DOCUMENT RESUME

ED 355 921 IR 016 017

AUTHOR Overbaugh, Richard C.
TITLE A BASIC Programming Curriculum for Enhancing Problem-Solving Ability.
PUB DATE 93
NOTE 88p.
PUB TYPE Guides - Classroom Use - Teaching Guides (For Teacher) (052) -- Reports - Evaluative/Feasibility (142)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS Computer Science Education; Computer Software; *Curriculum Development; High Schools; Lesson Plans; *Problem Solving; *Programing; *Programing Languages; *Secondary School Curriculum; *Skill Development; Teaching Methods
IDENTIFIERS *BASIC Programing Language

ABSTRACT

This curriculum is proposed to enhance problem-solving ability through learning to program in BASIC. Current research shows development of problem-solving skills from learning to program in BASIC. Successful treatments have been based on contemporary problem-solving theory, top-down, modular programing, and rigorous length and intensity. The proposed BASIC curriculum, incorporating these characteristics, is designed for secondary education and is adaptable to postsecondary education. The curriculum suggests beginning with an optional 6-week problem-solving software unit to overcome computer anxiety and introduce problem-solving theory. The major portion, the actual BASIC programing instruction, lasts 30 weeks with daily lessons, but is easily expanded or condensed. The program is designed to require computer access only in school. The nine major units of the curriculum and their subunits in the areas of graphics, animation, screen formatting, string variables, numeric variables, embedded DOS commands, mathematics, text files, dimensional arrays, and personal programs are included. (Contains 17 references.) (SLD)

A BASIC Programming Curriculum for Enhancing Problem-Solving Ability

Richard C. Overbaugh

Abstract: Current research shows development of problem-solving skills from learning to program in BASIC. The successful treatments were based on contemporary problem-solving theory, top-down, modular programming and rigorous length and intensity. The proposed BASIC curriculum, incorporating these characteristics, is designed for secondary education and is adaptable to post secondary education. The curriculum suggests beginning with an optional six-week problem-solving software unit to overcome computer anxiety and introduce problem-solving theory. The major portion--BASIC programming--lasts 30 weeks with daily lessons, but is easily expanded or condensed.

Overview

Introduction

Microcomputers were introduced to the public as a machine capable of extracting extraordinary intellectual powers from school children. They were supposed to lead to higher thinking skills and increased facility in problem solving abilities.To accomplish this amazing feat, prolific amounts of software appeared for computer aided instruction (CAI) and children of all ages were suddenly programming computers in languages such as Logo and BASIC.

The educational systems were quick to jump on the computer bandwagon because it was purported to be the answer to the inadequacies of American education. Unfortunately, the promises of the computer were not delivered, resulting in a backlash against computers in the early 1980's. In an effort to defend the use of computers in the schools, there was a rush to provide empirical evidence of the positive cognitive effects of programming language instruction, resulting in a large amount of research ranging from no effects to fantastic effects. In general this research was hastily thrown together and contained faulty methods and conclusions. Some of the major inherent problems were: (a) lack of sufficient exposure to the language, (b) lack of control groups, (c) lack of measures tailored to the computer environment, (d) insufficient instruction, or whether or not students really learned a programming language, and (e) computer anxiety (Burton & Magliaro, 1988, Palumbo & Reed, 1988; Reed & Palumbo, 1988; Reed, Palumbo & Stolar, 1988).

However, it is still currently believed that the teaching of programming languages promotes certain useful skills. Researchers have now had the opportunity to look at and compare earlier work and have begun new research which takes into account the aforementioned problems.

This paper looks at the work of professional computer educators who believe in and are dedicated to the positive aspects of computers in the schools.

Literature Review

Professionals in the field of educational computing maintain that learning to program has the potential to (a) develop higher-level thinking skills and problem-solving ability and (b) enhance the transference of these skills to non-programming domains.

Higher-level thinking skills and problem solving ability development

The first major prospect of learning to program is the utilization and exercising of higher-level thinking skills to enhance problem-solving ability. Problem-solving is broken down into four phases: specification, planning, coding and debugging (Dalbey & Linn, 1986; Dalbey, Tourniaire & Linn. 1986; Kurland, Pea, Clement & Mawby, 1986; McCoy & Orey, 1988; Reed & Palumbo 1988).

Problem specification. The first phase is problem specification which is determining exactly what the problem is and what the program needs to do. Students must evaluate the problem very carefully to know how to start the programming task. Their evaluation should involve questioning, clarifying the outcome of the program, and exploring the different ways to achieve the outcome (Dalbey, Tourniaire & Linn, 1986).

Planning. The second phase is planning, sometimes called procedural reasoning, and is often the most difficult step for students. In this phase, the problem is broken down into the steps that will actually be coded. To do this, the programmer must understand how commands are interpreted by the computer in order to correctly organize the steps of the program. It is widely accepted that programs should be designed using the top-down approach--a process beginning with problem specification, followed by successively breaking the problem down into levels of component parts (Clement, Kurland, Mawby & Pea, 1986; Dalbey & Linn, 1986; Dalbey, Tourniaire & Linn, 1986). At each level is a set of unique algorithms leading progressively to the final algorithms which are then coded, creating the program.

Coding. The third phase is coding--the actual conversion of algorithms into a specific language. Not only does a programmer need a large knowledge base of the programming language commands, but also efficient programming skill which is characterized by the use of templates

(Dalbey, Tourniaire & Linn, 1986; Magliaro & Burton, 1988). Templates are groups of commands or chunks of programs that students have used to solve previous problems that can be incorporated into the solutions of new problems. For example, in BASIC, a typical chunks are "page turn" subroutines used to clear the current screen in preparation for a new display, or the procedure to draw a particular color background.

Debugging. The last phase of programming is debugging--the task of finding the errors in a program that prevent its proper execution. Programming errors fall into two general categories: (a) syntax--usually typing errors, and (b) logic and wrong-command errors. Syntax errors are usually easy to locate and correct, but logic errors are much more difficult. To locate a logic error, programmers must be able to decenter, which is a process of separating (a) the function the programmer intended the computer to do from (b) what the computer actually did by analyzing the feedback provided by the computer. Becoming proficient in debugging requires a large amount of experience as it is a highly complex task involving abstract rational reasoning (Reed & Palumbo, 1988).

Transferring problem-solving skills. The second major prospect of learning to program is the transference of thinking skills and problem solving abilities to other areas. Transference is the reason for supporting a programming curriculum in the schools: If students become skilled in all phases of computer programming, it is believed that they will be able to apply the same thinking processes to non-computer problems. The teaching of computer languages to develop these skills is ideal because programming is not content directed as are other disciplines (Reed, 1986). For example, Social Studies and English have particular constituent parts that must be taught, whereas instruction in programming can be tailored to the task of enhancing problem-solving abilities.

Current research is directed toward showing empirical evidence that learning programming increases problem-solving abilities and transfers such skills to other areas. However, because the high level of programming ability needed for the transference of problem solving skills is not attained with present secondary school instruction (Clement, Kurland, Mawby & Pea, 1986; Dalbey, Tourniaire & Linn, 1986; Kurland, Pea, Clement & Mawby, 1986; Palumbo & Reed, 1988), the bulk of the research reviewed for this paper showed positive results in the enhancement

of problem solving ability when the measures and learning environments were tailored to the programming environment (Kurland, Pea, Clement & Mawby; McCoy & Orey,1988; Palumbo & Reed; Reed & Palumbo, 1988). One exception is research by Soloway, Lochhead and Clement (1982) that showed positive transference of problem solving skills to mathematics, which requires thinking process that are similar to those utilized in programming.

The learning environments for several of the studies were carefully constructed and controlled by the researchers in an effort to overcome, or at least reduce, problems that have led to poor results in the past (Dalbey & Linn, 1985; Dalbey, Tourniaire & Linn, 1986; Palumbo & Reed, 1988; Reed & Palumbo, 1988; Reed, Palumbo & Stolar, 1988). Since the results were positive, the instructional designs used by these researchers can be considered guidelines for educators involved in teaching programming languages. Three design aspects are particularly significant: (a) length and intensity of treatment, (b) top-down method of problem-solving, and (c) computer anxiety.

Length and intensity of treatment. The first aspect is length and intensity of instruction. Programming proficiency requires high levels of skill and cannot be achieved in short periods of time. The research examined here largely involved an instruction period covering at least one semester with daily lessons or an equivalent. In view of the positive results, this is a clear indication that a curriculum should last at minimum one semester with daily lessons (Palumbo & Reed, 1988).

Top-down approach. The second major aspect of programming instruction is the method of teaching the language. As reported, a top-down approach, using modular program construction is necessary for good planning, efficient program design and template formation (Clement, Kurland, Mawby & Pea, 1986; Lockard, 1986; Magliaro & Burton, 1988; Reed, Palumbo & Stolar, 1988). Instructors should teach their classes in a manner that develops planning skills through the use of algorithms and modular programming. The creation and use of templates can be promoted by using programs or parts of programs written by students in subsequent programming problems (Dalbey, Tourniaire & Linn, 1986, Lockard). It is here that a great deal of support for the use of Logo or Pascal can be found as they both require procedural or modular programming techniques, as opposed to BASIC which does not. However, it has been

found that if BASIC is taught in a modular form using subroutines there are no differences in the cognitive outcomes (Reed, Palumbo & Stolar). Therefore, the consideration of cost enters the picture. Logo and Pascal are expensive and must be purchased for each machine in a classroom, whereas BASIC is supplied with almost all machines.

Some researchers feel that certain concepts of programming, such as loops and conditionals, can be taught before a formal language is actually learned. To accomplish this, one group of researchers used Spider World, a simple program that uses commands similar to a programming language but requires minimum coding skills. After the students had a good grasp of the required procedures, they began learning a formal language with positive results (Dalbey & Linn, 1985).

Debugging skills increase as programmers become more skilled in coding and problem solving (Reed & Palumbo, 1988) and by understanding explicitly the function of commands (Dalbey, Tourniaire & Linn, 1986). This process can be started early by engaging students in the task of predicting the outcome of short programs written by an expert or by changing certain parts of a program (written by an expert) to produce different outcomes. A secondary advantage of this activity is exposure to well-planned programs written by an expert, thus giving students models of efficient programming and templates they can incorporate into their own work.

Computer anxiety. The third aspect of teaching programming that deserves attention is computer anxiety. It has been found that students who feel anxious toward the machine have more problems learning to program well (Reed & Palumbo, 1988). Efforts to overcome this anxiety should take place at the very beginning of a course and can be expedited with the use of easy, non intimidating software to acquaint students with operating techniques and general uses of the computer. Then, when beginning to program, a logical starting point is graphics (Reed & Palumbo) because graphics are perceived as enjoyable by most, are fairly easy to learn, and give an immediate, visual product.

Summary. In summary, this research supports the notion of the enhancement of higher-level thinking skills and increased problem-solving ability through the medium of computer programming instruction. Therefore, if the reason for teaching a computer language is the acquisition, development and transference of higher-level thinking skills, the positive results of this

research, conducted under conditions designed to promote these skills, should serve as a guide for curriculum development in this area. The fact that there is a lack of supporting evidence in situations not designed with similar objectives is an indication of the present condition of programming instruction. Most courses are taught by instructors whose training lies in other areas which usually results in curricula exclusively teaching how to code simple programs. Finally, that the transference of these abilities to other domains, even in research settings, remains to be shown is not surprising. If a high level of proficiency in programming is a requisite for positive results on measures tailored to the computer environment, a need for more curricular development is indicated before transfer to other domains will commence. Simply put, more research is needed to develop effective methods of teaching computer programming to specifically address the task of promoting problem-solving abilities. Just as important is the absolute necessity for trained professionals who can implement such a curriculum. Without specific curricula and qualified instructors, the potential for positive cognitive results from learning to program will not be realized.

Application

Learning to program in BASIC increases students' problem-solving ability when the instruction is based on the four components of problem-solving--(a) problem specification, (b) planning, (c) coding, and (d) debugging Other significant characteristics of the successful research designs are: (a) the top-down approach to programming, (b) modular program construction techniques, and (c) a treatment sufficiently rigorous to develop an adequate knowledge base of the language. The BASIC curriculum proposed is a non-content-oriented, cumulative, problem-solving approach that encourages higher-level thinking skills. In general, each successive unit requires or, at minimum, can incorporate the skills gained in the previous unit or units.

The next section gives a general description of each curriculum unit, followed by suggestions for class structure and procedures. The suggestions are designed to maximize the desired problem-solving approach to develop higher-level thinking skills.

Unit Descriptions

The curriculum begins with Low Resolution graphics in the immediate mode of execution to give the students instant reaction to the commands entered. This approach also reinforces the later use of line numbers to delay execution.

---

1. LO-RES GRAPHICS 3-4 Weeks

UNIT OBJECTIVES: The students will-
-initialize a blank disk to use as a work disk
-use simple commands to draw multicolored pictures on the screen in lo-res graphics.
-learn how to edit source code and what to do when error messages occur.
-SAVE, LOAD, edit and RUN programs.
-understand the difference between immediate mode and delayed mode of execution.

Unit two uses Lo-Res graphics to introduce numeric variables, subroutines, simple FOR/NEXT loops for delay, and animation loops. Unit three is High Resolution graphics which utilizes the same program structure and introduces graphics commands similar to those used in Lo-Res. Animation is included to give more practice with loops. Unit three also introduces embedded DOS commands and then segues into the text unit by using very simple text commands such as PRINT and VTAB.

---

2. ANIMATION IN LO-RES GRAPHICS 2 Weeks

UNIT OBJECTIVES: The students will-
-use FOR/NEXT Loops and numeric variables to animate simple one color and multi-color objects across the screen from all directions.

---

3. HIGH-RES GRAPHICS INCLUDING ANIMATION AND EMBEDDED DOS COMMANDS 5 Weeks

UNIT OBJECTIVES: The students will-
-be able to draw static pictures in Hi-Res graphics demonstrate an understanding of the compatibility of Hi-Res colors.
-animate single and multicolor objects in Hi-Res.
-animate objects on a background drawn in Hi-Res.
-use text commands to introduce a graphics program and embedded DOS commands to load and run the programs.
-use embedded DOS commands to alternate between Hi-Res page 1 (HGR) and Hi-Res page 2 (HGR2).

Unit four is a relatively brief text unit that introduces the remaining screen formatting commands such as FLASH, HTAB, SPEED= and NORMAL. More work with the text commands

occurs in the fifth unit, which introduces string variables and gives practice distinguishing string from numeric variables. The concept of conditional statements for program control is also begun in the fifth unit as students program interactive questions and answer programs. Unit six introduces saving data files on disk.

---

4. TEXT COMMANDS FOR SCREEN FORMATTING 2 Weeks
INRODUCTION TO STRING VARIABLES

UNIT OBJECTIVES: The students will-
-learn some text formatting commands.
-use a delay loop
-use simple text commands to introduce a graphics picture and embedded DOS commands to automatically run it.

---

5. STRING VARIABLES And NUMERIC VARIABLES , 3 Weeks
TEXT FILES, AND MORE EMBEDDED DOS COMMANDS

UNIT OBJECTIVES: The students will-
-use numeric and string variables to create a question and answer program.
-know the symbols for the four basic math functions.
-use conditional statements for responding to user input and error trapping.

---

6. TEXT FILES 22-2.5 Weeks

UNIT OBJECTIVES: The students will-
-save text to a text file on disk
-read text from a text file on disk

Math is the principle focus for unit seven. The four basic math functions were introduced in unit five, therefore, the purpose of this unit is simply to show how programming can be applied to other areas. Some ideas for useful programs are suggested, but the students should be free to use whatever level math they desire for their programs. All commands and skills learned up to this point can be easily integrated into games or interactive math problem-solving programs.

7. MATH UNIT 4 Weeks

UNIT OBJECTIVES: The students will-
-know 6 simple math symbols and their order of priority and how to change the priority with parethesis.
-use the LEN command to create a centering routine.
-use embedded DOS commands to alternate two Hi-Res images on the screen.
-use the RND command to create a number guessing game.
-combine the above commands in a single program to title and present any sort of number guessing game.
-use the INT command to round off decimal numbers.
-create programs to solve geometry problems.

Unit eight begins by introducing nested loops, then addresses creating and saving files of user inputted text on a floppy disk to be retrieved and used at another time. The initial task involves saving string variables in a text file and then reading them back. Next, one dimensional arrays are introduced for data storage and then two dimensional arrays for a data base problem.

8. 1 AND 2 DIMENSIONAL ARRAYS 2-3 Weeks

UNIT OBJECTIVES: The students will-
-understand nested loops.
-understand and use 1 and 2 dimensional arrays.

The final unit is the culmination of the entire curriculum. The students are to design their own program(s) utilizing as many concepts as possible.

9: PERSONAL PROGRAMS 2-3 Weeks

UNIT OBJECTIVE: The students will-
-use their knowledge to create applications programs that are of interest or use to them and instructor approved.

Class Structure

Discussion format. The class structure should be a discussion format as much as possible. Present new ideas and concepts as problems and work to solve them together. Create problems through discussion and supply new commands and techniques as solutions. As the course progresses, the structure of the programs and solutions will become familiar to the students. Make sure the four phases of programming/problem-solving are followed carefully: (a) problem specification, (b) planning, (c) coding, and (d) debugging

Problem-solving design. The first two, problem specification and planning, are the most critical. A great deal of care must be taken to make sure the students develop good methods for deciding what the programming problem is and planning a logical, workable solution.

The course does not include a unit on writing algorithms or flow charts. The reason for this omission is that individuals usually evolve their own form of problem-solving planning. Therefore, forcing them to utilize one particular method may inhibit individual development. In this course, the instructor guides the program structure initially by example and continues by providing templates or program chunks as well as complete programs throughout the course. These examples of expert programming, coupled with careful monitoring of code listings, will give the students the freedom to develop their own logically sound methods of problem-solving. Coding skills are strengthened through practice. Instead of teaching a few commands and writing short programs to use them and then moving on to new commands, the cumulative design of this course demands the repeated use of nearly all commands from the time they are introduced through the end.

The fourth step, debugging, is often the most difficult part of the problem-solving process. The students must be able to figure out what happened as opposed to what they intended to happen to correct the error. To do this, they need to understand exactly what each command does to be able to follow the flow of control to figure out where the problem occurred and make suitable modifications. This skill should be enhanced by having the students regularly debug faulty programs supplied by the instructor.

Cumulative learning. As discussed earlier, an important ingredient of problem-solving techniques is the top-down approach--breaking down a large problem into combinations of

small problems. Each part is then solved so that when combined they form the solution to the large problem. The proposed curriculum reflects this strategy by its cumulative design; the course in general, as well as each unit, begins with simple problems and proceeds to increasingly complex programs utilizing all previous commands and problem solutions.

Each unit is divided into sub-units that present one or two new concepts and related commands. The problem(s) as the end of each sub-unit can be used or, if desired, any other suitable problem that exercises the current concept can be substituted at the instructor's discretion. If it is determined that the students need more practice to acquire a working template of a particular concept, more problems should be solved before continuing.

Because the focus of this curriculum is to develop thinking skills, rather than simple knowledge of programming commands, the students should be encouraged to use the included problems as a basis for the development of their own problems. This allows a tremendous amount of flexibility and serves to motivate and hold interest. The freedom of this approach also lends itself to the customization of the course to students of differing abilities. The more advanced students can incorporate more concepts, employ more complicated programming templates and/or solve additional problems. The opportunities for problem-solving growth are limited only by time and the student's ability.

Tests. Tests are suggested at the end of and occasionally during each unit. Before discussing the several items that should be included to assess the student's knowledge and thinking skills, the one thing tests should NOT be is a simple test of BASIC commands. This is merely rote recall and gives no indication of the problem-solving skills they should be developing. A two-part exam, given on consecutive days (one day of written work and one day of on-line work), is probably the best way to adequately cover the following types of assessments.

Written questions should be included to test the problem definition and planning procedures utilized by the students: Present a problem and have the student define the problem specifically and write an outline or set of algorithms to solve it. The knowledge a student has of the commands will be evident in the coding part of the test and the explicit understanding the students have of the commands should be tested by predicting the outcome of and debugging supplied programs. Pass out appropriate programs or program segments and require students to determine

exactly how the program works and show the outcome. The debugging task is most effectively done on the computer. Supply a disk with buggy programs and have the students correct the bugs to render the program functional.

Student generated code should be evaluated for logical flow, modular programming and should include REM statements for each part of the program and all subroutines to make the program easy to read by others. The programs should also have a subroutine for any routine that is used more than once. Finally, to encourage modular, logical flow, the GOTO command should be used only as part of a conditional statement.

The course is divided into nine units. Each unit has a suggested length in weeks with the total equaling approximately 30. Since this curriculum was developed to be a problem-solving course, the first six weeks can be spent using problem solving software as a preliminary unit to the programming segment. If problem-solving software is not available or desired, the BASIC course can easily be expanded to 36 weeks by requiring more programming problems and/or more complex problems as the course progresses. The next portion suggests some appropriate problem-solving software and how to integrate it with the BASIC curriculum. The complete BASIC curriculum is included after the references.

Optional Problem-Solving Software Unit

As concluded in the review of research contained in the first portion of this project, the computer is an effective tool for developing problem-solving and higher-level thinking skills. There are two major ways to accomplish this: Computer programming and software designed to enhance these skills.

The ideas presented here are intended to comprise the first four to six weeks of a problem-solving course based on computer programming. Software may be used initially to familiarize students with the computer, thus reducing performance inhibiting anxiety and to raise their awareness of the process involved in solving problems. The four generally accepted problem-solving steps are: (a) analyze, (b) plan, (c) try, and (d) check. These four steps are equivalent to the four phases of programming/problem-solving--(a) specification, (b) planning, (c) coding, and (d) debugging--and should, therefore, be used in both the software section of the course as well as when teaching programming.

The first two weeks should be spent familiarizing students with the computers and the general process of solving problems. Games are a good way to overcome the initial anxieties of using the computer, learning the ins and outs of the machine, and laying a foundation for the development of problem-solving skills. There are many games that are not oriented toward subjects normally taught in high school, but are fun to use and do require problem-solving skills. There are also programs designed to develop problem-solving skills in a game format or make-believe situation. Try to have a variety of programs available and let the students choose which ones to use. Do not let them "cheat" by getting the answers or clues from other students. Encourage note taking and careful thinking rather than randomly trying solutions to solve the games. Once a program's problem has been solved, students should begin a new program rather than repeating the same one.

Some appropriate games:

Where in the World is Carmen San Diego?	Broderbund
Car Builder	Weekly Reader Family Software
Mystery Mazes	Educational Activities, Inc.
Heart of Africa	Electronic Arts

Specific problem-solving software in a game type format:

Incredible Laboratory	Sunburst Communications
Tribbles	Conduit
Discovery Lab	MECC
Discovery: Experiences in Scientific Reasoning	Millikan Publishing Co.

After the games have been used for one to two weeks, lead a class discussion about the four steps of solving problems. Discuss the games and relate them to the mental processes engaged by the students to work out their solutions. Talk about applying the processes to various games and problems presented by other school subjects such as math, science or English. Establish clearly the progression needed to be followed for all problems.

The next segment of the course will be similar to the first, except that the software will be oriented toward a subject normally taught in high school. The purpose is to apply the same problem-solving strategies to problems that are more relevant to the student. It will also help them see how to apply the strategies to different situations. Once again, it is important to let the students choose the first piece of software. This gives them a chance to develop the practical application of problem-solving in a field they enjoy. However, make sure successive programs are based on

different content to help the student see how the same processes apply to other areas. Some appropriate content oriented programs:

Math:	
Graphing Equations (Green Globs)	Sunburst
SemCalc	Sunburst
The Budget Simulator	EMC Publishing
Applying Mathematics	Laidlaw
The Whatsit Corporation	Sunburst
Survival Math	Sunburst
Social Studies and Geography:	
Social Studies Explorer Series:	Mindscape
Revolution and Constitution	
Westward Expansion	
Civil War and Reconstruction	
Central United States	
Western Europe	
Where in the World is Carmen San Diego?	Broderbund
Reading:	
Mystery Mazes	Educational Activities
Science:	
Botanical Gardens	Sunburst Communications
Oh, Deer!	MECC

The following part of the software unit is optional. If an intermediate step is desired between using applications software and computer programming, a logical choice would be the use of an authoring language or similar program requiring conceptual and procedural steps that are almost identical to those used in a programming language. Since most schools are limited to Apple II series computers, compatible authoring languages presently available are limited to Super Pilot. Authoring languages are simplified programming languages that are easy to use without requiring extensive time allotted to learning the program. If a program similar to Hypercard for the Macintosh becomes available for the Apple II , it would be an ideal choice. Authoring languages can help students learn concepts of program conditionals, structure, and program construction. The students can construct programs or games that they are interested in, as well as program problems presented by the instructor.

Target Audience

Age

The BASIC curriculum is designed primarily for students in grades nine through 12, but can be easily adapted for higher or lower levels (see curriculum extensions).

Ability and Prerequisite Skills

Unlike many programming courses, this curriculum will meet the needs of students with widely varying abilities because of its cummulative problem-solving design. Because graphics programming is intuitively appealing to beginners of all levels, the course begins with low-resolution graphics programming in the immediate execution mode for immediate feedback and then procedes quickly to delayed execution mode. After initial exposure to the minimum number of commands needed to solve simple programming problems and because programming for problem-solving emphasizes process, not product, students are encouraged to devise their own solutions. When students are given this sort of freedom, advanced students will create more complex programs while others will design less complex programs.

Curriculum Extensions

The proposed BASIC curriculum is easily adaptable to levels above or below high school. If the target audience is lower, the only difference will likely be difficulty in completing the curriculum because additional time may be required to complete the units. The extra time can be gained by either (a) eliminating the use of problem solving software as preliminary activities, (b) requiring less complex programs, (c) not completing all the units, or (d) a combination of the first three items.

If the curriculum is to be used for a post-secondary audience, it is recommended that the course be completed in one semester. This type of curriculum was used successfully in graduate classes that met once per week for three hours and required approximately six hours outside work (Reed, Palumbo & Stolar, 1988) and in undergraduate classes (Palumbo & Reed, 1988; Reed & Palumbo, 1988).

Possible Limitations

Lack Of Home Computers

The curriculum was designed to require computer access only in school. Homework assignments encompass (but are not limited to) planning in preparation for coding in class. An ideal situation would allow students access to computers outside class, either at home or during open times but is not necessary for effective implementation of the curriculum.

Insufficient Number Of Computers In Lab

The curriculum will also work when the ratio of computers to students is less than one to one. Students can work effectively in pairs, or small groups, possibly more effectively than working alone, due to student collaboration and peer feedback.

References

Burton, J.K. & Magliaro, S., (1988). Computer programming and generalized problem-solving skills: In search of direction. *Computers in the Schools, 4*(3/4), 63-90.

Clement, C.A., Kurland, M.D., Mawby, R. & Pea, R.D., (1986). Mapping the cognitive demands of learning to program, in R.D. Pea & K. Sheingold (eds.), *Mirrors of Minds*, 103-127.

Clement, C.A., Kurland, M.D., Mawby, R. & Pea, R.D., (1986). Analogical reasoning and computer programming, *Journal of Educational Computing Research, 2*(4), 473-485

Dalbey, J., & Linn, M.C. (1986). Cognitive consequences of programming: Augmentations to BASIC instruction. *Journal of Educational Computing Research, 2*(1), 75-93.

Dalbey, J., & Linn, M.C. (1985). The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research, 1*(3), 253-274.

Dalbey, J., Tourniaire, F., & Linn, M.C. (1986). Making programming instruction cognitively demanding: An intervention tudy. *Journal of Research in Science Teaching, 23*(5),427-436.

Kurland, D.M., Pea, R.D., Clement, C. & Mawby, R. (1986). A study of the development of programming and thinking skills in high school students. *Journal of Educational Computing Research, 2*(4), 429-455.

Lockard, J. (1986). Computer programming in the schools: What should be taught? *Computers in the Schools, 2*(4), 105-13.

Magliaro, S.B.,& Burton, J.K. (1988). Adolescents' chunking of computer programs. *Computers in the Schools, 4*(3/4), 129-138.

McCoy, L.P., & Orey, M.P. (1988). Computer programming and general problem solving by secondary students. *Computers in the Schools, 4*(3/4), 151-157.

Palumbo, D.B. & Reed, W.M. (1988). Intensity of treatment and its relationship to programming problem solving. *Computers in the Schools, 4*(3/4).

Peterson, C.G. & Howe, T.G.(1979), Predicting academic success in introduction to computers. *AEDS Journal, 12*, 182-191.

Reed, W.M., (1987). Teachers' attitudes toward educational computing: Instructional uses, misuses, and needed improvements. *Computers in the Schools, 3*(2), 73-80.

Reed, W.M. & Palumbo, D.B. (1988) The effect of the BASIC programming language on problem-solving skills and computer anxiety. *Computers in the Schools, 4*(3/4).

Reed, W.M., Palumbo, D.B. & Stolar, A.A. (1988). The comparative effects of BASIC and logo instruction on problem-solving skills. *Computers in the Schools, 4*(3/4).

Soloway, E., Lochhead, J. & Clement, J. (1982). Does computer programming enhance probelm-solving ability? Some positive evidence on algebra word problems. In R.J. Seidel, R.E. Anderson & S.B. Hunter (eds.), *Computer literacy: Issues and Directions for 1985* (pp.171-185).

Webb, N.M. (1985). Cognitive requirements of learning computer programming in group and individual settings. *AEDS Journal*, Spring 1985, 183-194.

UNIT 1

LO-RES GRAPHICS

UNIT 1
LO-RES GRAPHICS

UNIT OBJECTIVES:

The students will-

initialize a blank disk to use as a work disk
use simple commands to draw multicolored pictures on the screen in lo-res graphics.
learn how to edit source code and what to do when error messages occur.
SAVE, LOAD, edit and RUN programs.
understand the difference between immediate mode and delayed mode of execution.

UNIT LENGTH: 3-4 Weeks

TERMS:

Lo-Res Graphics
Pixels
Cartesian geometry
Immediate mode of execution
Delayed mode of execution

COMMANDS:

NEW
HOME
GR
LIST
COLOR=
PLOT X,Y
HLIN X(low),X(high) AT Y
VLIN Y(low),Y(high) AT X
REM statements
SAVE
END

MATERIALS:

Lo-res graph paper (X=0-39, Y=0-39)
List of colors
Printouts of sample programs and the results of their execution. Include a few programs in immediate mode for the first few sub-units and some delayed mode of execution for later programming when the students start using line numbers.

SPECIAL SUB-UNIT

INITIALIZING A BASIC PROGRAMMING DISK

SUB-UNIT OBJECTIVES:

The students will-

understand why a floppy disk must be formatted and how information(blocks) are organized on a disk.
know how to initialize a disk by writing 'Hello' program.

SUB-UNIT LENGTH: 1 Day

TERMS:

Initializing/Formatting
Source code
Sectors
Tracks

COMMANDS:

NEW
HOME
PRINT""
END
INIT xxxx

MATERIALS:

System masters
Floppy disks for each student
Floppy disk to take apart

PLAN:

Initializing is not part of Lo-Res graphics, but a disk must be prepared to boot the computer. You may wish to supply a disk for booting and save this sub-unit until you begin writing programs that will be saved.

All students will have a blank data disk. Discuss how a computer breaks up a disk into sectors and stores data. Show a diagram of a sectored disk. Different kinds of computers organize disks in different ways which is why the disk must be formatted for the particular computer being used. If you have a disk available for disection, show the parts inside. Talk about the storage of information being similar to audio tape. Also take the time to talk about the use and care of floppy disks.

PROBLEM:

Entering a 'Hello' program for the disk (Initializing a disk).

An example "Hello" program will be made available to the students to copy. This will not be an exercise in programming; rather, it will be mer· opying and following directions. Undoubtedly there will be some editing questions wh. will be dealt with as needed.

Boot the computers with Systems Masters and then collect the masters. Enter the NEW command that clears out memory and press Return. Now enter the "Hello" program:

```
10 HOME
20 PRINT "BASIC programming data disk for: (student name)"
30 END
```

After typing in the program, enter:

INIT HELLO and then press <RETURN>

Point out that the students may put their own message after the PRINT statement but make sure their name is included.

SUB-UNIT 1

OBJECTIVES:

The students will-

know the size of the screen (40 X 40)
understand pixels and how they are used to make pictures
orient themselves to the X and Y axis for drawing
learn BASIC commands for drawing

SUB-UNIT LENGTH: 1 day

TERMS:

LO-RESolution Graphics
Pixels
Cartesian Geometry
X and Y axis

COMMANDS:

NEW
GR
COLOR=
PLOT
HLIN
VLIN

PLAN:

Before beginning to code with program lines, two class periods will be spent programming in the immediate mode of execution. That is, the graphics mode will be entered and commands will be typed in without line numbers causing the results of the commands to appear immediately. This will not only provide immediate feedback to the students, but will also underscore the need for line numbers when introduced in the third lesson.

The first few classes will be more lecture than discussion. Begin by talking about the characteristics of Lo-Res graphics incuding the screen size, x axis, y axis (similarity to and difference from cartesian geometry). The next step is to discuss the commands that are used to draw pictures, starting with pixels and how they are turned on and off plus the following commands:

COLOR=
PLOT
HLIN, VLIN

Before beginning to program, you must go through the steps of booting the computer with the data disks, even though no programs are yet being saved.

PROBLEM:

As part of the discussion, a very simple graphics program will be started, programmed, and run as a group: The aforementioned commands will be included as well as the following commands:

NEW-to clear memory for new program
HOME
GR
COLOR= (students will find their own combinations of colors that look good on monochrome screens)
PLOT
HLIN
VLIN

Error Messages-What to do-the only error likely to occur at this point is 'SYNTAX ERROR'-resulting from an improperly entered command.

The remaining time will be spent by the students programming in immediate mode--either modifying or adding to the class program.

SUB-UNIT 2

OBJECTIVE:

The students will-

preplan a Lo-Res picture complete with coordinates and then program it in the immediate mode of execution.
follow a given program and predict the results of execution.

SUB-UNIT LENGTH: 1 day

PLAN:

Begin the class by having the students spend 10 or 15 minutes programming in the immediate mode to review yesterday's activity, and then review the commands and terms used yesterday. Next pass out a simple graphics program, have the class predict the outcome, and draw the results on the board.

Discuss the importance of preplanning. It may be useful to point out that it is easy to draw over a previously drawn entity if the drawing is not preplanned carefully.

PROBLEM:

Students will be given a few minutes to decide on something simple to draw in two colors and sketch it on graph paper with most coordinates marked. All 3 drawing statements must be used and no 2 adjacent dots may be drawn using the PLOT command The remainder of class time will be for programming by the students.

PROBLEM: -HOMEWORK

1. Students will be given 2 short graphics programs. They will sketch the results on the given graph paper.
2. Students will be supplied with several pieces of lo-res graph paper. They will sketch more detailed pictures than have been done thus far and write out the code on paper to be turned in.

EVALUATION:

The predictions of the short programs will be turned in for grading.

SUB-UNIT 3

OBJECTIVES:

The students will-

understand the difference between immediate and delayed mode programming.
program in delayed mode, edit source code by retyping lines, SAVE, CATALOG, RUN, RUN xxxx, and LOAD programs and use REMs.

SUB-UNIT LENGTH: 5 days

TERMS:

Delayed mode of execution
Source code
Editing
DOS commands
REM statements

COMMANDS:

SAVE xxxx
RUN
RUN xxxx
LOAD xxxx
REM
CATALOG

MATERIALS:

Student data disks

PLAN:

Review the term immediate mode of execution and lead into delayed mode. Discuss the use of line numbers by writing a simple graphics program on the board and adding the line numbers. Then introduce the RUN command to execute the program. Explain why the lines should be numbered by tens and illustrate by inserting some lines in the program on the board. Now, even though some students will know how, discuss editing the lines by retyping the line number and its command, the LIST command which outputs the program code to the screen, and how to stop the scrolling if the source code is long.

Talk about how confusing a program listing can be to someone who has not programmed the code and introduce the REM statement. The REM statement must be used liberally in the student's source code.

The final topic for the lesson is the DOS commands needed to SAVE and LOAD students programs, and CATALOG the work disk.

PROBLEMS:

INCLASS:
1. Enter the source code written for homework as delayed mode of execution source code. Do not forget REM's
2. Practice editing programs and the following comands: SAVE, RUN, LOAD.

HOMEWORK:
3. Plan and write a picture program in 5 colors utilizing graph paper. The source code must be complete and contain REM statements for each major portion of the picture.

EVALUATION:

The source code that was written for today will be turned in for grading.
When the picture programs are finished, have a Show and Tell. Assign grades for the programs andcheck the source code for REM statements.

SUB-UNIT 4

OBJECTIVES:

The students will-
understand program structure utilizing subroutines: organizing programs into chunks that are easily identifiable.

SUB-UNIT LENGTH: 2 days

TERMS:

Program structure
Sub-routines
Chunks

COMMANDS:

GOSUB
RETURN

MATERIALS:

A hardcopy of a sample picture program in lo-res utilizing subroutines and REM statments.

PLAN:

Discuss program structure that emphasizes logical sequence. Programmers must clearly define the problem and progressively break it down into logical, solvable, steps which in turn results in code that is easy to read and debug. Introduce subroutines as a way of putting parts of the program (different parts of a picture in this case), in their own part of the program. Put a portion of a program on the board and then separate part of it by putting it into a subroutine complete with a REM statement. Continue by using student input to create a more complex picture that places the major program components in subroutines.

PROBLEM:

Homework: Write the source code for a lo-res graphics program containing a minimum of two subroutines. The program will be coded in class.

SUB-UNIT 5

OBJECTIVES:

The students will-
understand and use the PRINT statement.
use a FOR/NEXT loop to delay parts of the program.

SUB-UNIT LENGTH: 2 days

TERMS:

Numeric variable
Delay loop

COMMANDS:

PRINT
FOR/NEXT

MATERIALS:

A sample program containing bugs to be debugged and the output predicted.

PLAN:

Explain the use of the PRINT statement to add a title to the lo-res programs in the text window. Now introduce the idea of putting a delay loop in the program by having the computer count. Explain the use of the FOR/NEXT loop, the concept of variables and the length of time required to count: It takes an Apple II 1 second to count from 1 to 750.

PROBLEMS:

Add a subroutine that prints the current programs title in the text window.
Add a subroutine to delay the presentation of some portion of the program.
HOMEWORK: Debug the supplied program and predict its output.

SUB-UNIT 6

OBJECTIVES:

The students will-

use a variable to draw a background color.
change the COLOR= command to change background color.
print out the source code of their programs.

UNIT LENGTH: 1 day

TERMS:

Hardcopy

COMMANDS:

FOR/NEXT
PR#1
PR#0

MATERIALS

Sample program containing a numeric variable and a FOR/NEXT loop to draw a large block on the screen.

PLAN:

Begin by pointing out how time consuming it would be to draw a complete background or large objects with separate HLIN or VLIN statements. Put several statements on the board as if a complete background were to be drawn. Ask what the emerging pattern is and then point out that there is an easier way using variables in a manner similar to the for next loop used for delays. If a particular statement could be executed over and over with the one value changing, then the program would be much simpler. Extend the concept of loops by re-explaining the for next loop and how it can be used to change the value of a variable in the particular statement that needs to change each time. The students should be able to come up with a working subroutine to draw a background from the discussion and the example program given.

Since the programs are beginning to be rather involved, it is time to print out the source codes to help visualize the entire program and aid in debugging. Show the students how to bring the printer on line by entering the PR#1 command, obtain a hardcopy of their programs by entering LIST, followed by PR#0 to close the printer.

PROBLEMS:

Add a subroutine to the current program to draw in a background color.

Make sure the students understand that the color can be changed very simply by either changing the color statement in the subroutine or putting the COLOR= statement in the main body and then sending the program to the subroutine to draw the background. Since the background is being added now, someone will undoubtedly cover their picture by drawing the background after the picture. This problem should be solved through discussion.

This is a good time to emphasize the importance of subroutines and numbering by tens as it would be impossible to add the background routine if there was no room to insert a GOSUB statement.

EVALUATION:

A hardcopyof the source code for the completed program is to be graded.

SUB-UNIT 7

OBJECTIVES:

The students will-

use a variable to draw large objects
use a variable to draw diagonal lines

SUB-UNIT LENGTH: 4 Days

PLAN:

Point out that the subroutine used to draw a background can be used to draw large objects on the screen, just as the sample program (in Sub-Unit 6) that led to the background routine did. Give some examples on the board.

Start a discussion on diagonal lines, beginning with the simplest (top-left to lower-right). Write several PLOT statements to begin a diagonal line. Students should quickly point out the pattern emerging for the X and Y axis. Steer the discussion to the loops to draw large objects and guide the students into coding a routine for drawing the diagonal line. After this is done, ask the class to figure out how to draw a diagonal line from the following directions:

lower-right to upper-left
upper-right to lower-left
lower-left to upper-right

Emphasize the importance of running the program each time a part is completed to help eliminate debugging time. (Also emphasize the importance of saving the program before running, particularly when working with loops as the program can become locked in an endless loop.)

PROBLEM:

Write a program that draws diagonal lines starting in different corners and/or different places on the screen.

HOMEWORK: Outline a program that includes subroutines to draw:
a background
diagonal lines from all corners
borders
picture(s)

EVALUATION:

A show and tell, as well as the hardcopy of the finished programs will be graded.

TIME FOR A TEST!!!
A work sheet with all the commands and terms learned thus far should be provided for review. Also included should be a worksheet with buggy program lines and short programs to debug in class and at home. The test should be part programming, part debugging and part predicting the result of given programs.

UNIT 2

ANIMATION IN LO-RES GRAPHICS

UNIT 2
ANIMATION IN LO-RES GRAPHICS

UNIT OBJECTIVES:

The students will-

use FOR/NEXT Loops and numeric variables to animate simple one color and multi-color objects across the screen from all directions.

UNIT LENGTH: 2 weeks

TERMS:

Animation
Animation loops

COMMANDS:

FOR/NEXT

SUB-UNIT 1

OBJECTIVE:

The students will-

use a FOR/NEXT loop and a numeric variable to animate a simple one color object across the screen from left to right.

SUB-UNIT LENGTH: 2 days

PLAN:

Begin by coding and drawing a simple object (such as a block car) on the screen/board. The COLOR= command will probably be placed inside a subroutine by the students. This is OK at this point. Help the class figure out how to move the object across the screen. It should not be difficult for them to realize that the X value needs to be increased to move the car. They will most likely NOT know that the object must be erased each time. When the discussion evolves to the point where the need to successively draw and erase the object, it should become clear that a loop is necessary. (The COLOR= statement must be issued for the object, the subroutine called to draw the object, then COLOR= the background color, and the subroutine called again to erase the object.) Finally, the issue of the limits must be addressed. Point out the size of the screen and what happens when those limits of 40 X 40 are exceeded.

PROBLEM:

HOMEWORK-code or at least diagram/outline a program to animate an object from left to right across the screen. The object should be simple (not larger than 8 x 10 pixels) to be coded and run tomorrow in class.

EVALUATION:

The show and tell, and the hardcopy of the source code will be graded.

SUB-UNIT 2

OBJECTIVE:

The students will-
animate an object of at least two colors across the screen.

SUB-UNIT LENGTH: 3 Days

MATERIALS:

A sample program that animates an object of at least two colors in one or more directions across the screen. Give the students a hardcopy of the source code.

PLAN:

Begin by drawing and coding a two-color object on the screen/board. Guide the discussion to the drawing and erasing process of animation and the need to create a subroutine for each part of the object. Include a diagram on which the students can base their own model.

The animation program will be at least partially planned during the remaining class time so make sure to monitor the student's activity closely.

PROBLEM:

Begin writing a plan for a program that will animate a 3-color entity across the screen.
HOMEWORK-complete the planning for the program so coding can begin in class tomorrow.

EVALUATION:

Grades will be given on 3 items:

Diagram/outline
Source code
Show and Tell

SUB-UNIT 3

OBJECTIVE:

The students will-
animate objects in four directions

SUB-UNIT LENGTH: 3-5 Days

PLAN:

Because of the similarity to using variables to draw diagonal lines, it should not be difficult to guide the discussion to the discovery of how to animate in different directions, adding or subtracting the X and/or Y value. Talk about adding a background color and stationary objects to the picture. You MIGHT point out that the animated object should not cross a stationary object as it will erase it.

PROBLEM:

Write a program that includes:

A background color
Stationary objects
one or more objects that move in all four directions.

BONUS: animate 2 or more objects in different directions at the same time.

EVALUATION:

The show and tell, and source code will be graded.
If animated objects are more than one color, extra credit will be given.

THIS CONCLUDES THE LO-RES UNITS. DEVELOP REVIEW WORKSHEETS AND A TEST. THESE SHOULD INCLUDE ALL TERMS AND COMMANDS, SAMPLE PROGRAMS TO PREDICT RESULTS, PROGRAMS TO DEBUG AND A PROGRAMMING SECTION.

UNIT 3

HI-RES GRAPHICS INCLUDING ANIMATION AND EMBEDDED DOS COMMANDS

UNIT 3
HIGH-RES GRAPHICS INCLUDING ANIMATION AND EMBEDDED DOS COMMANDS

UNIT OBJECTIVES:

The students will-

be able to draw static pictures in Hi-Res graphics
demonstrate an understanding of the compatibility of Hi-Res colors.
animate single and multicolor objects in Hi-Res.
animate objects on a background drawn in Hi-Res.
use text commands to introduce a graphics program and embedded DOS commands to load and run the programs.
use embedded DOS commands to alternate between Hi-Res page 1 (HGR) and Hi-Res page 2 (HGR2).

UNIT LENGTH: 5 Weeks

COMMANDS:

HGR
HGR2
HCOLOR=
HPLOT X,Y
HPLOT X,Y TO X,Y

Screen size is 280 x 160. (X=0-279 Y=0-159)
Colors = 0-7

MATERIALS:

Hi-Res graph paper

SUB-UNIT 1

OBJECTIVES:

The students will-

understand the commands to draw a static picture in Hi-Res graphics.
understand the color compatibility problems in Hi-Res graphics

SUB-UNIT LENGTH: 5 Days

TERMS:

High resolution
Pixels

COMMANDS:

HGR
HCOLOR=
HPLOT X,Y
HPLOT X,Y TO X,Y

MATERIALS:

Hi-Res graph paper
List of colors
List of commands and their function
Sample program to draw a Hi-Res picture

PLAN:

The class discussion should begin with a quick review of how Lo-Res programs draw pictures (by coloring pixels) and then move to the advantages of having higher resolution. Introduce Hi-Res graphics verbally and visually by showing a Hi-Res picture and comparing it to Lo-Res graphics. Give the screen size (280 x 160 {X=0-279 Y=0-159}).

To introduce the drawing commands, compare and contrast the Hi-Res commands with their Lo-Res equivalents.

GR = HGR
COLOR= = HCOLOR=
PLOT X,Y = HPLOT X,Y
HLIN X,X AT Y and VLIN Y,Y AT X = HPLOT X,Y TO X,Y

Explain the HPLOT command as drawing from one point to another. Point out the advantage of this for drawing non-vertical and horizontal lines. The HPLOT can also continue to draw connected line segments by linking the coordinates together with "TO":

HPLOT 20,0 TO 40,0 TO 40,20 TO 20,20 to 20,0

draws a box on the screen with the upper left corner at 20,0. Students must be careful to put the coordinates in the correct order to draw the desired entity. The same set of coordinates, combined in a different order will result in something completely different.

Finally, go through the steps of creating a subprocedure that uses a loop to draw a color background, followed by a second (brief start) procedure to draw a picture. Make sure you emphasize using loops to draw larger objects.

Discuss the available colors, the color families and their compatibility problems.

Colors	0=BLACK	4=BLACK
	1=GREEN	5=ORANGE
	2=VIOLET	6=BLUE
	3=WHITE	7=WHITE

The compatibility problems with Hi-Res graphics must be explained carefully to the students: If the X coordinate is an even number, the COLOR must also be even or the pixel will not change. If the X coordinate is an odd number, the color must be odd also. This does not cause a lot of problem but may be confusing. If a line is drawn vertically and does not match the X coordinate, it will not appear. The solution is to increase or decrease the X coordinate by one. It is suggested that students draw their entity in WHITE (#3)because white is compatible with odd and even numbers. After the entity is done, add or change the color statements and make any X corrections that are necessary. The other compatibility problem is the bleeding together of colors when the two families are mixed. When a color of one family is drawn over a color of the other family the edge where they meet does unpredictable things. The only color that is compatible with both families is the number 3 white.

Spend any remaining time experimenting with the Hi-Res graphics mode.

PROBLEM:

HOMEWORK-
Follow a given program and draw its results on paper, including coordinates and colors.
Develop a drawing using graph paper:
Sketch the picture on the graph paper-- a background is required.
Fill in the coordinates.
Write a plan for the program.
Write out the code.

THE INDIVIDUAL ENTITIES MUST BE PLACED IN SUB-ROUTINES
THE PROGRAM MUST HAVE REM STATEMENTS
CONNECTED LINE SEGMENTS SHOULD BE CONNECTED BY "TO" IN A SINGLE HPLOT STATEMENT

Subsequent classes will begin with a question and answer session on the homework and then move into a work session.

During the work session all homework will be checked by going around to compare what the students are actually coding to what is on their plans.

As many days as are necessary to satisfactorily complete at least one picture can be taken. Animation is the next sub-unit so additional proficiency will be gained in Hi-Res graphics.

EVALUATION:

The program plans will be graded.
The codes and the show and tell will be graded.

SUB-UNIT 2

OBJECTIVE:

The students will-
animate a single color entity in Hi-Res graphics.

SUB-UNIT LENGTH: 2 Days

COMMANDS:

STEP + X
STEP - X

MATERIALS:

2 sample animation programs: (a) a program to demonstrate an animation program's output and (b) a program to be used for output prediction.

PLAN:

Begin by showing an instructor-generated animation program. Then pass out a program listing of the second instructor-generated sample program and have the students predict what the program will do and what the similarities to Lo-Res are. Discuss the outcome of each program segment and then begin creating an animation program on the board using the same programming techniques found in the sample program. The first student created Hi-Res animation program will be the animation of a single color entity.

PROBLEM:

Write a program to animate a single color entity in HI-Res graphics and complete the code at home tonight.
NOTE: Animation in a diagonal direction will be worth extra points.

EVALUATION:

The show and tell will be graded. If it seems necessary, collect program hardcopies for grading.

SUB-UNIT 3

OBJECTIVE:

The students will-
animate a multi-color entity two directions in Hi-Res graphics.

SUB-UNIT LENGTH: 3 Days

MATERIALS:

Sample program that animates a multi-color object in Hi-Res that students will be able to use for reference.

PLAN:

Lead a discusion to define Hi-Res animation programming. Refer to the instructor-generated sample program while creating another program on the board based on student input. Compare the program's structure to the Lo-Res animation programs written earlier. Review the procedures (color changes and subroutines to draw and redraw [erase]) required for each differently colored part of the animated object.

PROBLEM:

Create a program in Hi-Res to animate a multi color entity in two successive directions.

EVALUATION:
The show and tell will be graded.

SUB-UNIT 4

OBJECTIVE:

The students will-
animate one or more multi-color entities in two or more directions.

SUB-UNIT LENGTH: 3 Days

MATERIALS:

A sample program for student reference.

PLAN:

Through a class discussion, point out the commands needed to animate two objects at the same time. Two variables may be used if necessary.

PROBLEM:

Modify the previously created program or create a new one that will animate at least two objects with a minimum of two colors each in different directions at the same time. An option for those having trouble is to animate one object then the other.

EVALUATION:

The (a) show and tell and (b) program listings will be graded.

Test: This is an appropriate place for a test on Hi-Res graphics. A programming only test should be adequate.

SUB-UNIT 5.0

SUB-UNIT OBJECTIVE:

The students will-
use DOS commands to save pictures drawn with graphics software on disk.

SUB-UNIT LENGTH: 5 Days

TERMS:

Binary files

COMMANDS:

PRINT:PRINT CHR$(4);"BSAVE";xxxx;",A$2000,L$2000"
PRINT:PRINT CHR$(4);"BSAVE";xxxx;",A$4000,L$2000"

MATERIALS:

A graphics package such as *E-Z draw* (public domain) or, if possible, *816 Paint* (requires a Koala pad).

PLAN:

Tell the students that they will be creating two pictures with a graphics package. They will then create a driver program to present each picture in succession and then alternate between the two at an interval specified by them and repeated as many times as they wish.

Discuss the procedure to be followed: Two pictures will be drawn. They may be completely different or very similar to produce a simple animation effect. By this point in the course, the instruction needed to for using the graphics package can likely be limited to learning to chang colors and draw a few simple shapes and lines on the screen. The students can solve most other problems they will run into.

Next, talk about how a graphics screen can be saved as a whole, separately from the program used to draw it, by saving it as a binary file. This is a good time to teach a little about how information is saved. Then write out and explain the commands needed to save the pictures. Hi-Res page 2 will have to be used, so make sure to point out that the only difference between page 1 and page 2 is that page 2 does not have a text window at the bottom of the screen.

PRINT:PRINT CHR$(4);"BSAVE";xxxx;",A$2000,L$2000"
PRINT:PRINT CHR$(4);"BSAVE";xxxx;",A$4000,L$2000"

A$2000 and A$4000 are for page 1 and page 2 respectively
L$2000 tells the computer to save the whole picture as a binary file.

PROBLEM:

Design two pictures and draw them. To save time, if the pictures are going to be similar, you can create the first one and save it, then alter it to create the second picture and resave it under a new name.

The remaining class time should be spent planning the pictures. The students should be ready to begin creating their pictures tomorrow.

SUB-UNIT 5.1

OBJECTIVE:

The students will-
use embedded DOS commands to load and present pictures created with the graphics software on pages 1 and 2 of Hi-Res.

SUB-UNIT LENGTH: 2 Days

COMMANDS:

PRINT:PRINT CHR$(4);"BLOAD xxxx,A$2000"
PRINT:PRINT CHR$(4);"BLOAD xxxx,A$4000"
POKE - 16300,0
POKE - 16299,0

PLAN:

A simple program needs to be written to alternately display the pictures. The program itself is not difficult but the commands are easy to make mistakes on when typing them because of the syntax. As a class, first write a program simply to display the pictures. This will give the students some experience with the commands and allow them to work out the bugs.

Tell the class what they are going to do and construct a sample program on the board. Make sure that there are delays between the pictures. The program will look similar to the one on the following page:

```
10 HOME
20 HGR
30 PRINT:PRINT CHR$(4);"BLOAD xxxx, A$2000"
40 HGR2
50 PRINT:PRINT CHR$(4);"BLOAD xxxx, A$4000"
60 FOR R = 1 TO 10
70 POKE - 16300,0
80 FOR T = 1 TO 750:NEXT T
90 POKE - 16299,0
100 FOR G = 1 TO 750:NEXT G
110 NEXT R
```

Explain that the BLOAD commands get the pictures from the disk and load them on to the graphics "pages", and the POKE commands are the ones that actually cause them to be displayed.

PROBLEM:

Write a program to alternately display two scenes drawn with the graphics package. The speed and number of repetitions is to be determined by the student.

SUB-UNIT 5.2

SUB-UNIT OBJECTIVE:

The students will-

create a picture with the graphics software and use it as a background for an animation program in Hi-Res graphics.

SUB-UNIT LENGTH: 5 Days

COMMANDS:

PRINT:PRINT CHR$(4);"BLOAD xxxx,A$2000"
PRINT:PRINT CHR$(4);"BLOAD xxxx,A$4000"
POKE - 16300,0
POKE - 16299,0

PLAN:

There should be no difficulty in explaining the idea of using a picture drawn with the graphics package for a background in Hi-Res. (They have already saved picturez and loaded them onto page 1 and 2.) Students should also know that a new entity can be drawn over an old one. The only point that needs to be emphasized is that the object to be animated needs to stay within a single color on the background to avoid the more complicated programming that successfully animates an entity across colors (the entity needs to be redrawn in a different color in order to be erased which is difficult as it makes the transformation from one color to the other).

PROBLEM:

Draw a picture with the graphics software, load it onto page 1 or page 2 and animate a multi-color (minimum two) object in any direction.

EVALUATION:

Grade the Show and Tell

UNIT 4

TEXT COMMANDS FOR SCREEN FORMATTING

AND AN

INTRODUCTION TO STRING VARIABLES

UNIT 4

TEXT COMMANDS FOR SCREEN FORMATTING AND AN INTRODUCTION TO STRING VARIABLES

UNIT OBJECTIVES:

The students will-
- learn some text formatting commands.
- use a delay loop
- use simple text commands to introduce a graphics picture and embedded DOS commands to automatically load and present the picture.

UNIT LENGTH: 2 Weeks

TERMS:

Embedded DOS commands
Text mode vs Graphics mode
Text formatting
Delay loop

COMMANDS:

TEXT
FLASH
INVERSE
NORMAL
SPEED=
HTAB
VTAB
: (colon for putting multiple commands on the same program line)
FOR x = 1 to 500 : NEXT x
PRINT:PRINT CHR$(4); RUN "xx"
PRINT:PRINT CHR$4(4); BLOADA$2000 xxxx,A$2000
PLOKE-16300,0

MATERIALS:

List of commands needed and their function
Sample program to show and pass out.

SUB-UNIT 1

OBJECTIVE:
The students will-

use the SPEED=, HTAB, VTAB, INVERSE, FLASH and NORMAL text formatting commands to create an attractive paragraph.

SUB-UNIT LENGTH: 3 Days

PLAN:

Point out that, although graphics may be used to accompany future projects, graphics programming instruction is complete. The task at hand will focus on text commands that will be used to manipulate the appearance of text on the monitor. The print statement was used before to "title" some Lo-Res graphics, but the computer has never been used in the text mode. Explain that this unit will consist of:

1. a paragraph written to introduce either a Hi-Res or Lo-Res graphics program that has already been created
2. a keypress to trigger the picture display
3. a delay loop to (a) present the picture for a specified time and then (b) cancel the picture and display another message.

The first step is to do some text formatting. Point out that one of the first steps done in graphics program was to issue a GR or HGR command to invoke the graphics mode. (The computer automatically goes into text mode when booted, which is why the graphics call had to be made.) Now the text mode will be used to present text--a paragraph explaining a previously drawn graphics picture. Give the screen size for text (40 characters/columns by 24 lines). Discuss the appearance of text making the following points:

1. Do not fill an entire screen with text-it is hard to read.
2. It is easier to read double spaced text than single.
3. It is often nice to leave some space at the top of the screen.
4. It is often nice to leave space at the left side of screen.
5. Attention can be drawn to words by changing their appearance.
6. INVERSE-creates a white background with black letters.
7. FLASH-alternates between the inverse and normal text.

The above can be accomplished with the following:

1. A PRINT statement without any text will print a blank line.
2. VTAB (y) moves the cursor to the specified line
3. HTAB (x) moves the cursor to the specified column.
4. INVERSE invokes the inverse mode
5. FLASH invokes the flash mode
6. NORMAL sets the mode back to normal.

7. SPEED= alters the presentation of successive characters on the screen. The default value is 255. Make sure SPEED is set back to 255 at the end of a program.

8. Any of the above can be combined to control the text.)

Mention that program lines/commands can be combined with a **colon.**
If a sentence has NORMAL and FLASH/INVERSE text, the **semi-colon** is used to keep it all on one line:

```
10 PRINT "normal text";FLASH; PRINT "flashing text"; NORMAL; PRINT"normal text
    again"
```

At this point involve the class in creating the beginning of a program to display text.

PROBLEM:

HOMEWORK: Create a paragraph of about 100 words to describe a picture of your choice. Decide what you want the screen display to look like and indicate these intentions on a hard copy of the paragraph. Utilize (a) HTAB, (b) VTAB, (c) FLASH, and (d) INVERSE.

INCLASS: The paragraph written for homework will be coded in class the next 2-3 days. The students should have a sample program to refer to.

SUB-UNIT 2

OBJECTIVES:

The students will-
use FOR/NEXT loops to delay text presentation.
use an embedded DOS command to run a program.

SUB-UNIT LENGTH: 2-3 Days

TERMS:

Delay loop
Variable
Embedded DOS commands

COMMANDS:

FOR/NEXT
PRINT:PRINT CHR$(4); "RUN xx"

PLAN:

The class has written a paragraph to introduce a graphics program. Explain that since their introductory paragraph is a different program from the graphics program, there must be a way of starting the graphics program after the text program without going through LOAD xxxx and RUN xxxx. They will be familiar with the term "DOS command" by now, so merely tell them that a DOS command may be placed in a program to be executed "automatically." Put an example of the

RUN "xx" command on the board and point out that it is necessary to tell the computer that a DOS command is about to be encountered by prefacing the command with PRINT:PRINT CHR$(4);. If the programmer wishes to run another program after the completion of one program the following command is necessary (assuming the last line of the program is line 390):

```
390 PRINT:PRINT CHR$(4); "RUN xx"
```

When the first program is complete and line 390 is encountered BASIC will automatically RUN xx.
CAUTION: Make sure to stress the importance of SAVING a program that contains the embedded DOS command RUN before running it. When a new program is run, the old program is purged from memory. If there were any changes made to the program before it was run, they will be lost.

Before the students add the embedded RUN command to their programs, it is necessary to explain Delay Loops. Begin by explaining that their introductory paragraphs will be seen only briefly because the computer will print the text and then automatically go to the RUN command. The computer can be slowed down to give the reader a chance to finish the text by making the computer count. The higher it has to count, the longer the delay. To make the computer count some lines need to be added after the introductory text and before the RUN command. Explain that a subroutine containing a counting loop is to be created in which the computer stays and counts until it reaches the amount specified by the programmer. Loop construction begins by naming a variable and specifying the range. Write on the board:

```
1000 REM ******* Counting routine ********
1010 FOR COUNT = 1 TO 500
1020 NEXT COUNT
1030 RETURN
```

Explain the use of the variable and the program control: (a) starts at 1,(b) adds 1 to itself when it reaches NEXT, and (c) goes back to FOR etc. until it reaches 500. Emphasize that the upper limit should be varied until the loop is an appropriate length.

Now insert the FOR/NEXT loop in the hypothetical program on the board between lines 380 and 390 by using the GOSUB command to send it to the counting subroutine.

PROBLEMS:

Add a subroutine to the text programs to delay the presentation of the second page of text if you have one.
Add the the embedded RUN command.

BONUS: Use a delay loop to delay certain parts of the text.
More subroutines must be created if different length waits are desired.

EVALUATION:

Programs will be printed and turned in for a grade.
The show and tell will be graded.

SUB-UNIT 3

OBJECTIVE:

The students will-

gain a rudimentary understanding of the difference between a numeric variable and a string variable.
understand and use the GET command to create a "page turn" subroutine.
understand that one subroutine may be called from another.

SUB-UNIT LENGTH: 3 Days

TERMS:

Numeric variable
String variable

COMMANDS:

GET x$

PLAN:

Begin by recapping all the modifications made to the animation program. Call attention to the fact that there are many different reading speeds as evidenced by the differences in the delay loops presented at the show and tell. The point of this class will be to make additional changes to allow the page to be turned when the READER wants. To do this a subroutine will be created that waits for the reader to press a key on the keyboard when he/she wishes to continue. Lead a discussion to create the subroutine on the board:

```
5000 REM*****Page Turn *****
5010 VTAB 24
5020 PRINT "Press any key to continue"
5030 GET A$
5040 HOME
5050 RETURN
```

Naturally, the explanation about String and Numeric variables will take place when line 5030 is encountered. Point out that previously the variable was numeric (in the delay loops) because the computer performed a math function with it. The word string refers to any alphanumeric character used for anything but a math function. When the GET command is encountered, the computer waits for any alphanumeric character to be typed in by the user. The character is stored under the variable name, just like the numeric variable was, but in this case it is not going to be used for anything but a signal for the computer to continue executing the program. (NOTE: This a brief discusion of string and numeric variables-more detail is contained in the next unit.) Before the discussion ends, draw attention to the fact that delay loops should still be used to delay the presentation of the page turn prompt. The length should be shortened appropriately. Demonstrate good program structure on the board, including a delay subroutine and a page turn subroutine.

PROBLEM:

Make the necessary changes to the text-graphics program to do the following:
The intro program should be split into two pages if not done already
Use appropriate margins and text placement
Incorporate the page turn routines
Write a short program to add text after the graphics
Insert another embedded DOS command in the graphics program that executes the closing text program.

OPTIONAL: Add some text to the text window in the graphics mode. Point out that if Hi-Res graphics is used, a VTAB command is necessary to place the text in the text window.

EVALUATION:

The show and tell will be graded
The program listings will be graded. Compare listings to the previous edition of the same program.

TEST: Create a one day test that includes:
Programming with the text formatting commands
Program prediction
Program debugging

UNIT 5

STRING VARIABLES, NUMERIC VARIABLES

AND

MORE EMBEDDED DOS COMMANDS

UNIT 5
STRING VARIABLES, NUMERIC VARIABLES AND MORE EMBEDDED DOS COMMANDS

UNIT OBJECTIVES:

The students will-

use numeric and string variables to create a question and answer program.
know the symbols for the four basic math functions.
use conditional statements for responding to user input and error trapping.

UNIT LENGTH: 3 Weeks

UNIT TERMS:

String Variable
Numeric Variable
Input statements

UNIT COMMANDS:

PRINT
INPUT x$
GET x$
INPUT x
GET x

UNIT MATERIALS:

Sample programs containing the new concepts

SUB-UNIT 1

OBJECTIVE:

The students will-
use the PRINT and INPUT commands to create a simple question and answer routine.

SUB-UNIT LENGTH: 3 Days

PLAN:

Let the class know that they will be using their knowledge of text formatting commands and variables to create programs that interact with the user. These programs will (a) ask questions, (b) remember the answers, (c) perform math functions on the answers, and (d) even check to see if certain answers are given in response to questions (error trap). The programs will start out fairly simple (like the graphics programs) and be added to and/or changed to become more complex.

The first program will be a simple question and answer routine. Begin construction of the program on the board using the class to provide the program lines. Use the standard first program lines: HOME, and REM statements to title the program.

```
10 REM***** Question and Answer *****
20 HOME
30 VTAB 5
40 PRINT "What is your name"
50 INPUT NAME$
```

The class will suggest using GET NAME$, so the difference between GET and INPUT can be explained. Review the characteristics of the string variable.

```
60 PRINT "How are you ";NAME$
```

Make sure you explain the use of semi-colons to insert variables and the blank spaces needed within the quotation marks for proper placement of the variables. Continue the program with something like:

```
70 PRINT "How old are you?"
80 INPUT AGE$
90 PRINT AGE$;" sounds like a good age!"
```

PROBLEM:

INCLASS-The remaining time is to be spent creating a question and answer routine. Save the program on disk.

HOMEWORK-Create and write out an outline of a question and answer routine that will (a) ask at least 5 questions, (b) save the answers under 5 different variable names, and (c) return the answers in a meaningful paragraph. Remember—the answers will have to make sense in a read-back paragraph so choose the questions carefully.

CAUTION: Make sure there is enough room allocated on the screen to accomodate lengthy answers without the sentences breaking in poor places. If long answers are anticipated, the program should accept the input on a blank line.

EVALUATION:

Programs will be graded

SUB-UNIT 2

SUB-UNIT OBJECTIVES:

The students will-
understand when the INPUT statement can be used in place of the PRINT statement.
perform simple math functions with user input.

SUB-UNIT LENGTH: 2 Days

PLAN:

Review the INPUT of string variables from yesterday. Write the line INPUT AGE$ on the board again. Ask the students if a string is appropriate for this INPUT. Explain that the use of a numeric variable is more appropriate because a math function can then be performed on the input. Change the variable to a numeric variable by removing the dollar sign. The next line might be something like:

```
90 PRINT "You are ";AGE * 12;" months old, ";NAME$;"!"
```

Give the symbols for the 4 basic math functions:

+ = Addition
= Subtraction
* = Multiplication
/ = Division

Discuss the order of priority for math functions: Left to right, multiplication and division before addition and subtraction and how to change the priority with parenthesis.
Now explain that the INPUT command maybe used in place of PRINT when the program expects a user input:

```
40 PRINT "What is your name?"
50 INPUT NAME$
```

is the same as

```
40 INPUT "What is your name?";NAME$
```

PROBLEM:

Change or add to the first question and answer program to input some information on which math will be performed. Try to incorporate as many of the functions as possible. Use imagination!!

SUB-UNIT 3

SUB-UNIT OBJECTIVES:

The students will-
understand and use the IF/THEN statement to respond to user input.
understand how the IF/THEN statement returns a boolean value to control the program flow.

SUB-UNIT LENGTH: 1 Day

TERMS:

Boolean
Conditional statements

COMMANDS:

IF/THEN

PLAN:

Refer once more to the original simple question and answer routine. If we wanted to make a response to the user input that differed depending on what answer the user gave we would need to set up the conditions that would cause a certain answer. Use wording similar to code when setting up the conditions. For instance discuss the possible responses to the question "How old are you?" Let the class come up with two or three responses and write them on the board. Explain the greater than, equal and less than symbols and how they can determine a response. The program lines will look something like:

```
50 PRINT "How old are you?"
60 INPUT AGE
70 IF AGE < 17 THEN PRINT "You are just a baby."
80 IF AGE = 17 OR AGE = 18 THEN PRINT "A great age to be!!"
90 IF AGE > 18 THEN PRINT "Boy are you old!"
```

Explain how each line is evaluated: If the expression is true the rest of the line is executed, if false, the rest of the line is skipped. In line 80 make sure the use of OR is understood and point out the repetition of AGE =. If line 70 read

```
IF AGE = 17 OR 18 THEN PRINT "A great age to be!!"
```

It would cause an error.

PROBLEM:

Modify the first Question and Answer program to include conditional statements.

SUB-UNIT 4

SUB-UNIT OBJECTIVE:

The students will-
use conditionals to check user input.

SUB-UNIT LENGTH: 3 Days

TERMS:

Conditional statements
Boolean values

COMMANDS:

IF/THEN

MATERIALS:

A sample quiz program for students to use as reference for sub-units 4 and 5.

PLAN:

Review the output of the two programs already written. Inform the class that the next program will be a quiz in which the user will be asked questions and the answers will be checked to see if they are right or wrong. Make a list of the characteristics of such a program. It should include the following:

PRINT statements to ask the questions
INPUT statements to get the answers
A way to check and see if the answer is correct or incorrect
A message to acknowledge a correct answer
A message to inform users that the answer is incorrect
A way to keep track of the number of correct answers
A message at the end of the program to tell the user how many answers were correct and possibly a percentage score
A way to handle keypresses that are not valid

Pass out the sample program and give a few minutes to look it over. Begin to construct the program on the board. Start with the by now familiar initial commands and then program the computer (on the board) to pose a multiple choice question with 4 possible answers. Guide the discussion using psuedocode (language similar to the program commands): If ANSWER$ equals B then the answer is correct. How do we code this? With a conditional statement such as:

IF ANS$ = "A" THEN PRINT "That is correct!!"

The expression is evaluated by the computer and if it is TRUE the rest of the statement is executed. If it is FALSE, the remaining part of the statement is skipped.

Now lead the discussion to build a correct answer subroutine and an incorrect answer subroutine.

IF ANS$ = "A" THEN GOSUB 2000

The line to send the program to the incorrect routine will have to meet several conditions:

IF ANS$ = "B" OR ANS$ = "C" OR ANS$ = "D" THEN GOSUB 3000

CAUTION: Once again emphasize the repetition of the above statement. One of the most common problems students have is in restating the entire expression in the IF/THEN statement, that is—many will want to write it:

IF ANS$ = "B" OR "C" OR "D" THEN GOSUB 3000

Also point out the difference between using OR and AND in the above statement.
Once the correct and incorrect answer subroutines are understood, add some program lines to allow the user to continue via a page turn subroutine by pressing a key or have the program wait a moment and then continue after a delay loop.

This is sufficient for the beginning of the quiz programs.

PROBLEM:

INCLASS-pick a subject for a 5 question quiz program. Write out the questions and answers on paper and, if necessary, refer to the sample program for correct structure.

HOMEWORK-Write out the code for the quiz program. The text should be presented in a pleasing manner using previously learned text formatting commands. The answer verifications will be placed in subroutines complete with REMs. Include a page turn subroutine.

NOTE: It is an appropriate time to discuss the non-biased type of language and responses that are appropriate for computer programs.

1 or 2 days may be spent working in class coding the quiz programs.

SUB-UNIT 5

SUB-UNIT OBJECTIVES:

The students will-

use conditional statements to build error traps.
create a procedure to keep score in the quiz program.
use math to give a percentage score.

SUB-UNIT LENGTH: 2 Days

TERMS:

Error-trap
Conditionals

COMMANDS:

IF/THEN
GOTO

PLAN:

Have the list of requirements for the quiz programs on the board again. The class may refer to the sample program. Review what the quiz programs already do and what still needs to be done. The first topic ought to be the error trap routine but does not have to be. Start a discussion on how to evaluate the users response to the questions. It should not be difficult to have the students create error traps. Build the program lines on the board for an error trap. Something like:

```
4000 REM***** Error Trap *****
4010 IF ANS$ <> "A" AND ANS$ <> "B" AND ANS$ <> "C" AND ANS$ <> "D" THEN
PRINT "Press only A,B,C or D":GET ANS$:GOTO 4010
4020 RETURN
```

Emphasize the repetion in the expression, and point out that since AND is used, every condition must be met for the expression to be true. Note the colon to continue the line and show why it would not work to split up line 4010. Also mention that the use of GOTO in this situation is the only acceptable use of that command as far as their programming is concerned.

Now the programs should be made to keep score. All that is needed is to place a counter in the correct answer routine:

```
SCORE = SCORE + 1
```

The last thing the program needs is a statement telling the user what his/her score is and the percentage. The class should be able to come up with something similar to the following:

```
650 PRINT "Out of 5 possible you answered correctly ";SCORE;" times."
660 PRINT
670 PRINT "The percentage is: ";SCORE / 5 * 100;"%."
```

PROBLEM:

Modify the quiz program to incorporate the features discussed.

A few days will be allowed to finish coding the programs.

EVALUATION:

A graded show and tell will take place and the program listings will be graded.

SUB-UNIT 6.0

OBJECTIVE:

The students will-
use embedded DOS commands to create a menu to run selected programs on their disks.

SUB-UNIT LENGTH: 2 Days

COMMANDS:

PRINT:PRINT CHR$(4); "RUN xx"

PLAN:

Lead a discussion about the programs that have already been written and saved on the work disks. This is an excellent time to sum up what has been accomplished so far and how much has been learned. Point out that someone who knows nothing about programming could not use the programs on the disk if they wanted. The next task will be to create a program that will make it easy for anyone to run the programs. There are no new commands needed to do this, only new applications for ones already learned. Organize the program on the board with student help, but do not go into much detail.

PROBLEM:

Write a program that describes several programs on your disk and presents a menu that allows a user to pick which one to run.
User selections must be error trapped.
A graphic may be included if desired, but make it the last step.

SUB-UNIT 6.1

OBJECTIVE:

The students will-
create a "Hello" program that catalogs the work disk and LOADS or RUNS the program by typing in the name of the desired program.
use the ONERR command to maintain program control if a wrong name is typed incorrectly.

SUB-UNIT LENGTH: 1 Day

COMMANDS:

PRINT:PRINT CHR$(4); "CATALOG"
PRINT:PRINT CHRS(4); "LOAD xx"
PRINT:PRINT CHRS(4); "RUN xx"
ONERR GOTO xx

PLAN:

Embedded DOS commands should be quite familiar to the students by now. Discuss ways to make their disks easier to use for someone who is not such an expert programmer as they. Tell the students that the CATALOG command can be an embedded DOS command and let them come up with the correct syntax. Then discuss ways to make the "HELLO" programs on their disks more friendly using the CATALOG, LOAD and RUN commands. Introduce the ONERR command and explain its use.

PROBLEM:

Modify the "Hello" program on your disk so that it will do the following:
Give a CATALOG listing of the programs on the disk
Automatically LOAD or RUN the program name when typed in by the user.
If the user types in a wrong name, make sure the computer does not respond with an ERROR message.

TEST: Develop a unit test that contains both written and programming problems. Open note testing may be used.

UNIT 6

MATH

UNIT 6
MATH

UNIT OBJECTIVES:

The students will-

know 6 simple math symbols and their order of priority and how to change the priority with parenthesis.
use the LEN command to create a centering routine.
use embedded DOS commands to alternate two Hi-Res images on the screen.
use the RND command to create a number guessing game.
combine the above commands in a single program to title and present any sort of number guessing game.
use the INT command to round off decimal numbers.
create programs to solve geometry problems.

UNIT LENGTH: 4 Weeks

UNIT COMMANDS:

LEN
INT
RND(1)
*,/,+.-,SQR,^
()
,
FOR/NEXT
IF/THEN
GOTO

SUB-UNIT 1

SUB-UNIT OBJECTIVES:

The students will-
- understand the priority order of the four simplest math functions and how to change it with parenthesis.
- understand the print zones created by the comma.
- use the four basic math functions to create programs to solve simple geometry or math problems.

SUB-UNIT LENGTH: 2 Days

TERMS:

Math operators
Math operator priorities
Print zones

COMMANDS:

Operators:
- *,/,+,-
- ()
- ,

MATERIALS:

A list of math operations to give the students practice prioritizing the math function. This could have at least two parts:
1. Similar equations with different priority for evaluation.
2. Equations in standard notation to convert to BASIC.

(Include only enough to take 10 or 15 minutes.)

PLAN:

The four math functions in this sub-unit have been learned before, so start with a quick review of the symbols and their priorities, followed by the use of parenthesis to change the priority. Pass out the worksheet andallow adequate time to finish and then go over it together.

After the worksheet, begin constructing a bare-bones program on the board to calculate the area of a rectangle. Get suggestions from the class to make the program look appealing and be user friendly. Do not add these features to the program, the purpose is merely to remind the students of what makes quality user-friendly programs.

The last thing to talk about are the "print zones" that are available to make formatting screen output easier. By putting commas between data to be printed on the screen you can acheive limited tabbing without the tab command. The Apple computer will place data at columns 1, 16 and 32 with the following restriction: The last column, 32, will be used only if columns 24 to 31 are blank. This is fairly restrictive, but is useful for formatting 2 or 3 columns of numbers with headings. For example:

```
60 PRINT "Length","Width","Area"
70 INPUT L,W
80 PRINT L,W,L*W
```

will result in:

Length	Width	Area
10	37	137

PROBLEMS:

Write a program that uses a formula to solve a math or geometry problem. The program should not be too complex, but should use as complicated a formula as you feel comfortable with. DO NOT use formulas that will result in lengthy decimal extensions such as pi. They will be used in the unit after the next one.

EVALUATION:

Hold a show and tell and grade the programs.

SUB-UNIT 2

SUB-UNIT OBJECTIVE:

The students will-
use the LEN command to create a centering routine.

SUB-UNIT LENGTH: 1 Day

COMMAND:

LEN

PLAN:

Begin by pointing out how cumbersome centering text can be (counting all the letters and HTABing) and then show how easily it can be done using the LEN command which simply counts how many letters are in a string. With this information, lead a class discussion to create the centering routine on the board.
For example:

```
20 A$ = "This will be centered"
30 GOSUB 1000
40 A$ = "This too!!"
50 GOSUB 1000

1000 REM***** CENTERING ROUTINE *****
1010 HTAB 20 - (LEN (A$) / 2)
1020 PRINT A$
1030 PRINT  REM *** JUST TO LEAVE A SPACE BETWEEN LINES***
1040 RETURN
```

Be sure to point out that this is only for horizontal centering. Vertical spacing will still need to be done with a VTAB.

PROBLEM:

Use the centering routine to make a title page for the math program finished yesterday.

SUB-UNIT 3.0

OBJECTIVES:

The students will-

use the INT command to round numbers to whole numbers or any decimal place desired.
know how to express a number in Scientific notation.
be aware of the more complicated math functions resident in BASIC and their priority.

SUB-UNIT LENGTH: 3 Days

TERMS:

Rounding
Exponent
Scientific Notation

COMMANDS:

Functions:
INT
SIN
COS
TAN
SGN-returns -1, 0 or +1 according to the sign of a number
ABS-returns the absolute value of a number
SQR-returns the positive square root of a number
EXP-raises a number to the indicated power to 6 decimal places
LOG-returns the natural logarithm of the number

Operator:
^ raises a number to the indicated power: 5^3 returns 125

MATERIALS:

Handout with a list of operator priorities:

()
+ - NOT unary operators
^
* /
+ -
> < >= =< => =< <> >< =
AND
OR

PLAN:

Begin by making comments about the short programs just completed and then start a discussion about a program containing a formula that might return an answer with many decimal places. An obvious choice a program to calculate the area of a circle (Area=pi*R^2). The point of the lesson is to create a template for rounding numbers to reasonable limits.
INT will return a whole number but it always rounds down to the nearest whole number:
INT(5.67) returns 5
INT(-5.67) returns -6
If this is confusing to some, describe it in terms of a number line and the next lower number is to the LEFT. To overcome this problem merely add .5 and the correctly rounded answer will be given.

To round to the nearest 10th use the following:
PRINT INT(2.68/.1 + .5) * .1 Explain this expression step by step to make sure everyone understands why 2.68 had to be divided by .1 and then multiplied by the same after adding .5. Now have the class figure out how to round off to the hundredths place, thousandths and ten thousandths places. Then go the other way and round to the 10's, 100's and 1000's places. Examples:

Round 123.345 to hundredths:
PRINT INT(123.345/.01 + .5) * .01 returns 123.35

Round 9825 to the nearest hundred:
PRINT INT(9825/100 + .5) * 100 returns 9800

The two problems that follow may be replaced with similar problems the students devise. (NOTE: they are not yet ready for repeating problems requiring loops for continuous input and one final answer.)

PROBLEM:

Write a discounting program to figure out how much an item will cost after a user specified discount is applied. Round the answer to cents.

PROBLEM:

Write a program that will add a user specified amount of tax to an item to be purchased. Round the answer to cents.

SUB-UNIT 3.1

SUB-UNIT OBJECTIVE:

The students will-
use a conditional statement to create a repeating loop within a program solving math problems.

SUB-UNIT LENGTH: 1 Day

TERMS:

Conditional statement
Flag
Controlled loop
Endless loop

PLAN:

Write a very short program on the board that results in an endless loop such as

```
10 PRINT "HOWDY"
20 GOTO 10
```

and discuss why this causes a loss of control over the program. Now put a slightly more complicated program on the board such as:

```
10 HOME
20 PRINT "Enter a number"
30 INPUT X
40 PRINT "Enter a second number"
50 INPUT Y
60 PRINT "X multiplied by Y =";X * Y
70 GOTO 20
```

Besides being an incorrect use of the GOTO command, the program is locked into an endless loop. Now introduce the idea of a flag to end the loop. For instance, if an unlikely number such as
999 was entered in response to line 20, it could be a flag to end the loop. Add the following line to the program:

```
35 IF X = -999 THEN END
```

or

35 IF X = -999 THEN GOTO 80

80 END

This should be sufficient to give the students a clear idea of how they can keep a program repeating until the USER decides to end it.

PROBLEM:

Alter the discount program so it loops until the user decides to end it. Make sure the program lets the user know what the flag is (the user is not a mind reader!!).

PROBLEM:

Write a program that will perform at least three different algebra or geometry problems using formulas that will produce long decimal answers. The programs should have the following features:

A menu to select what type of problem to solve.
At least two different rounding routines.
The option to do another problem or quit. (This is only slightly different than a program that automatically repeats until a flag is encountered.)

BONUS: Include a menu for the user to select how many decimal places to include in the answer.

EVALUATION:

Grade the programs and the program listings. A show and tell is optional.

SUB-UNIT 4

SUB-UNIT OBJECTIVE:

The students will-
use the RANDOM command to create a number guessing routine.

SUB-UNIT LENGTH: 5 Days

MATERIALS:

Several sample programs to be debugged, and several to (a) follow program flow, (b) predict output, and (c) use as templates.

COMMAND:

RND(1)

PLAN:

Begin by describing what the unit programs will be: Number guessing games that can

take many different forms from a simple heads and tails game to Black Jack. Student inventions are encouraged.

Explain the random command and how to make it produce the desired range of numbers. RND(1) returns a number larger than or equal to 0 and less than 1. The (1) is part of the command and does not reflect a range for the randomizing, that is, RND(5) would return the same thing as RND(1). To change the range of the random to produce a number from 0 to 9 use the following expression:

PRINT INT(RND(1) * 10)
If you desire a range of 1 to 10, add 1 to the above expression:

PRINT INT(RND(1) * 10 + 1)
(Note the use of parenthesis around the (RND(1) * 10) expression).

Ask the class to come up with the correct expression for different ranges.
Begin a simple 'Heads and Tails' program on the board with the classes help. It should look something like the following:

```
30 PRINT "Press any key to toss the coin"
40 INPUT KEY$
50 COUNTER = COUNTER + 1
60 X=INT(RND(1) * 2)
70 IF X = 0 THEN GOSUB (ANSWER IS HEADS SUBROUTINE)
80 IF X = 1 THEN GOSUB (ANSWER IS TAILS SUBROUTINE)
90 PRINT "Press any key to toss again or 'Q' to quit."
100 GET K$
110 IF K$ <> "Q" THEN GOTO 60
120 PRINT "Out of ";COUNTER;" coin tosses you"
130 PRINT "guessed right ";SC;" times for a "
140 PRINT "percentage of ";(SC * 100) / COUNTER
```

Right answer subroutine should contain the following:
The score counter
A message to inform the user they have guessed right
The option to continue or quit
Game ending comments and score with performance assessment such as: You were correct 68 % of the time and should be quite successful if you were to go to Las Vegas.

PROBLEM:

Create a game that generates a random numberfor the user to guess. The program will guide the user to the correct answer by telling them they are too high or too low and then waiting for a new guess until they get is right or enter a flag to end the program.

When the correct answer is guessed, tell the user how many guesses it took.

OPTIONAL: At the end of the program, give statistics on the users guessing ability and PRINT the percentages and a conditional statement assessing the user's gambling luck.

OPTIONAL: Include graphics. This would be easy as the actual text space needed is very small and can be contained in the text window of the graphics mode.

OPTIONAL: Instead of the above problem, program a more difficult game that you make up or already know such as BLACK JACK.

EVALUAION:
Grade the programs and the program listings.

SUB-UNIT 5

SUB-UNIT OBJECTIVE:

The students will-
use conditionals to create a loop that gets INPUT for a continuing (multiple input) math operations such as averaging.

SUB-UNIT LENGTH: 3 Days

PLAN:

This lesson does not introduce anything new, but combines conditional looping and math operations used in earlier programs. With this in mind, try to have students supply all the necessary steps and answers. Let the class know that they will be using previously learned skills to build more complex programs. Lead a discussion to build an averaging program on the board. It should contain some of the following features:

(leave some room to fill in lines 20-40 later as initializing has not been encountered yet)

```
20 C = 0
30 T = 0
40 N = 0

120 PRINT "Enter number ";C + 1
130 INPUT N
140 C = C + 1
150 T = T + N
160 IF N <> -999 THEN GOTO 120
170 PRINT "The average of ";C;" numbers is:"
180 T = T + 999
190 LET A = T/C
200 A = INT(A/.1 + .5) * .1
210 PRINT A
220 END
```

Explain initializing and go back and insert lines 20-40.

PROBLEM:

Write a program that is or might be useful to you that will calculate some sort of continuous math problem. Suggestions:

Checkbook or savings program.
Game statistics program such as baseball averages.
Budget program.

Programs must have the following features
Error control
User friendliness
Pleasing screen output format
Flag to signal the end of the input

EVALUATION:

Grade the programs and listings.

SUB-UNIT 6

OBJECTIVE:

The students will-
use FOR/NEXT loops to make bar graphs by using a variable in the loop to print a character on the screen.

SUB-UNIT LENGTH: 3-5 Days

COMMANDS:

FOR/NEXT

MATERIALS:

A handout with several short templates which also shows the results of using FOR/NEXT loops to PRINT repeating patterns or lines. Include one or two using the STEP - command.

A handout with programs for predicting output, and faulty programs to be debugged, by the students individually.

PLAN:

Tell the class that they will be creating bar-graphs with the FOR/NEXT command but before starting the actual graphing program some time will be spent learning a new use for the familiar FOR/NEXT command pair. Utlize class knowledge of the FOR/NEXT loop counting characteristics to create a program on the board that repeats a character or word several times:

```
20 FOR X = 1 TO 5
30 PRINT X
40 NEXT X
```

Now substitute a word for X:

```
20 FOR X = 1 TO 5
30 PRINT "Hello!!!"
40 NEXT X
```

The next step is to add a variable to allow the user to designate the limit of the FOR/NEXT. The students should be able to supply the needed commands to input a desired limit and execute a FOR/NEXT loop.
If you feel it is necessary, create a bare bones program to draw bar graphs for the students and let them fill in the blanks.

PROBLEM:

Create a program to draw bar graphs with the following characteristics:
Asks the user for a graph name.
The name is to be centered and printed.
Asks the user for a label for each bar and prints it to the screen.
Asks the user for data to create a bar of the correct length.
Automatically asks for a subsequent label and graph data until the user elects to quit.

BONUS: Instead of using Asterisks or Xs, think of a way to make a bar line look like a solid bar. (Print INVERSE blank spaces.)

EVALUATION:

Grade the programs and program listings.

SUB-UNIT 7

SUB-UNIT OBJECTIVES:

The students will-
- use FOR/NEXT loops to create a program that calculates and prints math tables such as multiplication, or conversion charts.
- understand the use of a constant to make changing a program easier.

SUB-UNIT LENGTH: 5 Days

TERMS:

Constants

COMMANDS:

LET
FOR/NEXT

MATERIALS:

A sample template to create an addition chart.

PLAN:

Make sure the class understands that they are continuing to build on the loop programs already written to thoroughly understand the looping process and create progressively more difficult programs. The next project is to use the FOR/NEXT command to make math tables that are useful to the students. A simple one will be done in class to clarify the concepts. Involve the class in a discussion to write a short addition program on the board. It will include a range of numbers to add and some formatting commands to make the output acceptable:

```
30 LET J = 1
40 FOR X = 1 TO 10
50 PRINT J;" + ";X;" = ";J + X
60 NEXT X
```

It should be clear that to change the program to output the "twos", only the value of J needs to be changed.

PROBLEMS:

Choose one of the following or have your own idea approved before starting to program. The problem you select should be interesting to you!!

Write a program that calculates and prints:
Math tables of any kind except addition or subtraction
Conversion from Metric values to SAE
Conversion from Kilometers to MPH (you may want to use the STEP command to compact the output)
Conversion from Celcius to Fahrenheit or Vice Versa

All programs must have the following features:
Pleasing output format
Easy to change constants if appropriate

EVALUATION:

Programs, Show and Tell (Optional) and listings will be graded.

TEST: Develop a unit test. It may be appropriate to make it strictly a programming test that lasts two days. It will not constitute "cheating" if the student has a chance to work at home on some of the problems they are experiencing and then finish their programs the next day.

UNIT 7

TEXT FILES

UNIT 7
TEXT FILES

UNIT OBJECTIVES:

The students will-
save text to a text file on disk
read text from a text file on disk

UNIT LENGTH: 2-2.5 Weeks

UNIT COMMANDS:

PRINT:PRINT CHR$(4); "OPEN xxxx"
PRINT:PRINT CHR$(4); "WRITE xxxx"
PRINT:PRINT CHR$(4); "CLOSE xxxx"
PRINT:PRINT CHR$(4); "READ xxxx"
PRINT
INPUT

SUB-UNIT 1

OBJECTIVES:

The students will-
create a Question and Answer program that saves the answers in a text file.

Optional-Students will-
use a variable to make the text file name user specified.

SUB-UNIT LENGTH: 2 Days

TERMS:

Data Base
Text files
Opening text files
Closing text files

COMMANDS:

PRINT:PRINT CHR$(4); "OPEN xxxx"
PRINT:PRINT CHR$(4); "WRITE xxxx"
PRINT:PRINT CHR$(4); "CLOSE xxxx"
PRINT

MATERIALS:

Sample program for student reference

PLAN:

A logical way to begin this unit is to review the quiz programs written before, particularly the personal questions program. Talk about the usefulness of saving information that is given by a user on disk, so it can be used later for some purpose or reviewed by someone else. Explain what a DATA BASE is and that the next project will be to construct a very small data base of their own: When information is entered by the user, it is usually in the form of text which can be saved in text files. All that is needed is a name for the file, which can be user supplied, and the commands to OPEN the file, PRINT (write) the information to the file, and to CLOSE the file. To get the information back, the same OPEN and CLOSE command are used, and the INPUT command reads the text from the file.

Begin a short example on the board with the students help which will include something similar to the following:

```
50 PRINT "What is your name?"
60 INPUT NAME$
70 PRINT "What kind of car would you like to own?"
80 INPUT CAR$
90 REM ***** SAVE INFORMATION IN TEXT FILE *****
100 PRINT:PRINT CHR$(4); "OPEN INFO"
110 PRINT:PRINT CHR$(4); "WRITE INFO"
120 PRINT NAME$
130 PRINT CAR$
140 PRINT:PRINT CHR$(4); "CLOSE INFO"
```

Make sure you explain that the information will be saved in the order that it is PRINTed to the disk, and that it will be retrieved in the same order.

PROBLEM:

Write a program that asks a minimum of 5 questions and writes the answers to a text file.

BONUS-make the text file name user supplied.

EVALUATION:

The program will not be evaluated until the next sub-unit is completed.

SUB-UNIT 2

OBJECTIVE:
The students will-

create a program to read text information from a text file and present it in a meaningful manner.

SUB-UNIT LENGTH: 10 Days

COMMANDS:

PRINT:PRINT CHR$(4); "OPEN xxxx"
PRINT:PRINT CHR$(4); "READ xxxx"
PRINT:PRINT CHR$(4); "CLOSE xxxx"
INPUT

PLAN:

Review what has been done so far on the text files. The information is now stored on the disk and a program must be written to retrieve and present the stored text. Since this program is extremely similar to the first, outline a very simple program on the board. The only differences are:
PRINT:PRINT CHR$(4); "WRITE xxxx" is replace by
PRINT:PRINT CHR$(4); "READ xxxx" and the PRINT statement is replaced by INPUT.

CAUTION: Make sure the students understand that the information was NOT stored with it's variable name, but rather just the text associated with the variable name is saved. Because of this, the variable names following the INPUT statement must be in the same order as when they were saved or the wrong information will be read back into the variables.

PROBLEM:

Write a program that retrieves the information and presents it in a meaningful and pleasing manner.

EVALUATION:

The programs and the listing will be evaluated. A show and tell is optional because of the time required.

PROBLEM:

Write a second DATA BASE program that uses a different text format to ask a minimum of 7 different questions that are not similar to the first program. It may be a quiz.
Include a menu to:
A. Run the question and answer segment.
B. Run the information retrieval segment.
C. Quit

Don't forget error trapping.
Include secondary menus to:
A. Name the file in which the information will be stored.
B. Catalog the files on the disk to choose which text file to read. Don't forget error control.

OPTIONAL TEST: You may wish to make sure these concepts have been learned well as they are essential to the next unit on arrays.

UNIT 8

1 AND 2 DIMENSIONAL ARRAY S

UNIT 8
2 DIMENSIONAL ARRAYS

UNIT OBJECTIVES:

The students will-
understand nested loops.
understand and use 1 and 2 dimensional arrays.

UNIT LENGTH: 2-3 Weeks

TERMS:

Arrays
Changing-value variables
1 and 2 dimensional arrays

UNIT COMMANDS:

DIM xxxx(x)
DIM xxxx(x,x)

SUB-UNIT 1

OBJECTIVES:

The students will-
use a loop to INPUT data.
use a 1 dimensional array to store and read data.

SUB-UNIT LENGTH: 3 Days

TERMS:

Array
Dimension
Conditional statements

COMMANDS:

DIM

PLAN:

To begin arrays, rely on the students' understanding of variables, and the question/routine programs already written. It should not be difficult to make the transition into single and two dimensional arrays.

Guide the class discussion in putting a simple question and answer routine of three questions on the board. Now discuss the problem presented by having many questions: The program would become very long and repetitious. To make a program more efficient, the information can be stored under one variable name. Explain the structure of the array and how data is stored in it by placing 'input' from the program on the board in a straight line separated by commas. Label this 'array' INFO and assign position numbers to the three bits of data:

INFO(Jack,blue,vanilla)
The next step is to create a simple loop to get the above information:

```
10 DIM INFO$(2)
20 FOR X = 1 TO 3
30 INPUT INFO$(X)
40 NEXT X
```

Do lines 20-40 first and then add line 10 and explain that the computer starts counting from 0 making the dimension seem one number smaller than the counting loop.
The next step is to add conditional statements to provide prompts for the user such as.

```
22 IF X = 1 THEN PRINT "Enter your name: "
24 IF X = 2 THEN PRINT "Enter your favorite color: "
26 IF X = 3 THEN PRINT "Enter your favorite flavor: "
```

The program lines necessary to read the information back to the screen is exactly like the first part except line 30 would read:

```
30 PRINT INFO$(X)
```

And the prompts would be different—now they would be explaining or presenting the responses.

This approach may not seem to address the usefulness of arrays for programming efficiency, but it will clarify the concept of the array without distracting the student with the nested loops required by two-dimensional arrays. Nested loops follow this unit.

PROBLEM:

Write a program that asks the user several questions and stores the responses in an array, then uses a loop to print the responses on the screen. Do not be overly concerned with text formatting—this assignment is to give you practice with arrays and counting loops.

EVALUATION:

No evaluation is necessary except for trouble shooting particular problems.

SUB-UNIT 2

SUB-UNIT OBJECTIVE:

The students will-
understand nested loops for outputting characters to the screen.

SUB-UNIT LENGTH: 1 Day

TERMS:

Nested loops

COMMANDS:

FOR/NEXT

PLAN:

This lesson will not involve reading data but will concentrate simply on making the concept of Nested loops clear by printing characters on the screen through nested loops. Begin by putting a simple loop on the board such as:

```
50 FOR X = 1 TO 5
60 PRINT X
70 NEXT X
```

and be sure everyone understands what will happen. Now place a second loop around the first one:

```
40 FOR Y = 1 TO 5
50 FOR X = 1 TO 5
60 PRINT X
70 NEXT X
80 NEXT Y
```

and discuss the output. Have the students enter the program above and RUN it. Add the following line to give the output further clarity:

```
45 PRINT Y
```

The problem that follows should be done in class:

PROBLEM:

Write a program that outputs the following:

Black
White
White
White
Black
White
White
White
Black
White
White
White

SUB-UNIT 3

SUB-UNIT OBJECTIVE:

The students will-
begin a Data Base program that uses nested loops to INPUT information from a question and answer program that places the information in a two-dimensional array.

SUB-UNIT LENGTH: 2 Days

PLAN:

Review yesterday's work with two dimensional arrays and relate it to the upcoming project which will be a Data Base to ask several people the same set of questions and save the information on disk. Lead a discussion to build a program on the board that asks 3 people 3 questions and places them in a two dimensional array. The program should look similar to the following:

```
10 HOME
20 DIM INFO$(2,2)
30 FOR I = 1 TO 3
40 FOR J = 1 TO 3
50 IF J = 1 THEN PRINT "..."
60 IF J = 2 THEN PRINT "..."
70 IF J = 3 THEN PRINT "..."
80 INPUT INFO$(I,J)
90 NEXT J
100 NEXT I
```

Go through the flow of the program and discuss what happens each time through the loop.

Reiterate the reasons for the array's dimensions appearing to be one less than the FOR/NEXT LOOPS.

PROBLEM:

Write a program that asks 5 people at least 3 questions each and stores the responses in a two-dimensional array.

EVALUATION:

It is not necessary to evaluate this program until it is completed by saving the data to disk, reading it back and presenting it.

SUB-UNIT 4

SUB-UNIT OBJECTIVE:

The students will-

use nested FOR/NEXT loops to present the information stored in a two dimensional array on the screen.

SUB-UNIT LENGTH: 2 Days

PLAN:

The coding needed to present the information in an array is almost identicle to the the program just completed. Lead a discussion to construct a nested loop template that the students can add to their program to present the information.

PROBLEM:

Add to the question program so that it presents the gathered information to the screen in an acceptable text format.

SUB-UNIT 5

OBJECTIVE:

The students will-

use embedded DOS commands and nested FOR/NEXT loops to write data in a two-dimensional array to disk.

SUB-UNIT LENGTH: 1 Day

COMMANDS:

```
PRINT CHR$(4);"OPEN";xxxx$
PRINT CHR$(4);"WRITE";xxxx$
PRINT CHR$(4);"CLOSE";xxxx$
PRINT CHR$(4);"DELETE";xxxx$
```

PLAN:

The template for saving information to the disk is very similar to the template for INPUTting information. Lead a class discussion to put the necessary lines of code on the board. It should look something like:

```
2000REM **** SAVE ROUTINE *****
2010 PRINT CHR$(4); "DELETE";xxxx$
2020 PRINT CHR$(4); "OPEN";xxxx$
2030 PRINT CHR$(4); "WRITE";xxxx$
2040 FOR I = 1 TO 3
2050 FOR J = 1 TO 3
2060 PRINT xxxx$(I,J)
2070 NEXT J
2080 NEXT I
2090 PRINT CHR$(4);"CLOSE";xxxx$
2100 RETURN
```

The DELETE command makes sure that there is no information left in the file from a previous use that may cause some problems when saving new information.

PROBLEM:

Write a new Question/Answer program that saves the information in a user named text file on disk. Include a menu that gives the following options:

Enter data
Save data to disk
Retrieve data from disk (this will be added next)
Print the data to the screen
Quit the program

EVALUATION:

The programs will be evaluated after the next sub-unit when they are completed (by adding a routine to read the information from the disk).

SUB-UNIT 6

OBJECTIVE:

The students will-

add a sub-routine to read data saved in a two dimensional array on a disk.
use the MON C,I,O and NOMON C,I,O commands to monitor the interaction between RAM and the disk drive to help locate programming errors.

SUB-UNIT LENGTH: 2 Days

COMMANDS:

```
PRINT CHR$(4); "OPEN";xxxx$
PRINT CHR$(4); "READ";xxxx$
INPUTxxxx$(I,J)
PRINT CHR$(4);"CLOSE";xxxx$
MON C,I,O
NOMON C,I,O
```

PLAN:

The last step, reading the data from the disk is almost identical to the saving routine so the discussion can be brief. With input from the class, build a sub-routine that looks similar to the following:

```
3000 REM ***** GET FILE ROUTINE *****
3020 PRINT CHR$(4); "OPEN";xxxx$
3030 PRINT CHR$(4); "READ";xxxx$
3040 FOR I = 1 TO 3
3050 FOR J = 1 TO 3
3060 INPUT xxxx$(I,J)
3070 NEXT J
3080 NEXT I
3090 PRINT CHR$(4);"CLOSE";xxxx$
3100 RETURN
```

Before ending this unit discuss the MON C,I,O and NOMON C,I,O commands for monitoring the interaction between RAM and the disk drive to locate problems causing system errors. This command is a toggle that is entered without line numbers before the RUN command and turned off after the program is run by entering NOMON C,I,O.

PROBLEM:

Add the routine for retrieving data from disk to the the Data Base program. Do not forget error control.

BONUS: Make the size of the Data Base user defined.

UNIT 9

PERSONAL PROGRAMS

UNIT 9
PERSONAL PROGRAMS

UNIT OBJECTIVE:

The students will-

use their knowledge to create applications programs that are of interest or use to them and instructor approved.

PLAN:

The final unit is intended to last the remaining time in the school year. There are no specific objectives, no new commands and no specific problems to solve. Hopefully, the students have mastered the concepts presented and used throughout the year. The remaining time is a chance for them to develop any type of program or programs they wish. The final software projects are to be planned carefully, fully outlined, and approved by the instructor before coding begins. The instructor's role is to be a resource and guide for the students.