"
3240  GOSUB 10000
3243  IF ANS = 4 THEN  RETURN
3245  ON ANS GOSUB 3260,3270,3280
3250  GOTO 3210
3255  REM
3260  GOSUB 3400
3268  RETURN
3270  ITEM$ = "LETTERS"
3275  GOSUB 3300
3278  RETURN
3280  ITEM$ = "NUMBERS AND LETTERS"
3285  GOSUB 3300
3288  RETURN
```

Line 3245 sends the program to a nearby subroutine which in turn calls another subroutine. Subroutines can call other subroutines.

```
3290  REM
3295  REM  ********************
3300  REM  * ALPHABETIC SORT *
3303  REM  ********************
3305  REM
3310  HOME : VTAB 5
3320  PRINT "OK, LET'S COMPARE ";ITEM$
3325  PRINT : PRINT "HOW MANY ";ITEM$
3326  PRINT "DO YOU WANT TO ENTER?"
3327  INPUT "(MUST BE NO MORE THAN 10 ITEMS)";NN
3330  IF NN > 10 THEN 3310
3335  REM  * INPUT THE ITEMS *
3338  REM
3340  FOR I = 1 TO NN
3345  PRINT "ENTER ITEM ";I;: INPUT " ? ";OLD$(I)
3350  NEXT I
3360  HOME : VTAB 8: HTAB 10
3361  PRINT "AS ENTERED     IN ORDER": HTAB 10
3362  PRINT "--------     --------"
3365  FOR I = 1 TO NN: HTAB 12: PRINT OLD$(I): NEXT I
3370  START = 1
3375  FOR I = 1 TO NN
3380  FOR K = START TO NN
3383  IF OLD$(I) <  = OLD$(K) THEN 3387
3385  TEMP$ = OLD$(I):OLD$(I) = OLD$(K):OLD$(K) = TEMP$
3387  NEXT K
3390  START = START + 1
3393  NEXT I
3395  VTAB 10: FOR I = 1 TO NN
3397  HTAB 28: PRINT OLD$(I): NEXT I
3398  GOSUB 13000: RETURN
```

Note that in line 3327, the student is instructed regarding the limitations of the response. Clear instructions to the student are necessary.

There are many ways to program an alphabetic sort. The sort here is known as a bubble sort. This type of sort is fine for small numbers of lines, but if large lists are to be sorted, then a more efficient method should be used.

```
3399  REM
3400  REM     ********************
3405  REM     * COMPARE NUMBERS *
3410  REM     ********************
3415  REM
3420  HOME : VTAB 5
3425  PRINT "OK, LET'S COMPARE NUMBERS"
3426  PRINT : PRINT "HOW MANY NUMBERS DO YOU WANT TO ENTER ?"
3427  PRINT : INPUT "(MUST BE LESS THAN 10 ITEMS )";NN
3430  IF NN > 10 THEN 3420
3435  FOR I = 1 TO NN: PRINT "ENTER ITEM ";I;: INPUT " ? ";NUM(I)
3438  NEXT I
3440  HOME : VTAB 8: HTAB 10
3441  PRINT "AS ENTERED       IN ORDER": HTAB 10
3442  PRINT "----------       ---------"
3445  FOR I = 1 TO NN: HTAB 12: PRINT NUM(I): NEXT I
3446  REM
3447  REM  ****************
3448  REM  * NUMERIC SORT *
3449  REM  ****************
3450  REM
3460  START = 1
3470  FOR I = 1 TO NN
3475  FOR K = START TO NN
3480  IF NUM(I) <  = NUM(K) THEN 3490
3485  TEMP = NUM(I):NUM(I) = NUM(K):NUM(K) = TEMP
3490  NEXT K
3495  START = START + 1
3500  NEXT I
3505  VTAB 10
3510  FOR I = 1 TO NN
3515  HTAB 28: PRINT NUM(I)
3520  NEXT I
3525  GOSUB 13000: RETURN
```

```
3989  REM
3990  REM  ****************
4000  REM  * DRILL MODULE *
4005  REM  ****************
4015  TIME = 1000
4020  HOME : VTAB 10
4025  PRINT "I CAN DRILL IN :": PRINT : HTAB 5
4026  PRINT "1. ADDITION": PRINT : HTAB 5
4027  PRINT "2. MULTIPLICATION": PRINT : HTAB 5
4028  PRINT "3. RETURN TO MAIN MENU"
4030  GOSUB 10000
4035  IF ANS = 3 THEN  RETURN
4038  ON ANS GOSUB 4100,4200
4040  GOTO 4020
4045  REM
4090  REM  ******************
4100  REM  * ADDITION DRILL *
4105  REM  ******************
4110  REM
4130  GOSUB 17000: REM  RANDOM NUMBER GENERATOR
4132  PROBLEM$ = ADD$(NUM):ANS$ = AANS$(NUM)
4135  GOSUB 18000: REM  PRINT PROBLEM
4140  RETURN
4190  REM
4195  REM  ************************
4200  REM  * MULTIPLICATION DRILL *
4203  REM  ************************
4205  REM
4220  GOSUB 17000
4230  PROBLEM$ = MULT$(NUM):ANS$ = MANS$(NUM)
4240  GOSUB 18000
4250  RETURN
```

```
9989  REM
9990  REM   **********************
10000  REM  * INPUT MENU CHOICE *
10003  REM  **********************
10004  REM
10020  ONERR  GOTO 16000
10050  VTAB 24: HTAB 1: PRINT "ENTER NUMBER FROM MENU";
10070  GET ANS$
10080  ANS =  VAL (ANS$)
10090  RETURN
10092  REM
10095  REM  **********************
11000  REM  * SCROLLING ROUTINE *
11010  REM  **********************
11011  REM
11020  GOSUB 15000
11050  FOR J = 1 TO 25
11060  PRINT
11070  FOR K = 1 TO 100: NEXT K
11080  NEXT J
11090  RETURN
11092  REM
11095  REM  *****************
12000  REM  * DELAY ROUTINE *
12005  REM  *****************
12010  REM
12050  FOR L = 1 TO TIME: NEXT L
12060  RETURN
12065  REM
12070  REM  ********************
13000  REM  * PRESS SPACE BAR *
13005  REM  ********************
13010  REM
13030  VTAB 22
13050  PRINT "PRESS SPACE BAR TO CONTINUE";
13060  GET A$
13070  IF A$ <  >  CHR$ (32) THEN 13060
13090  RETURN
```

The time delay here is generic. It can be set for different waiting periods based on the needs of the lesson at any given instance.

```
13095  REM
13990  REM  ******************
14000  REM  * COLOR GENERATOR *
14005  REM  ******************
14010  REM
14050  COLOR=  INT (14 *  RND (1)) + 1
14060  RETURN
14065  REM
14990  REM  **********************
15000  REM  * LASER SOUND MAKER *
15005  REM  **********************
15006  REM
15060  & T255,1
15090  FOR P = 250 TO 50 STEP  - 2
15100  & TP,2
15110  NEXT P
15120  FOR P = 50 TO 250 STEP 2
15130  & TP,2
15140  NEXT P
15150  RETURN
15151  REM
15160  REM  **********************
15490  REM  * POKE SOUND ROUTINE *
15495  REM  **********************
15496  REM
15500  FOR I = 768 TO 833: READ P: POKE I,P: NEXT I
15510  DATA  201,84,208,15,32,177,0,32,248,230,138,72,32,183,0,201,44,240,3
15511  DATA  76,201,222,32,177,0,32,248,230
15520  DATA  104,134,3,134,1,133,0,170,160,1,132,2,173,48,192,136,208,4,198
15530  DATA  1,240,7,202,208,246,166,0,208,239,165,3,133,1,198,2,208,241,96
15540  POKE 1013,76: POKE 1014,0: POKE 1015,3
15550  RETURN
```

```
15565  REM
15990  REM  *****************
16000  REM  * ERROR HANDLER *
16005  REM  *****************
16010  REM
16020  E =  PEEK (222)
16050  IF E = 16 OR E = 163 THEN  GOTO 16060
16055  END
16060  POKE 216,0: RESUME
16989  REM
16990  REM  *****************
17000  REM  * RANDOM NUMBER *
17001  REM  *   GENERATOR   *
17003  REM  *****************
17005  REM
17010  NUM =  INT (3 *  RND (1))
17020  RETURN
17990  REM
17994  REM  ******************
17995  REM  * GENERIC PROBLEM *
17996  REM  *     PRINTER     *
17997  REM  ******************
18000  REM
18006  FOR J = 1 TO 3
18007  HOME : VTAB 5
18008  PRINT "ANSWER THE FOLLOWING BY ENTERING THE"
18009  PRINT : PRINT "LETTER OF THE CORRECT RESPONSE"
18010  VTAB 10
18020  PRINT PROBLEM$
18030  GOSUB 19000: REM  GET ANSWER
18035  IF RES$ = ANS$ THEN 18060
18040  VTAB 18: HTAB 5: PRINT "NO, THAT'S NOT CORRECT..."
18045  GOSUB 12000
18048  NEXT J
18049  VTAB 18: HTAB 1: PRINT  SPC( 40): HTAB 1
18050  PRINT "THE CORRECT ANSWER IS: ";ANS$
18051  TIME = 2000
18052  GOSUB 12000
18055  RETURN
18060  REM
18065  GOSUB 2300: REM COLOR BARS
18070  TEXT : HOME : PRINT CRES$(NUM)
18075  GOSUB 12000: RETURN
```

```
18990  REM
18995  REM  *********************
19000  REM  * GET MATH RESPONSE *
19003  REM  *********************
19004  REM
19005  VTAB 22
19010  VTAB 24: PRINT "ENTER ";: INVERSE : PRINT "LETTER";: NORMAL : PRINT " OF
       CORRECT RESPONSE";
19020  GET RES$
19030  RETURN
19487  REM
19489  REM  *********************
19490  REM  * ADDITION PROBLEMS *
19495  REM  *********************
19496  REM
19500  DATA  2 + 2 = ?    A. 5     C. 6
B. 2     D. 4,4 + 5 = ?     A. 5    C. 8
B. 9     D. 10
19505  DATA  3 + 3 = ?   A. 9    C. 7
B. 0     D. 6,4 + 7 = ?    A. 11   C. 12
B. 9     D. 3
19790  DATA  D,B,D,A
19792  REM
19794  REM  *********************
19796  REM  * MULTICA. PROBLEMS *
19798  REM  *********************
19799  REM
19800  DATA  3 X 3 = ?      A. 16    C. 6                          B.  9    D.
8,4 X 4 = ?     A. 16   C. 15                        B.  8   D.  9
19805  DATA  2 X 4 = ?      A. 6     C. 12                         B. 8     D.
10,5 X 2 = ?    A. 8    C. 10                        B. 5    D. 11
19990  DATA   B,A,B,C
```

CHAPTER THREE

LESSON FEATURES; OR, USEFUL SUBROUTINES

Beginning programmers often see some very useful features of commercial programs that they would like to incorporate into their lessons to make their programs run more smoothly and look more professional. Many of these features are not explained very well in some of the common programming reference manuals but are well known to advanced programmers or can be devised by them as needed. The idea for this chapter is to provide a number of common techniques that can be used as is or modified to fit individual lesson needs. Add to this chapter as you see other techniques and figure out how they work. You might also wish to put these short programs on a utility diskette which would contain a collection of independent modules or subroutines ready to pull in and use during your lesson construction.

The reader should note that when sample programs are given in this chapter, parts of a line might be underlined. This is the signal that the programmer must supply some information for that line. The line should not be typed into the computer as printed.

Feature # 1:

PRESS ANY KEY TO CONTINUE
(getting the student to the next section of the lesson)

Necessary command:

```
10 PRINT "PRESS ANY KEY TO CONTINUE"
20 GET A$                    (any string variable name is ok)
```

Comments:

Any key pressed, whether on purpose or accidentally, will trigger the program to go to the next part of the lesson. The student does not have to press the return key. This method creates a problem because the program can advance without the student wanting it to do so. Notice that the GET command is used here. GET will accept only a single character as input from the keyboard. If more than one character is typed, only the first is regarded. Another problem is that a key can be accidently pressed before the GET statement appears on the screen. Applesoft will hold one keystroke in a buffer so when the GET statement comes up, the program will trigger.

Many educational programs use the GET command because children only need to press one key - they don't have to press RETURN after every answer.

Sample program:

```
10 HOME
20 PRINT "WHAT IS 2 + 2?"
30 VTAB 20
40 PRINT "PRESS ANY KEY TO SEE THE ANSWER";
50 GET B$
60 HOME
70 PRINT "4 IS THE ANSWER"
```

Advanced tip:

If you want to guard against a spurious key being pressed, change line 40 to the "PRESS THE SPACE BAR", then check to see if the space bar was pressed (see Feature # 3) after line 50 and then GOTO line 30 (reprint the request line).

Feature # 2:

PRESS RETURN TO CONTINUE
(getting the student to the next section of the lesson)

Necessary command:

```
10 INPUT "PRESS RETURN TO CONTINUE";A$
```
(A$ can be any string variable name)

Comments:

Using the input command here requires the student to press the return key to continue the lesson. The advantage here is that an accidental pressing of any key will not trigger the lesson to advance until the return key is pressed.

Sample program:

```
10 HOME
20 PRINT "WHAT IS 2 + 2?"
30 VTAB 20
40 INPUT "PRESS RETURN TO
   SEE THE ANSWER";B$
50 HOME
60 PRINT "4 IS THE ANSWER"
```

or

Sample program:

```
10 HOME
20 PRINT "WHAT IS 2 + 2?"
30 GOSUB 100
40 HOME
50 PRINT "4 IS THE ANSWER"
60 END
100 VTAB 20
110 INPUT "PRESS RETURN TO
    SEE THE ANSWER";B$
120 RETURN
```

Variation:

If you wish a single inverse "R" instead of the words "PRESS RETURN TO CONTINUE," the necessary command is:

```
10 VTAB 23 : HTAB 39 : INVERSE : PRINT "R"; : NORMAL
```

You can vary the position of the "R" on the screen but don't print lower than 23 or to the right more than 39. Be sure to include the semicolon after the "R" because this keeps the cursor waiting for the input on the 23rd line. If the semicolon on line 40 in the first sample program and line 110 in the second sample program is absent, the cursor will appear on the next line.

Feature # 3:

PRESS SPACE BAR TO CONTINUE
(getting the student to the next section of the lesson)

Necessary command:

```
10 PRINT "PRESS SPACE BAR TO CONTINUE";
20 GET A$
30 IF A$ = CHR$(32) THEN 50
40 GOTO 20
50 REM REST OF PROGRAM HERE  OR  RETURN IF A SUBROUTINE
```

Comments:

CHR$(32) is the ASCII code for the space bar. Any character on the keyboard with an ASCII code number may be used with the commands above (on the Apple II+, ASCII codes 32-94 may be used, on the Apple IIe or IIc, ASCII codes 32-127 may be used). See the ASCII list which follows for the code numbers and the characters they represent.

Sample program:

```
10 HOME
20 PRINT "WHAT IS 2 + 2?"
30 VTAB 20
40 PRINT "PRESS SPACE BAR TO
   SEE ANSWER";
50 GET Z$
60 IF Z$ = CHR$(32) THEN 80
70 GOTO 50
80 HOME
90 PRINT "4 IS THE ANSWER"
```

or

Sample program:

```
10 HOME
20 PRINT "WHAT IS 2 + 2?"
30 GOSUB 100
40 HOME
50 PRINT "4 IS THE ANSWER"
60 END
100 VTAB 20
110 PRINT "PRESS SPACE BAR TO
    SEE ANSWER";: GET Z$
120 IF Z$ = CHR$(32) THEN RETURN
130 GOTO 100
```

Variation:

You may wish to have an inverse "SB" in the lower right hand corner of the screen rather than using the full bottom line to indicate "PRESS SPACE BAR TO CONTINUE." Line 10 of the program at the top of the page would then read:

```
10 VTAB 21 : HTAB 38 : INVERSE : PRINT "SB"; : NORMAL
```

You can vary the position of the "SB" on the screen depending on what you have printed on the screen. VTAB 24 is the last line on the screen. HTAB 38 is the right-most position possible when printing two characters and still leave room for the cursor. If the cursor appears on the next line, you probably forgot the semicolon after the SB.

ASCII Character Codes

ASCII Code	On Screen	Key II+	Key c&e
0		ct-@	ct-@
1		ct-A	ct-A
2		ct-B	ct-B
3		ct-C	ct-C
4		ct-D	ct-D
5		ct-E	ct-E
6		ct-F	ct-F
7	(bell)	ct-G	ct-G
8	(left)	ct-H or	
9	(tab)	ct-I or tab	
10	(down)	ct-J or	
11	(up)	ct-K or	
12		ct-L	ct-L
13	c.r.	ct-M	ct-M
14		ct-N	ct-N
15		ct-O	ct-O
16		ct-P	ct-P
17		ct-Q	ct-Q
18		ct-R	ct-R
19		ct-S	ct-S
20		ct-T	ct-T
21	space	ct-U or	
22		ct-V	ct-V
23		ct-W	ct-W
24	canc.l	ct-X	ct-X
25		ct-Y	ct-Y
26		ct-Z	ct-Z
27		Esc	Esc
28		n.a.	ct-\
29		ctsM	ct-]
30		ct-^	ct-^
31		n.a.	ct-sft
32	space	spb	spb
33	!	!	!
34	"	"	"
35	#	#	#
36	$	$	$
37	%	%	%
38	&	&	&
39	'	'	'
40	(	(	(
41	)	)	)
42	*	*	*
43	+	+	+
44	,	,	,
45	-	-	-
46	.	.	.
47	/	/	/
48	0	0	0
49	1	1	1
50	2	2	2
51	3	3	3
52	4	4	4
53	5	5	5
54	6	6	6
55	7	7	7
56	8	8	8
57	9	9	9
58	:	:	:
59	;	;	;
60	<	<	<
61	=	=	=
62	>	>	>
63	?	?	?
64	@	@	@
65	A	A	sftA
66	B	B	sftB
67	C	C	sftC
68	D	D	sftD
69	E	E	sftE
70	F	F	sftF
71	G	G	sftG
72	H	H	sftH
73	I	I	sftI
74	J	J	sftJ
75	K	K	sftK
76	L	L	sftL
77	M	M	sftM
78	N	N	sftN
79	O	O	sftO
80	P	P	sftP
81	Q	Q	sftQ
82	R	R	sftR
83	S	S	sftS
84	T	T	sftT
85	U	U	sftU
86	V	V	sftV
87	W	W	sftW
88	X	X	sftX
89	Y	Y	sftY
90	Z	Z	sftZ
91	[	n.a.	[
92		n.a.	
93	]	sftM	]
94	^	^	^
95	und.l.	n.a.	sft-
96	`	n.a.	`
97	! or a	n.a.	a
98	" or b	n.a.	b
99	# or c	n.a.	c
100	$ or d	n.a.	d
101	% or e	n.a.	e
102	& or f	n.a.	f
103	' or g	n.a.	g
104	(or h	n.a.	h
105	) or i	n.a.	i
106	* or j	n.a.	j
107	+ or k	n.a.	k
108	, or l	n.a.	l
109	- or m	n.a.	m
110	. or n	n.a.	n
111	/ or o	n.a.	o
112	0 or p	n.a.	p
113	1 or q	n.a.	q
114	2 or r	n.a.	r
115	3 or s	n.a.	s
116	4 or t	n.a.	t
117	5 or u	n.a.	u
118	6 or v	n.a.	v
119	7 or w	n.a.	w
120	8 or x	n.a.	x
121	9 or y	n.a.	y
122	: or z	n.a.	z
123	; or {	n.a.	{
124	< or |	n.a.	|
125	= or }	n.a.	}
126	> or ~	n.a.	~
127	?or/d.	n.a.	del.

Note: 128-255 repeat codes 0-127.

Note: Codes above 95 on the Apple II+ produce strange screen characters. A program written on the IIe and run on the II+ will generate these characters when a lower case letter has been used.

n.a. = not available; sft = shift key; cts = control shift; ct = control key; und.l. = underline; spb = space bar; Esc = Escape; c.r. = carriage return; d. or del. = delete

Feature # 4:

TO HAVE QUESTION MARKS OR NO QUESTION MARKS; TO HAVE A CURSOR OR NO CURSOR

Necessary command:

```
10 INPUT "DO YOU SEE A QUESTION MARK ";A$      The answer is no.

20 INPUT "DO YOU SEE A QUESTION MARK? ";A$     The answer is yes.

30 PRINT "DO YOU SEE A QUESTION MARK ";        The answer is no.
40 INPUT A$

50 PRINT "DO YOU SEE A QUESTION MARK? ";       The answer is yes.
60 INPUT A$

50 PRINT "DO YOU SEE A QUESTION MARK?";        The answer is yes.
60 GET A$

70 PRINT "DO YOU SEE A QUESTION MARK ";        The answer is no.
80 GET A$
```

Decide early in your design whether the question mark is needed or not. Using one of the styles shown above from the beginning of your program will save much debugging time. Note that the semicolon in all the examples above will keep the cursor on the same line as the question.

Sometimes you do not want to have a cursor on the screen since you may feel that it is distracting. You can use a subroutine to make the cursor invisible. Below, we illustrate a subroutine which has the student press the space bar to go from one text page to another with no blinking cursor present.

First we will print normally on a page of text:

```
10 HOME
20 PRINT "THIS IS PAGE ONE OF TEXT"
```

Now we will run subroutine 1000, for "press the space bar to continue" and at the same time suppress the cursor.

```
30 GOSUB 1000
```

Now we will print the next page of text.

```
40 PRINT "THIS IS THE SECOND PAGE OF TEXT"
50 END
```

Here is the subroutine:

```
998 REM SPACE BAR TO CONTINUE AND SUPRESS CURSOR
1000 VTAB 21 : HTAB 39 : INVERSE : PRINT "SB" : NORMAL :
     REM PRESS SPACE BAR COMMAND
1010 REM THE NEXT SIX LINES HIDE THE CURSOR
1020 CURS$ = "    "          (Four spaces)
1030 C = 1
1040 L = LEN (CURS$)
1050 VTAB 10 : HTAB 29 : PRINT MID$(CURS$,C,1)
1060 KEY = PEEK (-16384)
1070 IF KEY < 128 THEN C = C+1 - L * (C=L) : GOTO 1050
1080 RETURN
```

Feature # 5:

END THE PROGRAM WHEN "END" IS ENTERED; PROVIDE HINTS WHEN "H" IS ENTERED

Necessary command:

```
10 INPUT "ANY MESSAGE";A$
20 IF A$ = "END" THEN PRINT "YOU HAVE ENDED THE PROGRAM.
   GOODBYE." : END
```

Comments:

By using the INPUT command rather than the GET command, the computer can test for more than one character entered at the keyboard. If you wish to allow the student to get out of the program at any time an input is requested, then you must test every input for the word "END". The easiest thing to do is put the test in a subroutine. It is much easier to type GOSUB line # than the complete test statement "IF A$ = "END"..."

Sample program:

```
10 HOME
20 INPUT "WHAT IS 2 + 2? "; A$
30 GOSUB 1000
35 IF A$ = "H" THEN PRINT "IF YOU HAVE TWO KITTENS AND YOUR SISTER HAS
   TWO KITTENS, HOW MANY KITTENS ARE THERE?"
40 IF A$ <> "4" THEN 10
50 PRINT "RIGHT"
60 HOME
70 INPUT "WHAT IS 4 + 4? "; A$
80 GOSUB 1000
85 IF A$ = "H" THEN PRINT "IF YOU HAVE FOUR DOGS AND YOUR NEIGHBOR HAS
   FOUR DOGS, HOW MANY DOGS ARE IN THE NEIGHBORHOOD?"
90 IF A$ <> "8" THEN 60
95 END
100 REM CONTINUE PROGRAM ASKING QUESTIONS
110 :
999 REM THIS SUBROUTINE TESTS INPUT FOR THE WORD "END"
1000 IF A$ = "END" THEN HOME : VTAB 20 : HTAB 5 : PRINT "YOU HAVE
     ENDED THE PROGRAM.  GOODBYE" : END
1010 RETURN
```

Variation:

In the same way, any input can be tested to send the student anywhere in the program. For example, the word "hint" might give a general hint at any

point (we have demonstrated a specific hint above which is unique to a given place). The word "menu" might take us there at any time. "Rule" might remind us at any point of a principle to be followed. All checks for these types of responses could be programmed in the same subroutine. There is one problem with this technique. It is not a good idea to send the program to a subroutine and then use a GOTO to get out of the subroutine. If you need a GOTO statement, then test for the condition at each input like we did with "hint" in the example above.

Feature # 6:

ERROR TRAPPING FOR CERTAIN NUMERIC KEYS
(you want the student to press a number 1 through 5 and no other keys)

Necessary command:

```
10 REM PRESENT A MENU HERE
15 PRINT "PLEASE ENTER NUMBER FROM MENU";
20 GET A$
30 A = VAL (A$)
40 IF A < 1 OR A > 5 THEN 15 (you could print an error message here)
50 REM REST OF PROGRAM OR RETURN IF A SUBROUTINE
```

Comments:

This technique works with the GET command - not INPUT. Line 30: A = VAL (A$) changes the student's response to a numerical value. Only the first character of the input is changed. All letters would be changed to the value zero. Numbers 0-9 remain unchanged. Line 40 will allow only the numbers 1-5 to proceed to the rest of the program, otherwise, the menu is repeated.

Why not just input an A (or integer variable) rather than an A$ or real variable? Inputing an integer variable and trapping it causes question marks and the message "reenter" to appear on the screen plus other annoying problems. That is why we recommend to input a string variable first and then convert it to a number - it is just a more flexible trap.

Sample program:

For a sample program, see sample program p. 3-11 which presents a menu and then traps for any key other than 1-5.

Feature # 7:

PRESENT A MENU AND ALLOW THE STUDENT TO SELECT AN OPTION

Necessary command:

10 ON B GOTO line#, line#, line#, etc.

or

10 ON B GOSUB line#, line#, line#, etc.

Comments:

A menu usually has numbered choices. For example:

Which would you like to do?
1. Play a game
2. Take a test
3. See a picture
4. Quit

If you type in a 1, the computer executes the first line number listed after the GOTO or GOSUB. If you type in a 2, the computer executes the second line number listed after the GOTO or GOSUB. Using this command eliminates a series of IF THEN statements such as: IF B = 1 THEN GOSUB 1000: IF B = 2 THEN GOSUB 2000...etc. If you use GOSUB, remember that the next line to be executed after a RETURN statement is right after the "ON" statement. If you use a "GOTO", you will need to tell the computer what line number to go to in the executed section of the program. GOSUBs are usually the preferable technique to use since this helps keep the modules independent and provides more control in the module that called the subroutine.

Other appropriate examples:

ON X GOSUB 750,850,900,1500,1950

ON X GOSUB 850,750,1500,900,1950

Sample program:

```
 10 HOME
 20 PRINT "WHICH WOULD YOU LIKE?"
 30 PRINT "1. PRINT A LIST"
 40 PRINT "2. PRINT MAILING LABELS"
 50 PRINT "3. ADD TO THE LIST"
 60 PRINT "4. DELETE FROM THE LIST"
 70 PRINT "5. END"
 80 VTAB 20
 90 INPUT "PLEASE ENTER THE NUMBER";A$
 95 A = VAL(A$) : REM CHANGES RESPONSE TO A NUMERIC CHARACTER
 96 IF A > 5 OR A < 1 THEN 10 : REM ERROR TRAPS FOR NUMBERS OUT OF RANGE
100 IF A = 5 THEN PRINT "THIS IS THE END" : END
110 ON A GOSUB 1000,1500,2000,2500
120 GOTO 10
1000 HOME : PRINT "HERE IS YOUR LIST" : FOR X = 1 TO 1000 : NEXT X : RETURN
1500 HOME : PRINT "HERE ARE YOUR MAILING LABELS" : FOR X = 1 TO 1000 :
     NEXT X : RETURN
2000 HOME : PRINT "YOU CAN NOW ADD TO THE LIST" : FOR X = 1 TO 1000 :
     NEXT X : RETURN
2500 HOME : PRINT "YOU CAN NOW DELETE FROM THE LIST" : FOR X = 1 TO
     1000 : NEXT X : RETURN
```

Note: If the student enters a letter for the response on line 90, line 95 will change that letter to a zero. Line 96 will treat a zero as out of range and will reprint the menu until a response 1-5 is entered. This technique is a very effective error trap. If you would like to print an error message, change line 96 to:

```
96 IF A < 1 OR A > 5 THEN VTAB 20 : PRINT "MUST BE 1-5";:
   FOR X = 1 TO 1000 : NEXT X : GOTO 10
```

Feature # 8:

PACE TEXT OR GRAPHICS
(i.e. slow down the printing of words or the drawing of graphics on the screen)

Necessary command:

For text:

```
10 SPEED =  a number from 0 to 255
20 PRINT "I WANT THIS TEXT TO BE PRINTED AS SLOW OR AS FAST AS
          I WISH IT TO BE"
30 SPEED = 255 : REM RESET THE SPEED BACK TO MAXIMUM
```

For graphics:

```
10 GR : COLOR = 2
20 FOR X = 0 TO 39 : REM LOOP TO MAKE BACKGROUND COLOR
30 FOR Y = 0 TO 100 : NEXT Y : REM LOOP TO SLOW GRAPHIC DRAW DOWN
40 VLIN 0,39 AT X
50 NEXT X
```

Comments:

For text printing, the computer sets a normal speed of 255. Once the speed is altered, it will remain so until it is changed or the RESET key is pressed. The speed command has no effect on graphics so a different technique must be used. Short time waits can be included at any point in the graphic drawing to slow down a sequence. The effect of these time delays is an illusion of someone painting color with a brush onto the screen. All types of tricks can be done. For example, the background color can be created in normal speed and then, at a slower speed, a car could be drawn something like an "etch a sketch" drawing over the top of the background. Pressing the RESET key will return the speed to its default setting of 255.

Sample program:

```
 10 REM A PICTURE OF A TABLE AND A JAR
 20 GR
 25 :
 30 COLOR = 8 : REM BACKGROUND
 40 FOR ROW = 30 TO 40
 50 HLIN 0,39 AT ROW
 60 NEXT ROW
 65 :
 70 REM DRAW FLOOR
 80 COLOR = 15
 90 FOR ROW = 0 TO 30
100 HLIN 0,39 AT ROW
110 NEXT ROW
115 :
120 REM DRAW TABLE
130 COLOR = 0
140 HLIN 10,29 AT 15
150 GOSUB 1000
160 VLIN 16,30 AT 14
170 GOSUB 1000
180 VLIN 16,30 AT 25
190 GOSUB 1000
200 FOR K = 14 TO 10 STEP -1
210 HLIN 17,21 AT K
220 GOSUB 1000
230 NEXT K
240 PLOT 19,8
250 END
999 :
1000 FOR X = 1 TO 200 : NEXT X : RETURN : REM TIME DELAY SLOWS
     DOWN DRAWING
```

Feature # 9:

ONE PART OF THE TEXT SCREEN REMAINS CONSTANT WHILE ANOTHER PART CHANGES. For example: a rule can remain at the top of the screen while the student is drilled on the rule at the bottom of the screen.

Necessary Command:

10 REM FIRST, SET UP THE PRINTING ON THE PART OF THE SCREEN THAT DOES NOT CHANGE.

20 POKE ___,___ : REM SET THE SCROLLING WINDOW. (use one of the number sets listed below)

30 REM THIRD, SET UP THE PART OF THE SCREEN THAT WILL CHANGE.

Scrolling window commands:

POKE 32,___ set left margin of screen from 0 to 39

POKE 33,___ set right margin of screen from 39 to 0

POKE 34,___ set top margin of screen from 0 to 23

POKE 35,___ set bottom margin of screen from 23 to 0

Comments:

These pokes only work on the text screen - not on the graphics screen. Once you use a POKE, you will have to reset it to normal. The normal settings are: POKE 32,0; POKE 33,40; POKE 34,0; POKE 35,23.

WARNING: Setting margins beyond the ranges shown above can erase part of your program from memory.

With the 80-column card active, the ranges for the windows become: left: 0-78, right: 2-80, top: 0-23, bottom: 23-0.

Sample Program:

```
 10 REM THIS PROGRAM DEMONSTRATES THE USE OF
 20 REM THE SCROLLING WINDOW ON THE TEXT SCREEN
 30 :
 40 HOME
 45 :
 50 REM FIRST, SET UP THE PRINTING ON THE PART
 55 REM OF THE SCREEN THAT DOES NOT CHANGE.
 60 VTAB 1
 70 HTAB 8 : PRINT "RULE :   I BEFORE E"
 80 PRINT : HTAB 16 : PRINT "EXCEPT AFTER"
 90 PRINT : HTAB 16 : PRINT "C OR WHEN"
100 PRINT : HTAB 16 : PRINT "SOUNDED AS A
110 PRINT : HTAB 16 : PRINT "IN NEIGHBOR"
120 PRINT : PRINT
130 PRINT "HERE ARE SOME PROBLEMS FOR YOU:"
140 :
150 REM DRAW BORDER AROUND THE RULE
160 POKE 48,32 : VLIN 0,19 AT 13 : VLIN 0,19 AT 28 :
    HLIN 13,28 AT 19
170 :
180 REM SECOND, SET THE SCROLLING WINDOW
190 POKE 34,15 : REM SET TOP OF SCREEN TO VTAB 15
192 :
195 REM THIRD, SET UP THE PART OF THE SCREEN THAT WILL CHANGE.
200 VTAB 17 : REM TAB DOWN TO NEW SCREEN
210 HTAB 15 : PRINT "LIE OR LEI"
220 PRINT : HTAB 10 : PRINT "(TO TELL AN UNTRUTH)
230 PRINT : PRINT
240 HTAB 6 : INPUT "TYPE IN THE CORRECT SPELLING:";A$
250 IF A$ = "LIE" THEN 290
260 HTAB 7 : PRINT "NO, THAT'S NOT CORRECT, TRY AGAIN."
270 FOR X = 0 TO 1000 : NEXT X
280 HOME : GOTO 210
290 HOME : HTAB 15 : PRINT "WONDERFUL"
300 FOR X = 0 TO 2000 : NEXT X
305 :
307 REM RESET THE SCREEN BACK TO NORMAL
310 POKE 34,0
320 HOME
330 END
```

Feature # 10:

KEEP SCORE DURING A QUIZ AND PRINT OUT THE NUMBER RIGHT AND % RIGHT

Necessary command:

Set your right answer variable to zero at the beginning of the test section: 10 RIGHT = 0

Ask your questions in the usual way.
Example: INPUT "WHAT IS 2 + 2?";A$

Immediately after each question, test the answer. If the answer was correct, then add one point to the right answer variable:
Example: IF A$ = 4 THEN RIGHT = RIGHT + 1

After all the questions have been asked, then you can print out the result by printing the value of RIGHT.
Example: PRINT "YOU GOT ";RIGHT; " ANSWERS CORRECT OUT OF 10 QUESTIONS"
PRINT : PRINT "YOUR PERCENT CORRECT IS ";RIGHT/10 * 100;"%"
(watch your spacing in the print statements)

When the next student uses the program, the RIGHT variable will be reset to zero at line 10.

Variations:

There are many variations possible. You may create a variable named WR (meaning wrong — you can't use WRONG because the reserved word ON is in the middle of wrong) and store the number of wrong answers in that variable. It is possible to store the scores of several quizzes during the lesson. Just use different variable names for each quiz and then print out the results at the end. You could also add up the variables and produce a grand percentage that were answered correctly:

```
1000 PRINT "YOUR TOTAL PERCENTAGE CORRECT FOR ALL THE QUIZZES IN THIS
     LESSON IS "; A + B + C + D /40 * 100;"%"
     (this assumes four quizzes and a total of 40 questions — A is
     the score on the first quiz)
```

Feature # 11:

TO PAGE BACK AND FORTH THROUGH TEXT WITH THE ESCAPE KEY USED TO GET OUT OF THE PROGRAM

Necessary command:

Tell the student that at the bottom of each text page, pressing a "B" will return to the previous page and pressing "F" will advance one page. At the bottom of each text page print the line:

```
4000 HTAB 27 : VTAB 22 : INVERSE : PRINT "B<- ->F"; : GET A$ :
     NORMAL  (this can be a subroutine that ends with
     a RETURN)
```

Now we will test the input for each page using the ASC command. If the input was a "B" then we will go back to the line number for the start of the previous page:

```
140 IF ASC(A$) = 66 THEN  line number of previous page
    (the 66 here is the ASCII value for the letter B)
```

Next we need to check if the escape key was pressed:

```
5000 IF ASC(A$) = 27 THEN  line number to end the program
     (the 27 is the ASCII value for the escape key)
```

Next we need to check if the F key was pressed:

```
5010 IF ASC(A$) = 70 THEN RETURN (continue on)
     (the 70 is the ASCII value for the letter F)
```

Lastly, we need to error trap for any other keys that might be pressed accidently.

Sample program:

```
 90 HOME : PRINT "THIS IS THE TITLE PAGE"
 94 FOR X = 1 TO 2000 : NEXT X
 95 :
 96 HOME
100 PRINT "THIS IS PAGE 1" : REM PUT PAGE ONE OF TEXT HERE
120 GOSUB 4000 : REM GET INPUT LINE
130 IF ASC(A$) = 66 THEN 90
140 GOSUB 5000 : REM GO FORWARD, ESCAPE AND ERROR TRAP
150 :
200 HOME
210 PRINT "THIS IS PAGE 2"
220 GOSUB 4000
230 IF ASC(A$) = 66 THEN 96
240 GOSUB 5000
250 :
300 HOME
310 PRINT "THIS IS PAGE 3"
320 GOSUB 4000
330 IF ASC(A$) = 66 THEN 200
340 GOSUB 5000
350 :
355 END
360 REM REST OF PROGRAM GOES HERE
3998 :
3999 REM INPUT SUBROUTINE
4000 HTAB 27 : VTAB 22 : INVERSE : PRINT "B<- ->F"; : GET A$ :
     NORMAL : RETURN
4998 :
4999 REM FORWARD, ESCAPE AND ERROR TRAP
5000 IF ASC(A$) = 27 THEN END : REM IF ESCAPE KEY PRESSED THEN END THE
     PROGRAM
5010 IF ASC(A$) = 70 THEN RETURN : REM IF AN F PRESSED THEN GO ON TO
     NEXT PAGE
5020 GOSUB 4000 : REM IF ANY OTHER KEY PRESSED THEN REPRINT THE INPUT
     LINE
5030 GOTO 5000 : REM RECHECK FOR ERRORS
```

Feature # 12:

TO USE INVERSE TO CREATE BORDERS OR BOXES AROUND TEXT; TO INVERSE A A SINGLE WORD WITHIN A SENTENCE -- ALL ON THE TEXT SCREEN.

Necessary command:

For borders and boxes use:

```
10 INVERSE              Note: we are on the text screen.
20 HTAB 5               Tab over to start of a horizontal line.
30 PRINT"             " Use spaces to print as long a line as desired.
40 FOR X = 1 TO 10      Position the beginning of a vertical line.
50 HTAB 5 : VTAB X      Tell how long the vertical line should be.
60 PRINT " "            Print one or several spaces for thin or
                           thick lines.
70 NEXT X
80 NORMAL               Get back to the normal mode and then print
                           words as usual.
```

Following the commands above results in boxes or emphasis lines anywhere on the text screen. You can vary your boxes by using inverse and any character on the keyboard instead of the space. For example, replace line 60 above with PRINT "#". You can also use keys like "%" without the inverse for patterned boxes.

For inversing a single word or phrase in a sentence:

```
10 PRINT "WE ONLY INVERSE ";        Words in normal print.
20 INVERSE                          Change to inverse.
30 PRINT "IMPORTANT";               Print words in inverse.
40 NORMAL                           Change back to normal.
50 PRINT " WORDS ON THE SCREEN"     Continue printing.
```

Note the use of the space after the word INVERSE in line 10 and the semicolon. The space prevents an inverse space (sloppy looking) and the semicolon keeps the next word on the same line. Note also the space before WORDS on line 50 for the same reason. We could write the entire command on one line but it is so easy to make errors in long lines and they are much more difficult to edit. It could read:

```
10 PRINT "WE ONLY INVERSE ";: INVERSE : PRINT "IMPORTANT";: NORMAL :
   PRINT " WORDS ON THE SCREEN"
```

Feature # 13:

SELECT A REINFORCING GRAPHIC OR MESSAGE AT RANDOM

(i.e., providing some variety as you praise a student)

Necessary command:

```
10 X = INT(HINUM * RND(1)) + LOWNUM
           |                   |
           |                   put minimum number you want here.
           |
           put maximum number you want here.

20 ON X GOSUB line#, line#, line#, line#
```

Examples:

```
10 X = INT(4 * RND(1)) + 1
```

will generate a random number, stored in X, between 1 and 4

```
10 X = INT(10 * RND(1)) + 1
```

will generate a random number stored in X between 1 and 10

Comments:

"RND(1)" picks a random number between 0 and 1. This is a decimal number. In order to get a whole number, we take the integer value: "INT (RND(1))." The problem is, we still can get a zero since zero is an integer. So, we add some value to the end to insure that we do not get a zero: "INT (RND(1)) + 1." Now we get only the number "one" from our command. To include numbers greater than one, we must add a multiplier. The multiplier will be the highest value that the computer will generate: "INT (6 * RND(1)) + 1" will generate a number between 6 and 1.

Sample program:

```
10 HOME:VTAB 10
20 INPUT "WHAT WAS PRES. LINCOLN'S FIRST NAME?"; A$
30 IF A$ <> "ABRAHAM" THEN 10
40 X = INT (4 * RND (1)) + 1
50 ON X GOSUB 1000,2000,3000,4000
60 PRINT : PRINT : PRINT "THAT'S THE END" : GOSUB 10000
70 END
```

```
 1000 PRINT PRINT : PRINT "GOOD" : GOSUB 10000 : RETURN
 2000 PRINT : PRINT : PRINT "GREAT" : GOSUB 10000 : RETURN
 3000 PRINT: PRINT : PRINT "WONDERFUL" : GOSUB 10000 : RETURN
 4000 PRINT : PRINT : PRINT "FINE" : GOSUB 10000 : RETURN
10000 FOR Y = 1 TO 5000 : NEXT Y : RETURN
```

Vatiations:

The subroutines 1000, 2000, 3000, 4000 could be four different graphics. Any number of different reinforcing or negative messages can be stored and used in this manner.

Another example (Sample program:

The following program does a number of things including the presentation of random error messages and random reinforcements. It is worth careful study.

```
10  REM *EXAMPLE RANDOM MSG PRINT*
20  BEEP$ =  CHR$ (7)
30  :
40  REM * STORE RANDOM MSGS *
50  :
55  FOR I = 0 TO 2
60  READ INCRES$(I)
70  NEXT I
80  FOR I = 0 TO 2
90  READ CRCTRES$(I)
100  NEXT I
110  DATA  "NO, THAT IS NOT CORRECT, TRY AGAIN."
120  DATA   "SORRY, THAT IS INCORRECT,CHOOSE ANOTHER"
130  DATA  "THAT IS NOT CORRECT, PLEASE TRY AGAIN."
140  DATA  "GREAT !!! THAT IS THE CORRECT ANSWER."
150  DATA "WONDERFUL!! YOU HAVE BEEN STUDYING!"
160  DATA "SUPER !!  GOOD WORK !!"
170  :
180  :
190  HOME
200  VTAB 2
210  PRINT : PRINT
220  PRINT "WHAT IS THE CAPITOL OF ARKANSAS ?"
230  VTAB 8: HTAB 2
240  PRINT "(A)  HOT SPRINGS";: HTAB 20: PRINT "(C)  BATESVILLE"
250  PRINT : HTAB 2
260  PRINT "(B)  LITTLE ROCK";: HTAB 20: PRINT "(D)  FAYETTEVILLE"
270  ANS$ = "B"
280  GOSUB 10000: REM  INPUT ANSWER
290  IF CHOICE$ = ANS$ THEN  GOTO 320
300  GOSUB 10500: REM  PRINT INC MSG
310  GOTO 200
320  GOSUB 11500: REM  PRINT CRCT MSG
340:
350 END
```

```
9990  :
9992  REM ********************
9994  REM *   INPUT  ANSWER  *
9996  REM ********************
9998  :
10000  VTAB 21
10010  INPUT "PLEASE CHOOSE FROM THE ABOVE: ";CHOICE$
10020  RETURN
10030  :
10490  :
10492  REM ********************
10494  REM * PRNT INC RES MSG *
10496  REM ********************
10498  :
10500  MAXRDUM = 3
10510  GOSUB 11000: REM  RANDOM NUMBER
10515  VTAB 23: PRINT  SPC( 40);
10520  VTAB 23
10525  PRINT BEEP$;
10530  PRINT INCRES$(RDUMNBR);
10540  RETURN
10550  :
10990  :
10992  REM ********************
10994  REM * RNDUM NUMBER GEN *
10996  REM ********************
10998  :
11000  RDUMNBR =  INT (MAXRDUM * ( RND (1)))
11010  RETURN
11020  :
11490  :
11492  REM **********************
11494  REM * PRNT CRCT RES MSG *
11496  REM **********************
11498  :
11500  MAXRDUM = 3
11510  GOSUB 11000: REM  RDUMNBR
11520  VTAB 23: PRINT  SPC( 40);
11530  VTAB 23: HTAB 1
11540  PRINT CRCTRES$(RDUMNBR);
11550  RETURN
```

CHAPTER FOUR

PROGRAMMING TIPS

As the programmer gains experience, there are a number of methods available to make life a little easier. No beginning textbook that we know, explains in plain English a wide variety of helps to make coding a program a little more bearable. The purpose of this chapter is to provide a number of tips that will streamline your hours of coding and debugging computerized lessons. The reader should note that any underlined part in sample programs must be supplied by the user.

Tip #1: Easy Editing

After you have entered your program and you wish to correct errors, you will find that Applesoft is not quite as friendly as some other versions of BASIC. One of the problems is that when Applesoft lists a line and that line is longer than 24 characters, Applesoft will leave some blank spaces at the beginning of the second line. These are hard to edit across. For example:

```
10 PRINT "THIS LINE IS TOO LONG TO
_____BE ON ONE LISTED LINE."
```

The example shows the blanks left by Applesoft at the beginning of the second line. To make the line easier to edit, and before you list out the lines to be

corrected, type:

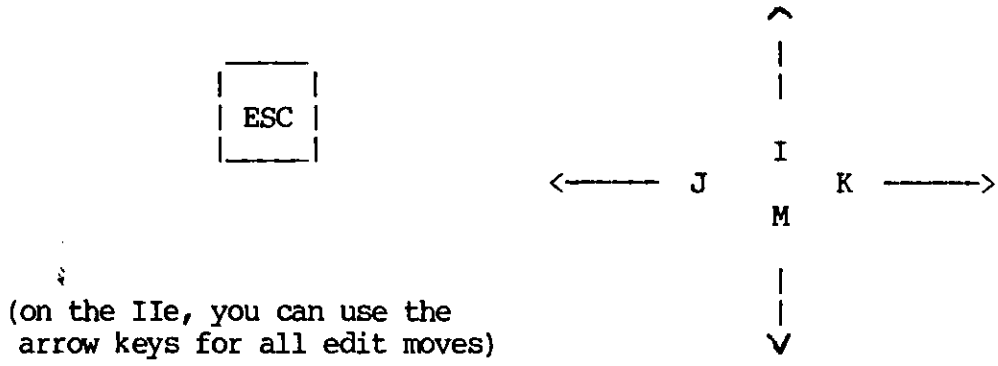
POKE 33,33

When you list line 10 above it will then look like this:

10 PRINT "THIS LINE IS TOO LONG TO
BE ON ONE LISTED LINE."

The POKE has eliminated the spaces at the beginning of the second line and editing becomes easier.

To edit, you must know how to move the cursor. Memorize the following keys that move the cursor: press escape (ESC) and then the "I,J,K,M" keys depending on the direction you want the cursor to move.



(on the IIe, you can use the arrow keys for all edit moves)

Here are the steps once you have mastered the cursor moves.

1. POKE 33,33
2. List the line(s) you wish to change.
3. Move the cursor <u>to the beginning</u> of the line you want to edit.
   (If you have an Apple IIe, press ESC again (this step is not necessary on the II+))
4. Use the right arrow to copy across the characters that are correct.
5. When you get to a character you wish to change, simply type over

the mistakes with the correct characters.

6. Continue copying with the right arrow key until you are at the end of the line.

7. Press the RETURN key.

Your line should be correctly edited. To be sure, list the line. Practice this method about ten times and you will be in business. Don't worry about having to restore Poke 33,33 back to its default value of Poke 33,40 because your Apple will do it for you when you run a program, press RESET or do a number of other tasks. How does Poke 33,33 work? It moves the right hand margin of the text screen over from 40 to 33 and forces Applesoft to delete the spaces when it lists out programs.

Another thing you can do while editing is to delete characters in a line and then continue copying. To do this, just press the ESC key and use the K or the right arrow to trace across the letters to be erased. When you want to begin copying again, press ESC again and continue with step 4 above.

While POKE 33,33 is a life saver, there are a couple of commercial programs which make editing and entering programs a breeze. "The Applesoft Tool Kit" has a program which provides automatic line numbering and the "Global Program Line Editor" makes editing wonderfully easy. Both of these software packages are highly recommended.

Tip #2: Making Your Programs Easier to Read

One of the best books to read which will help to code programs so that they are more easily read is:

Nevison, John M. The Little Book of Basic Style: How to Write a Program You Can Read. Addison-Wesley, 1978.

This book is a superb resource for learning how to arrange program lines. However, not all the tips in the book work in Applesoft. One suggestion the authors give is to put visual space between individual segments of the program. To do this, use a line number followed by a colon. For example:

```
10 REM THIS IS THE END OF SECTION ONE OF A PROGRAM
20 :
30 REM THIS BEGINS THE SECOND SECTION OF A PROGRAM
```

Line 20 merely divides the two sections visually for the programmer. Line 20 does not affect the computer; it merely skips over that line. It will slow the program down very slightly, so if speed in the program is your objective, then don't use this tip.

Tip #3: Numbering Subroutines

We often use large round numbers such as 8000 or 9000 for subroutines so that they are easily remembered and used in a program. In this case, our program would call the subroutines by commanding GOSUB 8000 or GOSUB 9000. When we program those subroutines, start numbering them a few lines before the 8000 or the 9000 number. For example, start them at 7998 and 8998. On the two lines before the subroutine starts, title the subroutine and explain what the subrqutine does. When the program calls the subroutine, it will never have to read the title lines. Thus, the program will run faster and the program documentation will not be lost in the code.

```
7990 REM TEST STUDENT ERRORS SUBROUTINE
7999 REM THIS SUBROUTINE TESTS STUDENT INPUT
     TO SEE WHETHER IT IS NUMERIC
8000 subroutine starts here
```

Tip #4: A Caution Concerning the Text Page

If you print a letter in the 40th column of the text screen, Applesoft will generate an extra line space, even though you did not intend this to happen - thus the rule: Never print a letter in the 40th column of the text screen. You can spend many minutes puzzling over that extra line when you are sure you only put one PRINT statement between text lines. That 40th column is the culprit.

Tip #5: Printing on the HIRES and LORES Text Windows

One detail often overlooked is that to print in the HIRES text window, you must VTAB down 21 lines before anything becomes visible. This is not the case when printing in the LORES text window. No VTAB is needed in LORES.

Another problem is to use all four lines of the text window. The presence of a cursor takes two lines usually - a blank line and a line for the cursor, thus, only two lines are available without some tricks. To get three lines, merely put a semicolon at the end of the third line of printing. To get a fourth line, you will need to begin the line with some sort of input so that the cursor stays at the end of the printing.

Tip #6: Running a Lesson When the Disk is Booted

After your lesson is created, you will want to make a lesson disk which, when booted (or turned on), will immediately run the lesson. The student will not have to CATALOG, LOAD, or RUN the lesson. To do this, load your program after it is complete. Put a fresh disk in the drive. Type INIT _program name. When the new disk is booted, your lesson will serve as the HELLO program and will run immediately. If you have split your lesson into a number of pieces, then the very first program which calls all the others should become the HELLO program.

Tip #7: A Simple Protection Scheme

If you want to prevent students from pressing RESET on purpose or accidently to get out of the lesson or prevent them from listing the program or loading it or cataloging the disk, use the following line at the beginning of your program.

```
10 POKE 214,255 : POKE 1010,102 : POKE 1011,213 : POKE 1012,112 : CALL
   1002
```

Once these POKES are made and the CALL is executed, you will not be able to load, list, catalog or reset a program. If you try, DOS will merely go back to the beginning and run your program again. WARNING: Do not add this command until you have made a backup copy of your program. It will prevent you from changing your program just as it will prevent students from doing so. This scheme will work only until your students become computer literate. When a student learns how to break this scheme, use that student as your programming assistant.

Here is the method of getting around this POKE command. This will be a secret between the authors and you. Boot the System Master or other initialized disk. Catalog your lesson disk. Load the lesson which is protected. Then list. Now you can delete the security POKE line #10. Be careful not to run the program during this procedure, otherwise, the protection scheme will be back in place.

Tip #8: Using Quotation Marks on the Text Screen

Problem: When you use quotation marks within your strings you are only able to print up to the second quote mark.

Solution: You have to tell the computer to print a quotation mark a little differently than other characters.

To print a line that looks like this:

SHE SAID, "ISN'T THAT WONDERFUL?"

you have to code this:

```
10 PRINT "SHE SAID, ";CHR$(34);"ISN'T THAT WONDERFUL?";CHR$(34)
```

CHR$(34) is the only way to tell the computer to print a quotation mark.

If you want to use this feature often, you can store the command in a variable which is easier to type. For example:

```
10 QUOTE$ = CHR$(34)          or      10 Q$ = CHR$(34)
300 PRINT "SHE SAID, ";Q$;"ISN'T THAT WONDERFUL?";Q$
```

Tip #9: Turning the Printer On and Off While a Program is Running

If you wish to turn on or off the printer during the execution of a lesson, you will need to issue a DOS command. Try this program:

```
10 HOME
20 INPUT "TURN ON YOUR PRINTER NOW AND THEN PRESS RETURN";A$
30 D$ = CHR$(13) + CHR$(4)
40 PRINT D$;"PR#1"
50 PRINT "THIS LINE WILL PRINT OUT ON THE PRINTER"
60 PRINT D$;"PR#0"
70 PRINT "THIS LINE IS ON THE SCREEN - NOT ON THE PRINTER"
```

Tip #10: Breaking Up Programs into Smaller Segments

Sooner or later, programs become too large to be held in the available memory space. What commonly happens is that a whole chunk of your program will be missing when you try to run it. If this happens, you are OK if you have not re-saved the program on your disk. Before the tragedy occurs and you think you have to re-enter a large piece of the program, there are a few techniques that will help you anticipate the problem and deal with it. First, keep track of about how much memory is left so that you can get a module with its own subroutines into memory at one time. Use Tip #12 to help do this. Another solution is to break up the program into smaller but logical chunks - each saved as a separate program. Here's how:

<u>"DOSing Out" - or Running one program from within another</u>

When you don't want to or can't merge two programs, it is possible to exit one program and run another automatically with very little disruption of the lesson flow. Many commercial programs do this. During the lesson, the red light on the disk drive comes on occasionally. It is loading new pieces of the program. Here is how that is done.

Necessary commands at the point where you want to exit one program and run another:

```
10 D$ = CHR$(13) + CHR$(4) : REM STORE CTRL-D IN D$
20 PRINT D$; "RUN  next program name "
```

Note: Some authors recommend that line 10 need only be declared once in the program. The authors recommend that it be repeated every time you need to move out of the program. If you have used a GET statement or hires graphics, strange things often happen and the computer will not remember what is in D$.

Comments:

The "program name" above is the name of the program you wish to run from within a program. Both programs must be on the same diskette or one could be on another diskette in drive 2 (then type "RUN program name ,D2"). Remember, if you are going to run one program from another, the first program will be erased from memory and the second one will be loaded in. This means that to get back to the first program, or another one, the second program must also contain the necessary commands listed above.

Usually, all values in the first program are lost when the second program is run. For example, if you asked the student's name in the first program, it would not carry over to the second program nor will it be there when the student has returned to the first program. If you wish to share the information stored in the variables between programs, use the CHAIN command described later.

The advantage of running one program from another is that the two programs might be in different languages, or they could be too large (too many K) to store as a single program. Another advantage is that a large wait at the beginning of the program for the computer to load can be replaced by a number of shorter waits at various intervals in the program.

Commercial publishers also have a clever reason for having programs "DOS out" regularly. Such programs require that a disk be in the drive at all times. This means that the teacher cannot load up a number of computers with one program and have the class use the program simultaneously. Supposedly, this means that the school will buy more copies of the program. It's a nice plan to earn more money - it might also earn the publisher's way right out of a sale as educators look for more flexible software.

Module by module breakup

One solution is to have each module of the program be a separate program on the disk. In this case, there might be an introductory module which has the main menu. Any choice on the menu would DOS the program out to the appropriate piece. If this method is used, all subroutines called by a module will have to be a part of that module. This means that you might have the same subroutine duplicated a number of times in different programs. In the main menu program, check each choice made by the student and then DOS to the appropriate program. For example:

```
10 D$=CHR$(13) + CHR$(4)
40 IF A = 4 THEN PRINT D$; "RUN program name which matches option 4"
```

At the end of the option 4 program, the commands would run the main menu program:

```
1550 PRINT D$; "RUN program name of the main menu "
```

A final thing you can do when you DOS from one program to another is to let the user know what is happening. Try this: just before you DOS out, clear the screen with the HOME command, then print a message something like: "just a moment, please, I'm loading." This will prevent having a blank screen with nothing on it and the student becoming concerned because nothing seems to be happening.

Many books recommend a simple D$ = CHR$(4) without the CHR$(13). If your program uses the GET command, then a simple CHR$(4) won't always work. Adding the CHR$(13) guarantees 100% predictable results.

Carrying variables across programs -- (Thank you, Rick Jones)

If you want to carry the student's name or scores or other information stored in variables from one program to another you will have to use the CHAIN command. CHAIN is a binary program which loads and runs a second program without erasing from memory the arrays and variables of the first program. Authors often state that CHAIN is an Integer Basic Command that is not available in Applesoft Basic. This is true, but somewhat misleading. There is a Binary program on the Apple System Master Disk that allows you to simulate the CHAIN command in Applesoft. To use it, do the following:

Step 1: Use FID or FILEM to copy the CHAIN program onto your disk. The CHAIN program must be on the disk where you intend to link programs.

Step 2: When you get to the point in the program where you wish to DOS to another program, add the following program lines:

```
10 D$ = CHR$(13) + CHR$(4)
110 PRINT D$;"BLOAD CHAIN, A520"
120 CALL 520" name of next program "
```

Note: A520 is the memory location for CHAIN

Note: do not put any spaces or punctuation marks between 520 and the opening quote)

Cautions:

1. When you chain from one program to another, the first program is erased from memory and execution begins at the lowest line number of the next program.
2. FLASH and INVERSE work by altering the ASCII codes. Be sure they are reset to normal before you call CHAIN. Failure to do so will result in garbled commands creating unpredictable results and the possibility of

uncontrollable frustration when you discover that the command didn't work.

3. If there is an ARRAY to be passed between programs, be sure that it is dimensioned in, and only in, the first program that uses it.
4. If the chained programs use a defined function, be sure that it is defined in each program.
5. Be sure to standardize your variable names from one program to the next. In other words, if you use N$ for NAME in the first program, N$ should also be used in the next program.

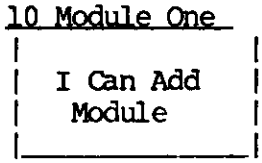
Sample Program Using CHAIN

```
Program "ONE"
   10 VTAB 10 : INPUT "TYPE YOUR FIRST NAME: ";N$
   20 HOME: D$ = CHR$(13);CHR$(4)
   30 PRINT D$;"BLOAD CHAIN,A520"
   40 CALL 520"TWO"

Program "TWO"
   10 HOME
   20 VTAB 10 : PRINT "HI, ";N$;", HOW ARE YOU TODAY?"
   30 PRINT : PRINT "I'M FINE NOW THAT I KNOW HOW TO REMEMBER YOUR NAME."
```

VTOC Changes

If you are going to break up the program into several pieces, then the line numbers on the VTOC should be the beginning line numbers of each module. It is a good idea to list the program name at the top of the module or program piece. For example:



Moving Memory

Another solution which may not require the segmenting of a program is to

move memory around in such a way as to make more room for a program. That may often be the most sensible solution, especially if your modules are already broken down into simple functional units. See the next tip for details.

Tip #11: Memory Maps and Moving Memory

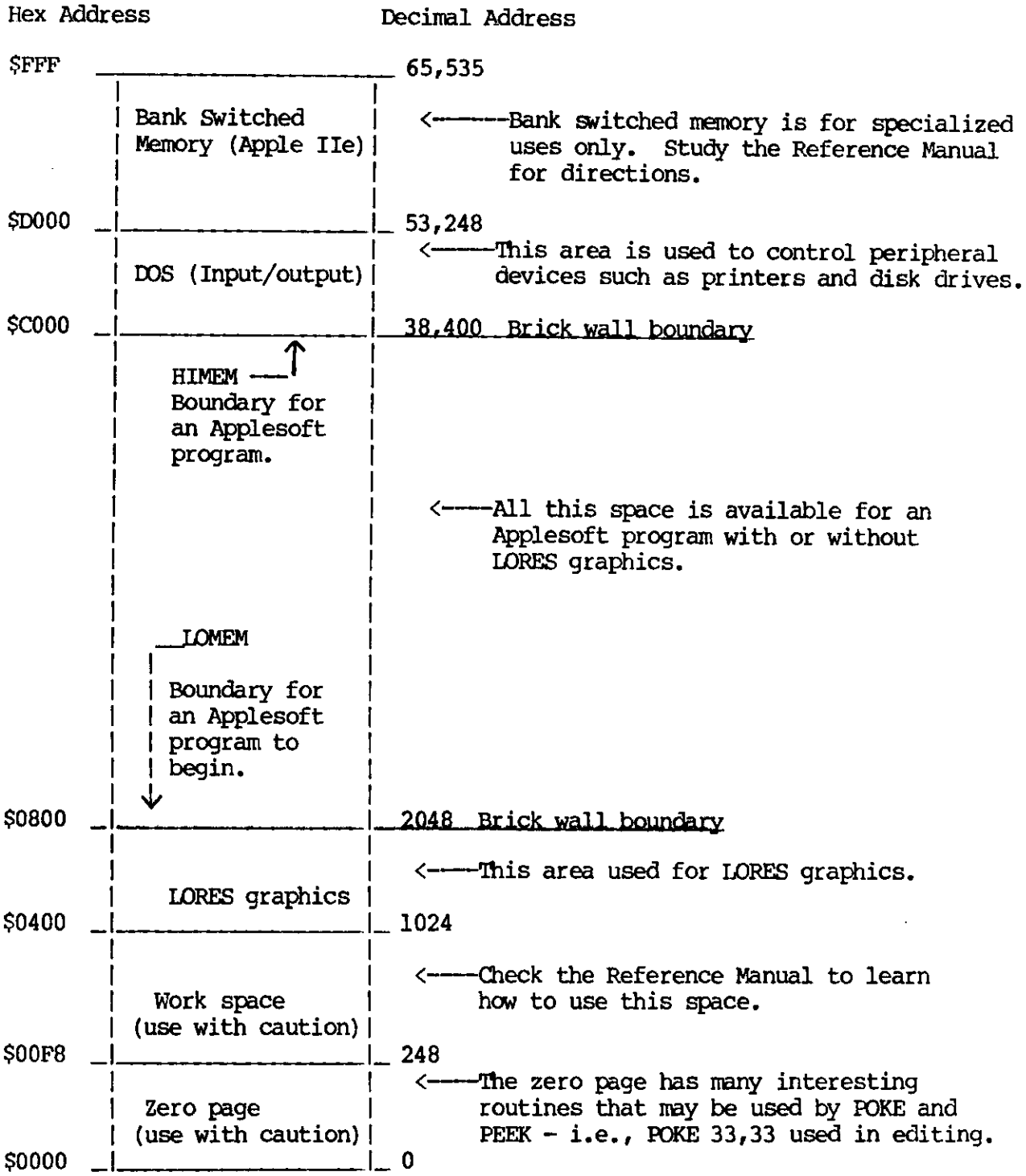
Memory is a mystery to most people, and it is usually not explained very well. Here is our attempt to take the mystery out of memory. One of the problems is that the memory of a computer is invisible. You can open the cover of the Apple and stare at the chips, but you will be no wiser. Our task, then, is to visualize what goes on in the mind of the Apple. Let us assume that our memory is a tall thin empty glass into which you can pour water (your program). Like water, your program will immediately start to fill up the glass (computer memory). Like a measuring cup, there are marks on our memory glass to tell us how full the glass is. But there is a major difference. We are not bound by gravity in the computer's memory, so we need not always fill the glass from the bottom up.

Spend some time studying the following diagrams. Usually, programming and reference books only present one diagram of memory. We have divided the diagram into two drawings. The first diagram illustrates two common types of programs - one is a program which uses neither LORES nor HIRES graphics and one which uses LORES graphics. The second diagram illustrates what happens when a program uses HIRES graphics and provides some solutions to some obvious problems.

MEMORY MAP WHEN USING APPLESOFT

WITH OR WITHOUT LORES GRAPHICS

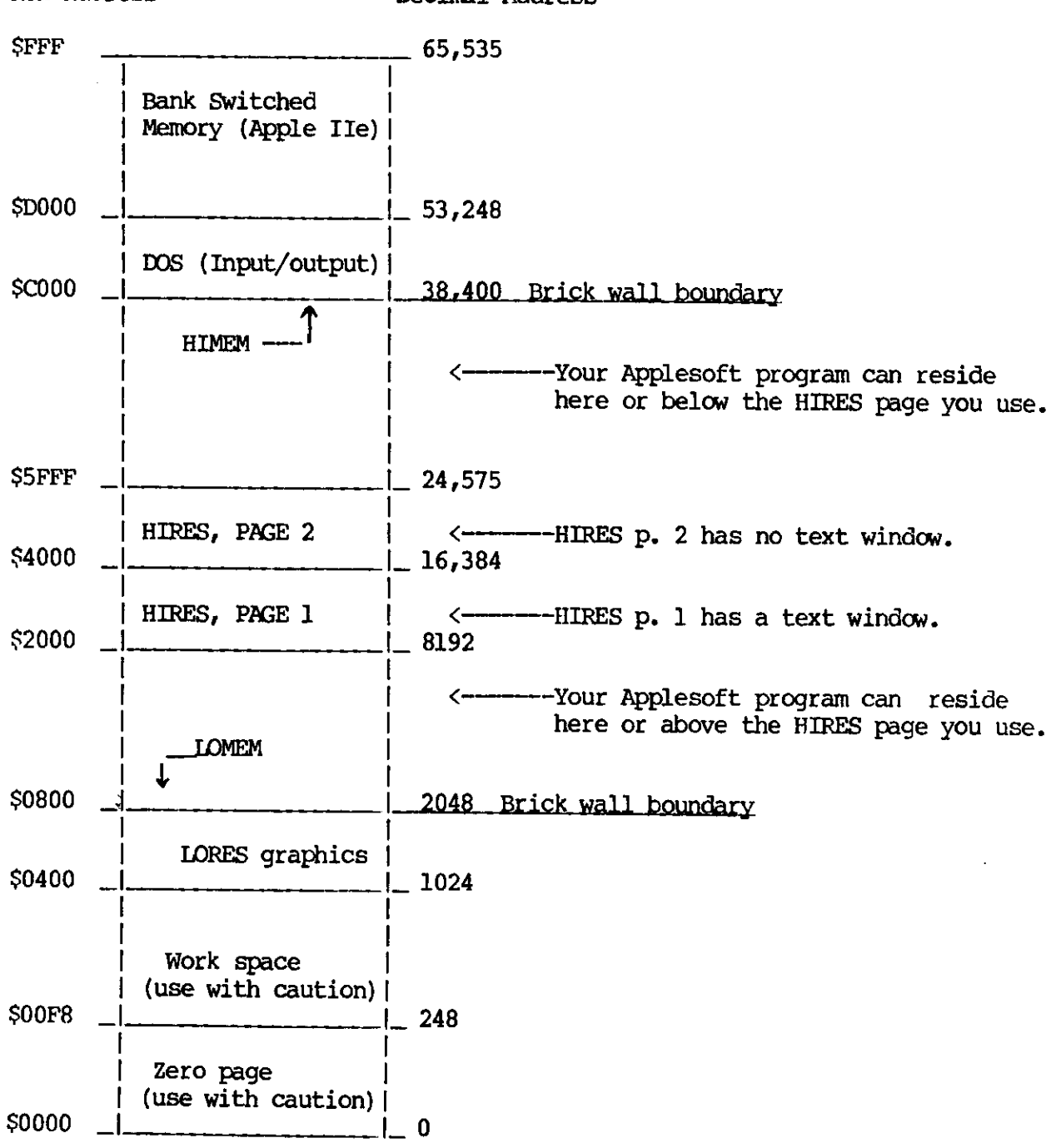
(Assuming 64K)



MEMORY MAP WHEN USING APPLESOFT

AND HIRES GRAPHICS

Because the space for the HIRES pages sits right in the middle of the space used for the Applesoft program, you must plan carefully or your graphics will collide with your programming lines and you will lose part of your program.



When you use HIRES graphics, your Applesoft program must be stored either below or above the HIRES page(s) you use. This means that if you change nothing, your Applesoft program will begin at 2048 and work its way up until it encounters a HIRES page in use. At that point, you have problems. There are a number of solutions:

1. Solution if you use only HIRES, page 1

   a. If you have a short program, move HIMEM down from 38400 to 8192. This will prevent your program from colliding with the HIRES graphic, i.e., you will get an "out of memory" message before any damage can be done. To use this solution, include the following lines in your control module and then DOS out to the program you have written.

```
 5 REM CONTROL MODULE
 8 D$=CHR$(13) + CHR$(4)
10 POKE 115,1
20 POKE 116,32
30 POKE 8192,0
40 REM NOW YOU CAN DOS OUT TO YOUR PROGRAM
50 PRINT D$;"RUN program name"
```

   b. The second solution is to move your Applesoft program above page 1 of HIRES and take advantage of the space from 16,384 to 38,400. This is much larger than the space you had below page 1. To do this, write a short program as part of your control module and then DOS out to your main program.

```
 5 REM CONTROL MODULE
 8 D$=CHR$(13) + CHR$(4)
10 POKE 103,1
20 POKE 104,64
30 POKE 16384,0
40 REM DOS OUT TO MAIN PROGRAM
50 PRINT D$;"RUN program name"
```

2. Solution if you use only HIRES, page 2

   a. If you have a short program, move HIMEM down from 38400 to 16384. This will keep your program from colliding with the HIRES graphic, i.e., you will get an "out of memory" message before any damage can be done. To use this solution, include the following lines in your control module and then DOS out to the program you have written.

```
 5 REM CONTROL MODULE
 8 D$=CHR$(13) + CHR$(4)
10 POKE 115,1
20 POKE 116,32
30 POKE 16384,0
40 REM NOW YOU CAN DOS OUT TO YOUR PROGRAM
60 PRINT D$;"RUN program name"
```

b. The second solution is to move your Applesoft program above page 2 of HIRES and take advantage of the space from 24575 to 38400. To do this, write a short program as part of your control module and then DOS out to your main program.

```
 5 REM CONTROL MODULE
 8 D$=CHR$(13) + CHR$(4)
10 POKE 103,1
20 POKE 104,64
30 POKE 24575,0
40 REM DOS OUT TO MAIN PROGRAM
50 PRINT D$;"RUN program name"
```

3. Solution if you use both HIRES page 1 and page 2

a. If you have a short program, move HIMEM down from 38400 to 8192. This keeps your program from colliding with the HIRES graphic, i.e., you will get an "out of memory" message before any damage can be done. To use this solution, include the following lines in your control module and then DOS out to the program you have written.

```
 5 REM CONTROL MODULE
 8 D$ = CHR$(13) + CHR$(4)
10 POKE 115,1
20 POKE 116,32
30 POKE 8192,0
40 REM NOW YOU CAN DOS OUT TO YOUR PROGRAM
50 PRINT D$;"RUN program name"
```

b. The second solution is to move your Applesoft program above page 2 of HIRES and take advantage of the space from 24575 to 38400. To do this, write a short program as part of your control module and then DOS out to your main program.

```
 5 REM CONTROL MODULE
 8 D$=CHR$(13) + CHR$(4)
10 POKE 103,1
20 POKE 104,64
30 POKE 24575,0
40 REM DOS OUT TO MAIN PROGRAM
60 PRINT D$;"RUN program name"
```

Tip #12: Determining Space Left on the Disk and Free Memory

Often, as your programs grow, you need to know how much free space there is on the disk. The program FID (old name) or FILEM (new name) on the System Master can tell us how many sectors are left on a disk - this feature is an option on the FID or FILEM menu. As a program becomes longer and longer, we also need to know how much more memory space (K) is available. If the program gets too large, you will have to break it into several smaller pieces. While you are programming, there are a couple of ways to find out how many K are left in memory.

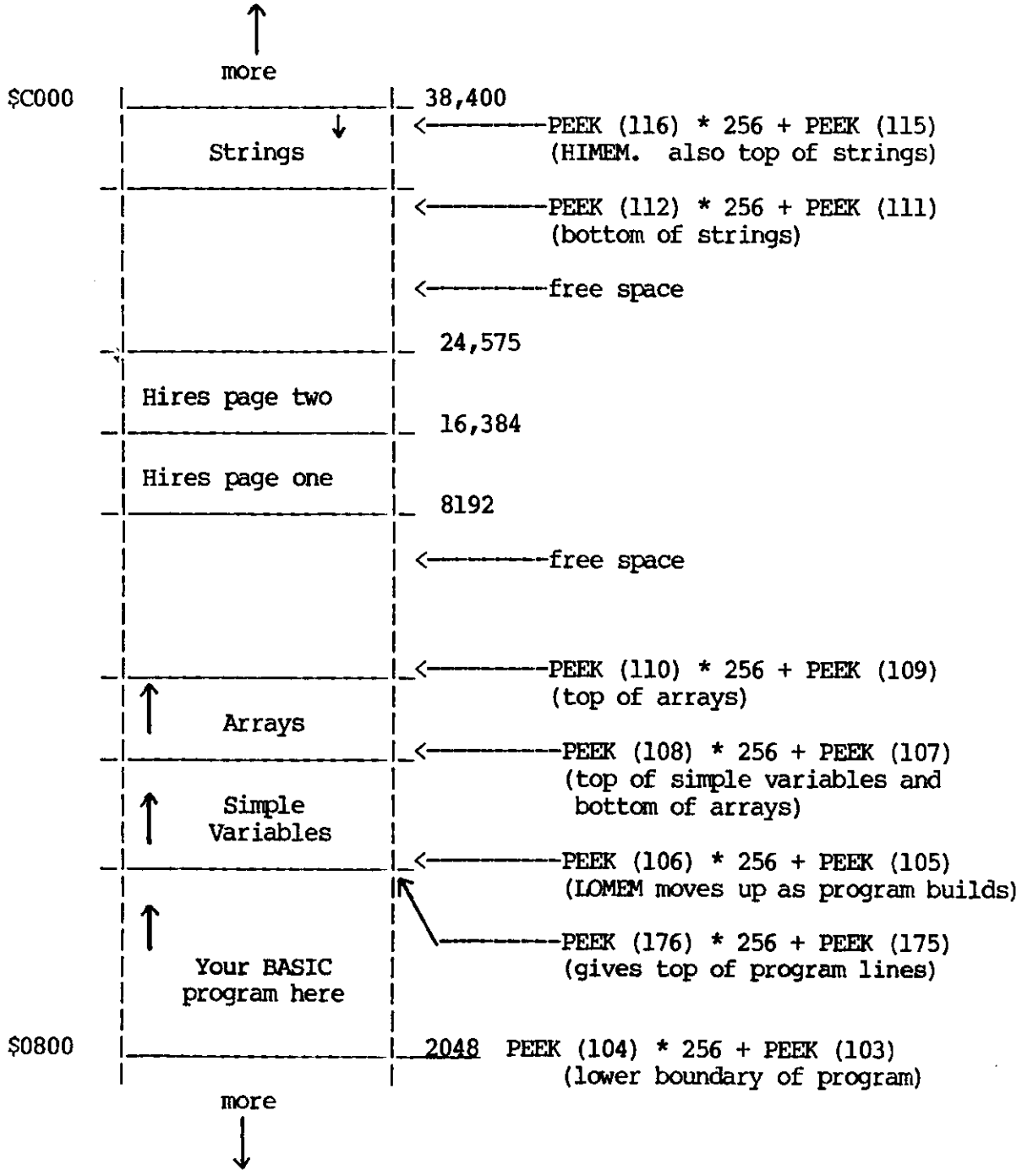
Method one:

Load your program and then type: PRINT FRE(0)
If the result is a negative number, then type: PRINT FRE(0) + 65536

Method two:

Study the following pointer address chart. A pointer is an indicator to the computer giving the location of a certain boundary. The pointers in this chart indicate the dividing line between strings, arrays, variables, the program, and free space.

POINTER ADDRESS CHART



If you understand the drawing above and the concept that memory is being filled up with every command you add to your program, you may find out at any given point how full the memory is. With the program loaded in memory you merely type in without giving a line number the command PRINT and any one of the PEEK statements from the chart above. The computer will reply with a number telling you where that boundary is. Let us say I want to know where the arrays end. I would type:

```
PRINT PEEK (110) * 256 + PEEK (109)
```

You can have even more fun if you include these statements as a subroutine so that anytime you run your program, it will automatically tell you how much memory you have used and how much you have left. Study the following program and then vary it to your needs:

```
   10 GOSUB 10000                          Put this line right at the first of
                                           your program.
   20 REM THE REST OF THE PROGRAM
      GOES HERE
 9999 END

10000 REM SUBROUTINE TO DETERMINE
      SPACE BETWEEN ARRAY BOUNDARY
      AND HIRES P. 1
10010 A = 256*PEEK(110)+PEEK(109)          This gives us the array boundary.
10020 B = 8191 - A                         B will now equal the amount of space
                                           left between the array boundary and
                                           8191 which is the lower boundary for
                                           hires p. 1 minus one.
10030 PRINT "THE TOP YOUR
      PROGRAM IS NOW AT ";A
10040 PRINT
10050 PRINT "YOU HAVE ";B;"
      MEMORY LOCATIONS BEFORE HIRES
      PAGE 1 STARTS."
10060 PRINT
10070 IF B < 0 THEN PRINT "SINCE THE
      ABOVE NUMBER IS NEGATIVE, YOU
      ARE USING THE WRONG SUBROUTINE
      TO DETERMINE THE MEMORY SPACE
      LEFT."
10080 FOR X = 1 TO 3000 : NEXT X           Here is a timer so you can read the
                                           message.
10090 RETURN
```

Tip #13: Using a Word Processor to Edit Your Program

Sometimes you will need to make some major changes in your program or edit it severely and you'll wish you could have the power of a word processor to do it. You can. If you use Apple Writer IIe or another word processor which uses normal Apple DOS text files to store its data on a disk, you are in luck. The idea is to transform your program into a text file, then call that text file into the word processor. You may then edit the program and resave it as you would any word processed file. You then get out of the word processor back into DOS and use the command EXEC (execute) to change the text file back into an Applesoft program. It really works!

Here in more detail are the directions.

1. You should have an Applesoft program on a disk you wish to edit. Be sure you have a backup of your program just in case something strange happens.

2. Load the program into RAM. List it just to make sure it is there.

3. Now add the following lines at or close to the beginning of the program but do not save these lines as a part of the program. These lines will only be temporary.

```
1 REM CHANGE FROM APPLESOFT TO TEXT FILE PROGRAM
2 D$ = CHR$(4)
3 PRINT D$;"OPEN ___text file name___    (not the same name as the
  Applesoft program name)
4 PRINT D$;"WRITE ___text file name___   (the same name as in line 3)
5 POKE 33,30
6 LIST
7 PRINT D$;"CLOSE ___text file name___   (the same name as in line 3)
8 TEXT : END
```

   We used line numbers 1-8. You can use any line numbers that don't conflict with your program. You can even put the whole mini-program on a single line number.

4. Now you have your Applesoft program in RAM with an extra little program added on the screen but not stored on the disk. Now type RUN (not

RUN program name !!!). The disk drive light will come on and your program will be converted to a text file.

5. Now catalog your disk. You should have both the original Applesoft program on the disk and a text file listed under the name you called it. If you have made a mistake in typing the program listed above, you will have a text file on your disk but it will be only one sector in length. Reason tells you that if your Applesoft program was very long, it could not fit into a one sector text file. If that happens, delete the textfile and try again.

6. Once you have the text file stored on the disk, boot up Applewriter IIe. Load the text file into Applewriter just as you would load any other file: Ctrl L - program name.

7. Edit the text file using Applewriter and save your changes as you usually do.

8. Now you are ready to change the program from a text file back to an Applesoft file. Get out of Applewriter and then boot the disk with the text file on it.

9. As soon as you get a cursor, type EXEC text file name

10. Magically, you will get an applesoft program on your disk named the name of the text file.

Readers may be interested to note that this technique was used to print out the Apple Demo program in chapter 2 of this book. The Applesoft program for Apple Demo was converted into a text file and then brought into Apple Writer. All the marginal comments were then added and the result was printed out in camera ready copy. Many hours of typing and proofreading were saved.

Tip #14: Clearing the Screen in LORES Graphics

Programming manuals usually recommend that to clear the screen from LORES graphics back to the text screen use: TEXT : HOME. Following this suggestion creates a bothersome flash of symbols on the screen. To eliminate this flash, try clearing the graphic screen to black first, then home. Use GR : HOME in the place of TEXT : HOME.

CHAPTER FIVE

GRAPHICS AND TEXT

The field of graphics for the Apple has seen an explosion of new ideas and commercial programs to help the designer of educational lessons. In this chapter, we will give our best suggestions, knowing full well that daily, new techniques may make some or even all that we say obsolete.

There are a number of commercial packages which assist in the creation of both HIRES and LORES graphics. Some of these packages include "Graphics Magician," "The Complete Graphics System," "The Power Pad," "The Koala Pad," and "Alpha Plot." The reader is urged to explore these packages. Some graphic-assist programs require peripherals such as graphics tablets and joysticks for drawing. Others use only keyboard controls. Such tools can make the creation of graphics and animation an enjoyable rather than tedious task.

Do It Yourself Graphics

Many times, you may wish to program your own graphics and not rely on a commercial graphics package. The following tips are designed to supplement current manuals dealing with graphics on the Apple. The first group of tips deals with LORES graphics and the second group deals with HIRES graphics.

Tip # 1: LORES Graphic and Text Screen Grids

Several types of grids and hints are given here to make the planning and programming of LORES graphics and text screens easier.

How Long is 40 Spaces?

Often as you are programming you will type the command:

```
10 PRINT "
```

If you start printing words, how will you know when you have printed 39 characters signaling the end of the text screen (remember, we have advised not to print in the 40th space)? A simple way to know is to type letters until the end quotation mark would be exactly under the beginning quotation mark on the screen. For example:

```
10 PRINT "I can keep print words until the last quote is under
the first"
```

The Text Screen

A copy of the text screen has been provided on page 5-6 which includes shading at every five columns and rows. This will help you plan and center your text.

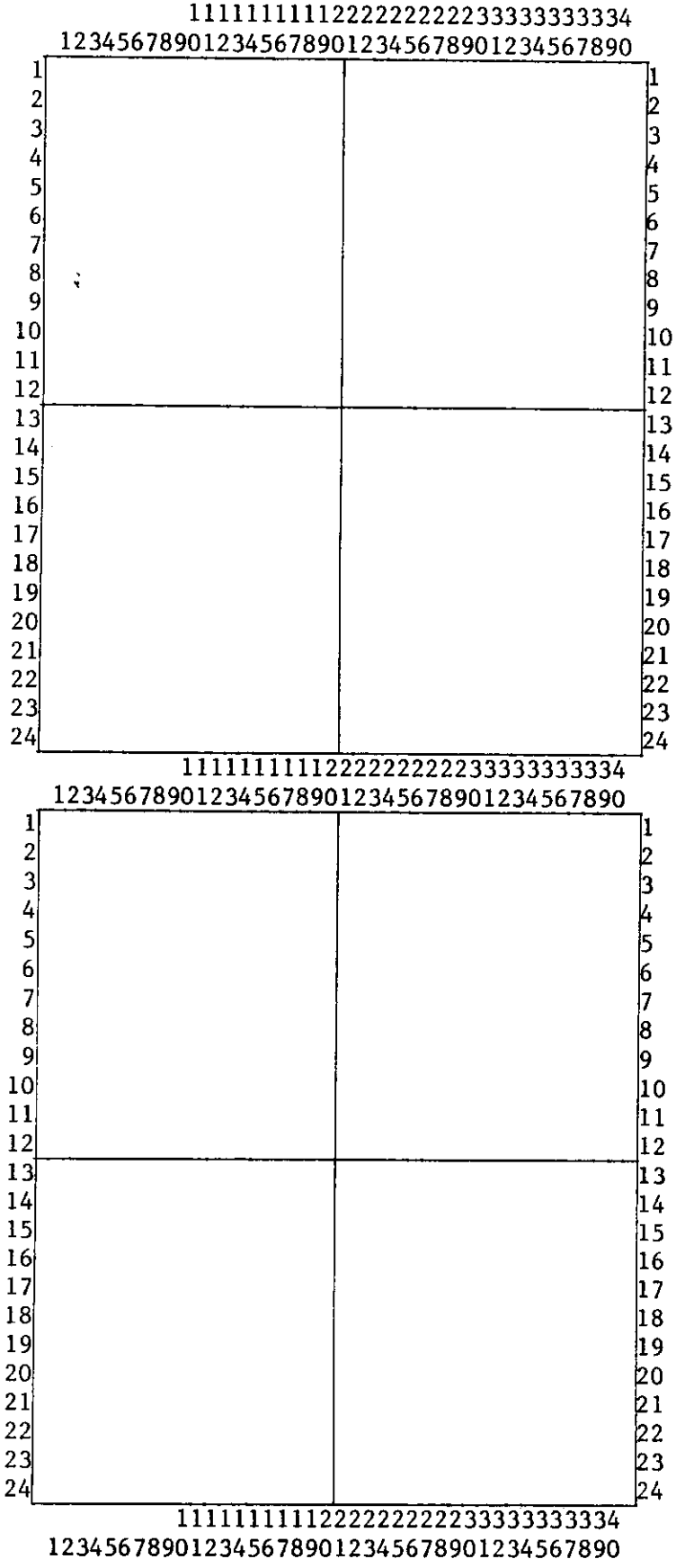
Text Screen Typing Guides

If you like to plan your text screens using a typewriter, you may use the two guides provided. One guide is for use with elite or 12 pitch typewriters, the other for pica or 10 pitch typewriters. To use the guide, make a photocopy of the page, insert it in the typewriter, center the crossed lines with your typewriter's vertical and horizontal line guides - then each letter you type will fit exactly in a text screen position. The guide makes it easy to plan for centering, line lengths, and vertical position on the screen.

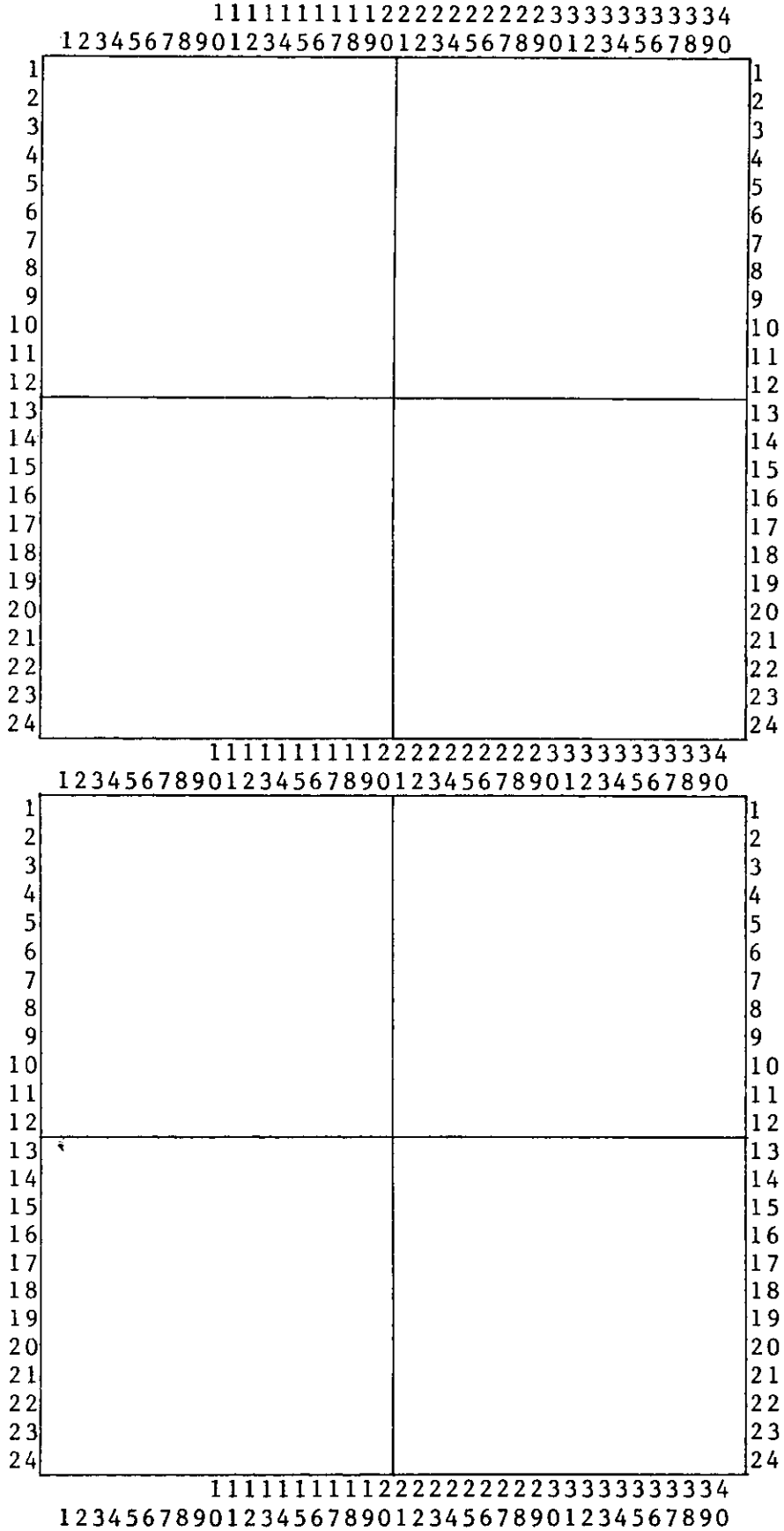
LORES Graphics Screen

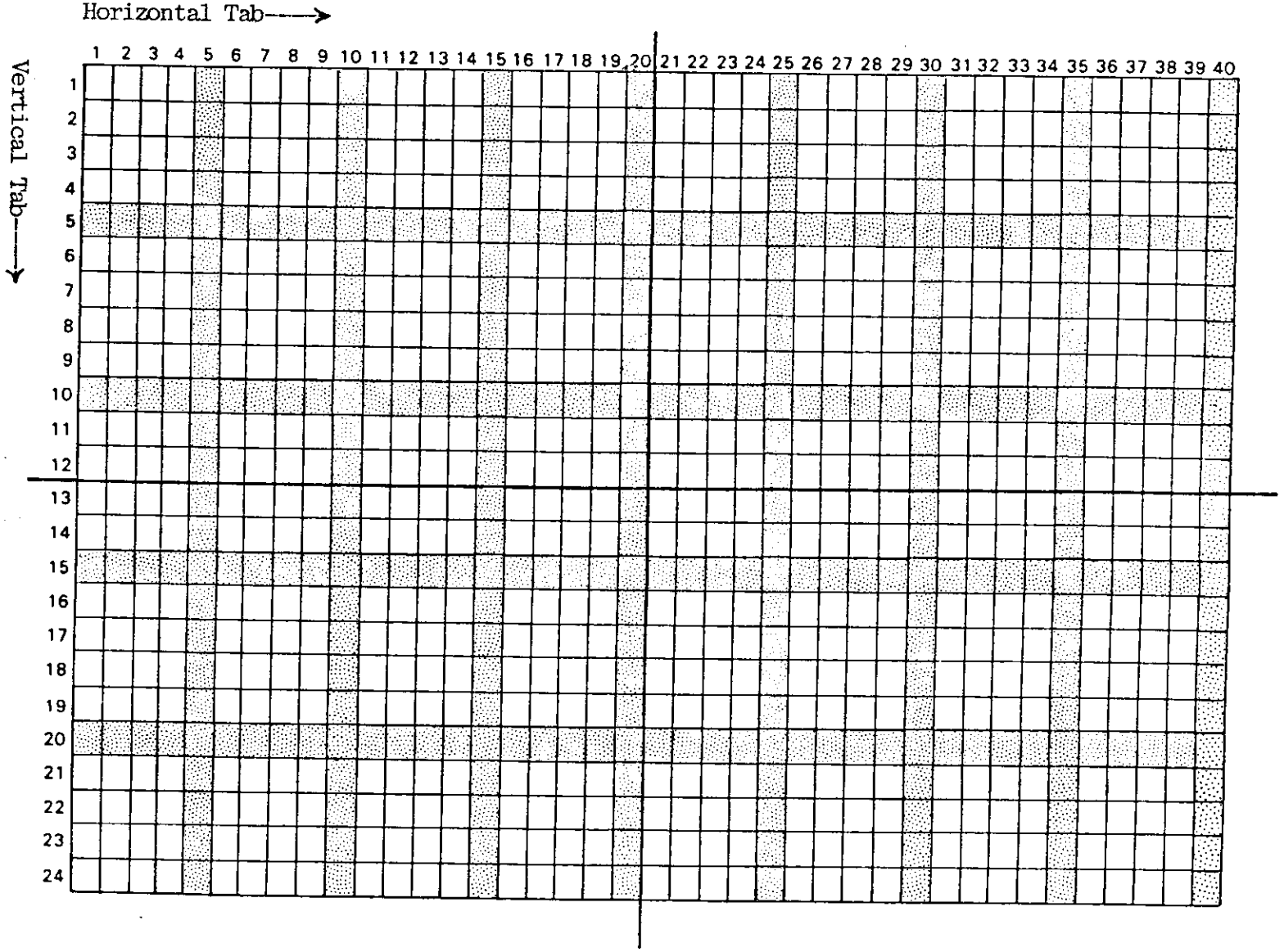
A copy of the LORES graphic screen has been provided on page 5-7 which includes shading at every five columns and rows. This will help you plan and center your graphics on the screen.

The third LORES grid page is for those of us who make many mistakes and don't want to use a whole sheet of paper every time we experiment. Six LORES screens are provided. If you use a medium point magic marker, you can fill one square of the grid with one swipe of the pen. You can experiment six times and waste only one sheet of paper.

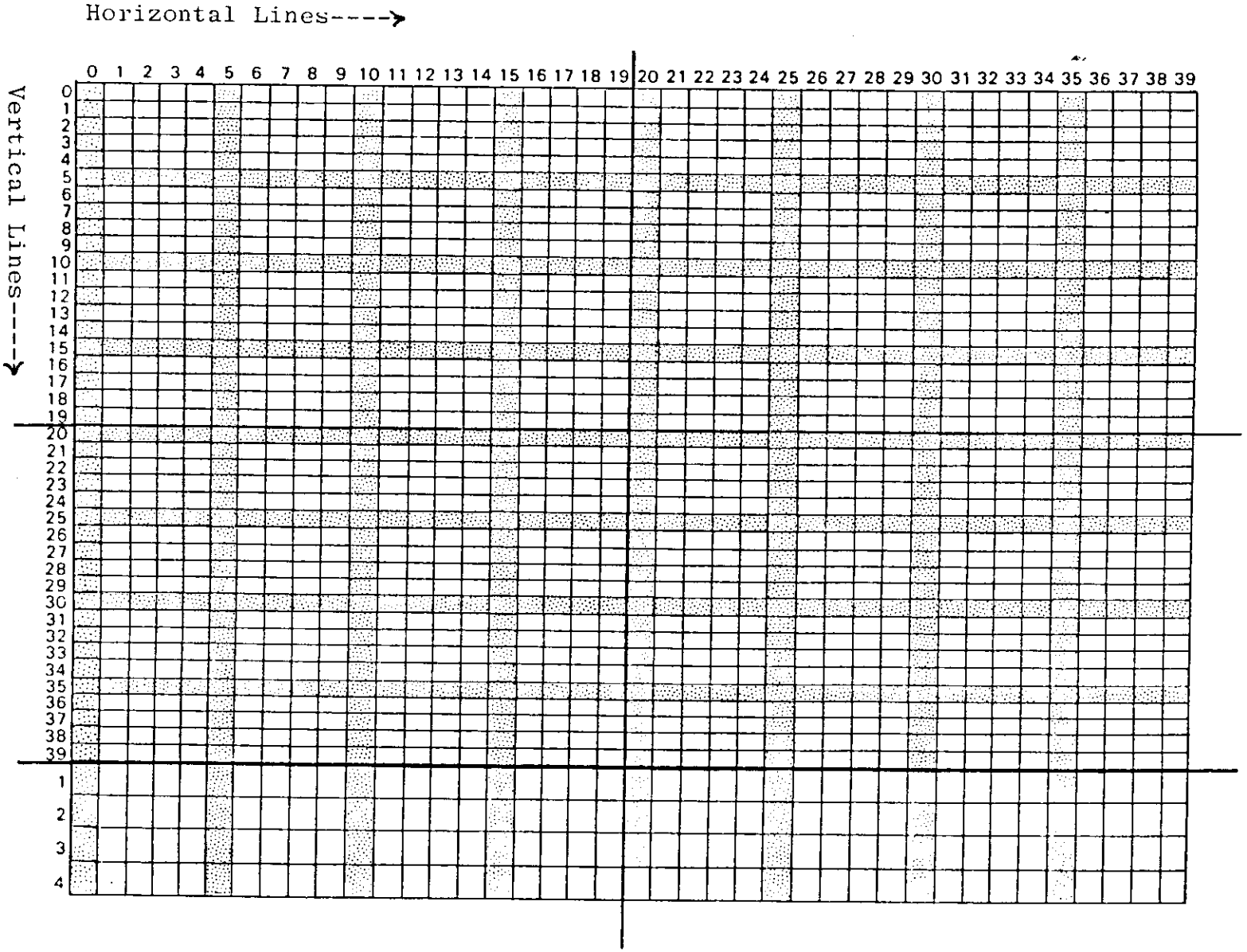

1111111111222222222233333333334
1234567890123456789012345678901234567890
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
1111111111222222222233333333334
1234567890123456789012345678901234567890
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
1111111111222222222233333333334
1234567890123456789012345678901234567890

Text Screen Typing Guide - PICA



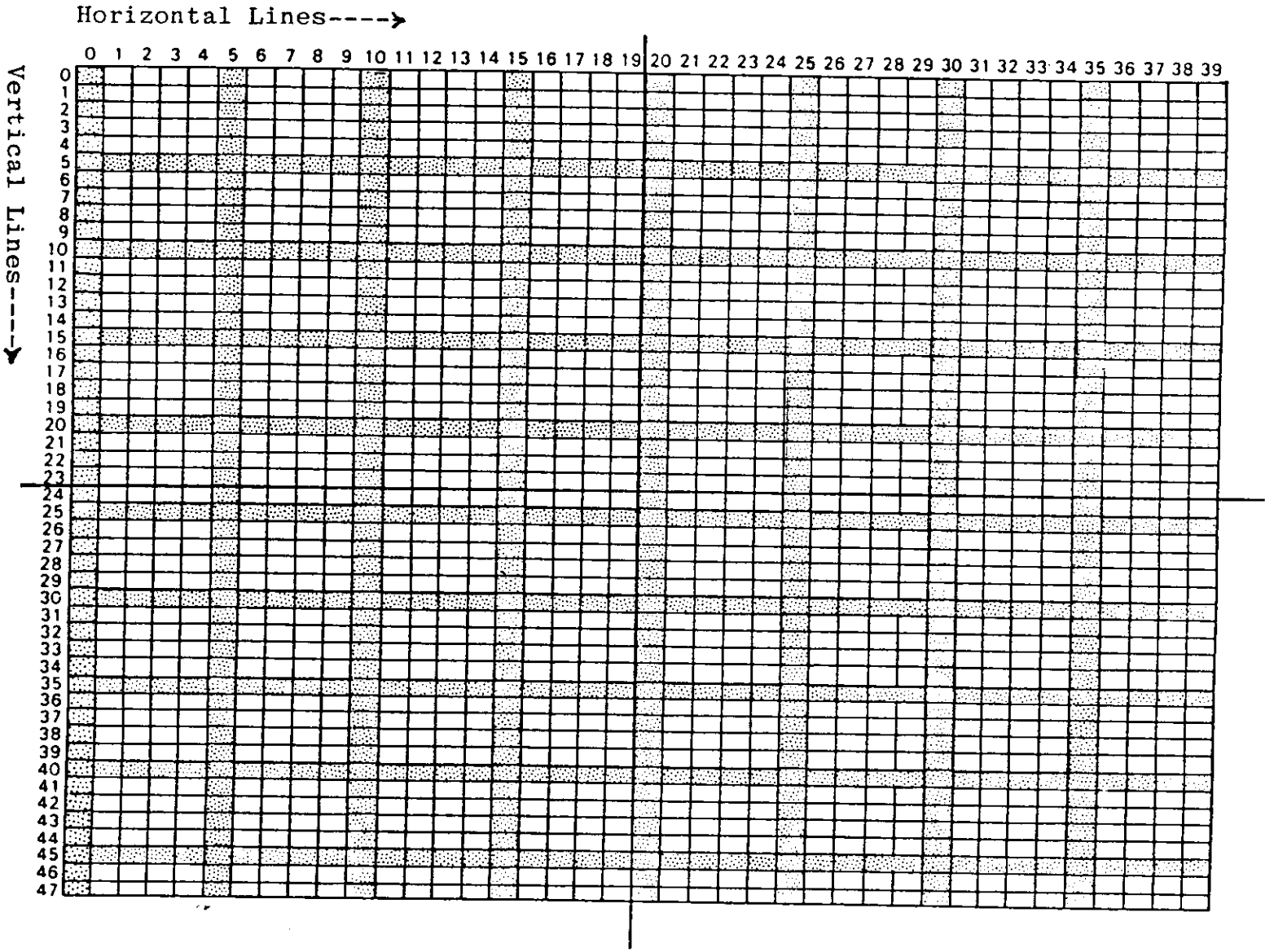

Horizontal Tab
Vertical Tab
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

TEXT SCREEN



LOW RESOLUTION GRAPHICS SCREEN

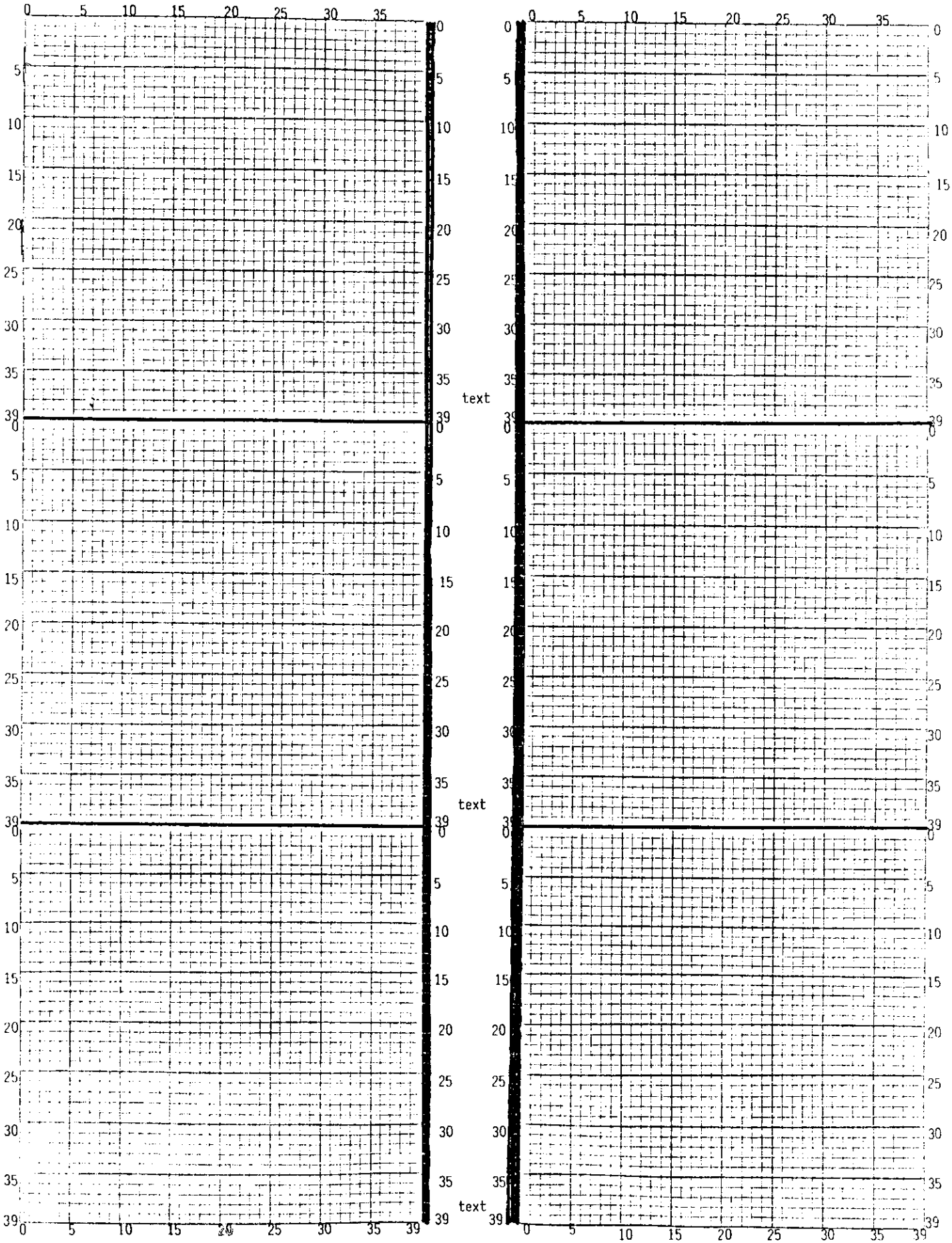
With Text Window



LOW RESOLUTION GRAPHICS SCREEN

Full Screen

APPLE LOW RESOLUTION SCREEN (please copy)



Tip # 2: Finding Designs for LORES Graphics

We have found that many students use counted cross-stitch books to help them design LORES graphic pictures. This works extremely well except for one problem: the Apple blocks of color are not square - they are rectangular. This means that your picture, if entered exactly as the cross-stitch book suggests, will turn out wider than it is tall and will be somewhat distorted. If you adjust for this factor, using designs is a great time saver.

Tip # 3: Debugging Graphics

One of the more bothersome problems with typing in the commands for graphics is that some typing mistake will be made and you will have one little square in your drawing that is the wrong color or out of place. How do you find it? We have two suggestions:

1. Put numerous remark statements in your program lines, i.e., REM ROOF; REM FLOWERS; REM SKY; REM MOVE CAR ACROSS SCREEN... This will help you locate problem sections faster.
2. As you type in the commands, run your program often - every several lines. You will be able to check the few lines you just entered to see that both color and position are correct. Instead of clearing the graphics screen everytime you run the program, just type your program lines in the text window. Much grief will be eliminated.

3. For complex designs, write your code out on paper before you enter it in the computer. This makes it easier to type in later.
4. Use FOR/NEXT loops for repetitive drawings, i.e.,

```
FOR X = 5 TO 10
VLIN 5,X AT 10
NEXT X
```

It is easier to correct height and width in such loops than correct many lines of code.

Tip # 4: Animation on the LORES Screen

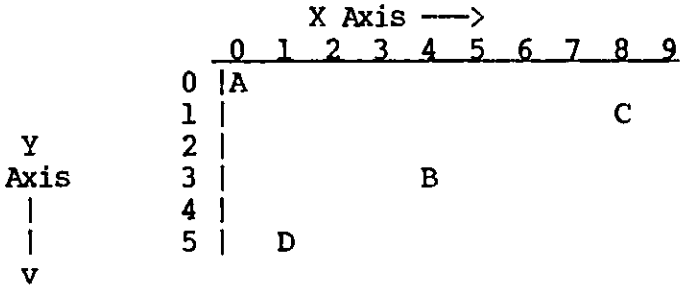
If you are going to animate computer graphics you should understand the general principle of animation in film. In a cartoon, thousands of drawings must be made to make the characters move. If you want Mickey Mouse's hand to move from his side to overhead, you must have numerous pictures showing the arm in positions from straight down, to outstretched, to raised overhead. Each of these pictures is shot with a camera and then flipped fast before our eyes. The principle of persistence of vision tricks our eye into believing that Mickey's arm really moved.

The same principle is used to animate LORES graphics. Suppose we have a car that we want to move across the screen from left to right. The car will be the same shape and color in each position and the background will be blue sky. The steps will be:

1. Draw the sky and the car at the left of the screen.
2. Re-draw the sky (covering over the car) and re-draw the car one position to the right.

3. Repeat the procedure until the car is at the right side of the screen. Read and think about the above three steps until you can visualize in your mind what should happen.

Now let us say you will have to have the car in ten different positions from left to right in order to make the animation look fairly smooth and not jerky. Will you have to plot the car ten different times? Fortunately not if you know something of the coordinate system you were taught in high school algebra. The coordinate system on the Apple is somewhat similar. Visualize the following axis having all positive numbers:



To give the address of a point on the grid we must give that address in relation to the zero point (where the two axes cross). Thus our A in the graph above is at 0,0. We give the X axis value first and the Y axis value second. Point B is at 4,3 (disregard negative values) or we can say B is at X+4, Y+3 from the zero point. Point C is at X+8, Y+1. Point D is at X+1, Y+5. Recall that the LORES screen starts counting from the upper left hand corner in the drawing above.

Now if we were trying to draw a small box on the LORES screen in the upper left hand corner of the screen, we could give two different sets of commands:

```
10 HLIN 0,4 at 0
20 HLIN 0,4 at 1
30 HLIN 0,4 AT 2
40 HLIN 0,4 AT 3
50 HLIN 0,4 AT 4
```

Or we can write the same thing in Xs and Ys:

```
10 X = 0 : Y = 0
20 HLIN X,X+4 AT Y
30 HLIN X,X+4 AT Y+1
40 HLIN X,X+4 AT Y+2
50 HLIN X,X+4 AT Y+3
60 HLIN X,X+4 AT Y+4
```

A still easier way is using the FOR NEXT commands

```
10 X = 0: Y = 0
20 For Y = Y to Y+4
30 HLIN X,X+4 AT Y
40 NEXT Y
```

Now we can move our box anywhere on the screen by changing the values of X and Y:

```
10 X = 5: Y = 10
20 For Y = Y to Y+4
30 HLIN X,X+4 AT Y
40 NEXT Y
```

Now let's put the drawing of the box in a subroutine and call it whenever we want it. Try this:

```
10 GR
15 REM DRAW BOX IN UPPER LEFT OF SCREEN
20 COLOR = 2
30 X = 0: Y = 0: GOSUB 220
40 :
50 REM DRAW BOX NEAR THE CENTER OF THE SCREEN IN A DIFFERENT COLOR
60 COLOR = 4
70 X = 20 : Y = 20 : GOSUB 220
80 :
90 END
218 :
219 REM DRAW BOX
220 For Y = Y to Y+4
230 HLIN X,X+4 AT Y
240 NEXT Y
250 RETURN
```

Now we are ready to attempt to move the object across the screen. Let us move our box across the top of the screen from the left edge to the right edge. We will move the box one position at a time, changing the background color and replotting the box each time to make it disappear before the next box is plotted. In our example, the background color is black and our box is magenta.

```
10 GR
20 REM SET THE Y POSITION TO 0 WHICH WILL REMAIN CONSTANT
30 Y = 0
40 REM USE A FOR NEXT LOOP TO CHANGE THE X POSITION
50 FOR X = 0 TO 15
```

```
 60 COLOR = 2 : GOSUB 220 : REM DRAW COLORED BOX
 70 COLOR = 0 : GOSUB 220 : REM ERASE BOX
 80 NEXT X
 90 END
218 :
219 REM DRAW BOX
220 For Y = 0 TO 4
230 HLIN X,X+4 AT Y
240 NEXT Y
250 RETURN
```

Warning: None of our box can be printed outside the limits of the screen. Our X value must never be more than 39 or the program will stop and we will get the error message: "ILLEGAL QUANTITY ERROR."

Tip # 5: Using HIRES Graphics

Using HIRES graphics with the Apple computer is quite different than using LORES graphics. It also comes with its own unique set of benefits and problems.

One of the biggest problems is that the Apple computer was designed so that the part of memory designated to handle the HIRES graphics mode is right in the middle of programmable memory. (See Memory Mapping in chapter 4: tip #11) This is a distinct disadvantage in that programs must be short. Breaking up large programs is necessary. (See tip # 10 in chapter 4)

There are several ways to draw pictures using the Apple's HIRES capabilities. First there are the Apple HIRES commands (HPLOT, HCOLOR, HGR, HGR2). These commands are good to use when the HIRES picture is simple, i.e., using straight lines, or using mathematical formulas to plot curved lines. A disadvantage of using this method from your BASIC program is that the commands take up a great deal of the precious memory space left for your BASIC program. A way to get around this problem is to write the code and run the program to

draw the picture that you want, and then BSAVE the picture into a binary, screen-image file that can be called into a BASIC program. To do this, run the program that draws your picture and while the picture is still on the screen, issue this command:

BSAVE picture name,A$2000,L$2000

(to draw on HIRES page 1)

or

BSAVE picture name,A$4000,L$2000

(if you want to draw on HIRES page 2)

(the A means the address and the L is the length of the picture)

If you catalog your disk after the above command has been issued, there will be a binary file with the same name you called your picture. To use the picture from your BASIC program, all you need are two commands:

HGR or HGR2

BLOAD picture name,A$2000 or BLOAD picture name,A$4000

There are several very good packages that aid the programmer in creating very professional looking HIRES graphics. One of the greatest features of these packages is the ability to fill-in shapes you have created with almost any color imaginable. You don't have to be a programmer to create the pictures but you have to know a little about programming to be able to use them in your programs.

Some of the best packages are the Graphics Magician by Penguin Software, Alpha Plot by Beagle Brothers, and The Complete Graphics System by Penguin Software. In addition, VersaWriter, the Apple Graphics Tablet, the Power Pad and the Koala Pad are important mechanical tools. Each of these software and mechanical aids is easy to use if you practice.

Alpha Plot's biggest advantage is its ability to be used with only the keyboard as an input device. The other software listed in the above packages require paddles, tablets, or joysticks in addition to the keyboard to draw pictures. The tablets have an advantage over the joystick, keyboard and paddles in that you can "trace" the picture that you want to draw as long as you have it in the correct scale. The additional peripheral devices seem to be well worth the investment because of their extended capabilities.

When you use a peripheral device such as the Koala pad, saving one picture requires 34 sectors on the disk. This is a major limitation when trying to save more than a few pictures on the disk. Luckily, there are several scrunch programs available which will compress these 34 sector pictures down to 7-15 sectors. One of these scrunchers is on the Alpha Plot disk, another commercial program is named "Picture Packer." Watch for other programs as they become available. Before you can use the "scrunched" picture in your program, you must "unscrunch" it using the directions in the package.

Versawriter, Alpha Plot, and The Complete Graphics Systems all have text writing capabilities built in that are superior to the Graphics Magician. They allow you to change the size, direction, color and spacing of the letters that you type onto a HIRES picture. There are several companies that offer disks which contain various fonts (differing lettering styles) to be used on HIRES pictures. Some of these are Fontrix, Applesoft Tool Kit, and Apple Mechanic.

You can add text, using these different fonts to pictures created with any of the packages or pictures you have created using the HIRES commands as long as they are saved in the 34 sector format. To add different styles of text to a picture:

1. Save a HIRES picture on the disk in the 34 sector format (see directions on the previous page)

2. Load one of the commercial font programs and follow the directions for placing lettering where you want it on the picture you created.
3. Re-save the program on your disk in the 34 sector format.
4. "Scrunch" the picture if you desire.

Versawriter's strongest point is one of its drawing features. It allows you to draw lines between two points like the Graphics Magician and Alpha Plot and, in addition, it has a "pen up" and "pen down" feature that allows you to draw curved lines of varying heaviness. You can draw a wide, heavy line or a light, narrow one. It also has a reverse color feature that is nice (Alpha Plot has it too). You can draw with white on black; then if you don't like it, change it to black on white.

The Graphics Magician is by far the nicest package of the three for several reasons. First, the Graphics Magician saves picture drawing commands rather than screen images. This means that each picture you draw takes up very little space on a disk. Even the most complex pictures take less than 5 sectors. This means that you can put up to one hundred pictures on a single disk. It also means that the pictures draw faster than the 34 sector type.

Alpha Plot and Versawriter provide no easy way to erase mistakes that you make while drawing. If you draw something over a part of the screen that shouldn't have been drawn, too bad, start over. The Graphics Magician provides two ways for editing your pictures that are fast, easy and very handy. You can delete steps that you have made so if you don't like the last step, just press the D key and delete it! What you have drawn to that point will still be intact.

Graphics Magician also allows for pictures to be laid, one on top of the other without erasing what is already there, like using transparencies. This can be an extremely useful tool in educational software. It also has an animation system that is more for the advanced programmer but is not impossible for the beginner to understand. Educational programs must compete with exciting arcade games so the more graphics we can put into the educational programs the better.

Tip # 6 : Instant Graphics

You may want HIRES graphics to pop onto the screen instantly rather than draw slowly. A couple of simple PEEKs and POKEs along with a little fancy programming will do the trick. First you have to understand that just because you cannot see a picture on one of the HIRES screens, it doesn't mean that a picture is not there. A HIRES picture will only disappear after a HGR or HGR2 command. For example, you are looking at a picture from page 1 and you want to have a picture ready to show from page 2 immediately upon some preset condition. Here is a sample program to handle that:

```
10 HGR                          Set page 1 graphics
20 BLOAD picture #1,A$2000      Load a picture and show it on page 1
30 VTAB 21                      Print something in the text window
40 PRINT "THIS WILL APPEAR
   IN THE TEXT WINDOW OF PAGE 1"
```

Now while our learner is looking at the picture on page 1 and reading what the text window has to say, we'll draw the second picture on page 2. The order in which the commands are executed is the most important part.

```
50 POKE 230,64                  This poke says, draw the next
                                picture loaded on page 2.
60 BLOAD picture 2,A$4000       Load the picture that you want on
                                page 2 but don't show it.
70 GOSUB 1000                   Print a page turning command in the
                                text window of page 1.
80 POKE -16299,0                This poke switches you from page 1
```

```
                                    to page 2 without clearing either
                                    page - instantly.
   90 POKE -16301,0                 Add the text window to page 2
                                    (POKE -16302,0 would take it away)
** 95 GOSUB 1000                    Hit the space bar when you want
                                    to go on.
  100 POKE -16300,0                 Look at what was previously drawn
                                    on page 1.
  999 END
 1000 REM SUBROUTINE FOR SPACE BAR
      :
      :
 1050 RETURN
```

**If you wanted to draw a new picture to page 1 while looking at page 2, then between 90 and 95 add these lines:

```
   91 POKE 230,32                   This will draw the next picture on page 1
   92 BLOAD picture 3,A$2000        This tells the computer which picture
                                    to draw on page 1
```

Tip # 7 : Drawing on the HIRES Screen

When you are planning a line drawing for a HIRES screen, it is hard to find a piece of graph paper that is 280 by 160. The following page of graph paper is one eighth of the HIRES screen. It can be used in a number of different ways. If your picture takes up only part of the screen, just draw your figure on the

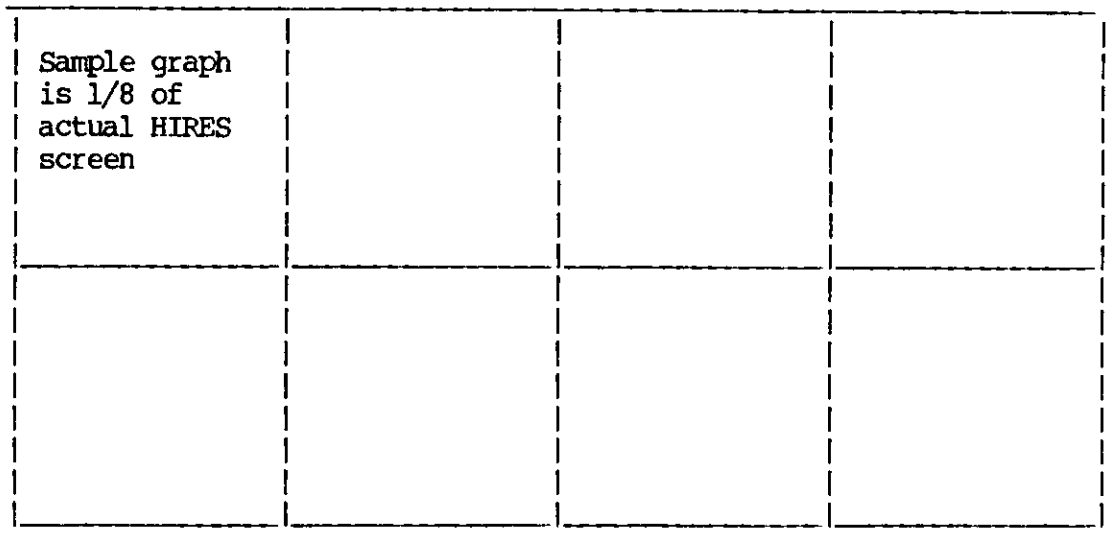
graph paper and then figure out the coordinates of where to place it on the screen. You can use one or several graphs pasted together for this purpose.

Another technique is to let each square on the graph equal five dots on the screen or ten - whatever will help you visualize what you need.

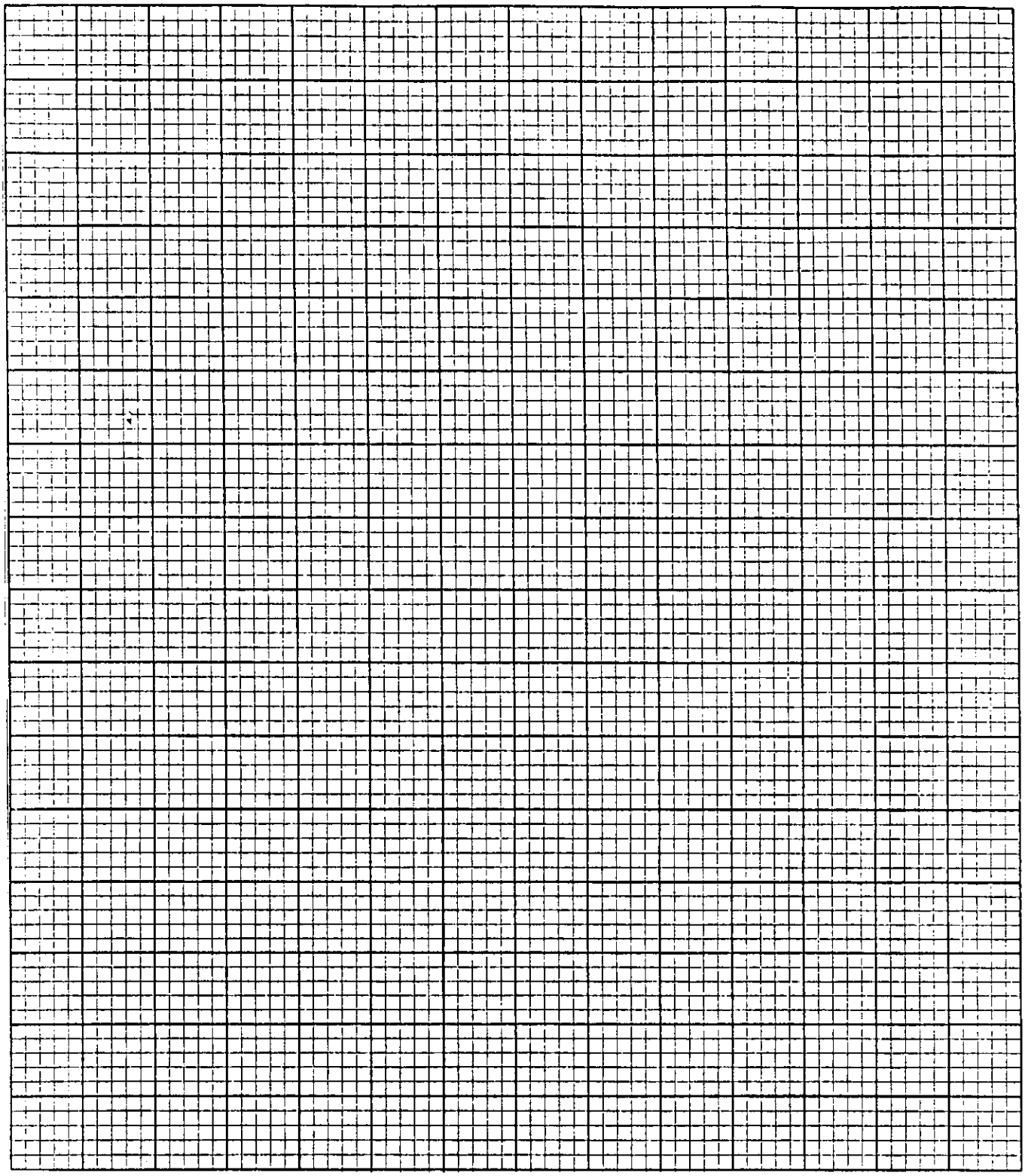
If you need to see the entire HIRES screen, you will have to piece 8 of the following graphs together and then add the row and column numbers.

This graph can also be a handy tool in planning the colors you can and cannot use in certain columns of the HIRES screen. Even numbered columns can display black, purple or blue (even color numbers 0, 2, 4 and 6). Odd numbered columns can display black, green or orange (odd color numbers 0, 1, 4 or 5).

Sample graph:



1/8 of HIRES SCREEN (4 across, 2 down) please copy



CHAPTER SIX

SIMPLIFIED GUIDES TO WORD PROCESSORS

WORD PROCESSING WITH APPLE IIe WRITER
— a short guide by David V. Loertscher —

Equipment - an Apple IIe computer with an 80 column text card inside the computer, at least one disk drive, a monitor, one disk marked Apple Writer II, and one blank disk.

The purpose of this short guide is to introduce the most commonly used features of Apple Writer IIe. Using this guide will prepare you to consult the Apple Writer Manuals with more understanding. Our objective is to help you get your feet wet.

Getting Started

1. Put the Apple Writer disk in Drive One (label side up and close drive door).
2. Boot the disk by turning on computer (adjust monitor if necessary).
3. After Copyright Screen comes on, press RETURN and wait until red light on drive goes off. The word processing program is now available in your computer.
4. Remove Apple Writer disk from drive and insert blank disk to initialize it.
5. Type CTRL O (Hold down the key marked CONTROL and type the letter O).
6. Type G then type S6,D1 and press RETURN and wait until red light goes off.
7. Press RETURN to exit - you are ready to "process words".
8. You now have initialized a storage disk for your files. The next time you start with Apple writer you will not do steps 4-6. Instead, you will insert your storage disk after step 3.

Writing on the Screen

Your blank screen is now an electronic piece of paper. The blinking cursor is your electronic pencil. The strip across the top of the screen is the data line. It contains seven pieces of information that will be helpful later on.

Use the keyboard much as you would a typewriter including the shift key for capital letters and caps lock to make all caps. The biggest difference is not having to use a carriage return at the end of each line. As you type a line, the program will push words on to the next line when it gets too full -- its magic -- its called word wrap around. Each line will take up to 80 characters (including spaces).

Do the following:

1. Push the CONTROL key and the letter Q down at the same time. This should give you the "Additional Functions Menu." Now push G. Your cursor now has a bracket inside it. When you push the return key from now on, it will leave a bracket there. This will show you where your carriage returns are.

After you get used to word wrap around you can turn off the bracket by pressing CONTROL Q and G again.

2. Hold the space bar down and notice how the postion counter (POS) works. On what number does the cursor return to the left line? (answer:80)

3. To get rid of all those spaces you just made on the screen, press DELETE and hold it down until the POS number returns to zero.

4. Now type in the following joke but do not type in a carriage return until you are at the end of the joke. Put five spaces on your screen to indent the first line of your joke.

What did the baby porcupine say when it backed into the cactus? Is that you, Mother?]

Does your joke look just like the one above? If you made any mistakes or you have a carriage return you should not have, use the DELELTE key to erase and do it over again.

Now add a new joke as a new paragraph:

What is black and shiny and lives in trees and is dangerous? A crow with a submachine gun.]

5. You can move your electronic pencil (the cursor) around by using the arrow keys. Try skipping through the text with the four arrows. You can move one letter at a time by pressing the arrow once. You can move faster by holding the arrow key down. You can skip even faster by holding down the FILLED APPLE and the arrow keys. Get your cursor back to the end of the second joke.

6. We can add words in the middle of sentences as if by magic. Let's do it. Use the arrows to move the cursor until it is on the first "b" of "baby" in the first joke. Now type the word "little" and add a space at the end. Notice how all the words in the paragraph are pushed aside to make room for the new word. Now add these words to the second joke: usually lives ; in big tall trees.

7. Now you need practice in deleting and adding. The DELETE key only works from right to left. In the first joke, delete the word "little" by putting your cursor on the first letter of "baby" and deleting left. Now delete the words "big tall" in the second joke.

8. Now add a new joke right as a new paragraph between the other two jokes:

Why does an elephant have a trunk? So that it has someplace to hide when it sees a mouse.]

9. Now get your cursor at the end of your document - CONTROL E is the fastest way. Look for a little arrow at the left of the data line at the top of your screen. Change the direction of this arrow by pressing CONTROL D. It should be pointing left. Now press CONTROL X. You just erased your third joke. Press CONTROL X twice more and your screen will be blank. You now know how to erase full paragraphs.

10. Now you are ready to do fancy things. Type a friendly letter to someone you know. Use a date in the upper right hand corner and indent as you should. Make your letter about 50 words long.

11. Add a sentence in the middle of the letter.

12. Proofread your letter and correct your errors.

Summary

CAPS LOCK -- when down - all uppercase letters
— when up - all lowercase letters

SHIFT — use to get uppercase letters when CAPS LOCKS is up

4 arrow keys (up, down, left, right) -- move the cursor without erasing anything

DELETE -- deletes letters from right to left

Filled Apple and arrow keys — move cursor faster

RETURN — returns cursor to beginning of next line - not needed until the end of a paragraph of section

CTRL B -- go to the beginning of the file

CTRL E -- go to the end of the file

Saving and Loading Files to and from the Disk

1. Type CTRL S and then a name for the file (such as Letter) and press RETURN.

2. To make sure the file is saved, type CTRL O and select A for CATALOG and press RETURN. Is your file on the disk? If not, get help. To get back to your file, press RETURN, and press RETURN again. Notice that the file name has been added to the data line. If you are writing a long document, you will want to save every 15-20 minutes (this is insurance against loss).

3. Before loading another file, type CTRL N and then Y to get the old file off the computer's screen. If you don't clear your screen before loading a new document, you will have two documents on your screen - the old and the new.

4. To load a file type CTRL L and the name of the file (such as Letter) and press RETURN. The Mem on the data line shows how much space is left on that file. The Len show how many characters are presently stored in that file.

5. Resaving is done after editing or making additions. It can be done under the old name or a new name. A shortcut to resave the old name file is to use CTRL S = . This will save the file with the name exactly as it is shown on the data line. If you use a new name then you will have two files, one with the old name and one with the new name.

6. To delete a file, go to the DOS commands by using CTRL O.

Summary

CTRL S — save a file onto a disk

CTRL S = — save a file under the name up in the data line

CTRL O — DOS commands (catalog, rename, verify, lock, unlock, delete, or initialize)

CTRL L — load a file from the disk into the computer

CTRL N — clear file off the screen

Printing Out a File

You are now ready to print your letter out on the printer.

1. Turn on the printer.

2. Now you need your letter on the screen. If it is there, skip step #3.

3. Press CONTROL L. It will ask for a file name. Type in the name of your file (I hope you remember what you named it - if not type CONTROL O and option A to see what names are on the disk). Press RETURN. Your letter should be on the screen.

4. Press CONTROL P

5. Type np and then RETURN —The document will be printed out on the printer. np means new printing.

Summary

Ctrl P — gets into the print mode.

? — gives you a list of formatting codes (how you want the document printed).

np — prints out the document on your screen starting at the beginning.

cp — continues to print on to the end of a previous document.

Getting Better at Word Processing

You now know the rudiments of word processing. You might want to practice the creating and printing simple documents several times before you go on. The rest of this document shows some common things you can to with Apple Writer. Look through the headings so that you know what you can do. When you need to add paging or move paragraphs, go to that section of the handout and practice. Soon you will need to know more advanced techniques to make your word processing easier. You should now be able to read the Apple Writer manuals with some understanding.

Numbering Pages

Apple Writer will start counting on page 1 with number 1 unless you give it a different number for the first page. To do that, change the PN command on the print command list to the number you want. For placement of numbers, begin with CTRL P and ? followed by:

TL///#/ — top right corner of the top of the page

TL//#// — top middle of the page

TL/#/// — top left corner

BL///#/ — bottom right corner

BL//#// — bottom middle of the page

BL/#/// — bottom left corner

After typing one of the above, press RETURN. Did the display change? If not, try again.

Replacing Words or Phrases

Checking again for errors.

1. Use CTRL B to get to the beginning of the file.

2. Use CTRL D to point the beginning arrow on the data line in the direction you are checking.

3. Use CTRL F to find words you want to check for spelling, or usage, etc. Put a slash mark at the beginning and end the word or phrases you want to check. Example -- /Sincerely/

4. CTRL F /mispelled word/correctly spelled word/ -- will find the mispelled word or phrase and ask if you wish to replace it with the correctly spelled word. Y - replaces. RETURN - looks for the next occurrence. Any other key - cancels.

5. CTRL F /mispelled word/correctly spelled word/a -- will replace all occurrences of the misspelled word or phrase in the document.

6. CTRL F <>< and RETURN will make the cursor jump from paragraph to paragraph.

7. CTRL R will let you type right over the top of any character - won't insert any spaces. This is really helpful when creating columns.

Moving and Deleting Text

When you use the DELETE key, everything you delete is gone forever. Fortunately there are ways of temporarily deleting material which can then be retrieved when needed.

1. Use CTRL D to change direction of the arrow on the data line. Left is for temporarily deleting, right is for retrieving.

2. For moving a few words (less than a paragraph), hold down the Open Apple key and press the left arrow key. 128 characters can be temporarily hidden this way. To retrieve the hidden words, position the cursor where you want the words placed, change data line arrow to the right, and hold down the Open Apple key and press the right arrow key until they all reappear.

3. Ctrl X deletes and retrieves a paragraph from right to left. Change data line arrow to the left, position cursor to the right of the parts to be temporarily deleted, use CTRL X, move cursor to place for insertion, change data line arrow to the right, and press CTRL X.

4. To copy a paragraph and duplicate it somewhere else, position cursor at end of part to be copied, change data line arrow to the left, hold down the Filled Apple key and use CTRL X. Then move cursor to desired place for insertion, change data line arrow to the right and use CTRL X for retrieval.

Word Wraparound

CTRL Z — Turns on and off the word wraparound function (automatic carriage return). The Z will appear at the top of the screen if wraparound is ON.

Underlining

To underline text, type the backslash at the beginning and end of what you want underlined. The backslash is printed out as a space at the end of the word,so do not put a space after what is being underlined.

Example:

The name of the book is \Morning is Nigh\and it's funny.

produces:

The name of the book is Morning is Nigh and it's funny

Centering a Line

Use .cj in the text on a line all by itself at the left margin and just before the line you want centered. Then type .lj on the next line to stop the centering. Example:

```
.cj
This line will be centered.
So Will this line.
.lj
This line will not be centered.
```

Setting and Clearing Tabs

Apple Writer automatically sets tabs for you at every eight spaces. Use the TAB key to move the cursor nine times across the screen. Notice the number that appears on the data line after TAB. You can also set and clear your own tabs.

1. CTRL T and typing P will purge the existing tabs.

2. Use the space bar to position the cursor where you want TABS.

3. CTRL T and typing S will create a tab position, repeating for each setting.

4. CTRL T and typing C will clear the tab where your cursor is.

Making and Using a Glossary

This is a joy . A glossary allows you to type common phrases and sentences with one keystroke. You store the phrases and then retrieve them anytime.

Building a glossary:

1. CTRL N (clear memory)

2. Type in definitions (composed of a one letter designator followed by the definition).

Example:

aaccounts payable
sSincerely yours,
lWe love you very much.

Any key on the keyboard can be the designater. Even upper case letters can be designators.

Saving a glossary:

1. CTRL S and name the glossary (glossary is a good name).

Using a glossary:

1. CTRL L to load your document.

2. CTRL Q, choose E, and type the name of your glossary. When you press RETURN, nothing happens on the screen but your glossary is stored and is ready to use.

3. As you are typing along and you need a phrase from the glossary, just press CTRL G and type the letter which represents the phrase.

Example: CTRL G a prints accounts payable.

Changing the Way a File is Printed

1. CTRL P puts the computer into print mode.

2. Type ? to get a list of Print/Program Commands. The values listed for each command are the "default" values — the ones that Apple Writer will use to print text unless you change the values.

3. To change any one of the command values, type the two letter code and the value you want changed. No space or = sign is needed.

Set left margin	LM___	default 9
Indent paragraph margin	PM___	default 0
Right margin from left edge	RM___	default 79
Top line margin printed under page #	TM___	default 1
Bottom margin before footer	BM___	default 1
Set page number	PN___	default 1
# of printed lines per page	PL___	default 58
# of lines from top of one page to next	PI___	default 66 (assumes 11" paper)
# of spaces between lines	LI___	default 0 (single space) 1=double space 2=triple space
Single or continuous paper	SP___	0=continuous paper 1=single sheets fed in one at a time
Printed on screen or printer	PD___	0=screen 1=printer
Carriage return on	CR___	default 0 1=add a carriage return

4. Turn printer switch on and type np (for new print) to cause the text of the document in memory to be printed.

5. To print a document at the position on the page where the last one ended, type cp (for continuous print).

To change print commands within the document (for example, if you want some lines single spaced within a double-spaced document) then the commands listed above must be embedded or contained within the body of the text itself.

1. Carriage return to begin a new line.

2. Type a period on the left margin and the command wanted.

3. Carriage return.

Examples:
.li0 — would turn on single spacing
.li1 — would turn double spacing on again
.ff — Form Feed. This will eject the paper from the printer and start printing on a new sheet.
.in — Input. This will stop the printer and will display a message on the screen, waiting for you to press RETURN before it continues printing.

Printing Out Only Part of a Document

Use the embedded command .ep (enable print) to print out specific parts.

.ep0 -- placed at the beginning of the part you don't want printed (on the left margin always).

.ep1 -- placed at the beginning of the part you do want printed. This means you must put an .ep0 at the end of what you want printed so that printing won't continue beyond that part.

Another way to print out only part of a file is to load only a part of the file first. This is done by specifying the beginning words and ending words of the part you want. You must specify enough words to distinguish that section from any other section.

1. Type CTRL L and file name followed by a comma and d1 (for drive one) then a slash mark used as a delimiter followed by the specific beginning words, another slash mark followed by the specific ending words followed by another slash mark.

Example: CTRL L Weather,d1/Today is gloomy/much effort./

These commands cause the paragraph chosen to be loaded, then the CTRL P command can by used along with np to get a printout of that paragraph.

Printing Several Documents as One

1. CTRL N to clear memory.
2. CTRL L and name first document.
3. CTRL P and np to print.
4. CTRL N to clear memory.
5. CTRL L and name second document.
6. CTRL P and cp to print next document.

Printing to the Screen - Not to the Printer

You can check you document before you waste printer paper.

1. CTRL L and name file.
2. CTRL P and ?
3. Change command PD (print destination) from 1 to 0
4. Print out using np
5. Use CTRL S to stop and continue scrolling.

COMMAND SUMMARIES

CAPS LOCK — when down - all uppercase letters
-- when up - all lowercase letters

SHIFT -- use to get uppercase letters when CAPS LOCKS is up

4 arrow keys (up, down, left, right) -- move the cursor without erasing anything

DELETE -- deletes letters from right to left

Filled Apple and arrow keys — move cursor faster

RETURN -- returns cursor to beginning of next line - not needed until the end of a paragraph of section

CTRL B -- go to the beginning of the file

CTRL E -- go to the end of the file

CTRL S -- save a file onto a disk

CTRL S = -- save a file under the name up in the data line

CTRL O -- DOS commands (catalog, rename, verify, lock, unlock, delete, or initialize)

CTRL L -- load a file from the disk into the computer

CTRL N -- clear file off the screen

Ctrl P -- gets into the print mode.

? -- gives you a list of formatting codes (how you want the document printed).

np -- prints out the document on your screen starting at the beginning.

cp -- continues to print on to the end of a previous document.

BEGINNER'S GUIDE TO BANK STREET WRITER

by Sherry Greathouse

Lesson One
Entering Text and Printing

1. Boot up the Bank Street Writer Tutorial program. It is on the back side of the master disk.

DISK MUST STAY IN THE DISK DRIVE WHEN TUTORIAL IS IN USE.

---

2. Do Lesson One only.

3. After completing Lesson One, remove the tutorial disk, turn it over and boot up Bank Street Writer.*

   *Bank Street Writer from this point on will be denoted as BSW.

4. Type the following adaptation of the "The Camel" from Nobody Really Likes A Nervous Cow by Elinor Goulding Smith, including all spelling and typing errors:

   I itch. I itch froom head to feet. BBoy do I itch. I'm telling you, it's enough to drive yu crazy. Itch, itch, itch and noting to scratch with. It's this kamel's hair I'm cobered with. And furthermore, my feet hurt.

   It's so hot today. It's hotter in New York City than it was in Egypt, and nobody stared at you there.

5. Remove the BSW disk and replace it with a blank disk. If the disk being used to store files has already been initialzed, go to #6. (The disk being used for file storage does not have to be initialized with BSW. It can be initialized with a "Hello" program.) If the disk is to be initialized with BSW, do the following:

   a. press ESC
   b. select TRANSFER MENU by using the apple key on IIe or the arrows on the II+
   c. press RETURN
   d. select INIT
   e. press RETURN

6. Go to TRANSFER MENU if not already there. Select SAVE. Press Return. Follow the directions on the screen to name and save the file.

7. To print the file do the following:

   a. get in TRANSFER MENU
   b. select PRINT-FINAL
   c. press RETURN
   d. answer the questions on the screen by pressing RETURN if you agree with the value shown OR entering the value desired then pressing RETURN. The following can be changed:

      1. Characters per line?: 40-126 (the choice will depend on the size of paper and the type of printer being used)

      2. Spaces between the lines?: single, double, or triple

      3. Continuation of a previous file?: yes or no ("yes" allows the connection of a file that is now being printed to one that has just been printed; printing begins at the end of the previous file and continues with line and page number)

      4. Numbering of pages?: yes, if it is to be numbered, or no, if it's not to be numbered

      5. Where the numbering begins?: number 1 if it is to begin with the first page, or number 2 if there is a cover sheet and the first numbered page is to be the actual second page

      6. Numbering at top or bottom?: "T" for top, or "B" for botom

      7. Pause bewteen pages?: yes or no (select yes if individual sheets of paper are being used, so the printer will stop after each page it prints and another piece of paper can be rolled in)

      8. Eject last page?: yes or no ("yes" sets the printer at the top of the following page; "no" will cause the printer to stay at the last printed word)

      9. Page heading?: useful if the file is long; the heading entered appears on all pages following the first page

10. Print entire file?: yes or no ("no" will allow for only a portion of the file to be printed; follow the directions on the screen to indicate which portion)

11. Where the text ends on a page?: yes or no (follow the directions on the screen to make changes)

e. Turn on the printer.

f. When the printer is ready press RETURN

g. Follow the directions on the screen to print more than one copy

Lesson Two
Cursor Movement and Correction

1. Boot up the BSW Tutorial program.

2. Do Lesson Two only.

3. After completing Lesson Two, remove the tutorial disk, turn it over and boot up Bank Street Writer.

4. Press ESCAPE to go to EDIT menu. Select TRASNSFER MENU press RETURN. Select RETRIEVE press RETURN.

5. Remove BSW and replace with the disk that has on file the text from "The Camel". Following the directions on the screen retrieve this file.

6. Once the text has been retrieved, correct all errors in the text using the cursor and correction techniques explained in lesson two of the tutorial.

EDIT MODE ALLOWS THE CURSOR TO MOVE AROUND IN THE TEXT WITHOUT ERASING. WRITE MODE ALLOWS FOR THE CORRECTIONS TO BE MADE. IT WILL BE NECESSARY TO BACK AND FORTH FROM ONE MODE TO THE OTHER WHILE MAKING CORRECTIONS.

There are six errors: from, Boy, you, nothing, camel's, covered

7. Go to WRITE MODE. Add the following paragraph to the file:

> I really don't like camels much. When we look at them, with their hair all over them, I itch even more. I just might lick one of them when no one's looking.

8. Make the following corrections in the paragraph that has just been added:

   a. At the beginning of the second sentence change "we" to "I"
   b. At the end of the second sentence change "more" to "worse"
   c. In the middle of the third sentence change "lick" to "bite"

9. Save the additions and corrections by doing the following:

   a. Get into EDIT MODE
   b. Select TRANSFER MENU press RETURN
   c. Select SAVE press RETURN
   d. Follow the directions on the screen

THE COMPUTER WILL ASK IF THE ENTIRE FILE IS TO BE KEPT AND IF THE SAME FILE NAME IS TO BE KEPT. THE ANSWER IS "YES". WHEN THE SAME FILE NAME IS USED, THE OLD VERSION OF THE TEXT IS REPLACED WITH THE NEW VERSION OF THE TEXT, THUS SHOWING ADDITIONS AND CORRECTIONS.

---

10. When the screen indicates the file has been saved, then select CLEAR and press RETURN. This clears the workspace in the computer's memory and clears the screen. If the workspace is not cleared after a file has been saved it will be included as a part of any other work that may be done. BE SURE TO SAVE BEFORE THE COMPUTER IS CLEARED AND BE SURE TO CLEAR BEFORE A NEW FILE IS BEGUN.

Four other keys which can move the cursor around in the text:

B = beginning of the text

E = end of the text

U = up twelve (12) lines at a time

D = down twelve (12) lines at a time

Lesson Three
Using Erase and Unerase Commands

1. Boot up the BSW Tutorial program.

2. Do Lesson Three only.

3. After completing Lesson Three, remove the tutorial disk, turn it over and boot up BSW.

4. Press ESCAPE to go to EDIT mode. Select TRANSFER MENU press RETURN. Select RETRIEVE press RETURN.

5. Remove BSW and replace with the disk that has on file the text from "The Camel". Following the directions on the screen retrieve this file.

6. Go to EDIT mode.

7. Erase the second paragraph by doing the following:

   a. Select ERASE press RETURN

   b. Take cursor to the beginning of the second paragraph so that it's under the "I" and press RETURN

   c. Take cursor to the end of the paragraph by using the down arrow so that it is highlighted by lines. Press RETURN and follow the directions on the screen.

If ERASE has been selected in EDIT mode then the arrow keys will do the following:

right arrow = highlights letter by letter
left arrow = removes highlighting letter by letter
down arrow = highlights line by line
up arrow = removes highlighting line by line

---

8. The second paragraph should not have been erased. It can be put back by doing the following:

   a. Go to EDIT mode (should already be there)
   b. Select UNERASE press RETURN
   c. Answer "yes" to the question that appears on the screen

---

IN ORDER FOR THE UNERASE COMMAND TO WORK, IT HAS TO BE USED DIRECTLY AFTER THE ERASE COMMAND. IF MODES ARE CHANGED OR ANY OTHER CORRECTIONS MADE AFTER THE ERASE COMMAND HAS BEEN USED, IT IS NOT POSSIBLE TO GET BACK THE PART WHICH WAS ERASED.

---

9. Go to EDIT mode. Select ERASE press RETURN. Erase the second sentence of the third paragraph by following the directions on the screen.

10. Replace the sentence that has just been removed. Select UNERASE press RETURN and follow the directions on the screen.

When using the ERASE command, up to 15 lines can be removed at a time. Should more than 15 lines need to be removed, it will have to be done in blocks. (one block = 15 lines of text)

If more than one block of text is erased (15 lines), the UNERASE command only works in replacing the text of the last block or portion removed.

Lesson Four
Using Move and Moveback Commands

1. Boot up the BSW Tutorial program.

2. Do Lesson Four only.

3. After completing Lesson Four, remove the tutorial disk, turn it over and boot up BSW.

4. Press ESCAPE to go to EDIT mode. Select TRANSFER MENU press RETURN. Select RETRIEVE press RETURN.

5. Remove BSW and replace with the disk that has on file the text from "The Camel". Following the directions on the screen, retrieve this file.

6. Go to Edit Mode.

7. Move the third paragraph so that it becomes the second paragraph by doing the following:

    a. Select MOVE press RETURN

    b. Take the cursor to the line right before the third paragraph. Press RETURN. (You have to remember to move the spaces prior to the text being moved.)

    c. Take the cursor to the end of the paragraph by using the down arrow or the right arrow. Text will become highlighted. Press RETURN.

d. Using the up arrow, take the cursor to one line above the second paragraph. This will allow for the space between paragrapahs. Press RETURN.

e. Answer "yes" to the question on the screen.

8. This paragraph should not have been moved. It can be put back in its original position by doing the following:

a. Go to EDIT MODE (should already be there)
b. Select MOVEBACK press RETURN
c. Answer "yes" to the question that appears on the screen

IN ORDER FOR THE MOVEBACK COMMAND TO WORK, IT HAS TO BE USED DIRECTLY AFTER THE MOVE COMMAND. IF MODES HAVE BEEN CHANGED OR ANYTHING ELSE DONE AFTER THE MOVE COMMAND HAS BEEN USED, THE MOVEBACK COMMAND WILL NOT WORK.

---

9. Go to EDIT MODE. Select MOVE press RETURN. Move the last sentence of the first paragraph to the end of the second paragraph by following the directions on the screen. DON'T FORGET TO HIGHLIGHT THE SPACE OR SPACES BEFORE THE SENTENCE YOU ARE MOVING.

10. Return the sentence to its original position. Select MOVEBACK press RETURN and follow the directions on the screen.

When using the MOVE command, up to 15 lines can be moved at a time. Should more than 15 lines need to be moved, it will have to be done in blocks. (one block = 15 lines of text)

If more than one block of text (15 lines) is moved, the MOVEBACK command only works on the text of the last block or portion moved.

Lesson Five
Using Find and Replace Commands

1. Boot up the BSW Tutorial program.

2. Do Lesson Five only.

3. After completing Lesson Five, remove the tutorial disk, turn it over and boot up BSW.

4. Go to EDIT MODE. Select TRANSFER MENU press RETURN. Select RETRIEVE press RETURN.

5. Remove BSW and replace with the disk that has on file the text from "The Camel". Following the directions on the screen retrieve this file.

6. Go to EDIT MODE.

7. Select FIND and press RETURN.

8. Type in the word "itch" and follow the directions on the screen. Answer "yes" to the question. A beep will sound when there are no other occurrences of the word.

NOTE: If the word "the" is typed, not only does BSW find every occurrence of that word, but also every occurence of those three letters. Therefore, "the" is highlighted in words such as "whether", "they", and "either". TO KEEP THIS FROM HAPPENING, PRESS THE SPACEBAR ONCE BEFORE AND ONCE AFTER THE WORD YOU WISH TO FIND.

Decisions will need to be made regarding this since BSW will not find a word that has punctuation following it if there was a space put after the word that is to be found. Since upper and lower case letters are considered different, BSW would find only "the" and not "The" unless specifically told to do so.

---

9. Go to EDIT MODE.

10. Select REPLACE and press RETURN.

11. Replace the word "itch" with the word "stink". Follow the directions on the screen.

When checking the text, it is noticed that the same word has been misspelled all the way through. Use the REPLACE command to correct all the occurrences of that particular word.

The REPLACE command may be used to delete words or characters in the text. It will delete up to 29 characters at a time. Just press RETURN when it asks what is the replacement.

The FIND and REPLACE command can be used at any time while working on the text.

Additional Notes on the Bank Street Writer

1. Words CAN NOT be underlined in the text.

2. A disk does not have to be initialized by Bank Street Writer in order to store files. Any DOS 3.3 initialized disk may be used.

3. The BSW disk can be removed from the disk drive after it has been booted up if files are being worked on. IF THE TUTORIAL IS BEING USED, THE DISK MUST STAY IN THE DISK DRIVE THE ENTIRE TIME THE TUTORIAL IS IN USE.

   Note: If one computer is being used to type the text, but another computer will be used to print, the computer being used to print will have to be booted with BSW first before the files can be printed out.

4. The following can be done while in WRITE MODE:

   a. Press CONTROL (CTRL) and 'C' and any characters on that line will appear centered on the print out.

   b. Press CONTROL (CTRL) and 'I' and the characters will be indented 8 spaces. This can be used up to 4 times per line. Text can be indented 8 spaces, 16 spaces, 24 spaces and 32 spaces. (Apple II or Apple II Plus only

   c. Press TAB and the characters will be indented 8 spaces. This can be usedup to 4 times per line. Text can be indented 8 spaces, 16 spaces, 24 spaces and 32 spaces. (Apple IIe only)

5. The Apple II+ (64K) or the Apple IIe will take approximately 3,200 words in a file. When entering text and this capacity is about to be reached, a message will be placed on the screen indicating how much space is left. To check the capacity level as text is being entered, press CONTROL (CTRL) and 'S'.

   If the text is going to exceed the capacity level that can be stored in a file, SAVE the text that has been entered before the maximum has been reached. After this first part has been saved, then CLEAR the workspace and proceed with the typing of the rest of the file. SAVE this second part <u>under a different name.</u> When printing, this second file will be a <u>continuation of the first file</u>.

6. When a file is retrieved and there is already text in the workspace, the file that has been retrieved is added where the cursor is located.

7. To combine files, use the following procedure:

   a. Make sure the workspace is CLEARED

   b. RETRIEVE the file you need to combine with another file

   c. Select SAVE press RETURN

   d. The BSW will ask if the whole file is to be saved, press 'N' and follow the directions on the screen to save only the part that is to be combined with another. Be sure and give it a new file name.

   e. CLEAR the workspace

   f. RETRIEVE the file that is to have text added

   g. Place the cursor where the added text is to appear

   h. RETRIEVE the text just saved, and it will be inserted into the text at the point where the cursor was placed, thus combining the text just retrieved with what is in the workspace.

      Note: If something is too large to be combined, the screen will display: FILE TO LARGE TO RETRIEVE. When this happens it is necessary to erase some of the text in order to combine files.

8. It is possible to RENAME a file that is saved on a disk by going to TRANSFER MENU; selecting RENAME, and following the directions on the screen.

9. If files need to be removed from the disk, go to TRANSFER MENU, select DELETE, and follow the directions on the screen.

10. If PRINT-DRAFT is selected, the copy will look just like what is printed on the screen; that is 38 characters a line. It will also be printed with large margins and spaces between the lines of text, so that editing can be done away from the computer.

11. Files can be protected by using a password. A file may be given a password when it is saved. Be sure to keep a written list of the passwords if they are going to be used, since it is not possible to retrieve a file unless the password is known, or the BSW Utility program is accessible.

12. The Bank Street Writer Utility Program may be retrieved by pressing and holding ESCAPE while the BSW is loading. The four options on the Utility Program are:

    1. Change Set Up Items - This option allows for the changing of the pre-set values of Bank Street Writer.

    2. Display Passwords - If passwords are used to protect files, they are listed here. Follow the directions on the screen to utilize this option.

    3. Convert Writer File - This option allows for the conversion of Apple text to Bank Street Writer text, or the other way around.

    4. Quit - As long as the BSW master disk is in drive one the Utility Program may be aborted and Bank Street Writer automatically reloaded, if this option is selected.