



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

Production and Evaluation of a Self-Instructional Package
to Introduce Computer Programming
Through a Discussion of Problem Solving
and Logo Turtle Graphics

Patricia Chwartkowski

A Thesis-Equivalent
in
The Department
of
Education

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Arts at
Concordia University
Montréal, Québec, Canada

November 1985



Patricia Chwartkowski, 1985

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-30643-2

ABSTRACT

Production and Evaluation of a Self-Instructional Package to Introduce Computer Programming Through a Discussion of Problem Solving and Logo Turtle Graphics

Patricia Chwartkowski

This thesis-equivalent presents the formative evaluation of a self-instructional package produced for an undergraduate computer literacy course at Concordia University. The author was asked to develop materials that would accomplish three broad objectives:

1. introduce computer programming through a discussion of problem solving,
2. familiarize the learner with the programming language Logo, and
3. provide a direct hands-on experience at the computer.

Dick and Carey's (1978) procedure for the systematic design of instructional materials was used to design and evaluate the resulting materials: two self-instructional booklets, Unit 1--Problem Solving, and Unit 2--Programming with Logo.

Specific evaluation questions were developed to verify the assumptions underlying the design of the materials. The materials were field-tested during the first offering of the course in January 1985. Evaluation data was collected from

forty-six students randomly selected from those registered for the course. The data collected came from a variety of sources: pre- and posttest scores, a final project, student questionnaires, interviews, and informal observation.

An in-depth analysis of this data revealed the areas of strength and weakness in the instruction. This thesis-equivalent concludes with a discussion of the recommended revisions.

I would like to thank the following people
for their help:

Mariela Tovar

Terrance Hughes

Jean-Claude Deguise

Isabelle Deguise

Philippe Deguise

TABLE OF CONTENTS

	Page
CHAPTER 1: Introduction	1
CHAPTER 2: Literature Review	4
Problem Solving	4
Problem Solving and Computer Programming	5
Problem Solving and Logo	6
Formative Evaluation	7
CHAPTER 3: The Instructional Design	12
Educational Objectives	12
Instructional Analysis	13
Target Audience	14
Rationale for Media Selection	15
Outline of the Content	16
Instructional Strategy	17
CHAPTER 4: Method	19
Preliminary Evaluation	19
Field Evaluation	20
Evaluation Questions	20
Instrumentation	21
Pretest and posttest	21
Final project	23
Personal information form	24
An observation sheet	24
Evaluation questionnaires	24
Informal interviews	25
Evaluation Procedures	25
The Sample and Sampling Procedures	26

	Page
CHAPTER 5: Results	27
Preliminary Evaluation	27
Field Evaluation	29
The General Picture: Overall Learning	
Gains	29
The Central Focus: Objective-by-Objective	
Analysis	30
Subordinate objectives 1 to 9	30
Terminal objectives 10 to 13	33
Analysis of the Qualitative Data	33
The Target Population	35
CHAPTER 6: Discussion	36
Evaluation Question 1: Were There Statistically	
Significant Learning Gains on Each Objective?...	36
Unit 1--Problem Solving	36
Unit 2--Programming with Logo	37
Evaluation Question 2: Which Objectives did NOT	
Produce the Desired Results and Why?	37
Subordinate objectives 1 to 7	37
Objective 2: An example of the revision	
decision process	40
Terminal objectives 10 to 13	42
Evaluation Question 3: How did the Learners React	
to the Material?	43
Unit 1--Problem Solving	43
Unit 2--Programming with Logo	44

	Page
Evaluation Question 4: Were the Materials Self-Instructional?	46
Unit 1--Problem Solving	46
Unit 2--Programming with Logo	46
Evaluation Question 5: Were the Assumptions About the Target Population Valid?	47
Evaluation Question 6: How Long did it Take the Learners to Complete the Lesson?	47
CHAPTER 7: Conclusions and Recommendations	49
Recommendations	50
Unit 1--Problem Solving	50
Unit 2--Programming with Logo	51
The Influence of Learning Style	52
Suggestions for Research	53
The formative evaluation process	53
The revision decision process	53
Unit 1 and Unit 2	54
REFERENCES	55

LIST OF APPENDICES

	Page
Appendix A: Instructional Analysis	59
Appendix B: List of Subskills, Behavioral Objectives and Test Items	61
Appendix C: Table of Contents for Unit 1--Problem Solving	67
Appendix D: Table of Contents for Unit 2--Programming with Logo	68
Appendix E: Pretest	69
Appendix F: Posttest	74
Appendix G: Personal Information Form	79
Appendix H: Observation Sheet	80
Appendix I: Evaluation Questionnaire: Unit 1-- Problem Solving	81
Appendix J: Evaluation Questionnaire: Unit 2-- Programming with Logo	85
Appendix K: Item Response Patterns for Those Subordinate Objectives NOT Performing as Expected	91
Appendix L: Unit 1--Problem Solving	92
Appendix M: Revised Edition of Unit 2--Programming with Logo	126

LIST OF TABLES AND FIGURES

	Page
Table 1: Test Items and Mastery Criterion for Each Subordinate Objective 1 to 9	23
Table 2: Descriptive Statistics on Pre- and Posttest Scores	29
Table 3: Percentage of Students Showing Four Different Patterns of Pre-Posttest Results on Subordinate Objectives	32
Table 4: Analysis of the Terminal Objectives (10-13) Assessed by the Final Project	34

	Page
Figure 1: Percentage of students mastering subordinate objectives 1 to 9 on the pretest and posttest	31

CHAPTER 1

Introduction

In the fall of 1984 Concordia University decided to begin developing materials for a general computer literacy course available to undergraduate students in the Arts and Science faculty. The aim of the course was to provide a basic knowledge of computers and their usage to students who might not otherwise have the opportunity or prerequisites for a mainstream computer course.

The general course content and structure had already been determined by a subcommittee of the Arts and Science faculty. This subcommittee had met with faculty members to discuss what basic computer skills each considered necessary or desirable in his or her area of interest. Curriculums for other college-level computer literacy courses had also been reviewed in order to determine course content.

The course was to be self-instructional with as much hands-on experience as possible. To this end a computer lab was set up with fifteen IBM PC's and three video terminals. An IBM AT was available for the eventual implementation of a networking system and a computer-based testing system.

In October 1984 the author was approached by the project coordinator to develop materials that would accomplish three main objectives:

1. introduce the learner to computer programming through a discussion of problem solving,

2. familiarize the learner with the programming language Logo, and

3. provide a direct hands-on experience at the computer.

The following restrictions were imposed: a) the materials were to be self-instructional; b) only print, computers, or video could be used; c) instruction time was limited to four hours; d) no money was available for a computer programmer; and e) the production deadline was December 1984.

The author was particularly interested in this work as materials evaluation was an important component of the project. Consequently, the author produced two self-instructional booklets: Unit 1 deals with problem-solving strategies and techniques useful for programming in any language; Unit 2 is an application of this information to programming using Logo turtle graphics.

The first offering of the course was Winter 1985. A total of 100 students took the course over the 13 week period. Each student chose one 4-hour lab session per week. Three sessions were offered per day: morning, afternoon, and evening. A teaching assistant was on hand to answer any questions. All the materials were distributed from one central location by the personnel of the language lab.

The materials produced by the author constituted the third lesson within the overall course structure:

Lesson 1: Introduction to the hardware and the Disk Operating System (DOS); how the machine

works

Lesson 2: Applications software: spreadsheets and wordprocessing

Lesson 3: Introduction to computer programming through a discussion of problem solving and Logo turtle graphics

Lesson 4: Logo list-handling

Lesson 5: Programming in BASIC; computer graphics

Lesson 6: Ethics and social issues

After each lesson, the students took a quiz. The project coordinator considered a lesson complete once the quiz was taken and any required projects handed in. The final course grade was based on the number of lessons completed and on the results of a mid-term and final exam.

The author carried out a preliminary evaluation of the materials she produced for this course in December 1984. The field test was conducted during the first offering of the course, January to April 1985. This thesis-equivalent details the procedures and results of this formative evaluation process.

CHAPTER 2

Literature Review

Problem Solving

The content of the unit on problem solving is based on a review of current literature dealing with problem solving and with problem solving and computer literacy.

The nature of general problem solving is still under investigation; nevertheless, from this research a set of problem-solving heuristics has emerged. Schoenfeld (1979) gives the following definition of a heuristic:

A heuristic is a general suggestion or strategy, independent of subject matter, that helps problem solvers approach, understand, and /or efficiently marshal their resources in solving problems. (p. 315)

Polya (1957, 1962) was the first to discuss problem-solving heuristics. It is those mentioned in his works that are most prevalent in the literature.

It should be noted that the academic community is divided over the issue of whether general problem solving can be taught independently of specific subject matter (Lochhead & Clement, 1979; Tuma & Reif, 1980). Since the author's mandate was to discuss general problem-solving strategies and techniques, this debate was not explored further. Rather a closer look was taken at those advocating the teaching of general problem-solving skills (Polya, 1957, 1962; Rubinstein, 1975; Schoenfeld, 1979; Whimbey & Loch-

head, 1981).

In addition to an in-depth discussion of problem-solving strategies and techniques, these authors also propose a methodology for teaching general problem solving. They all emphasize the use of a systematic (or step-by-step) procedure. Each author suggests steps to follow in the solution of a problem.

Research by Lochhead and Whimbey (Lochhead, 1981; Lochhead & Clement, 1979; Whimbey, 1980; Whimbey & Lochhead, 1981) has revealed that the critical attributes of good problem solvers are carefulness and the use of a step-by-step approach. They also found that the most effective strategy for teaching problem solving stressed active participation by the students in their problem-solving processes and included the teaching of a specific problem-solving strategy to use. These elements were, therefore, integrated into the instructional strategy of the unit on problem solving.

Problem Solving and Computer Programming

Which of the problem-solving heuristics outlined by the above authors would be most appropriate to computer programming in the context of a computer literacy course? To answer this question, several computer literacy and programming textbooks were reviewed. A few approached programming from a problem-solving perspective (Gledhill, 1981; Heller & Martin, 1982; Horn & Poirot, 1981; Koffman & Friedman, 1979). These authors describe programming as a problem-

solving activity and deal at length with general problem-solving heuristics useful for programming.

Again, all of them suggest a systematic strategy to use when attacking a programming task. The steps vary in number from author to author but the following four were always stressed:

1. understand the problem
2. relate and organize the data
3. find a solution
4. check your results

It is this condensed version that is presented in the unit on problem solving.

Problem Solving and Logo

The choice of which heuristics to present was also influenced by the necessity of introducing the programming language Logo. Several studies have been done to see if learning Logo improves general problem-solving skills (Krasnor & Mitterer, 1983; Pea, 1983; Rampy, 1984; Statz, 1973). However, these studies were not relevant to the materials produced as the objective of learning Logo in Unit 2 is not to improve the learner's problem-solving skills, but rather to demonstrate the application of certain problem-solving heuristics to a programming task.

Certain aspects of Logo make it a good vehicle for discussing problem-solving heuristics (referred throughout Unit 1 and Unit 2 as "techniques"). Papert (1980) points this out: "...Turtle geometry lends itself so well to

Polya's principles that the best way to explain Polya to students is to let them learn Turtle geometry" (p. 64).

Procedures and subprocedures are a concrete representation of the problem-solving technique: break a problem down into smaller more manageable parts. This is referred to in the materials as "stepwise refinement". The interactive mode allows the learner to continually practice the technique: check your results or, in computer jargon, debug your program.

Because of the imposed constraint of an instruction time of four hours, it was decided to discuss in detail only these two problem-solving heuristics as well as the use of a systematic plan. It was further decided to limit instruction only to Logo turtle graphics.

Papert (1980) advocates that Logo should be learned in a free exploratory environment with little direct teacher intervention. Some studies have been done that suggest this is not an optimum strategy (Pea, 1983). Moreover, due to the time constraints of the course and to the overall course objectives, such an approach was deemed unfeasible. A more directed approach to learning Logo was adopted. Exploration and discovery are encouraged but only within the limited scope of a final project.

Formative Evaluation

The evaluation of the instructional materials discussed in this thesis-equivalent is formative in nature. The definition of formative evaluation is that of Dick (1977a):

Formative evaluation may be ... defined as a process of systematically trying out instructional materials with learners in order to gather information and data which will be used to revise the materials. (p. 311)

The purpose of the formative evaluation process is to use the collected data to revise and, thus, improve the instructional materials. This is in contrast to summative evaluation which aims to decide the instructional effectiveness of a finished product:

Summative evaluation is conducted after a product or instructional system has been completed... (Dick, 1977b, p. 338).

The formative evaluation procedures followed by the author were those outlined by Dick (1977a) and Dick and Carey (1978). In these works, the three phases of the formative evaluation process are described in detail.

In the one-to-one stage, the designer goes through a rough draft of the materials with two to three members of the target audience. Ideally these learners should represent the range of abilities found in the population. The objective of this phase is to discover major typographical errors, difficulties in content presentation, and any confusion arising from the wording of instructions, test items, or examples. The designer works through the materials with the learner encouraging comments and, if a difficulty arises, attempts a clarification. The materials are also given to a content expert to determine their accuracy. Subsequent to this one-on-one evaluation and consultation with the content expert changes are made and the product is

improved for the next stage of the formative evaluation process--small group.

In the small group phase, the designer works with ten to twenty members of the target population in an environment that resembles as closely as possible the actual context and manner in which the materials will be used. The objective of this stage is to reveal any difficulties not only in the materials but also in their implementation and use. Revisions are made as a result. If the changes are major, it may be necessary to begin the formative evaluation process anew. If not, then one moves to the final stage--the field evaluation.

In this last stage, the materials are used in a real-life situation with a minimum sample of thirty learners representative of the target population. The designer does not participate in the administration of the materials, but rather watches over the smooth running of the collection of evaluation data. Data is collected on cognitive gains by the student, on the content and use of the materials, as well as on student attitudes. Dick (1977a) and Dick and Carey (1978) describe in detail the type of data to be collected.

Research in the field of instructional design and evaluation has shown that formative evaluation is a valuable process. Materials which have been revised on the basis of student feedback are more effective than materials which have not been evaluated at all (Dick, 1977a; Nathenson & Henderson, 1980).

However, is it necessary to go through all three stages? What is the optimum number of try-outs? At which stage is the most useful information to the revision process collected? It is important to find answers to these questions as the constraints of the real-life working environment have led to a large discrepancy between theory and practice. After an extensive review of the current literature on formative evaluation, Nathenson and Henderson (1980) state: "In practice, it is hard to find a single report in the literature of the formative evaluation of a product which has passed through the full three-stage try-out model..." (p. 42).

Some researchers have begun to address these questions. Wager (1983) reviewed two studies done to test the relative effectiveness of the one-to-one and small group stages of formative evaluation. These studies indicated that it may be possible to eliminate one of these stages and still have effective revision of materials.

Wager's own study explored this tentative conclusion further. She tried to ascertain which stage, one-to-one or small group, produced the most useful data for revision purposes. She was also interested in exploring the relationship between learner aptitude level and the effectiveness of learner feedback in the formative evaluation process.

Her results showed that materials revised from the feedback obtained in the one-to-one sessions with a mixed aptitude group were as effective as materials revised from

feedback from a combination of one-to-one and small group, and more effective than materials revised from feedback from a small group session alone. She concluded that it may be possible to eliminate the small group stage if revisions are made on the basis of the data collected from the one-to-one with learners of high, average, and low ability.

Once the data has been collected, how does one determine what revisions to make? Wager (1983) and Dick (1977a) both note that there is as yet no theoretical base on which revision decisions can be made. Melton (1982) and Nathenson and Henderson (1980) further add that the revision of learning materials is still an essentially creative process, more art than science.

Thus, a review of the current literature on formative evaluation produced few guidelines for the practitioner who is working with limited resources and strict deadlines.

CHAPTER 3

The Instructional Design

The instructional design of the materials followed the procedures set out by Dick and Carey (1978) in their book The Systematic Design of Instruction. In the design of the materials, certain constraints were imposed by the project coordinator:

1. All the materials were to be self-instructional.
2. Only print, video, and/or the computer could be used as instructional media.
3. All tests were to be designed for a computer-based system; therefore, only selected-response test items could be used.
4. The instruction time was to be four hours with extra lab time available for practice and the final project.

Educational Objectives

The materials produced were used as an introduction to computer programming in the context of a computer literacy course.

The broad goals given for these materials by the project coordinator were outlined in the Introduction. These were researched and analyzed to obtain more specific objectives. The result: programming is presented throughout the materials as a problem-solving activity involving three basic skills:

1. finding the solution to a given problem,
2. translating the solution into a form and a language the computer can understand, and
3. operating the computer so as to enter, run, and print out the program.

```

*****
*                                     *
*           Computer                 *
*                                     *
*       Programming                  *
*                                     *
*****

```

```

*****      *****      *****
*           *           *           *
*   find a   *   put the solution *   enter and *
*   solution *   \* into a language *   \* correct *
*   to the   *   /* the computer  *   /* the program *
*   problem  *   * understands   *   *           *
*           *           *           *
*****      *****      *****

```

These became the general terminal objectives of the instructional materials.

Instructional Analysis

A final activity was chosen to incorporate these three objectives. An instructional analysis was then performed to clearly identify the prerequisite subskills and the necessary entry behaviors. (See Appendix A.)

The skills identified in this analysis were subsequently grouped and translated into behavioral objectives. Appendix B presents a list of the terminal and

subordinate objectives derived from the instructional analysis as well as the test items constructed to measure mastery of these objectives. The final project was used to evaluate mastery of the terminal objectives.

Target Audience

The materials were designed to be used by undergraduate students in the faculty of Arts and Science at Concordia University. The course, an introduction to computer usage in the Arts and Science, is interdisciplinary, yet it was assumed that few, if any, students were majors in math or science, and that this was the first course about computers that they had taken. It was also assumed that the target audience had had no previous formal instruction in problem-solving techniques nor in computer programming.

The level of language used in the materials was based on these assumptions. The test items were also written for an undergraduate population; they were made intentionally more challenging. Furthermore, a mathematical approach to problem solving and programming was not taken in order to avoid alienating those who hate mathematics. Because of the educational level of the target audience, a less directed instructional strategy was used. Learners were not given full explanations but rather encouraged to find out first for themselves before reading the detailed discussions.

The materials evaluated in this thesis-equivalent constituted lesson 3 within the course structure. The first lessons of the course introduce the hardware, explain how

the computer works and lets the learner try out various applications software. The two quizzes of these previous lessons essentially test for the entry behaviors identified in the instructional analysis. Although the students had only to complete, not pass, the quizzes in order to continue onto the next lesson, it was assumed that they had the required entry skills.

Rationale for Media Selection

As mentioned earlier, certain constraints were imposed upon the instructional design process. The course is entirely self-instructional; therefore, all the instructional activities (motivation, content presentation, examples, practice and feedback, and follow-through activities) had to be contained within the materials themselves. Secondly, the choice of medium was limited to video, computer, and/or print.

The author was given a production deadline of 2 1/2 months. There was no money involved, nor was there any available to pay a computer programmer. These restrictions eliminated the option of a CAI tutorial to present some or all of the information. No currently available IBM PC software nor video was found which would have met the instructional goals set out by the project coordinator. Therefore, it was decided to develop two self-instructional booklets.

Unit 1--Problem Solving--uses only print. Unit 2--Programming with Logo--uses print to direct the students in

their learning of Logo turtle graphics. In this unit, all the learner's time is spent at the computer carrying out the instructions, examples, and practice exercises outlined in the booklet.

Outline of the Content

Unit 1 deals with problem solving. The main objective of this unit is to present information to the learners about problem-solving strategies and techniques useful to computer programming. The primary purpose of Unit 1, therefore, is information presentation.

Unit 1 introduces the concept of programming as a problem-solving activity. The terms "problem" and "problem solving" are defined according to Polya (1962). A four-step problem-solving strategy is discussed in detail. The techniques of stepwise refinement (breaking a problem down into smaller parts) and of debugging are presented in the context of these four steps. Appendix C contains the Table of Contents for this unit. The whole unit is presented in Appendix L. The content of Unit 1 covers subordinate objectives 1 to 6 described in Appendix B.

Unit 2 introduces the learner to Logo turtle graphics. It is an application of the strategy and techniques presented in Unit 1 to programming. Only a minimum of commands are introduced to allow the learner to complete a fun and interesting project. The emphasis throughout is not on learning Logo per se but on using the systematic problem-solving strategy and techniques outlined in Unit 1 for the program-

ming tasks. Appendix D gives the Table of Contents for this unit. Subordinate objectives 7 to 9 and terminal objectives 10 to 13 (see Appendix B) are presented in Unit 2.

The lesson is considered complete when the final quiz on Units 1 and 2 is taken and the final project handed in.

Instructional Strategy

Cartoons are used throughout to motivate and enhance the content of the print materials. Motivation is further encouraged by designing into the materials a detailed rationale for learning about problem solving and Logo, by stressing a non-mathematical approach, and by intentionally using a conversational tone of language rather than a didactic one.

The active participation of the learners is encouraged throughout the two units. In the unit on problem solving, the learners are asked to think about a problem, and/or to respond to a question or a situation. Only after having attempted an answer, do they read the appropriate feedback. In this way, the learners are not passively reading, but interacting with the print materials. In the unit on Logo, the learners are never told how to do something, rather they are encouraged to solve a problem (for example, draw a square) using the problem-solving strategy and techniques presented in the previous unit. Only after the learners have tried to draw a square, circle, or triangle is the step-by-step explanation given.

The presentation of the content for each objective is

followed by an example and then a practice exercise with immediate feedback. In this way the learners can test their understanding of each objective and monitor their own progress through the two units. It also provides a structure that allows the learners to move through the materials at their own pace according to their ability and interest.

At the end of each unit, a summary/review sheet reminds the learners of the concepts they are required to know. In Unit 2, summary sheets are used more frequently as the hierarchical nature of the learning is more crucial to the successful completion of the final project.

To reach and interest learners of all ability levels, a special remedial/enrichment section was included. In Units 1 and 2 the learners are presented with a list of recommended readings. As well, in Unit 2, they are encouraged to consult the IBM Logo tutorial available at the reserve booth if they have any problems or if they simply want to learn more Logo commands.

CHAPTER 4

Method

Preliminary Evaluation

The author conducted an initial evaluation of the materials when these were still in their draft form. A copy of the materials was shown to the project coordinator. At the same time the instructional analysis, behavioral objectives, test items, and a copy of the materials were given to a content expert for evaluation.

A one-to-one evaluation with three members of the target population was then carried out. Two had a minimum knowledge of the computer (some keyboard experience, no programming experience). The third represented high-ability learners as this person had programming experience but knew nothing about Logo and had had no formal training in problem solving. The author sat with the three as they completed the module noting any difficulties and asking specific questions in order to elicit the learners' reactions to the materials.

The revisions made to the original materials as a result of this preliminary evaluation are discussed in the next chapter.

Field Evaluation

The materials, revised according to the feedback from the preliminary evaluation, were then field-tested during the first offering of the computer literacy course in January 1985. Evaluation questions were designed and instruments developed to collect the necessary data.

Evaluation Questions

The basic evaluation design for the field test was a pretest-posttest single group. The following questions were formulated:

1. Were there statistically significant learning gains, on each objective?
2. For which objective(s) did the instruction NOT produce the desired results and why?
3. How did the learners react to the materials? (Particular attention was paid to the file management and the subprocedure sections to check if the problems indicated in the one-to-one evaluation had been effectively dealt with.)
4. Were the materials truly self-instructional?
5. Were the assumptions regarding the target population valid?
6. How long did it take the learners to complete the whole lesson?

Instrumentation

A pretest and a posttest.

For each subordinate objective 1 to 9, test items were developed (see Appendix B). Appendix E contains the pretest, Appendix F the posttest. An attempt was made to have equivalent rather than identical items and to vary their presentation order on the pretest and posttest to avoid the problem of testing effect.

The author had to decide two important issues: a) what score on each objective would constitute mastery of that objective; and b) what percentage of students need master each objective in order to conclude that the instruction for that objective was appropriate.

1. determining mastery criterion. A review of current literature on criterion-referenced measurement revealed the lack of a clear-cut theoretical base on which to make decisions about the appropriate number of test items for each objective and the appropriate criterion level for mastery of an objective (Berk, 1980; Briggs & Wager, 1981; Dick & Carey, 1978; Gagné & Briggs, 1979; Hambleton, 1980; Morris & Fitz-Gibbon, 1978; Popham, 1981). All these authors concur that the decision is a subjective one based on the importance of the objective in the learning hierarchy, the specificity of the objective, and the importance of the decisions to be made from the results.

The purpose of the posttest was to help locate any weaknesses in the instructional materials, not to pass or

fail the learner, nor determine the value of this product over another. The test also dealt only with subordinate objectives and was one in a series of measures whose data was used to determine instructional effectiveness. The author, therefore, established criterion scores for mastery on each objective according to the number of items used.

Table 1 presents the test items and mastery criterion raw score for each subordinate objective. Based on this criterion score, the learners were classified into two categories, mastery or non-mastery, for each of the subordinate objectives 1 to 9.

2. setting an acceptable standard. What percentage of students must master a given objective? Hambleton (1980) gives a detailed summary of several standard-setting methods but advises that the first step is to consider the decision-making context and the resources available. The author did this and found the methods described by Hambleton to be too elaborate for her purpose.

Briggs and Wager (1981) describe the 30%-80% convention (30% pass on the pretest; 80% pass on the posttest). They note that these levels could be higher or lower depending on the importance of mastering the performance and on the nature of the subject matter. Considering the performances being tested and the context in which the pretest and posttest were used, the author decided that the 30%-80% convention was suitable.

Table 1.

Test Items and Mastery Criterion for Each SubordinateObjective 1 to 9

Objective	Test Item Number		Mastery Score on Pre- and Posttest
	Pretest	Posttest	
Unit 1			
1	1 & 2	3 & 5	2/2
2	10	10	1/1
3	5 & 7 & 9	1 & 4 & 7	2/3
4	4 & 8	6 & 8	2/2
5	6	2	1/1
6	3	9	1/1
Unit 2			
7	11 & 14	12 & 17	2/2
8	12a-e & 13	13a-e & 14	4/6
9	15 & 16 & 17a-e	11a-e & 15 & 16	5/7

Note. Each test item is worth 1 point. Appendix B lists the contents of each objective. Appendix E contains the pretest, Appendix F the posttest.

A final project.

For the terminal objectives 10 to 13 (see Appendix B), a final project was developed:

Write a program using Logo turtle graphics to draw a means of transportation; for example, a car, a boat, a truck, a bicycle. The program must run without error and contain subprocedures. A printout of your program

must be handed in along with a brief description of the strategy used to arrive at your solution. Refer to Section 11.3 of Unit 2 for an example of such a description.

The final project was considered successfully completed when: (a) there were no bugs in the program; (b) sub-procedures were used; and (c) in the description of the problem-solving strategy used, the learners indicated that they had attempted to use a step-by-step plan rather than just random guessing.

A personal information form.

This was attached to the pretest and was used to verify the assumptions made regarding the target audience. See Appendix G.

An observation sheet.

This is contained in Appendix H. The teaching assistants on duty in the computer lab were instructed to record all comments and/or questions made to them by the students. This data was used to pinpoint any trouble spots and to determine to what extent the materials were self-instructional. Also, for this same purpose, specific questions on the evaluation questionnaires asked the learners to specify if they needed help and why.

Evaluation questionnaires.

Appendix I contains the questionnaire for Unit 1 and Appendix J for Unit 2. These were used to determine the learners' reactions to the materials, to identify any areas of difficulty, and to test the design decisions outlined in

chapter 3. Questions were included to gain a more accurate estimate of completion time.

Informal interviews.

These were conducted by the author. For Unit 1 three students were randomly selected to complete the Unit 1 evaluation questionnaire with me. For Unit 2 the author chose eight students who did not use subprocedures in their final project. After determining that the students knew what a subprocedure was, the author attempted to find out why they had not used them and what would have made them do so.

Evaluation Procedures

As mentioned in the Introduction, the instructional materials produced by the author constituted lesson 3 of the course program. The posttest became the quiz for this lesson. Therefore, all the 100 students registered in Interdisciplinary Studies C-300 for the 1984-85 winter session went through the materials and completed the posttest.

Sixty copies of the pretest and the two evaluation questionnaires were drawn up. The language lab personnel were instructed to give the students the pretest before handing out Unit 1. After Unit 1 was completed, the relevant questionnaire was given before they began Unit 2. Once Unit 2 was completed and the final project handed in, the posttest and the evaluation questionnaire for Unit 2 were

distributed.

The Sample and Sampling Procedures

The self-instructional, self-paced nature of the course made it difficult to control the distribution of the data collection instruments without the full cooperation of the project coordinator, the language lab personnel, and the three teaching assistants. Inevitably someone would forget to hand out the instruments or to collect them.

In the end, this was a positive point as the author had initially hoped to distribute the instruments evenly throughout the 100 students, that is, not just to the first 60 students who may have worked faster because they were more motivated or more able learners. Thus, the sample was truly randomly selected from the 100 students registered.

However, this method of distribution also resulted in some students in the sample completing only two or three, not all, of the instruments. Therefore, the sample consisted of the 46 students who completed both the pretest and the posttest. Of these, 37 filled out the questionnaire for Unit 1 and 27 for Unit 2. All completed the final project. The sample was 63% male and 37% female, having a mean age of 23.

CHAPTER 5

Results

Preliminary Evaluation

The project coordinator, after a look at the draft copy of Unit 1 and Unit 2, recommended some changes. He wanted to see the material rearranged so that the major ideas stood out. This was done by putting the key concepts in starred boxes. He also felt that the language of the content presentation for Unit 1 was too academic and so several passages were rewritten to present the material in a more straightforward manner.

The content expert approved the instructional analysis and content presentation. However, she pointed out that the instruction and test items for Objective 2 were not clearly stating the author's intent. As a result, the objective, relevant instruction, and related test items were considerably reworked.

The one-to-one evaluation with three members of the target audience gave a rough estimate of the minimum time to complete the lesson: two hours for Unit 1 and four hours for Unit 2. Only the experienced programmer completed all the materials in one session.

All three felt that Unit 1 was interesting and challenging (none had ever reflected upon what a problem was). They enjoyed being asked to think about a situation before

being given the answer. All three enjoyed Logo turtle graphics and commented spontaneously on the format--they liked trying to draw a triangle first before being told how to do it.

As a result of the one-to-one evaluation the following changes were made to Unit 1:

1. Errors in spelling and punctuation were corrected.

2. The passage summarizing characteristics of good and bad problem solvers was found to be redundant and overly negative by all three; therefore, it was omitted.

3. Polya's strategy was found to be unclear, yet they felt presenting two strategies was a useful exercise. Parts of Polya's strategy were paraphrased to make it clearer to non-mathematicians.

The following changes were made to Unit 2:

1. Major errors in the sample programs were corrected.

2. The file management section was rewritten as all three felt it was unclear.

3. The exercise to explain subprocedures was felt to be too difficult so it was modified.

The instructional materials used for the field evaluation included the above changes.

Field Evaluation

The aim of the quantitative and qualitative analyses which follow was to pinpoint areas of weakness in the instruction for revision purposes. Consequently, Unit 1--Problem Solving, and Unit 2--Programming with Logo--were treated as two separate entities.

The General Picture: Overall Learning Gains

For mastery learning the author expected strongly skewed distributions on the pretest and on the posttest. To obtain this general view, descriptive statistics were calculated on the pre- and posttest scores for Unit 1 and Unit 2 (see Table 2) and the distribution of the scores compared.

Table 2

Descriptive Statistics on Pre- and Posttest Scores

	Mean	Medium	Mode	SD
Unit 1				
Pretest	4.4	4.5	3.0	1.7
Posttest	6.9	6.9	6.0	1.7
Unit 2				
Pretest	2.2	1.8	0.0	1.8
Posttest	7.6	7.8	8.5	1.4

Note. Maximum score = 10 on the pretest and on the posttest.

For additional information, the change in scores was tested for statistical significance. The pre-posttest score gain for Unit 1 (Wilcoxon matched-pairs signed-ranks test, $z = -5.10$) and for Unit 2 (Wilcoxon matched-pairs signed-ranks test, $z = -5.91$) was significant at $p < .01$, one-tailed.

The Central Focus: Objective-by-Objective Analysis

Since the purpose of the field evaluation was to identify as precisely as possible any areas of difficulty in the instruction, the results of the objective-by-objective analysis yielded more useful information than the general information given above.

Subordinate objectives 1 to 9.

In the statistical analyses which follow the performance of each student was grouped into one of two categories: 0 for non-mastery or 1 for mastery, according to the criterion raw score given in Table 1. Figure 1 summarizes the performance on the pretest and posttest for each of the subordinate objectives 1 to 9.

A sign test was done on the pre-posttest change for each of these objectives. Only the gain for Objective 2 was not significant (one-tailed $p = .059$ using the binominal distribution with $n = 15$ and $x = 4$). All the other gains were significant (Objective 1: $p < .05$, one-tailed; Objectives 3 to 9: $p < .01$, one-tailed).

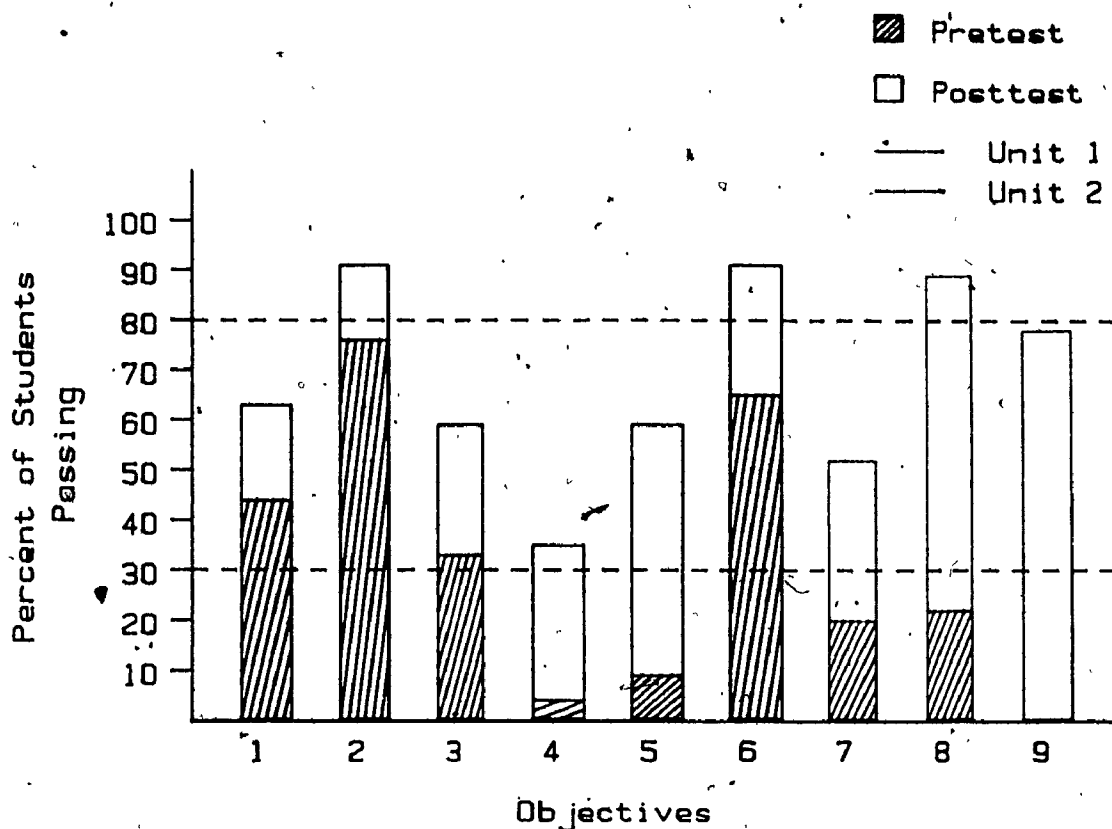


Figure 1. Percentage of students mastering subordinate objectives 1 to 9 on the pretest and the posttest. (See Appendix B for the content of each objective, Table 1 for the mastery criterion raw score. $N = 46$.)

The 30%-80% standard discussed in the previous chapter was then applied to evaluate the performance on each objective. The author had expected less than 30% of the sample to master each objective on the pretest and 80% or more to master the same objective on the posttest. As shown in Figure 1, several objectives did not meet these expectations.

In an attempt to discover why this happened, the author sought additional information. A frequency table was drawn up to examine more closely the pattern of change for each objective. Table 3 presents this information. Also, the item response patterns for those objectives NOT performing as expected was looked at to verify the assumption of test item validity. This data is presented in Appendix K.

Table 3

Percentage of Students Showing Four Different Patterns of Pre-Posttest Results on the Subordinate Objectives

Pattern of Results	Subordinate Objective								
	Unit 1						Unit 2		
	1	2	3	4	5	6	7	8	9
I Failed Pretest/ Passed Posttest	33	24	33	30	52	30	46	70	78
II Passed Pretest/ Passed Posttest	30	67	26	04	07	61	07	20	00
III Passed Pretest/ Failed Posttest	13	09	06	00	02	04	13	02	00
IV Failed Pretest/ Failed Posttest	24	00	35	65	39	04	35	09	22

Note. Appendix B lists the objectives. N = 46.

Terminal objectives 10 to 13.

Terminal objectives 10 to 13 (see Appendix B) were assessed by the final project, which was completed by everyone. Table 4 gives the results of the analysis of the final project. All mastered Objectives 10 and 11; however, only 24% of the sample mastered Objectives 12 AND 13 as well.

Analysis of the Qualitative Data

It has already been mentioned that there were difficulties in the collection of completed evaluation instruments from this self-paced, self-instructional course. All the students were willing to "take the tests" but many refused to fill in the evaluation questionnaires. Most complained that the process was too time-consuming; several just could not be bothered. As a result, from the sample of 46 who submitted both the pretest and posttest, 80% completed the evaluation questionnaire for Unit 1 (n = 37), and 59% that for Unit 2 (n = 27). The low response rate for Unit 2 was compensated by the informal interview data and the observation sheet data which dealt almost exclusively with Unit 2.

An initial look at the first questionnaires handed in showed that several students were unfamiliar with the bipolar adjective format. More precise instructions had to be included in order to get the desired response. The response frequencies for the Unit 1 questionnaire are given in Appendix I and those for Unit 2 in Appendix J.

Table 4

Analysis of the Terminal Objectives (10-13) Assessed by
the Final Project

Objective	Results of Analysis		
Objective 10	100%	mastery	BUT
	39%	student's final project is the file saved and printed out.	
	61%	student saved everything in RAM, not just the final project.	
Objective 11	96%	mastery	
		2/46 had a bug. Upon questioning, it was found that the error was due to lack of knowledge regarding file management NOT debugging skills.	
Objective 12	76%	mastery - final project consists of one superprocedure with only subprocedures as in FACE example.	
	24%	no superprocedure; a list of commands with occasional use of subprocedures given in text: SQUARE, TRIANGLE, CIRCLE	
Objective 13	48%	mastery - description indicates the final project was developed from a written plan; an algorithm (i.e., guess-estimates of angles and distances) was written first; the computer used only for debugging.	
	26%	the written plan took the same form as above (i.e., a drawing with the whole broken into parts) BUT no algorithm made first; computer used to determine angles and distances on a trial-and-error basis.	
	19%	no written plan; everything done on computer.	
	2%	not stated if algorithm or plan used.	
	4%	not submitted.	

Target Population

Through an analysis of the data on the personal information sheet (Appendix G), it was found that 78% of the sample had no previous programming experience and that 48% had not taken any math nor science courses in the last two years. The results for the question regarding previous formal instruction in problem solving were not used as several respondents misunderstood "problem solving" to mean "math". 70% had English as their mother tongue, 13% French and 13% another language.

A series of crosstabulations were done to see if any significant relationships existed between previous programming experience, the number of math/science courses taken, and the mastery of terminal objectives 12 and 13. This analysis revealed a significant relationship between previous programming experience and mastery of Objective 12, the use of subprocedures in the final project, (corrected chi square [1, N = 46] = .0461, $p < .05$). The Phi statistic (Phi = .362) showed a low positive relationship.

CHAPTER 6

Discussion

Evaluation Question 1: Were There Statistically Significant Learning Gains on Each Objective?

Unit 1--Problem Solving.

The objective-by-objective analysis showed that only the pre-posttest gain on Objective 2 was not significant. The gains on all the other objectives covered in Unit 1 were significant. These results are strong indicators that learning took place as a result of the instruction and not by chance.

The author had expected that less than 30% would master each objective on the pretest and 80% or more on the post-test. For Unit 1 none of the objectives performed as expected (see Figure 1).

The design of the instruction was based on the principles of mastery learning. Consequently, the author expected strongly skewed distributions of test scores. Unit 1 did not produce these results (see Table 2).

Therefore, although the significance tests showed that learning did take place in Unit 1, it was not in the pattern anticipated. Possible explanations are presented in the next section.

Unit 2--Programming with Logo.

Unit 2 did produce the desired results. The distribution of scores was in the pattern anticipated (see Table 2). The pre-posttest gains were significant on the subordinate objectives 7 to 9 covered in Unit 2. Only Objective 7 did not meet the 30%-80% expectation (see Figure 1). For Objective 9 the ~~78%~~ pass on the posttest is acceptable seeing that no one achieved the objective on the pretest.

The performance on the final project was on the whole very satisfactory. (The analysis of Objectives 10 to 13 as shown in Table 4 are discussed in detail in the next section.) The learners obviously acquired the knowledge and skill to produce a short program using Logo turtle graphics.

Evaluation Question 2: Which Objective(s) did NOT Produce the Desired Results and Why?

Subordinate objectives 1 to 7.

The preceding section identified Objectives 1 to 6 of Unit 1 and Objective 7 of Unit 2 as not performing as expected. There are three possible explanations for these results:

1. The entry behaviors may not have been correctly identified. Table 3 shows that 67% and 61% of the subjects passed the pretest AND the posttest items for Objectives 2 and 6 respectively. This would seem to indicate that the learners already knew the related content and thus instruction may not have been necessary.

2. The instruction for some objectives may not have been adequate. More than one-third of the learners failed both the pretest AND the posttest items related to Objectives 3, 5, and 7. An astonishing 65% did the same for Objective 4 (see Table 3). A look at Objectives 1 and 7 in Table 3 indicates that the corresponding instruction may even have been detrimental to 13% of the learners. Perhaps there are serious weaknesses in the instruction related to these objectives.

3. The tests may have weak validity. The determination of mastery or non-mastery of the subordinate objectives 1 to 9 was based strictly on the pre- and posttest scores. Were these tests valid measures? The author hypothesizes that several factors seriously weakened pretest and posttest validity, thus making this explanation the most likely one for the unexpected results:

(a) poor learner motivation. Personal observation and student feedback showed that motivation to succeed on the test had an influence on the test results. It was clearly stated in the instructions to the pretest and posttest (see Appendices E and F) that the results would not be used to grade the students. This coupled with the pressure to complete the rest of the course may have led many students to perform at a low standard. For example, to master Objective 4 the learner must have closely studied the related content in Unit 1. A cursory reading of the unit and a quick completion of the posttest may have produced the result that 65% failed the pretest and the posttest (see

Table 3).

(b) too few test items. It can be argued that it is not logical to determine mastery on the basis of performance on one test item. The author in designing the tests was aware of this issue (refer to chapter 4--Instrumentation). Nonetheless, it may be that the unexpected pattern of results is in part due to the fact that too few items were used to adequately measure certain objectives.

(c) weak test items. Two measures of test item validity are reported in the literature on criterion-referenced measurement: item-objective congruence and item sensitivity (Berk, 1980; Hambleton, 1980; Morris & Fitz-Gibbon, 1978; Roid & Haladyna, 1982).

Item-objective congruence was carefully dealt with at the design stage. The match between items and objectives was submitted for review to a content expert and approved at the one-to-one evaluation stage. Hence this aspect of test-item validity is satisfactory.

Item sensitivity refers to the ability of a test item to distinguish between masters and non-masters of the corresponding objective. Berk (1980) and Roid and Haladyna (1982) suggest a detailed item response analysis of both pretest and posttest items to determine item sensitivity. An item is considered weak if there is a very small or negative movement from pretest to posttest (indicated by the percent pass). Distractors that do not perform effectively also indicate a weak test item. Appendix K shows that several objectives did indeed have weak test items.

Take Objective 7 as an example. It was the only objective of Unit 2 that did not perform as expected. The related pretest item 14 and posttest item 12 have poor item sensitivity because of the ineffective functioning of the distractors. The unexpected results for Objective 7 could then be attributed to the weak validity of the corresponding test items and not to ineffective instruction.

Informal student feedback supports this hypothesis of weak test item validity as the most likely explanation for the unanticipated results from the pre- and posttest scores. Several students sought me out to complain about the similarity of choices within questions (especially posttest item number 10 for Objective 2 and posttest item number 12 for Objective 7). The author mentioned in chapter 3 that the test was deliberately made challenging to appeal to a university audience. Perhaps this quest for challenging questions was too zealous.

Given these serious threats to test-item validity, the author, in the revision decision process, relied heavily on the analysis of the final project and on the qualitative data. This process is illustrated in the following discussion of Objective 2.

Objective 2: An example of the revision decision process.

There are two most likely explanations for the results for Objective 2. A look at Table 3 shows one possible

explanation. 67% passed the pretest and the posttest. It may then be inferred that the content of this objective was known by the majority of learners and thus need not have been dealt with in the materials. A look at the item response patterns for the related test items (see Appendix K) uncovers a second possible explanation. The distractors on the test items for both the pretest and posttest did not work very well. Perhaps the items were too easy. Or perhaps the result came from a combination of these two explanations.

The feedback from the learners indicated that this section of the unit did have an impact. When asked what they remembered, all said the problem-solving strategy AND the literalness of the computer. Many positive comments were received about the exercise dealing with the necessity of making your assumptions clear (the chicken and hen problem). All said that they knew the material on a subconscious level but appreciated having the concept verbalized and related to the computer. The material, then, although easy, did serve a useful purpose for them. Student feedback also indicated that the corresponding question on the posttest was particularly ambiguous.

It was therefore decided that no change would be made to the instructional materials as the qualitative data clearly showed that the objective was achieved. The related test items, however, need to be substantially revised.

Terminal objectives 10 to 13.

1. Objectives 10 and 11. The results of the analysis for Objectives 10 and 11 (see Table 4) are as expected. The author, at the design stage, had decided not to explain the fact that all of RAM is saved in a Logo file. Consequently, the learners were not told to save their project in one file and no instruction as to how to do this was included in the materials. This explains in part the file management problems encountered by the learners.

From the type of questions asked and the comments made, however, it was evident that many did not have the required entry behaviors specified in the instructional analysis, especially a clear concept of RAM and ROM. Several learners could follow the instructions, but once on their own showed by their questions that they had not assimilated an understanding of what it meant to save a program and reload it from a diskette.

2. Objectives 12 and 13. Table 4 reveals a relatively poor performance on Objectives 12 and 13. In light of the feedback from the learners, it is quite acceptable. There were early indicators that most of the learners found the instructions for the final project too vague. This clearly affected the performance on Objectives 12 and 13.

For Objective 12, the instructions did not specify that ONLY subprocedures should be used as in the FACE example. The fact that 76% of the sample did this on their own, however, shows that the instruction for Objective 12 had

some effect.

For Objective 13, supplemental instructions had to be written on the blackboard as no one knew exactly what was wanted. Nevertheless, part of the objective was accomplished as 74% did start the problem-solving process by writing a plan and did use the technique of stepwise refinement.

From the qualitative data it was found that other factors, such as learning style and personal motivation, also came into play in the mastery of Objective 13. Several learners indicated that time was a factor. Their objective was simply to get the lesson finished. (It should be noted that their final grade was based on the number of lessons completed and the final exam.) This meant that they did the final project as quickly as possible, eliminating the planning/algorithm stage. Others, under the same time constraint, worked more diligently and thoroughly.

Only 24% mastered all the terminal objectives 10 to 13. This is very low, yet not unexpected given the difficulties with Objectives 12 and 13 discussed above.

Evaluation Question 3: How did the Learners React to the Material?

Unit 1--Problem Solving.

The qualitative data collected for Unit 1 (see Appendix I) supported the design decisions outlined in chapter 3. The learners enjoyed the instructional strategy. For

several the actual problem solving was the best part. Of those who answered the relevant question, 61% felt print was adequate and sufficient.

The vast majority felt that the information presented was easy yet nonetheless worthwhile as it made explicit unconscious processes. The most helpful sections were the ones dealing with the systematic problem-solving strategy and its component parts. Algorithms were mentioned next.

The learners showed a positive attitude toward the unit. There were some negative comments about the unit being tedious because of the repetition, and there was one very strong dissenting voice. This person, a major in math, found the unit too easy and thus boring.

On the whole, the learners had very little difficulty with this unit. More actual problem-solving exercises were requested by 6 of the 37 respondents; 3 wanted more practice with algorithms. The only area of weakness identified from this data was the section introducing algorithms. For one learner, it was unclear whether an algorithm was a product or a procedure and for some learners the examples were not very clear.

Unit 2--Programming with Logo.

All the students enjoyed this unit, especially the final project. (See Appendix J for the results of the Unit 2 questionnaire.) The level of language was suitable and the instructional strategy popular with the learners.

From the observation sheet, informal interviews, and evaluation questionnaire four main areas of difficulty were identified:

1. missing essential commands. For example, many students got into the definition mode (the > prompt) and did not know how to get out.

2. unclear instructions. For example, in the introduction the learner was reminded to have a formatted diskette. Several reformatted their diskette and lost all previous work. Many were confused by the instructions to keep a copy of the INITIALS exercise for later use. Also, as mentioned earlier, the instructions for the final project were too vague.

3. the practice exercise FACE and the concept of a program made up only of subprocedures as illustrated by this example. Many learners did not fully understand the concept because the example itself was confusing. This was probably due to the fact that one of the subprocedures had a bug and that the instructions were not clear.

From the informal interviews of those who did not use subprocedures in their project it was found that they knew what a subprocedure was and how to define one in the Logo editor. The concept of stepwise refinement was also well understood. But the link was not made between defining each part of the problem as a subprocedure and putting these parts or subprocedures together into a program. All suggested rewriting the instructions for the final project. Only a very few wanted more examples like FACE. Many cited

this example as confusing because of the bug and the instructions.

4. file management. There were many problems with saving, loading, and editing files. Possible reasons for this were discussed in the above section subtitled Terminal Objectives.

Evaluation Question 4: Were the Materials Self-Instructional?

Unit 1--Problem Solving.

Data from the observation sheet and the evaluation questionnaire confirmed that Unit 1 was self-instructional. The vast majority of the learners completed the unit on their own.

Unit 2--Programming with Logo.

Unit 2, on the other hand, was not entirely self-instructional. The learners asked numerous questions in the four areas discussed in the preceding section. Some of the revisions outlined in chapter 7 were made to remove these difficulties so that the learners would be better able to complete Unit 2 with a minimum of assistance.

Yet the author also observed that many learners asked questions that were obviously answered in the materials. These students seemed to have a difficult time with the self-instructional nature of the course. They were either not used to obtaining all the necessary information from the instructional materials, no matter the medium, or not used

to working completely on their own. Once again personal learning style came into play. Given this influence, can material be truly self-instructional? The author makes some recommendations in the next chapter.

Evaluation Question 5: Were the Assumptions About the Target Population Valid?

The author was surprised at the proportion of learners having programming experience (22%) and previous math/science courses (52%). The instruction had been written with the assumption that the target population had neither programming experience nor math/science courses. The significant correlation between previous programming experience and the use of subprocedures in the final project (mastery of Objective 12) was, therefore, expected. The difficulties with Objective 13 outlined earlier precluded a similar significant result for this objective.

Student feedback, however, indicated that even with programming experience and a heavy math background the content of Unit 2 was effective and enjoyable. As for Unit 1, the self-paced nature of the course allowed these students to spend less time on areas they were familiar with.

Evaluation Question 6: How Long did it Take the Learners to Complete the Lesson?

The average completion time for Unit 1 was one and a half hours, for Unit 2 six and a half hours, and for the

final project three hours--a total of 11 hours. The coordinator had originally assigned one 4-hour lab session for this lesson (Unit 1 and Unit 2) with extra time available, if needed, to complete the final project.

Within the course structure, this lesson was the learner's first programming experience. Many were still apprehensive and insecure while using the computer. Consequently, more time was spent on Unit 2 and the final project than anticipated. The results indicate that, if the lesson remains in the same place within the course, two 4-hour lab sessions would be needed to complete the instructional materials in their present form.

CHAPTER 7

Conclusions and Recommendations

The results of the field evaluation indicate that the materials produced by the author are educationally sound--learning did take place. The broad instructional goals presented in the Introduction were achieved: the learners acquired knowledge of a specific problem-solving strategy and two techniques useful for computer programming, and they acquired the skill to use this information to solve a programming task on the computer with Logo turtle graphics.

The results further show that the instructional materials are suitable for a wider target population, that is, for learners with some programming experience and some math/science courses. However, these two learner characteristics need to be more precisely defined in order to specify more accurate entry behaviors for this lesson, as well as for the course as a whole.

The materials are appropriate for the context for which they were designed--a university-level computer literacy course. The author feels that this package could also be used in other contexts as long as the general terminal objectives and learner characteristics remain the same.

The field evaluation also revealed specific areas of weakness that need to be dealt with (see chapter 6--Discussion). The process by which the author arrived at the revisions and recommendations discussed below has already

been described in chapter 6.

Recommendations

Unit 1--Problem Solving.

The material in its present form is adequate for the broader goal of information presentation on problem-solving strategies and techniques related to computer programming.

From the analysis of the evaluation data on this unit and from a consideration of the overall course structure, the author recommends a substantial reworking of the section on algorithms. This section should be broadened to include a presentation of flowcharting (presently given in lesson 5: Programming in BASIC) and a rationale on the importance of algorithms and flowcharts in the programming process.

The concept of different computer languages to solve various types of problems could be introduced at this point. Examples and practice exercises should deal with more general problems, not just drawing shapes. More practice is also needed in breaking a given problem into smaller parts, writing a flowchart and algorithm for each part, then putting the parts together into one program.

Another medium should be used to present this information as the amount of print for Unit 1 is sufficient. A short video could introduce the concept of different computer languages to solve different problems and also to present the rationale for the flowcharting/algorithmic process. A CAI tutorial could then provide the necessary practice and

feedback.

Finally, the testing procedures and the test items for subordinate objectives 1 to 6 covered in this unit need to be reviewed and revised so that test performance reflects actual achievement. This certainly must be done and the resulting instruments validated before any summative evaluation is attempted. Because of the intense relationship between producers and their materials, it is often difficult for producers to find alternative wording or test items. Thus, the author suggests that the course development team work together on reviewing the test items and testing procedures.

The above recommendations have not been implemented. The decision to do so rests with the project coordinator.

Unit 2--Programing with Logo.

This unit was successful in consolidating and achieving the general terminal objectives stated in chapter 3--The Instructional Design. All the students learned enough Logo commands to solve a simple problem (the final project).

Revisions were made: (a) to eliminate the difficulties encountered by the learners discussed in chapter 6, (b) to enhance the self-instructional nature of the unit, and (c) to be consistent with terms and explanations given in the two previous lessons. Appendix M presents the revised edition of Unit 2--Programming with Logo.

The sections dealing with file management and subprocedures (the FACE example) gave trouble both in the one-to-one

evaluation and in the field test. The file management problems were addressed by rewriting the sections dealing with computer memory and storage devices in the earlier lessons of the course and by including more detailed explanations in the file management section of Unit 2.

The directions for the FACE example were rewritten and the bug in the example removed. Due to space limitations, the author decided not to include additional examples but rather to clarify the explanation of subprocedures.

The author recommends another field test of the file management and subprocedure sections to ensure that the difficulties have been eliminated.

The instructions for the final project as well as Objectives 12 and 13 were rewritten so that the students would be aware of exactly what was wanted from them. (See Appendix B for the revised Objectives 12 and 13 and the revised instructions for the final project.)

The author recommends that Objective 13 and the relevant test item (a description of the problem-solving strategy used on the final project) be revised so as to be consistent with the course goal of implementing a computer-based testing system.

The Influence of Learning Style

Chapter 6 showed how the self-instructional nature of the materials was affected by personal learning style. To encourage self-instruction and to cater to different learning styles, specific learning activities could be designed to

enable learners to work in pairs and/or in a group.

Redundancy of key concepts throughout the course (saying the same thing in a different way and/or in a different medium) would help those learners unaccustomed to using self-instructional materials.

Suggestions for Research

The formative evaluation process.

It is the author's opinion that the three-stage formative evaluation model presented by Dick and Carey (1978) is too cumbersome a process for the real-life working environment which often includes strict deadlines and limited budgets. More research is needed along the lines of Wager's study (1983) in order to make the model more accessible and appealing to the practitioner.

The revision decision process.

Detailed guidelines are available to the practitioner on the procedures to follow and the data to collect in the formative evaluation process. Suggestions can be found on how to design the data collection instruments and how to analyze the data. Yet very little is said on how to derive the end-product, the revisions, from all that has gone before.

The author can only concur with researchers in the field of instructional design and evaluation (Dick, 1977a; Gagné & Briggs, 1979; Melton, 1982; Nathenson & Henderson,

1980; Wager, 1983) that the choice of what revisions to make is essentially a creative process relying heavily on the evaluator's interpretation of the data and on the constraints of the working environment. Research in the area of linking evaluation data to revisions is needed in order to increase the possibility that the revisions made actually do lead to more effective learning materials.

Unit 1 and Unit 2.

The author would like to see research done on this lesson to test the hypothesis that Unit 1 (in its present form) is necessary for the successful completion of Unit 2. In other words, is a general introduction to problem-solving as presented in Unit 1 necessary to the acquisition of the programming skills covered in Unit 2?

The results of the field evaluation seem to indicate that Unit 1 provides interesting and useful, but not necessary, information. If only Unit 2 were presented, could the general terminal objectives outlined for this lesson in chapter 3 still be achieved?

In conclusion, the author notes that the materials she produced did achieve the general objectives set out by the project coordinator. In the context of the computer literacy course the materials were an effective introduction to computer programming.

References

- Berk, Ronald A. (1980). Item Analysis. In Ronald A. Berk (Ed.), Criterion-referenced measurement: The state of the art (pp. 80-123). Baltimore, Maryland: The John Hopkins University Press.
- Briggs, Leslie J., & Wager W. (1981). Handbook of procedures for the design of instruction (2nd ed.). Englewood Cliffs, New Jersey: Educational Technology Publications.
- Dick, W. (1977a). Formative evaluation. In L. J. Briggs (Ed.), Instructional design: Principles and applications (pp. 311-333). Englewood Cliffs, N.J.: Educational Technology Publications.
- Dick, W. (1977b). Summative evaluation. In L. J. Briggs (Ed.), Instructional design: Principles and applications (pp. 337-348). Englewood Cliffs, N.J.: Educational Technology Publications.
- Dick, W., & Carey, L. (1978). The systematic design of instruction. Glenview, Illinois: Scott, Foresman and Company.
- Gagné, R. M., & Briggs, L. J. (1979). Principles of instructional design. New York: Holt, Rinehart and Winston.
- Gledhill, V. X. (1981). Discovering computers. Singapore: Science Research Associates Ltd.

- Hambleton, R. K. (1980). Test score validity and standard-setting methods. In Ronald A. Berk (Ed.), Criterion-referenced measurement: The state of the art (pp. 80-123). Baltimore, Maryland: The John Hopkins University Press.
- Heller, R. S., & Martin, C. D. (1982). Bits'n bytes about computing: A computer literacy primer. Rockville, MD: Computer Science Press.
- Horn, C., & Poirot, J. (1981). Computer literacy; problem solving computers. Austin, Texas: Sterling Swift Publishers.
- Koffman, E. B., & Friedman, F. L. (1979). Problem solving and structured programming in BASIC. Reading, Mass.: Addison-Wesley Publications.
- Krasnor, L. R., & Mitterer, J. O. (1983). LOGO and the development of general problem solving skills. Unpublished manuscript, Brock University, Ontario.
- Lochhead, J. (1981). Research synthesis on problem solving. Educational Leadership, 39(1), 68-70.
- Lochhead, J., & Clement, J. (Eds.). (1979). Cognitive process instruction: Research on teaching thinking skills. Philadelphia, Penn.: The Franklin Institute Press.
- Melton, R. F. (1982). Instructional models for course design and development. Englewood Cliffs, New Jersey: Educational Technology Publications.

- Morris, L. L., & Fitz-Gibbon, C. T. (1978). How to measure achievement. Beverly Hills: Sage Publications Ltd..
- Nathenson, M. B., & Henderson, E. S. (1980). Using student feedback to improve learning materials. London, Eng.: Croom Helm Ltd..
- Nickerson, R. S. (1981). Thoughts on teaching thinking. Educational Leadership, 39(1), 21-24.
- Papert, S. (1980). Mindstorms: Children, computers and powerful ideas. New York: Basic Books, Inc..
- Pea, R. D. (1983). LOGO programming and problem solving (Technical Report No. 12). New York: Bank Street College of Education, Center for Children and Technology.
- Polya, G. (1957). How to solve it: A new aspect of mathematical method (2nd ed.). Garden City, N.Y.: Doubleday.
- Polya, G. (1962). Mathematical discovery: On understanding, learning, and teaching problem solving (Vols. 1 & 2). New York: John Wiley & Sons.
- Popham, W. James (1981). Modern educational measurement. Englewood Cliffs, N.J.: Prentice-Hall Inc..
- Rampy, L. M. (1984, April). The problem-solving style of fifth graders using Logo. Paper presented at the annual meeting of the American Educational Research Association, New Orleans.
- Roid, G. H., & Haladyna, T. M. (1982). A technology for test-item writing. Academic Press.
- Rubinstein, M. F. (1975). Patterns of problem solving. Englewood Cliffs, New Jersey: Prentice-Hall, Inc..

Schoenfeld, A. (1979). Can heuristics be taught? In Lochhead, J., & Clement, J. (Eds.), Cognitive process instruction: Research on teaching thinking skills (pp. 315-338). Philadelphia, Penn.: The Franklin Institute Press.

Statz, J. (1973). The development of computer programming concepts and problem-solving abilities among 10-year olds learning Logo. Unpublished doctoral dissertation, Syracuse University.

Tuma, D. T., & Reif, F. (Eds.). (1980). Problem solving and education: Issues in teaching and research. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Wager, J. (1983). One-to-one and small group evaluation. Performance and Instruction Journal, 22(5), 5-7.

Watt, D. (1984). Learning with Apple Logo. New York: McGraw-Hill Book Company.

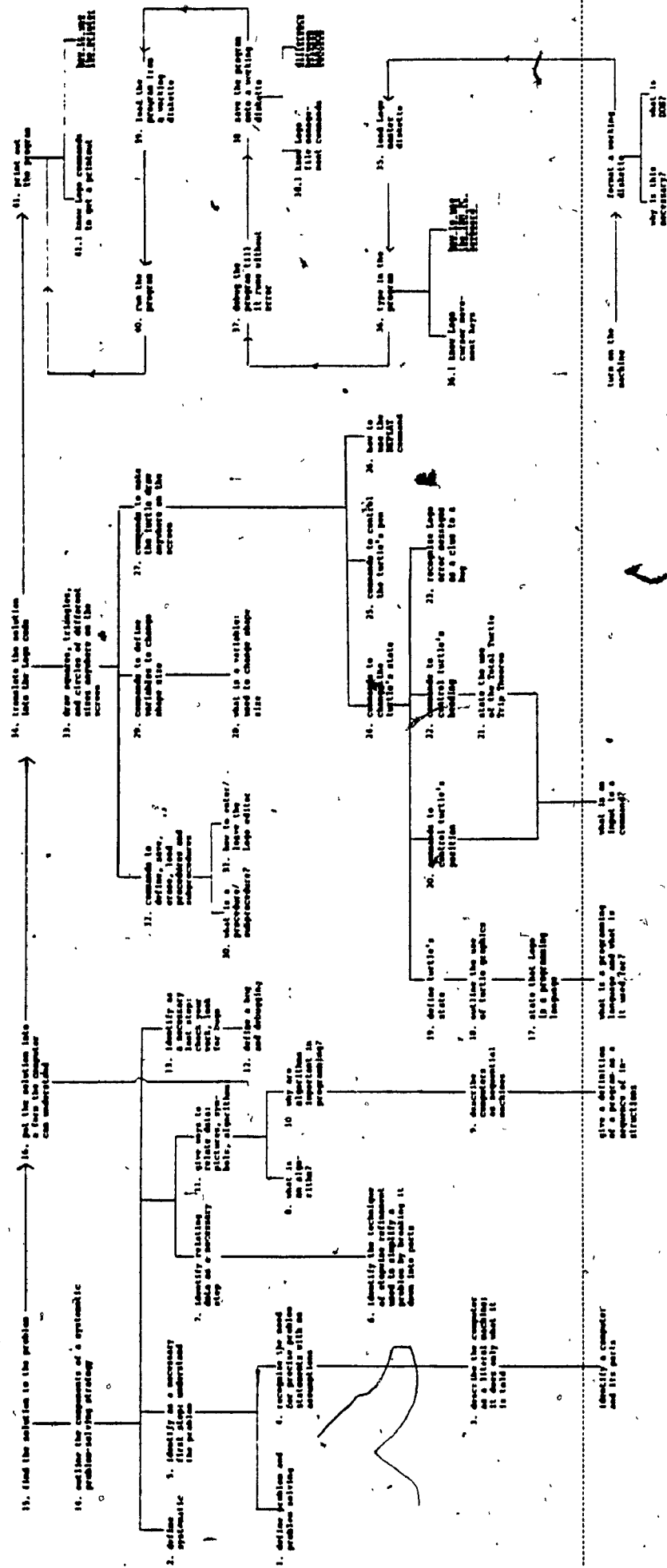
Whimbey, A. (1980). Students can learn to be better problem solvers. Educational Leadership, 37, 560-562.

Whimbey, A., & Lochhead, J. (1981). Problem solving and comprehension: A short course in analytical reasoning (2nd ed.). Philadelphia, Penn.: The Franklin Institute Press.

Appendix A.

The Instructional Analysis

The learner will write, run, and edit out a program to solve a given problem using Loop Turtle graphics.



101. Numbers indicate the teaching sequence; entry numbers are below the dotted line or underlined.

Appendix B

List of Subskills, Behavioral Objectives and Test Items

Refer to Appendix A--the Instructional Analysis for a description of the subskill.

Refer to Appendix E for the pretest and Appendix F for the posttest.

Unit 1--Problem Solving

Objective 1. (Subskills 1 and 2)

1. Given a list of definitions, the learners will select the correct definitions for problem, problem solving, and systematic.

Test items: pretest 1 and 2; posttest 3 and 5

Objective 2. (Subskills 3 and 4)

2. Given an example of a problem statement, the learners will correctly identify the need for a more precise problem statement with no assumptions.

Test items: pretest 10; posttest 10

Objective 3. (Subskills 5, 7, 13, 14)

3. Given a model of a systematic problem-solving strategy, the learners will correctly identify its component parts.

Test items: pretest 5, 7, and 9; posttest 1, 4, and 7

Objective 4. (Subskill 6)

4. Given a problem situation, the learners will identify the use of the technique stepwise refinement.

Test items: pretest 4 and 8; posttest 6 and 8

Objective 5. (Subskills 8 to 11)

5. The learners will recognize the use of an algorithm to put the solution into a sequence the computer understands.

Test items: pretest 6; posttest 2

Objective 6. (Subskill 12)

6. The learners will choose the correct definition of the word bug and debugging.

Test items: pretest 3; posttest 9

Unit 2--Programming with Logo**Objectives 7 and 8.** (Subskills 17 to 19, 21, 23, 28, and 30)

7. The learners will be able to choose a correct definition for the following terms: Logo, turtle graphics, turtle's state, variable, procedure, and subprocedure.

Test items: pretest 11 and 14; posttest 12 and 17

8. The learners will identify the correct use of a variable, an error message, the total turtle trip theorem, and the Logo editor.

Test items: pretest 12 and 13; posttest 13 and 14

Objective 9. (Subskills 20, 22, 24 to 26, 29, and 32)

9. The learners will be able to match the appropriate command to a description of that command.

Test items: pretest 15, 16, and 17; posttest 11, 15, and 16

Terminal Objectives

Objective 10. (Subskills 35 to 41)

10. The learners will be able to write, run, and print out a program to solve a given problem using Logo turtle graphics.

Test item: the final project (see the final page of this appendix)

Objective 11.

11. Given a definition of the technique debugging and a problem to solve with the computer, the learners will demonstrate their ability to use the technique of debugging by writing a program that runs without errors.

Test item: the final project

Objective 12.

12. Given an explanation of the technique stepwise refinement, the learners will demonstrate their ability to use this technique by using subprocedures in their program.

Revised Objective 12: Given an example of the use of the technique stepwise refinement in programming, the learners will demonstrate their ability to use this technique by using only subprocedures in their program as in the example program.

Test item: the final project

Objective 13.

13. Given a model of a systematic problem-solving strategy and a problem to solve with the computer, the learners will demonstrate their ability to use a systematic problem-solving strategy by providing a description of the steps used to arrive at the solution.

Revised Objective 13: Given a model of a systematic problem-solving strategy and a problem to solve with the computer, the learners will demonstrate their ability to use a systematic problem-solving strategy by providing a description of the steps used to arrive at the solution.

The learners must indicate that (a) they worked from a written plan, (b) wrote an algorithm in the form of guess-estimates of angles and distances, and (c) used the computer only to debug their algorithm.

Test item: the final project

The Final Project

Write a program using Logo turtle graphics to draw a means of transportation; for example, a car, a boat, a truck, a bicycle. The program must run without error and contain subprocedures. A printout of your program MUST be handed in along with a brief description of the strategy used to arrive at your solution. Refer to Section 11.3 of Unit 2 for an example of such a description.

Revised Instructions for the Final Project:

1. Write a program using Logo turtle graphics to draw a means of transportation; for example, a car, a boat, a truck, a bicycle, etc..

** your program should consist of one main procedure and several subprocedures as in the FACE example in Section 15.1.

** it should have NO bugs.

** it should be saved in a separate file on your own diskette. Use your last name as the file name. (The extension .lf is automatically added so you know it's a Logo file.)

Your drawing need not be complicated. You will be evaluated strictly on the above three points.

2. Write a brief description of the problem-solving strategy you used to arrive at your solution.

** how did you attack the problem? Did you work from a written plan?

** did you write an algorithm first before you used the computer?

** how did you use the computer: to debug your algorithm? to determine angles and distances on a trial and error basis? to immediately draw your shape?

** did you use any of the techniques described in Unit 1--Problem Solving?

There is no one correct answer. The objective of this exercise is to get you thinking about how you solve programming problems so that you become more conscious of the process.

Appendix C

Table of Contents for Unit 1--Problem Solving

1. Introduction
 - 1.1 Why Study Problem Solving?
 - 1.2 Programming and Problem Solving
2. Do You Have a Problem?
3. Problem-Solving Strategies
 - 3.1 What is a Systematic Problem-Solving Strategy?
 - 3.2 Two Examples of Problem-Solving Strategies
4. A Systematic Problem-Solving Strategy To Use
 - 4.1 Understand the Problem
 - a) Be Precise: Know What is Wanted
 - b) Breaking a Problem Down--Stepwise Refinement
 - 4.2 Relate the Data
 - 4.3 Find a Solution
 - a) Algorithms
 - 4.4 Check Your Results
 - a) What is a Bug?
5. A Final Word on Problem Solving
 - 5.1 Recommended Reading
6. Summary

Appendix D

Table of Contents for Unit 2--Programming with Logo

1. Introduction
 - 1.1 A Review of a Systematic Problem-Solving Strategy to Use
2. What is Logo?
 - 2.1 Logo Turtle Graphics
 - 2.2 A Turtle???
3. How to Load Logo
 - 3.1 Cursor Movement Keys
4. Let's Meet the Turtle
 - 4.1 Commands to Make the Turtle Appear and Disappear
 - 4.2 Commands to Control the Amount of Text on the Screen
5. The Turtle's State
 - 5.1 Commands to Change the Turtle's Position
 - 5.2 Commands to Change the Turtle's Heading
6. Error Messages
7. SHOOT
 - 7.1 How to Play
 - 7.2 Playing SHOOT Using a Systematic Problem-Solving Strategy
8. Review of Logo Commands
9. Commands to Control the Turtle's Pen
 - 9.1 Drawing Letters
10. Some Helpful Hints: The Total Turtle Trip Theorem
11. Drawing Shapes
 - 11.1 A Square
 - 11.2 The REPEAT Command
 - 11.3 Practice in Drawing Shapes Using a Systematic Problem-Solving Strategy
 - a) a Triangle
 - b) a Circle
12. A Quick Review
13. Procedures
 - 13.1 Defining Procedures in the Logo Editor
 - 13.2 The Command POTS
 - 13.3 Debugging Procedures
14. Variables
15. File Management
 - 15.1 Saving Procedures
 - 15.2 Erasing Procedures
 - 15.3 Loading Procedures From Your Working Diskette
 - 15.4 The Command DIR
 - 15.5 Erasing Files From Your Working Diskette
16. Subprocedures
 - 16.1 Subprocedures: A Problem-solving Technique in Action
17. A Final Review
- 17.1 Summary
18. Recommended Reading
19. Final Project
- Appendix A: Loading Logo
- Appendix B: Formatting a File Diskette
- Appendix C: Cursor Movement Keys

Appendix E**Pretest****NAME:****I.D. NUMBER:**

The following short quiz is on the contents of the module **Introduction to Computer Programming**. Its objective is to find out how much you already know about the topic **BEFORE** you study the materials.

You are not supposed to know all the answers, so don't be discouraged. When you complete the module, you'll have the answers.

The results will NOT be used to grade you. They will be used only to improve the quality of the instructional materials.

WHEN YOU ARE FINISHED, HAND THIS IN AT THE RESERVE BOOTH AND PICK UP LESSON 3:

An Introduction to Computer Programming**Unit 1****Problem Solving**

CIRCLE THE LETTER OF THE CORRECT ANSWER! IF YOU DON'T KNOW THE ANSWER, LEAVE IT BLANK AND GO ON TO THE NEXT QUESTION. DO NOT GUESS.

1. To solve a problem means

- A. to relate given data.
- B. to understand the difficulty to be overcome.
- C. to obtain missing data.
- D. to achieve a desired goal.

2. A systematic approach to problem solving means that you try to solve the problem

- A. by using pictures to clarify any relationships.
- B. by working out the solution on paper first.
- C. by following a step-by-step plan.
- D. by guessing intelligently.

3. In the following context, what is the meaning of the word "bug"?

Bob has been working on this program for an hour and he still hasn't found the bug.

- A. the reason why the program won't run
- B. the way to code the program
- C. the desired solution to the problem
- D. the missing part

4. Paul wants to write a program that will output a game called TARGET. The computer will draw a circle and a man at different places on the screen. The player must tell the computer how far to move the man so that he will end up inside the circle. Paul begins by writing a small program to draw the circle and another program to draw the man. Next he writes a program to place the circle at a random position on the screen. What is he doing?

- A. revising his program
- B. using the technique stepwise refinement
- C. stating his problem in precise terms
- D. organizing and relating his data

5. When using a systematic problem-solving strategy, the FIRST thing to do is obtain
- A. a knowledge of problem-solving techniques.
 - B. experience in problem solving.
 - C. a complete algorithm.
 - D. a complete understanding of the problem.
6. Which of the following is used to organize in a logical sequence the steps of a solution to a problem?
- A. a systematic strategy
 - B. stepwise refinement
 - C. algebra
 - D. an algorithm
7. John is given the following problem.
- The combined ages of a husband and wife are 96. He is twice as old as she is. What are the ages of the husband and wife?
- As part of his problem-solving strategy, he writes this:
- $$X + Y = 96 \quad \text{and} \quad X = 2Y$$
- Which step in the plan is he working on?
- A. making explicit the assumptions made in the problem statement
 - B. revising his solution
 - C. writing down the available data
 - D. relating the given information
8. For which of the following purposes would you use the technique "stepwise refinement"?
- A. to relate parts of the problem to the whole
 - B. to break the problem down into smaller parts
 - C. to revise each step
 - D. to debug the solution
9. When using a systematic problem-solving strategy, a necessary FINAL activity is
- A. to write your algorithm.
 - B. to check your results.
 - C. to run your program.
 - D. to obtain your solution.

10. A sales manager sends a note to the computing center requesting the sales figures for the area. One week later he receives 25 pages of printout listing sales figures for each salesperson working in the province. The manager is furious because the data is useless to him.

Which statement BEST explains the reason for the misunderstanding?

- A. The sales manager did not send a long enough note.
- B. The sales manager did not explain exactly what he wanted.
- C. The computing center personnel did not follow instructions.
- D. To impress the manager, the computing center personnel gave more information than was necessary.

11. What is Logo?

- A. a word that can assume a given value
- B. a name for a program that allows you to draw on the screen
- C. a program to draw a turtle
- D. a means of communicating with the computer

12. Directions: Column A contains a list of uses for the terms in column B. On the line at the left of each statement, write the letter of the term in column B that best fits the use listed in A. Each response in Column B may be used once, more than once, or not at all.

Column A

Column B

- | | |
|------------------------------------|----------------------|
| () change procedures | A. a variable |
| () indicate something is wrong | B. an error message |
| () help find the turtle's heading | C. total turtle trip |
| () define procedures | theorem |
| () give clues to a bug | D. Logo editor |

13. A variable can be used
- A. to vary the name of a shape.
 - B. to edit a procedure.
 - C. to correct a mistake.
 - D. to change the size of a shape.

14. What is a procedure?

- A. a command that allows you to change the size of your drawing
- B. a series of instructions to the computer
- C. a subset of Logo
- D. the name of a program which outputs a drawing

For the next two questions use the following

```
TO 6FLAG :SIZE
REPEAT 6 [FLAGR :SIZE RT 60]
END
```

15. Which of the following indicates the variable?

- A. REPEAT
- B. RT 60
- C. FLAGR :SIZE
- D. :SIZE

16. Which one indicates a subprocedure?

- A. 6FLAG :SIZE
- B. FLAGR :SIZE RT 60
- C. FLAGR :SIZE
- D. :SIZE

17. Directions:

Column A contains a list of descriptions. On the line at the left of each one, write the letter of the Logo code in column B that best fits the description. Each response in Column B may be used once, more than once, or not at all.

Column A

- () changes the turtle's position
- () changes the turtle's heading
- () controls the turtle's pen
- () groups a list of instructions
- () defines a procedure

Column B

- A. REPEAT
- B. PU
- C. HT
- D. BK 60
- E. SAVE "SQUARE
- F. LT 60
- G. TO SQUARE

Appendix F**Posttest**

NAME:

I.D. NUMBER:

Now that you've finished the module, **INTRODUCTION TO COMPUTER PROGRAMMING**, let's see how much you've learned.

Take the following short quiz. Like the beginning quiz, the results of this one will NOT be used to grade you. The results of the two quizzes will be compared to see if the instructional materials were effective.

To get credit for completion of this module, you must have a. handed in the final Logo programming project
and
b. completed this quiz.

Indicate if any questions are confusing or not relevant to the materials.

HAND THIS IN AT THE RESERVE BOOTH WHEN YOU HAVE FINISHED.

CIRCLE THE CORRECT ANSWER. IF YOU DON'T KNOW THE ANSWER, LEAVE IT BLANK AND GO ON TO THE NEXT QUESTION. DO NOT GUESS AND DO NOT LOOK UP AN ANSWER IN THE MODULE--THAT'S CHEATING!!

1. When using a systematic problem-solving strategy, a necessary FINAL activity is
 - A. to write your algorithm.
 - B. to check your results.
 - C. to run your program.
 - D. to obtain your solution.
2. An algorithm is an important part of programming because it puts the steps of a solution into
 - A. a code the computer can understand.
 - B. a sequence the computer can understand.
 - C. a language the computer can understand.
 - D. an instruction the computer can understand.
3. To solve a problem means
 - A. to reach a clearly defined goal.
 - B. to organize given information.
 - C. to understand the difficulty to be overcome.
 - D. to obtain missing data.
4. John is given the following problem.

The combined ages of a husband and wife are 96. He is twice as old as she is. What are the ages of the husband and wife?

As part of his problem-solving strategy, he writes this:

$$X + Y = 96 \quad \text{and} \quad X = 2Y$$

Which step in the plan is he working on?

- A. making explicit the assumptions made in the problem statement
- B. revising his solution
- C. writing down the available data
- D. relating the given information

5. Which of the following is characteristic of a systematic problem-solving strategy?
- A. the use of intelligent guessing
 - B. the use of pictures or graphs
 - C. the use of a step-by-step plan
 - D. the use of the technique stepwise refinement
6. For which of the following purposes would you use the technique "stepwise refinement"?
- A. to relate parts of the problem to the whole
 - B. to break the problem down into smaller parts
 - C. to revise each step
 - D. to debug the solution
7. When using a systematic problem-solving strategy, the FIRST thing to do is obtain
- A. a knowledge of problem-solving techniques.
 - B. experience in problem solving.
 - C. a complete algorithm.
 - D. a complete understanding of the problem.
8. Joan wants to program her computer to play a game called RACE. The objective is to have the player race a car at varying speeds through a maze without crashing it. She begins by writing a procedure to draw a maze. Her next procedure draws the car in the startup position. What is she doing?
- A. stating her problem in precise terms
 - B. organizing and relating her data
 - C. revising her program
 - D. using the technique stepwise refinement
9. Bob is having trouble with his program. Can you help him debug it? You are being asked to help Bob
- A. find the missing procedure in his program.
 - B. write his program.
 - C. enter his program correctly into the computer.
 - D. find the reason why his program won't run.

10. A publisher asks a graphic artist to design the cover of a book. In two weeks, she gets a bill for \$1000 and an elaborate computer-designed graphic. She is shocked because she was expecting only a simple drawing.

Which statement BEST explains the reason for the misunderstanding?

- A. The graphic artist assumed he could do anything he wanted.
- B. The graphic artist knew exactly what he wanted to do.
- C. The publisher did not know exactly what she wanted.
- D. The publisher's instructions were too vague.

11. Directions: Column A contains a list of descriptions. On the line at the left of each one, write the letter of the Logo code in column B that best fits the description. Each response in column B may be used once, more than once, or not at all.

Column A

- () changes the turtle's position
- () changes the turtle's heading
- () controls the turtle's pen
- () groups a list of instructions
- () defines a procedure

Column B

- A. PD
- B. FD 50
- C. REPEAT
- D. ERASE "TRI
- E. ST
- F. TO TRI
- G. RT 50

12. What is Logo?

- A. a word that can assume a given value
- B. a name for a program that allows you to draw on the screen
- C. a program to draw a turtle
- D. a means of communicating with the computer

13. Directions: Column A contains a list of uses for the terms in column B. On the line at the left of each statement, write the letter of the term in column B that best fits the use listed in A. Each response in Column B may be used once, more than once, or not at all.

Column A

Column B

- | | |
|------------------------------------|----------------------|
| () change procedures | A. a variable |
| () indicate something is wrong | B. an error message |
| () help find the turtle's heading | C. total turtle trip |
| () define procedures | theorem |
| () give clues to a bug | D. Logo editor |

14. A variable can be used
- A. to vary the name of a shape.
 - B. to edit a procedure.
 - C. to correct a mistake.
 - D. to change the size of a shape.

Use the following to answer the next two questions.

```
TO 6FLAG :SIZE
  REPEAT 6 [FLAGR :SIZE RT 60]
END
```

15. Which one indicates the variable?
- A. REPEAT B. RT 60 C. FLAGR :SIZE D. :SIZE
16. Which one indicates a subprocedure?
- A. 6FLAG :SIZE B. FLAGR :SIZE RT 60
C. FLAGR :SIZE D. :SIZE
17. What is a procedure?
- A. a command that allows you to change the size of your drawing
 - B. a series of instructions to the computer
 - C. a subset of Logo
 - D. the name of a program which outputs a drawing

Appendix G

Personal Information Form

AGE: _____ SEX: Male _____ Female _____

MOTHER TONGUE: _____

PRESENT COURSE OF STUDY: Year _____ Degree _____
Major _____In your last 2 years of study, how many math and/or science
courses have you taken? _____

Have you ever had formal instruction in problem solving?

Yes _____ No _____

Have you had any previous experience in writing computer
programs? Yes _____ No _____If yes, specify what language was used and in what
context (for a course, for work, for fun, etc.).

Appendix H
Observation Sheet

DATE:

MODULE

SECTION

PAGE

QUESTION/COMMENT

Appendix I

Evaluation Questionnaire:

Unit 1--Problem Solving

Kindly take the time to carefully fill out this questionnaire. The information is needed to improve the materials. We are especially interested in any problems you had.

Here's your chance to let us know exactly what you think!

CIRCLE YOUR RESPONSE TO EACH STATEMENT. CONSULT THE MATERIALS IN THE UNIT WHEN NECESSARY. WRITE ON THE BACK OF THE PAGE IF MORE SPACE IS NEEDED.

 *
 * FOR ALL YOUR COMMENTS, CLEARLY SPECIFY *
 *
 * THE EXACT SECTION AND PAGE NUMBER. *
 *

(Note. n = 37. The percentage of "No Answer" for a particular question is not reported.)

1. a. Did you find the material in this unit

Too Easy
5%

Average
89%

Too Difficult
0%

b. Where was it too easy or too difficult?

only 2 negative comments (we already know this)
yet felt a worthwhile unit

2. a. Was the information clear or confusing?

Clear	Alright	Confusing
73%	10%	0%

- b. Where was it confusing? Which section(s) did you have a hard time understanding?

only positive comments about the clarity of the information

3. a. Did you find this unit

Interesting	Alright	Boring
68%	30%	3%

- b. Which section(s) were boring and why?

only one person replied: boring because of a strong math background

- c. Which section(s) did you like the best?

78% replied - all positive comments. The actual solving of problems and the systematic problem-solving strategy got highest comments, algorithms next.

4. a. How did you find this unit?

Helpful	OK	Unnecessary
70%	19%	3%

- b. Which section(s) were unnecessary and why?

only negative comment from student who had taken a lot of math. 14% of respondents wrote nothing was unnecessary.

- c. Which section(s) were the most helpful?

the systematic problem-solving strategy and its component steps mentioned most often, algorithms next.

5. a. Do you feel that this unit should be part of a computer literacy course?

Yes	No
97%	3%

- b. If no, please explain why not?

no replies

6. a. How were the practice exercises?

Too Few
22%About Right
59%Too Many
0%Useful
65%OK
22%A Waste of Time
3%

b. Where were they too few or too many?

too few: more actual problems to solve and more
practice with algorithms

c. Which ones were a waste of time?

16% replied. Of these half were positive (none
were a waste of time)

7. a. Did you receive sufficient feedback on your practice exercises?

Too Little
5%About Right
95%Too Much
0%

b. Where was the feedback too little or too much?

too little: algorithms

8. a. This unit is intended to be self-instructional. Did you have to ask for help from the T.A. or other students?

No, not once
95%Two or Three
Times
3%Often
0%

b. Specify why you asked for help?

algorithms

9. Did you find the pictures

Interesting
43%Alright
54%Distracting
0%

10. The unit uses only print material. Would you prefer another medium such as film, computers, etc.? Explain your choice. (n = 31/37)

61.0% print adequate, sufficient

6.5% print plus film as reinforcement

12.9% print plus computer to practice solving problems

12.9% film

6.5% computers

Any other comments?

19% replied: no pattern to responses, repeated
what was said above

THANK YOU FOR YOUR COOPERATION

PLEASE RETURN THIS TO THE RESERVE BOOTH

Appendix J

Evaluation Questionnaire:Unit 2Programming with Logo

NAME:

DATE:

I.D.:

Kindly take the time to fill out this questionnaire. Your comments are important. We need to know if there are any bugs (to use computer jargon) in the materials. Filling this out carefully and in detail will allow us to improve the materials.

Thank you for your cooperation,
the course designers

CIRCLE YOUR ANSWER TO EACH STATEMENT. CONSULT THE MATERIALS
IN THE UNIT WHENEVER NECESSARY. WRITE ON THE BACK OF THE
PAGE IF MORE SPACE IS NEEDED.

*
* FOR ALL YOUR COMMENTS, CLEARLY SPECIFY *
*
* THE EXACT SECTION AND PAGE NUMBER. *
*

(Note. n = 27. The percentage of "No Answer" to a
particular question is not reported.)

1. a. Approximately how long did it take you to go
through Unit 2? _____

1 - 4 hrs	30%
4 1/2 - 8 hrs	41%
more than 8 hrs	26%

- b. Did you complete the unit in one session at the
computer? Yes No
30% 70%

2. a. How did you find the final project? Circle one
from EACH set of adjectives.

i) Interesting	OK	Boring
78%	15%	0%

ii) Easy	Alright	Difficult
0%	44%	11%

- b. How much time did it take you to complete the
project?

less than 1 hr	19%
1 to 2 hrs	30%
2 1/2 - 3 hrs	15%
more than 3 hrs	19%

- c. To do your final project, did you use the systematic problem-solving strategy outlined in Unit 1?

Yes
81%

No
7%

- d. If no, explain why not.

Both respondents commented that they felt no need to use it

3. a. Was it a useful exercise to write a brief description of the problem-solving strategy you used to do your final project?

Yes
59%

No
30%

- b. What did you think about this exercise?

85% replied: it was clear that this question was not understood by all; some thought "this exercise" meant "the final project".

4. a. Did you find the material in this unit

Too Easy
4%

Average
93%

Too Difficult
4%

- b. Where was it too easy or too difficult?

only 6/27 replied: difficulty with defining procedures, subprocedures, and the FACE example.

5. a. Was the information clear or confusing?

Clear
52%

Alright
9%

Confusing
15%

- b. Which section(s) did you have a hard time understanding?

44% replied. Of these half were positive (nowhere was it difficult), the majority of the remaining replies indicated the FACE example.

6. a. Did you find this unit

Interesting
89%

Alright
11%

Boring
0%

- b. Which section(s) were boring and why?

22% replied. 2 commented that the explanations of the problem solving became tedious.

- c. Which section(s) did you like the best?

85% replied. Majority wrote the final project (actually doing it on their own) and drawing shapes.

7. How did you find the game SHOOT? Circle one from EACH pair of adjectives.

i) helpful	63%	unnecessary	15%
ii) fun	63%	boring	2%
iii) easy	44%	difficult	0%

8. a. How were the practice exercises? Circle one from EACH set of adjectives.

i) Too Few	About Right	Too Many
26%	63%	0%
ii) Useful	OK	A Waste of Time
44%	41%	0%

- b. Where were they too few or too many?

52% replied. Of these, almost half were positive (just the right amount); the others indicated too few in the subprocedures section.

- c. Which ones were a waste of time?

Of the 15% who replied, all were positive (none were a waste of time).

9. a. Did you receive sufficient feedback on your practice exercises?

Too Little	About Right	Too Much
4%	89%	0%

- b. Where was the feedback too little or too much?

The 2 replies were not specific enough.

10. How did you find the review/summary sections throughout the unit?

Helpful
67%

OK
33%

Unnecessary
0%

11. a. Did you use the IBM tutorial Logo: Programming with Turtle Graphics?

Yes
30%

No
70%

- b. If yes, why did you?

For extra help
15%

For enjoyment
4%

Both
7%

12. a. Do you feel that this unit should be part of a computer literacy course?

Yes
93%

Don't Care
0%

No
7%

- b. If no, please explain why not?

One found Logo boring; the other found Logo commands too specific to be transferred to other languages.

13. a. This unit is intended to be self-instructional. Did you have to ask for help from the T.A. or other students?

No, not once
30%

Two or Three Times
59%

Often
11%

- b. Specify why you asked for help?

Majority mentioned file management; the others how to get out of < prompt and subprocedures.

14. a. Was it easy to get all the necessary materials from the reserve booth?

Yes
96%

No
4%

- b. Indicate any problems you had.

The one reply didn't like waiting because nobody was there.

Any other comments?

11: replied - all positive. Unit was fun and interesting

PLEASE RETURN THIS TO THE RESERVE BOOTH

Appendix K

Item Response Patterns for Those Subordinate ObjectivesNOT Performing as Expected

Pretest							Posttest						
Obj	Item No.	% Pass	A	B	C	D NA	Same Item	Item No.	% Pass	A	B	C	D NA
Unit 1													
1	1	44%	7	9	5	<u>20</u> 5	No	3	72%	<u>33</u>	4	4	5 0
	2	96%	0	0	<u>44</u>	0 2	No	5	85%	0	3	<u>39</u>	2 2
2	10	76%	0	<u>35</u>	4	0 7	No	10	91%	1	0	1 <u>42</u>	2
3	5	63%	6	0	3	<u>29</u> 8	Yes	7	94%	1	0	1 <u>43</u>	1
	9	33%	0	<u>15</u>	9	5 17	Yes	1	57%	4	<u>26</u>	11	3 2
	7	26%	12	0	17	<u>12</u> 5	Yes	4	33%	19	0	11 <u>15</u>	1
4	4	15%	1	<u>7</u>	3	14 21	No	8	48%	4	10	3 <u>22</u>	7
	8	17%	0	<u>8</u>	10	0 28	Yes	6	63%	1	<u>29</u>	5	8 3
5	6	9%	22	1	2	<u>4</u> 17	No	2	59%	2	<u>27</u>	7	8 2
6	3	65%	<u>30</u>	1	4	6 5	No	9	91%	4	0	0 <u>42</u>	0
Unit 2													
7	11	26%	4	8	1	<u>12</u> 21	Yes	12	52%	1	21	0 <u>24</u>	0
	14	59%	0	<u>27</u>	0	0 19	Yes	17	89%	1	<u>41</u>	0	3 1

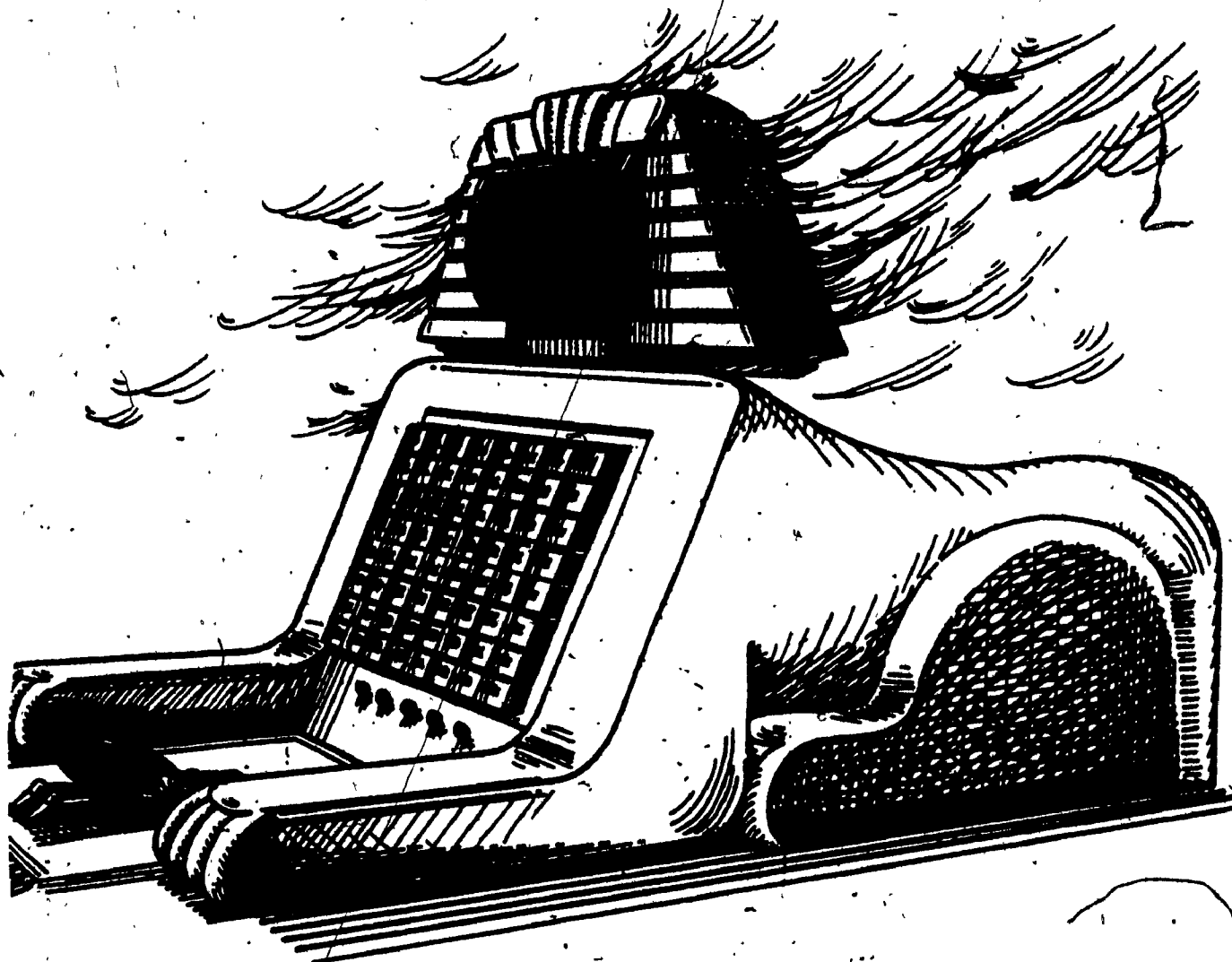
Note. The correct answer is underlined. Appendix B lists the content of the objectives. N = 46.

Appendix L

AN INTRODUCTION TO COMPUTER PROGRAMMING

Unit 1

Problem Solving



An Introduction to Computer Programming

Up to now you have been a relatively passive user of pre-packaged software -- software that has been written by others. Now it is time to see what you can do as a programmer.

This module won't make you an expert problem solver nor a first-class programmer. That takes a lot of time and practice. It is an introduction, a taste of what it is like to be in control!

This module is divided into two units. Unit 1 deals with problem-solving techniques useful for programming. Unit 2 introduces you to the computer language Logo. At the end of the module you will be required to write a program in Logo to solve a given problem.

What is Computer Programming

When you program a computer, you do three things.

1. You decide what problem the computer is to handle, then you find its solution.
2. You translate this solution into a form and a language the computer understands. The result is your program.
3. You work at the computer to enter and correct the program so that it runs without errors.

You will have plenty of opportunity to practice these three steps as you work through the two units of this module.

```

*****
*                                     *
*      Computer                      *
*                                     *
*      Programming                   *
*                                     *
*****

```

```

*****      *****      *****
*          *          *          *
*  find a  *  put the *  enter and *
*  solution *  into a *  correct  *
*  to the  *  the computer *  the program *
*  problem *  understands *          *
*          *          *          *
*****      *****      *****

```

Table of Contents

1. Introduction
 - 1.1 Why Study Problem Solving?
 - 1.2 Programming and Problem Solving
2. Do You Have a Problem?
3. Problem-Solving Strategies
 - 3.1 What is a Systematic Problem-Solving Strategy?
 - 3.2 Two Examples of Problem-Solving Strategies
4. A Systematic Problem-Solving Strategy To Use
 - 4.1 Understand the Problem
 - a) Be Precise: Know What is Wanted
 - b) Breaking a Problem Down -- Stepwise Refinement
 - 4.2 Relate the Data
 - 4.3 Find a Solution
 - a) Algorithms
 - 4.4 Check Your Results
 - a) What is a Bug?
5. A Final Word on Problem Solving
 - 5.1 Recommended Reading
6. Summary

Unit 1

Problem Solving

1. Introduction

1.1 Why Study Problem Solving?

As a university student you have obviously had some success in problem solving. How else would you have gotten this far in your academic career? Yet how much of your problem solving is conscious?

** Could you describe to someone how you attack a problem?

** Could you describe what techniques you used to get the answer?

** Could you write clearly and precisely each step in the solution so that someone else could solve the same problem just by reading your solution statement?

You may indeed find that this unit will merely verbalize what you already know. That's great, for by being aware of your problem-solving processes, you can more consciously apply them when it comes to writing computer programs.

Problem solving is a skill. It can be learned just as one learns how to swim or to play tennis. Techniques and strategies exist to help you improve your problem-solving ability. This unit introduces you to those that are particularly appropriate to Computing.

```

*****
*
*   Problem solving is a skill   *
*
*   that can be learned        *
*
*   and improved                *
*
*****

```

1.2 Programming and Problem Solving

You may be asking yourself, "What does problem solving have to do with programming?"

```

*****
*
*   Programming is a problem-solving activity *
*
*****

```

How do you get the computer to do what you want it to do? You want the computer to calculate your monthly interest payments, or express in graphic form the data you collected on your last field trip. In order for the computer to do this, you must write the program; that is, YOU must first solve the problem and then tell the computer exactly HOW to do it. Remember the computer is a tool. It will do only what you tell it to do.

```

*****
*
*   A program is the solution to a problem
*
*   written in a language
*
*   the computer understands
*
*****

```

Programming demands the same type of mental activity and mental discipline that is involved in solving such problems as:

A man, a fox, a goose and some corn are on one side of a river with a boat. The boat will carry the man and one other thing. The fox and the goose cannot be left together, as the fox would eat the goose. Similarly, the goose and the corn cannot be left together. How does the man get them all safely across the river?

or

You are facing east, you make an about-face, and then you turn left. Which direction is now on your left side?

Sound familiar? These types of problems are found on such standardized tests as the SAT (Scholastic Aptitude Test), the GRE (Graduate Records Exam), and the LSAT (Law School Admissions Test) to name a few. These problems are intended to test analytical reasoning, your ability to analyze a problem in order to find the solution.

2. Do You Have A Problem?

Before we begin our discussion of problem solving let's be sure we are talking about the same thing. Take a few minutes to reflect upon what exactly is a problem and what is problem solving. Jot down your ideas on a piece of paper. Be sure to give a definition not just examples. Turn the page when you are ready to continue.



Feedback

Compare your definitions with the ones given below by George Polya. An eminent mathematician, he was the first to write about problem-solving strategies and techniques.

"Getting food is usually no problem in modern life. If I get hungry at home, I grab something in the refrigerator, and I go to a coffeeshop or some other shop if I am in town. It is a different matter, however, when the refrigerator is empty or I happen to be in town without money; in such a case, getting food becomes a problem. In general, a desire may or may not lead to a problem. If the desire brings to my mind immediately, without any difficulty, some obvious action that is likely to attain the desired object, there is no problem. If, however, no such action occurs to me, there is a problem. Thus, to have a problem means: to search consciously for some action appropriate to attain a clearly conceived, but not immediately attainable, aim. To solve a problem means to find such action.

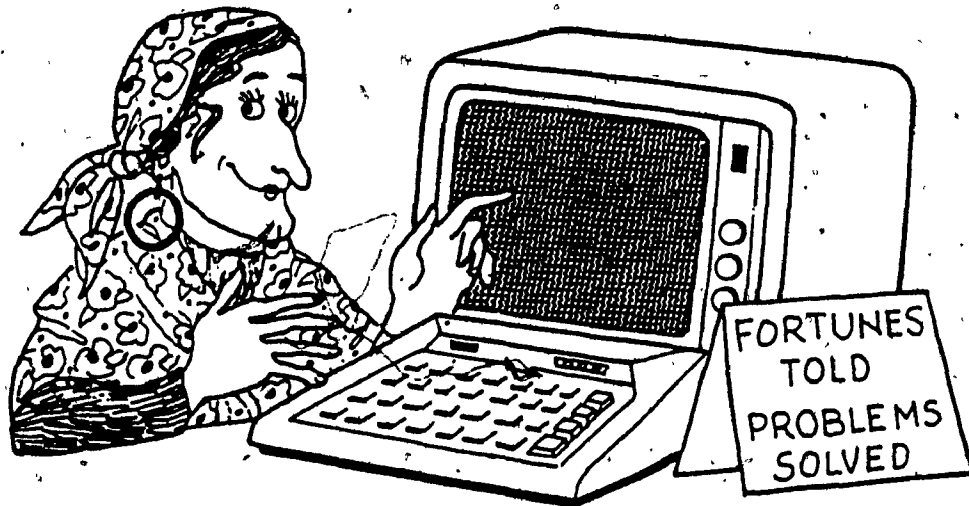
A problem is a great problem if it is very difficult, it is just a little problem if it is just a little difficult. Yet some degree of difficulty belongs to the very notion of a problem: where there is no difficulty, there is no problem."¹

¹ Polya, George (1962). Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving. (vol. 1) p. 117.

Problems are not just of the type $2 + 2 = ?$. Having a problem implies not being able to immediately achieve a desired goal. Some degree of difficulty is involved. To solve a problem is to find the way to achieve the goal.

```
*****
*                                     *
*   To solve a problem is to find   *
*                                     *
*   a way to get what you want     *
*                                     *
*****
```

Let's turn now to a look at problem-solving techniques that will be useful to you when you begin to program in the next unit.



3. Problem-Solving Strategies

How do YOU approach a problem?

- ** Do you attack it with a certain plan in mind: first I'll do this, then that?
- ** Do you jump right in and take a guess?
- ** Do you write down your first answer?
- ** Do you redo the problem to be sure you haven't overlooked anything?

Here's a problem to solve. The main objective is not to get the right answer but, to think about HOW you go about finding the answer. It may help you to think outloud or to imagine that you have to tell someone else HOW to solve this problem. After you arrive at a solution, try to answer the above questions. Turn the page when you have finished.

There are 3 separate, equal-sized boxes, and inside each box there are 2 separate small boxes, and inside each of the small boxes there are 4 even smaller boxes. How many boxes are there altogether?

- a. 24 b. 13 c. 21 d. 33
- e. some other number

Feedback

Getting the right answer is not as important as the procedure you used to get the answer. (The correct answer is 33.) Were you able to identify a particular type of approach or problem-solving strategy that you used?

3.1 What is a Systematic Problem-Solving Strategy?

Good problem solvers have a plan of attack. They break the problem-solving process down into a series of steps. Each step leads them closer to a solution. They use a SYSTEMATIC problem-solving strategy.

A systematic problem-solving strategy is one that follows a method or orderly procedure. You solve the problem by following a step-by-step plan.

```

*****
*                                     *
*           When you use a systematic problem-           *
*                                     *
*           solving strategy, you follow a                 *
*                                     *
*           step-by-step plan to get                       *
*                                     *
*           an answer.                                     *
*                                     *
*****

```

There is no ONE plan or strategy that can be used to solve all problems. Various problem-solving strategies have been proposed. Following these plans will help you arrive at a solution. Notice I said help, NOT give you an answer. In effect the plans are simply a series of commonsense guidelines that have been shown to be useful in solving problems. Following them never guarantees an answer, but it may bring you closer to one.

Practice

Two suggested problem-solving strategies are presented below. Read each carefully and answer the following questions.

1. Could both be described as a systematic approach to problem solving? If yes, in what way are they systematic?
2. Identify at least two essential components or steps that are present in both of the methods.

3.2 Two Examples of Problem-Solving Strategies

A. A Prescription for Problem Solving

1. Read and reread the problem carefully.
2. State the goal in your own terms.
3. List in short, declarative sentences the facts that are (explicitly or implicitly) given.
4. Try to make a picture (table, graph, diagram) that represents the known facts and relationships.
5. Try to infer some additional facts or relationships from those that are given; add to the list and incorporate it in the picture.
6. Determine what additional information would be sufficient to reach a solution; see if that information can be inferred.
7. Try to infer something about the solution (for example, it must be positive; it must be less than X; it cannot be Y).
8. If it is a numerical problem, try extreme cases (for example, solve for 0, 1).
9. If you're stuck, try to find another way to think about the problem; generalize it; particularize it.
10. If you're still stuck, try to solve a similar but simpler problem.
11. If you're still stuck, do something else for awhile.
12. Check your work.²

²Nickerson, R. (1981). Thoughts On Teaching Thinking. Educational Leadership, 39 (1). p. 23

B. How to Solve Math Problems

FIRST

You have to understand the problem. Is it possible to solve the problem with the given information?

SECOND

Find the connection between the data and the unknown. You may be obliged to consider auxiliary problems if an immediate connection cannot be found. You should obtain eventually a plan of the solution.

Have you seen the problem before?

Do you know a related problem?

Look at the unknown! And try to think of a familiar problem having the same or a similar unknown.

THIRD

Carry out your plan. Check each step.
Can you see clearly that the step is correct?

FOURTH

Examine the solution obtained.
Can you check the result?
Can you derive the result differently?
Can you use the result, or the method, for some other problem?

³polya, George (1945). How To Solve It. p. xvi.

Feedback

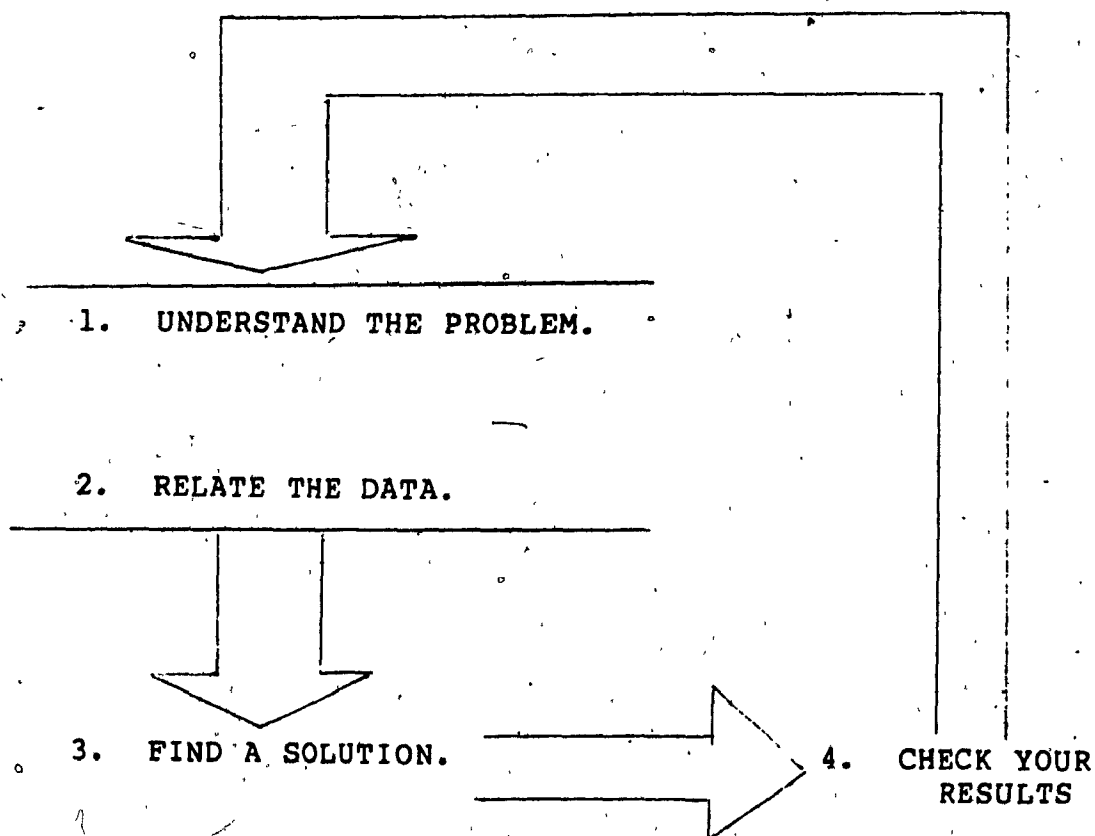
1. Both are examples of a systematic approach in that they suggest a step-by-step procedure to use when solving a problem. Following the given plan will bring you closer to a solution than just guessing or using no plan at all.

2. Both suggest as a first step -- understand the problem, and as a last step -- check your results. A necessary intervening step is to relate the data in some way. You may have picked out other steps as well.

*
* Be systematic: work from *
* *
* a plan *
* *

4. A Systematic Problem-Solving Strategy To Use

Below are the essential steps in any systematic problem-solving strategy. You will use this strategy repeatedly as you program in Unit 2.



4.1 Understand The Problem

1. A farmer has some chickens and some sheep. Altogether, a count shows a total of 100 heads and 280 feet. How many chickens and sheep are there?

To solve this problem, you need to make certain assumptions?

What are they?

2. Mr. Smith brings the family car to his local garage and asks the mechanic to service it. The mechanic has been given the problem "service the car". He changes the oil. When Mr. Smith returns, he is annoyed that the mechanic didn't check the water and the battery.

Who was to blame for the confusion and why?

Turn the page when you're done.

Feedback

1. This problem seems very straightforward. Yet to arrive at a solution, you have to assume a. a chicken has 1 head and 2 feet

and

b. a sheep has 1 head and 4 feet.

2. Both parties were to blame for the misunderstanding. Mr. Smith assumed the mechanic knew what he meant by the instructions "service it". The results showed that he assumed too much. What he gave was a GENERAL problem statement. What he should have given was a PRECISE problem statement saying exactly what needed to be done - "change the oil, check the battery and the water level".

The mechanic also contributed to the confusion because he did not ask for clarification of the problem. The mechanic assumed he knew what was wanted. He should have checked that his understanding of the problem was the same as Mr. Smith's.

```

*****
*
*          UNDERSTAND THE PROBLEM          *
*
*          make explicit any assumptions      *
*
*          Be precise: know exactly what is wanted  *
*
*****

```


4.1 a) Be Precise: Know Exactly What is Wanted

When programming the computer this attention to detail and precision is very important. The computer is a blank slate and must be told everything, right down to the last detail. You, as the programmer, must understand exactly what you want the computer to do. You should make explicit any assumptions and you should be precise. YOU must fully understand the problem you want the computer to help you solve.

Let's look at the difference between a general problem statement and a precise problem statement.

General Problem Statements

1. Convert pounds to kilograms.
2. Calculate the monthly payments on my mortgage.
3. Analyze the results of my questionnaire.
4. List those students absent for the exam.

Precise Problem Statements

1. Convert 10 pounds into kilograms.
2. Calculate the monthly payments on a 10-year mortgage of \$25,000 at an interest rate of 19% per year.
3. Give me the mean and the standard deviation for the data from my questionnaire given in May 1984 to first-year sociology students.
4. List in alphabetical order all those absent for the final exam in Economics 250.

4.1 b) Breaking a Problem Down -- Stepwise Refinement

We have seen that a good way to approach a problem is to have a plan, to be systematic. This involves breaking the problem-solving process into a series of steps or guidelines to be followed. The same technique can be used to break the problem itself into smaller, simpler parts.

```

*****
*
*           Stepwise Refinement:  the process
*
*           of breaking a complex problem
*
*           down into a series of
*
*           smaller problems
*
*****

```

"This is one of the most important problem-solving techniques in Computing. In fact all problems are broken down by stepwise refinement to a level where the solution is known--a single computer instruction.... Let us take a practical example.

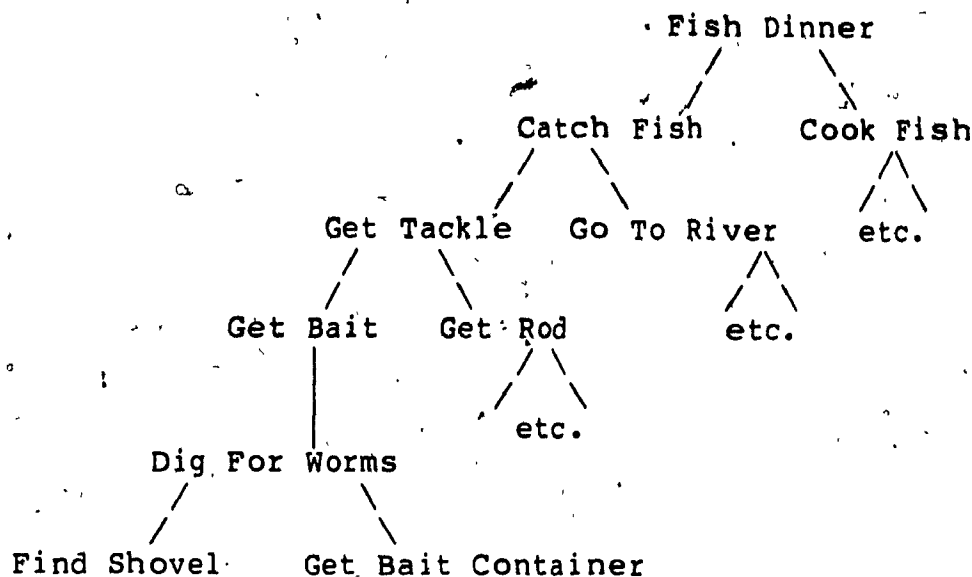
Suppose you are camping out near a lake for a few days and decide you want fish for dinner. You have no fish, and so your first problem is to catch one. Also, before you eat the fish, you have to cook it...already our initial problem has become two sub-problems: to catch the fish, and to cook it.

We will take one of these problems and examine it in more detail.

In order to catch the fish, we need to get our fishing tackle together and go to the river. Again, we have further problems that need solution.

Stepwise refinement of the problem of getting the tackle divides that problem into two: one to find some bait, the other to get the fishing rod.

The process continues. It can be partly illustrated by the diagram below.



If we follow one arm of the refined problem, we come to the basic problems of getting the shovel and a bait container. If we work back up through the chain of problems and all other sub-problems, we shall solve our problem of having a fish dinner. If we fail in any one of the sub-problems, the total problem cannot be solved."⁴

Practice

How would you break up or refine the problem of drawing a face?

⁴Gledhill, V.X. (1981). Discovering Computers. p. 32.

4.2 Relate the Data

Once the problem is clear in your mind, the next step, is to list what data is given in the problem and to specify how the data is related or organized. This is often made easier by using graphs, drawings or symbols.

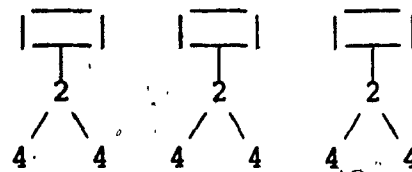
Let's return to the problem of the 3 boxes.

There are 3 separate, equal-sized boxes, and inside each box there are 2 separate small boxes, and inside each of the small boxes there are 4 even smaller boxes. How many boxes are there altogether?

- a. 24 b. 13 c. 21 d. 33
e. some other number

The first step is to find out what is wanted: the total number of boxes. Next, itemise and organize the data to make clear the relationships. One way is to draw a picture.

1. There are 3 separate boxes.
2. Inside each there are 2 boxes
3. Inside each there are 4 boxes



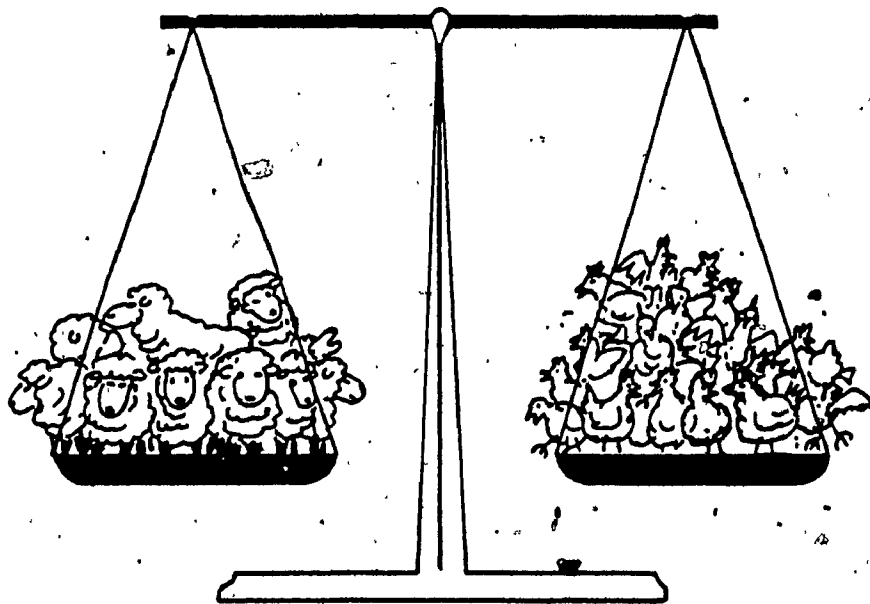
If we count all the numbers and each big box we get 33.

*
* Use drawings, symbols, graphs *
* to organize and relate *
* the data. *
*

Practice

Look at the next two problems. First, state in your own words what you want to know. Next, organize the data in whatever way helps you clarify the relationships.

1. Tom is heavier than Fred but lighter than Marty. If from this information you can determine which of the 3 men is the heaviest, write his name. Otherwise write indeterminate.
2. A farmer has some chickens and some sheep. Altogether, a count shows a total of 100 heads and 280 feet. How many chickens and sheep are there?



Feedback

1. We want to know who is the heaviest; that is, who weighs the most. We have four choices: Tom, Fred, Marty, indeterminate.

Tom is heavier than Fred

(heavy)

—	Tom
—	Fred

(light)

but lighter than Marty

(heavy)	—	Marty
	—	Tom
(light)	—	Fred

The answer becomes clear from the picture. Marty is the heaviest.

2. What we want to know is pretty clear. But remember the assumptions you have to make. The relationships expressed in this problem statement can best be shown by using algebra.

Let the number of chickens be X .

Let the number of sheep be Y .

There are 100 heads. $X + Y = 100$.

There are 280 feet. $2X + 4Y = 280$.

To solve the problem solve for X and Y . Hopefully you remember a little algebra. $X = 100 - Y$

Substitute this in the second equation and divide by 2:

$$100 - Y + 2Y = 140$$

$$Y = 40$$

$$X = 100 - 40 = 60$$

There are 60 chickens and 40 sheep.

4.3 Find a Solution

A program has been defined as a solution to a problem. If you want the computer to draw a square, you have to tell it HOW to do it one step at a time.

The important thing to remember is that the computer is a SEQUENTIAL machine. It takes one instruction at a time in an order, or sequence, specified by the programmer.

```

*****
*                                     *
*      The computer is a           *
*      sequential machine          *
*                                     *
*****

```

4.3 a) Algorithms

An algorithm is a procedure used to organize the steps of a solution to a problem. These steps are related sequentially; that is, the steps must follow each other in the order specified in the algorithm. Following an algorithm exactly as it is written will ALWAYS give you the answer.

Writing an algorithm forces you to think about HOW you solve the problem. You must specify each step.

** No steps must be missing

** The steps must be in the right order
or sequence.

You know your algorithm is correct, if the end result is what you want.

As a programmer, you must first find the answer, know each step in the solution, and organize these steps sequentially BEFORE you begin to write the program.

```
*****
*
*   An Algorithm:  a procedure for organizing
*
*   sequentially the steps in a solution
*
*****
```

An Example

Here's a problem statement: draw a rectangle.

1. ** What do you want? a rectangle

** Make explicit any assumptions: a rectangle has four 90 degree angles and the length and width are different.

** Be precise: how big a rectangle do you want?

It's not specified, so you can choose any number -- length 25 cm and width 10 cm.

Now we know exactly what we want!

2. ** Relate the data: to help us clarify the relationship of the data, let's draw a picture.

10 cm

25 cm

90 degrees



3. ** Find the solution. Since we have to tell the computer HOW to draw the rectangle, let's write the algorithm. Pretend that you are telling a friend how to do it.

```
Put your pen on the paper
Go forward 10 cm
Turn right 90 degrees
Go forward 25 cm
Turn right 90 degrees
Go forward 10 cm
Turn right 90 degrees
Go forward 25 cm
Take your pen off the paper
```

Anyone following this algorithm would be able to produce a rectangle with sides of 10 cm and 25 cm.

Compare it with this one:

```
Put your pen on the paper
Go forward 10 cm
Turn right 90 degrees
Go forward 25 cm
Take your pen off the paper
```

Would the above algorithm produce a rectangle? No. It is incomplete. There are steps missing.

What about this one?

```
Put your pen on the paper
Go forward 10 cm
Turn right 90 degrees
Take your pen off the paper
Go forward 25 cm
Turn right 90 degrees
Go forward 10 cm
Turn right 90 degrees
Go forward 25 cm
```

This one is incorrect because the steps are not in the right sequence.

More than one algorithm could be written to draw the same shape. You could turn left instead of right or go forward 25 cm first and then 10 cm. There is often more than one way to solve a problem. The important thing about an algorithm is that no matter the solution path, the output or end result is always the same.

Practice


Below are two algorithms. Follow through each one. What is the output? What is the general problem statement?

A

Pen down
Turn left 90 degrees
Go forward 50 cm
Turn left 90 degrees
Go forward 50 cm
Turn left 90 degrees
Go forward 50 cm
Turn left 90 degrees
Go forward 50 cm
Pen up

B

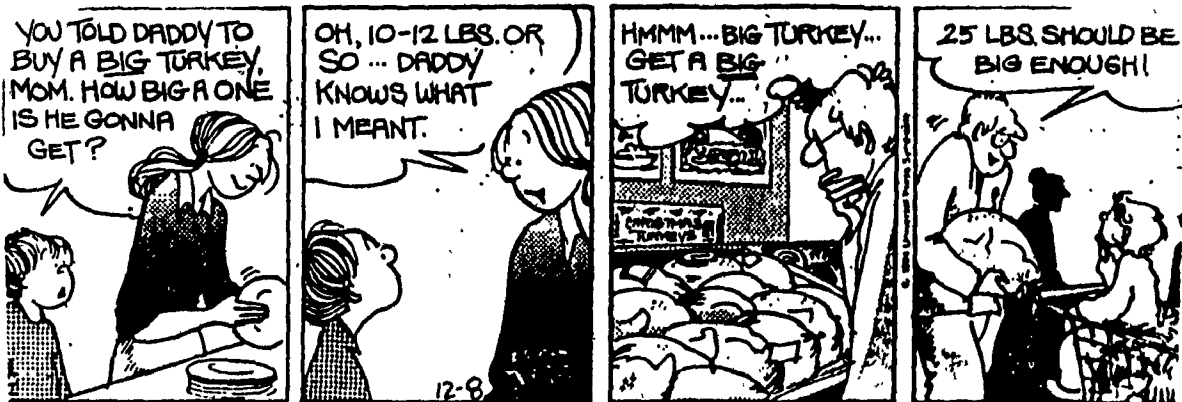
Pen down
Go forward 300 cm
Turn right 90 degrees
Go forward 150 cm
Go back 300 cm
Pen up



For the answers see the next page. You'll get plenty more practice using this type of algorithm in the next unit where you will actually program the computer to draw these shapes.

Feedback

- A. The output is a square with 50 cm sides. The problem statement is "draw a square".
- B. The output is a T. The problem statement is "draw the letter T" or "draw the letter coming after S in the alphabet".



4.4 Check Your Results

Let's review our systematic problem-solving strategy.

1. Understand the problem fully
2. Relate the data
3. Find a solution
4. Check your results

We've discussed steps 1 to 3 of this plan. Now let's look at step 4 -- check your results.

Good problem solvers will check and recheck every step of the solution to be sure that nothing has been overlooked. They rarely write down their first answer as the final one.

4.4 a) What is a Bug?

When you begin to program, a lot of time and energy can be saved by continually checking every step of the process. There must be no errors in your program if you want to get your desired output. In computer jargon, when a program does not run we say it has a **bug**.

A bug could be an error in the programmer's logic.

** are there any missing steps in the solution?

** is the sequence right?

Or it could be an error in the grammar rules of the programming language.

** did you misspell a word?

** did you forget a space?

** did you use the wrong command?

The process of finding the error is called **debugging**.
Why is it called debugging? Here is one explanation.

The Origin of "the Bug"

(somewhere around 1944)

Mark 1 was the first electromechanical computer. Many of its components were relays that moved to open and close electrical circuits. The story goes that one day the system stopped operating efficiently. The technicians looked around and found a moth caught in a relay. They removed the moth, recorded the incident in their log, and jokingly said that they were "debugging" the system. The tradition continues today; a "bug" usually means an error in human logic. Nevertheless, the moth may be seen taped to the log's record of the event at the U.S. Naval Bureau in Washington, D.C. ⁵

You will have ample opportunity to practice your debugging skills in the next unit. (We're almost there.)

```

*****
*
*      A Bug:  an error in a computer program.
*
*      Debugging:  the process of eliminating
*
*                  any program errors.
*
*****

```

⁵Horn, C. and Poirot, J. (1981). Computer Literacy: Problem Solving Computers. p. 17.

5. A Final Word on Problem Solving

You may be wondering if you are a good problem solver. The only way to find out is to solve problems. For those who want to work on their problem-solving skills, these books are recommended.

5.1 Recommended Reading

1. Polya, George (1962). Mathematical Discovery: On understanding, learning and teaching problem solving Vol 1 and 2. New York: John Wiley and Sons.

This book deals with ways to solve mathematical problems. Many examples are given.

2. Rubinstein, Moshe F. (1975). Patterns of Problem Solving. New Jersey: Prentice-Hall Inc.

This is a more general treatment of problem solving strategies and techniques.

3. Whimbey, A. and Lochhead, J. (1981). Problem Solving and Comprehension: A short course in analytical reasoning. 2nd edition. Philadelphia: the Franklin Institute Press.

This book contains over a hundred academic reasoning problems to solve. A solution analysis is given for many of them.

6. Summary

This unit deals with the relationship between problem-solving and computer programming. Computer programming is a problem-solving activity. Techniques that have proven useful in problem solving can also be applied to programming.

You should be thoroughly familiar with the following problem-solving strategy.

1. Understand the problem

- * make explicit any assumptions
- * be precise: know exactly what you want
- * use stepwise refinement to simplify a complex problem

2. Organize and relate the data

- * use pictures, graphs, symbols to show relationships

3. Find a solution

- * when programming, write an algorithm to put the steps of a solution into the correct sequence

4. Check your results

You should also be able to give a definition of the following:

- | | |
|--------------------|--------------|
| -- a problem | -- algorithm |
| -- problem solving | -- bug |
| -- systematic | -- debugging |

'PICK UP UNIT 2 OF THIS MODULE:

PROGRAMMING WITH LOGO

FROM THE RESERVE BOOTH --

Appendix M
Revised Edition of Unit 2

AN INTRODUCTION TO COMPUTER PROGRAMMING

Unit 2

Programming With Logo



Table of Contents

1. Introduction
 - 1.1 A Review of a Systematic Problem-Solving Strategy to Use
2. What is Logo?
 - 2.1 Logo Turtle Graphics
 - 2.2 A Turtle???
3. How to Load Logo
 - 3.1 Cursor Movement Keys
4. Let's Meet the Turtle
 - 4.1 Commands to Make the Turtle Appear and Disappear
 - 4.2 Commands to Control the Amount of Text on the Screen
5. The Turtle's State
 - 5.1 Commands to Change the Turtle's Position
 - 5.2 Commands to Change the Turtle's Heading
6. Error Messages
7. SHOOT
 - 7.1 How to Play SHOOT
 - 7.2 Playing SHOOT Using a Systematic Problem-Solving Strategy
8. Review of Logo Commands
9. Commands to Control the Turtle's Pen
 - 9.1 Drawing Letters
10. Drawing Shapes
 - 10.1 A Square
 - 10.2 The REPEAT Command
 - 10.3 Practice in Drawing Shapes Using a Systematic Problem-Solving Strategy
 - a) a Triangle
 - b) a Circle
11. A Quick Review
12. Procedures
 - 12.1 Defining Procedures in the Logo Editor
 - 12.2 Editing (Changing) Procedures
 - 12.3 Checking Your Work with PO and POTS
 - 12.4 Erasing Procedures
 - 12.5 The Immediate Definition Mode (or the > prompt)
 - 12.6 More Error Messages
13. Variables
14. File Management
 - 14.1 Saving Your Work
 - 14.2 The Commands DIR and ERALL
 - 14.3 Loading a File From Your Working Diskette
 - 14.4 Editing a File
15. Subprocedures
 - 15.1 Subprocedures: A Problem-solving Technique in Action
16. A Final Review
 - 16.1 Summary
17. Recommended Reading
18. Final Project

Unit 2

Programming With Logo

1. Introduction

Unit 1 of this module discussed problem-solving techniques appropriate to computer programming. Programming was defined as a problem-solving activity -- how to get the computer to do what you want it to. You saw the importance of problem presentation and definition, the need to adapt your way of thinking to the sequential, literally-minded machine. Now it's time to put those techniques into action.

Unit 2 is an introduction to Logo. You will learn a few basic commands to permit you to program in Logo. You will see what it's like to go from a problem to a program that solves it. The emphasis throughout is on applying the techniques learned in Unit 1, in developing the type of thinking needed for programming in any language.

For this unit, all your time will be spent at the computer. Make sure you have with you your own diskette.

In trouble? Refer to the table of contents for the section you need.

This unit will require a minimum of 4 hours at the computer. Don't try to do too much at one session.

Computing should be fun, not tiring!

1.1 A Review of a Systematic Problem-Solving Strategy to Use

```

*****
*                                     *
*      Computer                     *
*                                     *
*      Programming                   *
*                                     *
*****

```

```

*****      *****      *****
*      *      *      *      *
* find a  *      * put the solution *      * enter and *
* solution *      * into a language *      * correct  *
* to the  *      * the computer   *      * the program *
* problem *      * understands   *      *              *
*      *      *      *      *      *
*****      *****      *****

```

Our systematic problem-solving strategy applied to programming adds one more step.

1. Understand the problem.
 - ** be precise: know exactly what you want
 - ** make explicit any assumptions
 - ** break down a complex problem
2. Organize and relate the data by drawing a picture.
3. Write an algorithm for the solution. When you follow it through, is the end result what you want?
4. Check your work -- look for bugs.
5. Enter, run, and debug your program.

2. What is Logo?

Logo is the name of a powerful computer language. You have already learned that a computer language allows you to communicate with the computer. Like any other language, such as French or German, programming languages have their own vocabulary and grammar rules.

```
*****
*
*           Learning to program in Logo means
*
*           learning Logo's vocabulary and rules.
*
*****
```

Fortunately it is very easy to learn Logo, as we speak almost the same language.

2.1 Logo Turtle Graphics

This unit deals only with the subset of Logo called Turtle Graphics. You will learn how to make an electronic turtle draw on the screen. It is a unique, fun way to learn and consolidate basic programming techniques. Turtle graphics was designed to teach programming skills to learners with no prior mathematical knowledge.

2.2 A Turtle???

As you load Logo into your computer you may start looking for a turtle. You'll need a lot of imagination to find it. What you'll see is a small triangular pointer sitting in the middle of the screen. That's the turtle!!

Before the advent of personal computers with graphic screens, Logo outputted to a robot in the shape of a turtle. The turtle had a pen that could move up or down. When the pen was down, the robot turtle would trace intricate graphics in response to commands given to it on the terminal. See Figure 1 below.

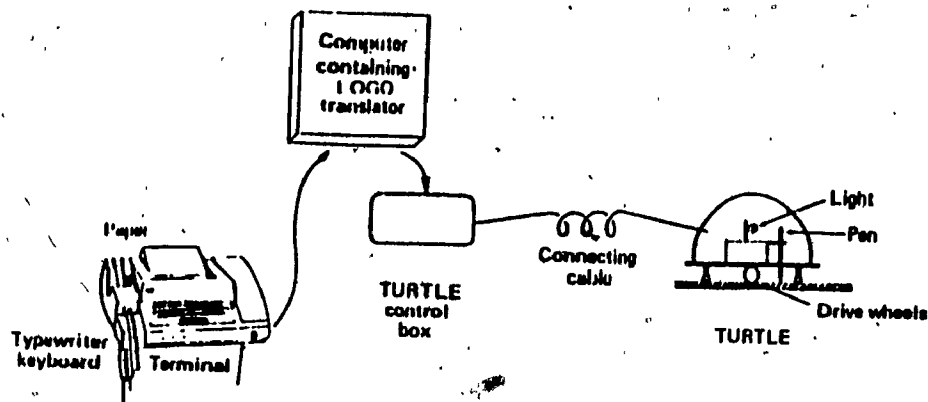


Figure 1. The original turtle.

Adapted to the microcomputer's video display terminal, the robot turtle takes the shape of a triangle. The turtle behaves in the same way. You imagine that it has a pen which leaves a trace over the path you have told the turtle to follow.

3. How to Load Logo

1. Take the diskette labelled LOGO MASTER and insert it into disk drive A. Turn on the machine.
2. After about 40 seconds, the drive A light will go on. This lets you know that Logo is being loaded into the computer's memory (RAM).
3. After a few seconds, the drive light will go off. The copyright statement and serial number will appear on the screen as well as a message saying:

WELCOME TO LOGO
?

4. Remove the LOGO MASTER diskette from disk drive A and put it back into its jacket. Insert your own working diskette in disk drive A.
5. Every time you begin a session at the computer you must repeat steps 1 through 4.

You are ready to begin to program when you see the message:

WELCOME TO LOGO
?

The question mark is the Logo prompt. It means "what do you want me to do next". Logo is waiting for a command. The flashing box is the cursor. It shows where the next character you type will appear.

3.1 Cursor Movement Keys

You should already be familiar with the keys that move the cursor. Some keys are special to Logo. Look carefully at Appendix A.

Practice

Type in this line.

MARY HAD A LITTLE LAMB (do NOT press Return)

1. Press the Del key in the lower right-hand corner.

Don't worry! The beep just means this key only works in the Logo editor. You'll learn all about the editor in a later section.

2. Erase the word "little".

3. Now press Return. You will see this:

I DON'T KNOW HOW TO MARY
?

This error message will be explained shortly.

```
*****
*                                     *
*          Forget which key to use?  *
*                                     *
*          Look at Appendix A.       *
*                                     *
*****
```


4. Let's Meet the Turtle

4.1 Commands to Make the Turtle Appear and Disappear

You may be wondering where the turtle is. It's hiding, so let's call it up. Type

SHOWTURTLE (or the short form ST)

** Be very sure of your spelling.

** Always check your work.

After each command you must also remember to press Return. This tells the computer "do it - execute my command". You can type more than one command on a line, but as a beginner it is better to press Return after EVERY command. This way you can see the immediate result so debugging is much easier.

```
*****
*                                     *
*      PRESS RETURN AFTER EVERY COMMAND      *
*                                     *
*****
```

The turtle will appear as a triangle pointing straight up in the center of the screen.

You can make the turtle invisible by typing

HIDETURTLE (or HT)

It reappears when you type SHOWTURTLE.

4.2 Commands to Control the Amount of Text on the Screen

Notice that the ? prompt and the cursor have moved to the bottom of the screen. From now on, your typed instructions will appear only on the last six lines of the screen, leaving the rest free for graphics.

Type the following commands and note what each does.

TEXTSCREEN (TS)

The whole screen is now available for text.

MIXEDSCREEN (MS)

Now you have the graphics and text back.

FULLSCREEN (FS)

The whole screen is blank ready for only graphics.

Practice

Play with these commands until you are thoroughly familiar with what they do.

SHOWTURTLE	ST	TEXTSCREEN	TS
HIDETURTLE	HT	MIXEDSCREEN	MS
		FULLSCREEN	FS

5. The Turtle's State

The important thing to remember about the turtle is that it has a specific position and a specific heading that you can control:

- * The POSITION is where the turtle is placed on the screen.
- * The HEADING is the direction in which the tip of the triangle (that is, the turtle) is pointing.
- * The position and heading together are called the turtle's STATE.

5.1 Commands to Change the Turtle's Position

FORWARD or FD BACK or BK

These commands move the turtle on the screen. Each command needs an input to tell the turtle how many steps to take, that is, how far to move. Type

FORWARD 50 or FD 50

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X                                                                X
X          Did you spell the command correctly?                X
X                                                                X
X          Did you leave a space between the two words?       X
X                                                                X
X          Did you press Return?                                X
X                                                                X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

You must always type a space between a command and its input. If you misspell a command or forget the space, Logo is friendly enough to let you know something is wrong by giving you an error message. More about these later on.

Type

FORWARD 100 or FD 100

What happened?

Your input number was too large, causing the turtle to go off the screen and reappear at the bottom. It takes a little time to realize how far the command FD 1 moves the turtle. You'll soon get lots of practise moving the turtle around the screen.

To start again, type

CLEARSCREEN or CS

This command erases any drawings on the screen and moves the turtle back to its original state. You will use it often!!

Practice

Use the commands FORWARD (FD) and BACK (BK) with different inputs.

5.2 Commands to Change the Turtle's Heading

RIGHT or RT

LEFT or LT

These commands do not move the turtle but turn it to face in another direction. Type

CS

RIGHT 90

or

RT 90

The input to these commands is the number of degrees you want the turtle to turn (or if math scares you, the number of turns you want the turtle to make: 1 turn equals 1 degree).

Type the following and note what happens. Don't forget to press Return after each command.

CS

FD 75

RT 90

FD 53

LT 25

BK 100

Remember Logo was designed to be used by people with NO previous mathematics. Kids as young as six can program with Logo Turtle Graphics. If you can't see right away which direction to turn the turtle, play turtle! That is, pretend

that you are the turtle and walk through your instructions. Or if you are feeling lazy or shy, turn your head in the direction you want it to go.

Practice

Try playing with the commands that change the turtle's state: FD, BK, RT, and LT. Try different inputs for each one. Remember to use the command CLEARSCREEN (CS) when you want to start over. Then answer these questions.

1. What command will make the turtle turn around and face the opposite direction?
2. What command will make the turtle turn all the way around and face the same direction again?



"I think there's a bug in here somewhere."

Feedback

1. RIGHT 180 or LEFT 180
2. RIGHT 360 or LEFT 360 (did you see the turtle move?
If you did, you've got good eyesight!)

6. Error Messages

Logo has been programmed to tell you when you have typed in something it doesn't understand. The error message is a clue to what you did wrong. Let's look at some error messages you may encounter.

1. Type

FORWARD50 as is and press Return

You'll get this message:

I DON'T KNOW HOW TO FORWARD50

Logo was expecting a command and didn't recognize FORWARD50. Why not? You forgot the space between the command and its input. Now type

FORWD 50 as is and press Return

Logo is still protesting with the same error message. You have the space, so what's the problem? There is a spelling mistake.

Remember the computer will do only as you say. The burden is on you to be accurate and precise at all times. In order to avoid wasting your time and energy on debugging typing and spacing errors, always check your work.

2. Type

FORWARD (press Return)

The message reads

NOT ENOUGH INPUTS TO FORWARD

Logo commands such as FORWARD, BACK, RIGHT, LEFT require inputs immediately after the space. If you forget to tell the turtle how far to move or how much to turn, you'll get this message.

3. Type

FORWARD CS (press Return)

Logo will clear the screen and then give you this message:

CLEARSCREEN DIDN'T OUTPUT TO FORWARD

In other words, the turtle didn't move forward because the input was wrong. Here Logo expects a number.

4. Now type just the number 50 and press Return. Logo outputs the message:

I DON'T KNOW WHAT TO DO WITH 50

Logo is waiting for you to tell it what to do with the number 50. You must type in a command, such as FORWARD 50.

Practice

Type the following commands as shown. Correct them using the error message as a clue to the bug.

1. BACK
2. BK TS
3. BAK 60
4. BK100
5. 70

Feedback

1. no input: BACK 20 2. wrong input: BACK needs a number
3. wrong spelling: BACK 60 4. no space: BK 100
5. no command: FD 70

7. SHOOT

Let's play the game SHOOT. It's intended to give you practice in moving the turtle on the screen and in applying the problem-solving techniques discussed in Unit 1.

7.1 How to Play SHOOT

1. Type START (Don't forget to press Return)
 The computer will draw a target somewhere on the screen and place the turtle somewhere else. Every time the game is played the computer will change the position of the target and the state of the turtle.
2. Aim the turtle toward the target using RIGHT (RT) or LEFT (LT) commands.
3. When you think the turtle is pointing directly at the target, type SHOOT
4. The computer will ask HOW FAR? You type a number. Then the computer will move the turtle the distance you type and show whether you have hit the target.
 THE TURTLE MUST LAND INSIDE THE CIRCLE. If you miss, the turtle goes back to its starting point and you try again. The computer keeps count of how many tries it takes you to score a hit. Wait until you see the ? prompt before you input your number.

5. To stop at any point in the game, just type CLEARSCREEN (CS).
6. To begin with a new target and a new turtle state, type START.

7.2 Playing SHOOT Using a Systematic Problem-Solving Strategy

One way to approach this problem is to start guessing at random. You simply type in your guess and see what happens. Try this approach once or twice.

As you saw in Unit 1, a more efficient approach is a systematic one.

- ** Understand exactly what is wanted. You must do three things:
1. turn the turtle RT or LT,
 2. a specified number of degrees so the point of the triangle faces the target center,
 3. indicate how far the turtle is to move in order to land inside the circle.

You have effectively divided the problem into 3 subproblems or 3 instructions to the turtle.

Since changing the turtle's state depends a lot on your perceptual ability and experience with the turtle, you have to guess. But write each guess down so you can remember it the next time. Writing down each attempt and then guessing according to your previous experience is also a systematic approach -- you are following a plan rather than just guessing blindly.

Play the game now using a systematic approach. It may take you longer than random guessing, but it should bring you closer to the solution.

8. Review of Logo Commands

FORWARD	FD	CLEARSCREEN	CS
BACK	BK	SHOWTURTLE	ST
RIGHT	RT	HIDETURTLE	HT
LEFT	LT	TEXTSCREEN	TS
		MIXEDSCREEN	MS
		FULLSCREEN	FS

9. Commands to Control the Turtle's Pen

PENUP or PU PENDOWN or PD PENERASE or PE

PENUP (PU) raises the turtle's pen and allows you to move the turtle without leaving a trace. But always remember to PENDOWN (PD) to start drawing again. Try this:

```

CS
FD 50
RT 90
PU
FD 100
LT 90
PD
BK 50
HT
*****
*                                     *
*      Don't forget to PENDOWN      *
*      after the commands PU or PE  *
*                                     *
*****

```

What if you want to erase a line? One way is to use CLEARSCREEN but then you lose all your work! Instead use the command PENERASE (PE). The turtle's pen now is an eraser. Bring the turtle back on the screen with the ST command. Clear the screen and type this:

```

FD 50
RT 90
FD 50
PE
BK 50
LT 90
BK 50
PD

```

9.1 Drawing Letters

Let's try to put together everything you've learned up to now. Your problem is to write a program that outputs the letter T.

- ** Be precise: know exactly what you want.
- ** Relate the data by drawing a picture.
- ** Write an algorithm for the solution. Here's one example:

```
FD 50
RT 90
FD 25
BK 50
```

** Check your work -- look for bugs. A possible bug may be the inputs to FD and BK. How big do you want the letter to be?

- ** Enter, run, and debug your program.

Practice

Write a program that will draw your initials. Remember to use a systematic approach. Follow the outline in the example above.

Keep a written copy of your program on a separate piece of paper. In a later section you will use it to create a procedure called INITIALS. It is also a good programming habit to always have a written copy of any program.

10. Drawing Shapes

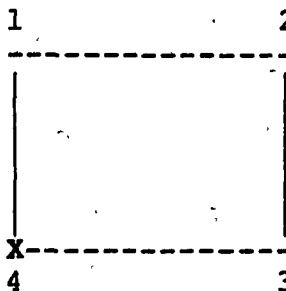
10.1 A Square

Your task now is to have the computer draw a square.

**** Be precise:** know exactly what you want.

A square has 4 equal sides. You must specify the length. Let's choose 50. You also need to know how much to turn and in what direction.

**** Relate the data by drawing a picture.** The "X" is the turtle with its tip pointing toward the top of the screen.



Remember you do NOT need to know math to figure out how much to turn the turtle. The Total Turtle Trip Theorem was devised for those of you who hate math. It is used to determine the inputs for the commands RIGHT or LEFT. Here it is:

```

*****
*
*           If the turtle takes a trip
*
*         around the boundary of any area and
*
*       ENDS UP IN THE STATE IN WHICH IT STARTED,
*
* then the sum of all the turns is 360 degrees.
*
*****

```

So, to draw a square and have the turtle end in the same state (position and heading) as it began, it needs to turn four times. Each turn would be $360 / 4 = 90$. In the above drawing you turn RIGHT.

**** Write the algorithm:**

```
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90 (remember to bring the turtle
       back to its original heading)
```

**** Check your work -- follow through your algorithm on paper first. Did you get what you wanted?**

**** Type it in checking for spelling, typing, and spacing errors.**

10.2 The REPEAT Command

Time can be saved by using the REPEAT command where appropriate. It takes 2 inputs: the number of times the commands are to be repeated and the commands enclosed in square brackets.

Your program for the square becomes:

```
REPEAT 4 [FD 50 RT 90]
```

Note the exact syntax. Remember to use **square brackets**. Type in this command and see what happens.

Practice

1. Use a REPEAT command to reduce the following:

A

FD 100
LT 90
FD 100
LT 90
FD 100
LT 90
FD 100
LT 90

B

FD 100
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
BK 50

2. Using the REPEAT and PU commands, draw 2 squares of different sizes. They should not touch one another.

Turn the page for the answers.

Feedback

1. A. REPEAT 4 [FD 100 LT 90]

B. FD 100
 REPEAT 3 [RT 90 FD 50]
 RT 90
 BK 50

2. To vary the size of the square you change the input to FD. Try these two:

REPEAT 4 [FD 75 RT 90]

REPEAT 4 [FD 25 LT 90]

To make sure the two squares do not touch, use the PU command. Try this:

CS
 REPEAT 4 [FD 75 RT 90]
 PU
 LT 90
 FD 50
 RT 90
 PD
 REPEAT 4 [FD 25 LT 90]

This is only one possible answer. Your answer need not be exactly the same. If the two squares are of different sizes and do not touch one another, your answer is correct.

10.3 Practice in Drawing Shapes Using a Systematic Problem-Solving Strategy

Use a systematic approach to solve the following problems. Fill in the outline below BEFORE you use the computer. (Use a separate piece of paper.)

1. Understand the problem.
 - ** be precise: know exactly what you want
 - ** make explicit any assumptions
 - ** simplify the problem by breaking it down
2. Relate the data by drawing a picture.
3. Write an algorithm of the solution. When you follow it through on paper, do you get what you want?
4. Check your work -- look for bugs.
5. Enter, run, and debug your program.

Turn the page when you have finished. Be sure not to get too frustrated trying to do this exercise. Try for about 15 minutes, then go on to the discussion of the answer. It is better at this stage to understand HOW to attack a programming problem. And remember, you do not need math to solve these problems.

a) Draw an equilateral triangle (all the sides are equal) with the point of the triangle facing the top of the screen.

b) Draw two circles of different sizes. They may overlap.

HINT: use the REPEAT command and the Total Turtle Trip Theorem.

Feedback

10.3 a) A Triangle

** Understand the problem:

You want an equilateral triangle with the point facing the top of the screen. Let's break this problem into two simpler problems and attack each one separately: 1) draw an equilateral triangle, 2) be sure the point is facing the top of the screen.

What's an equilateral triangle? If you recall any geometry, it has 3 equal sides and all angles are 60 degrees. If you don't remember any geometry, that's great. Logo is just for you. Let's handle both cases.

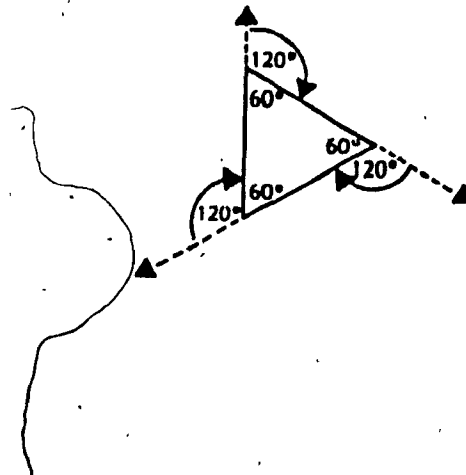
i) THE MATH LOVER: she wants three equal sides of 25 and three angles at 60 each. Because she's convinced this is an easy problem to solve, she enters her program right away:

```
REPEAT 3 [FD 25 RT 60]
```

Does she have an equilateral triangle? Try it and see.

There's a bug somewhere. So now she decides to draw a picture. Taking out her pencil and ruler she starts to calculate.

Finally she sees that the turtle must turn 120 not 60.



ii) THE MATH HATER: she's forgotten all about triangles and angles. But she remembers the Total Turtle Trip Theorem. She begins by drawing a picture. From the picture, she sees a triangle has 3 turns and each turn must be 120 ($360 / 3 = 120$). She's right on her first try because she used her knowledge NOT of geometry but of the computer and the turtle.

**** Check your results:**

Rereading the problem statement, you realize the problem is still not solved. The point of the triangle is not facing the top of the screen. The final solution should be like this:

```
LT 90
REPEAT 3 [FD 25 RT 120]
RT 90
```

If you wanted to vary the size of the triangle, you need only change the input to FD.

Take a few minutes to draw triangles of different sizes in different places on the screen.

10.3 b) A Circle

**** Understand the problem:**

You must draw two circles of different sizes. Let's again divide this into two subproblems: 1) draw a circle 2) vary the size of the circle.

So, what is a circle? The math lover replies gleefully that a circle is 360 degrees with the radius equal to one-half the diameter etc., etc. You have already seen that

this is the more difficult path to a solution.

The math hater looks at the circle from the turtle's point of view. Drawing a circle on paper or walking through a circle, she realizes that a circle is actually a repetition of one step, one turn. For a big circle she takes big steps; for a small circle she takes small steps.

The Total Turtle Trip Theorem states that the turtle will turn 360 times to return to its same state. Using this to find the input for the heading command, you get

$$360 / 360 = 1.$$

```
REPEAT 360 [FD 1 RT 1]
```

or

```
REPEAT 360 [FD 1 LT 1]
```

Good for you, if you came up with this answer. Drawing circles in Logo is a difficult task with your limited Logo vocabulary.

Notice these two commands take a long time to run. Hiding the turtle first will speed things up considerably.

There are other ways to program Logo to draw circles. If you're interested in pursuing this topic, the books at the end of this unit will show you how. For the purposes of this module, it suffices to know the commands below.

To vary the size of the circle you again change the input to FD. Try them out to see the difference.

```
HT
```

```
REPEAT 360 [FD 1 RT 1]
```

```
REPEAT 360 [FD .5 RT 1]
```

```
REPEAT 360 [FD .15 RT 1]
```

```
ST
```

11. A Quick Review

Before you go on, review the following concepts and commands. You should be very clear about their meaning and use. If you are not, take some time now to review and practice.

Logo	turtle graphics
the turtle's position	the turtle's heading
the total turtle trip theorem	

FORWARD	FD	CLEARSCREEN	CS
BACK	BK	SHOWTURTLE	ST
RIGHT	RT	HIDETURTLE	HT
LEFT	LT	PENUP	PU
REPEAT		PENDOWN	PD
		PENERASE	PE
		TEXTSCREEN	TS
		MIXEDSCREEN	MS
		FULLSCREEN	FS

You should also be able to draw squares, triangles, and circles of varying sizes anywhere on the screen.

12. Procedures

One of the most powerful features of Logo is that it allows you to teach the computer new words, that is, to add to its vocabulary of commands. This process is called defining a procedure.

```
*****
*                                     *
*           A procedure is a series   *
*                                     *
*           of instructions or commands. *
*                                     *
*****
```

For example, REPEAT 4 [FD 50 RT 90] is a procedure to draw a square with sides 50. To draw a circle you used the procedure REPEAT 360 [FD 1 RT 1]. The list of commands used to draw your initials is also a procedure.

```
*****
*                                     *
*           To define a procedure is to teach *
*                                     *
*           Logo a new command., *
*                                     *
*****
```

When a procedure is defined you need only type the procedure's name and the turtle will execute all the commands listed in that procedure. Let's look at an example.

12.1 Defining Procedures in the Logo Editor

The easiest way to define a procedure is to use the Logo editor. This is a feature of the language that allows you not only to define procedures but to change them as well. You will use it often.

Let's define a procedure to draw a square.

1. Choose a name for your procedure. It can be any name you like but it is preferable to use one that describes the output of the procedure. The procedure to draw a square would be called SQUARE, the one for a circle, CIRCLE etc.

2. Enter the Logo editor. Type

EDIT "SQUARE or .ED "SQUARE

Note carefully the syntax used. A space always follows EDIT and the name of the procedure must be preceded by quotation marks.

Notice that your turtle drawings have disappeared and the words LOGO EDITOR appear at the bottom of the screen. You now have the whole screen for typing. Look where the cursor is.

3. The first line of a procedure definition must always be TO. Note that TO is added automatically as soon as you enter the Logo editor, so you need not retype it. Simply move the cursor down one line by pressing Return. Type in the commands to draw a square.

REPEAT 4 [FD 50 RT 90] (press Return)
END

4. Always conclude your definition with the

command END, alone on the last line.

```
*****
*
*       A procedure definition always begins with TO
*
*               and finishes with END
*
*****
```

5. There are two ways to get out of the Logo editor:

- i) if you ~~do~~ NOT want to keep your work, press at the same time the Ctrl key and the Break key (it's the same as the Scroll Lock key in the upper right-hand corner).

- ii) if you want to keep your work, press the ESCape key.

You want to keep this procedure, so press ESC. (If you jumped the gun and pressed the Ctrl and Break keys, start over again from step 1.)

You should see the message:

SQUARE DEFINED

SQUARE is now stored in RAM, ready to be used.

6. To check your work, clear the screen with CS and then type the new command:

SQUARE

The turtle should draw a square with sides 50. SQUARE has become a new command like FORWARD 100 or CLEARSCREEN.

Practice

1. Let's use our new command. Type

```
CS
SQUARE
PU
LT 90
FD 100
RT 90
PD
SQUARE
```

2. Use your new command to draw squares anywhere on the screen.

12.2 Editing (Changing) Procedures

Any changes or corrections are done in the Logo editor. You now want to make a bigger square. To do this you need to change the input to FD. Let's try it.

1. Enter the Logo editor. Type

EDIT "SQUARE

2. Move the cursor to the number 50. (See Appendix A for the cursor movement keys.)

3. Change 50 to 100.

4. Exit the editor by pressing the ESC key. This replaces your old version of SQUARE.

5. Check your results. Clear the screen and type

SQUARE

Does this edited procedure produce a bigger square? If not, recheck every step to try and find the bug.

12.3 Checking your Work with PO and POTS

To find out what procedures you have already defined, type POTS (that's short for Print Out Titles) You should see the reply:

TO SQUARE

This means that the procedure SQUARE has been defined and is now stored in RAM.

To see what instructions are in a given procedure, you simply use the PrintOut command: PO "procedure name. This

is quicker than using the Logo editor. Try it. Type

PO "SQUARE

12.4 Erasing Procedures

You decide that you don't want to keep the procedure SQUARE. You want to erase it from RAM. Type

ERASE "SQUARE (go ahead and do it)

Now check that it is gone by typing POTS.

You should get no reply.

12.5 The Immediate Definition Mode (or the > prompt)

There is a second way to define a procedure in Logo. It is called the immediate definition mode because it bypasses the Logo editor AND because you cannot correct mistakes or make changes without stopping the definition process and beginning again. For this reason it is not recommended for beginners. But you should know it exists, if by accident you fall into it. Let's redefine our SQUARE procedure using this mode.

1. Type TO SQUARE (press Return)

```
*****
*
*       Note the ? prompt has become a > prompt.
*
*****
```

2. Practice returning to the ? prompt, by pressing the Ctrl and Break keys together. This stops the definition. Begin again by typing TO SQUARE.

3. Type in the instructions for a square with sides 50.

4. Be sure your last line is END. You should see the message: SQUARE DEFINED and the ? prompt. Now check your work with POTS. SQUARE should be there.

12.6 More Error Messages

Type the following as is: SQU (Press Return)

You get the message I DON'T KNOW HOW TO SQU

This means that you have not defined a procedure named SQU. To check the spelling of all your defined procedures, use POTS.

Now let's try to define a procedure to draw a window.

Type ED "WINDOW

You'll get this message WINDOW IS A PRIMITIVE

This simply means that WINDOW is already a Logo command; it's part of the original Logo vocabulary so you cannot redefine it. Choose another name for your procedure.

Practice

1. Define procedures to draw a triangle and a circle of any size. Use the ones from the previous exercises.
2. Define a procedure which outputs your initials. Call your procedure INITIALS.
3. Check that you have defined the following: SQUARE, TRIANGLE, CIRCLE, INITIALS.

```

*****
*
*   Want to stop the execution of your procedure?
*
*   Want to exit the Logo editor WITHOUT keeping your work?
*
*   Want to change the > prompt back to the ? prompt?
*
*   PRESS THE CTRL AND BREAK KEYS TOGETHER.
*
*****

```

Feedback

For 1. and 2. refer to the section "Defining Procedures in the Logo editor" to check that you followed the correct steps.

3. POTS is the command used to check that your procedures are defined.

13. Variables

Suppose you want to draw a smaller square. You could edit the procedure SQUARE and change the input to FORWARD. But that's the long way to do it. A better way is to use a variable to change the size of your shape.

```
*****
*                                     *
*           A variable is a word that   *
*                                     *
*           can assume a given value.   *
*                                     *
*****
```

Using a variable creates ONE procedure which needs an input to tell it how big the square should be. A variable is like an empty container. You specify what to put into it (the value of the variable). A variable is ALWAYS preceded by a colon. Let's look at an example.

You want your procedure SQUARE to be able to draw a square of any give size. Remember to change the size of a shape you changed the input to FORWARD. Let's use a variable for the input.

1. Enter the editor to make the change. Type

EDIT "SQUARE

2. Type :N (N for Number - you can label the variable as you like but its better to use a name that helps you remember that the command FORWARD needs a number as an input) Your top line should look like this:

TO SQUARE :N

3. Now change the input to FORWARD. Your new procedure should look like this:

TO SQUARE :N
REPEAT 4 [FD :N RT 90]
END

4. Press the ESC key to exit the Logo editor.
5. Check your work. To make the turtle draw a square, type the command SQUARE followed by an input number telling the turtle how far to move.

SQUARE 10
and
SQUARE 25

```
*****
*
*   There are 2 special rules for the use of the : symbol
*
*   1. Always type :N without a space after the :
*
*   2. Don't type the : when you give a value to the
*       variable. Watch what happens if you do.
*
*           SQUARE :30
*
*           30 HAS NO VALUE
*
*****
```

Practice

Redefine your procedures to draw triangles and circles using a variable. Follow the above steps.

Feedback

Your new procedures should look like this in the Logo editor:

```
TO TRIANGLE :N  
  LT 90  
  REPEAT 3 [FD :N RT 120]  
  RT 90  
  END
```

```
TO CIRCLE :N  
  REPEAT 360 [FD :N RT 1]  
  END
```

Practice these two commands with different values for :N.

14. File Management

14.1 Saving Your Work

All your defined procedures are stored in RAM. If you were now to turn off your machine, you would lose all of your work. So be sure to save any procedures you need onto your working diskette. Here's how to do it.

When you save, the computer creates a file on your working diskette for future use. There is a difference between a file and a procedure.

```
*****
*
*   The file you save on your working diskette
*
*   will contain
*
*   all the procedures in RAM at the time of the save.
*
*****
```

Let's save all the procedures you have defined so far:

1. To check what procedures are currently in RAM, type

POTS

The reply should be:

```
TO CIRCLE :N
TO TRIANGLE :N
TO SQUARE :N
TO INITIALS
```

2. Choose a name for your file. Let's choose the name SHAPES.

3. To save all the procedures listed above in the file named SHAPES, type

SAVE "SHAPES

Note the syntax of the command SAVE -- a space and then a quotation mark. The file name must be preceded by quotation marks.

14.2 The Commands DIR and ERALL

To see what files you have saved on your working diskette, type DIR (for DIRectory). This command outputs the names of all the files you have saved. You should see the file name SHAPES.

Since you now have all your procedures on your own diskette, you can erase them from RAM. Type

ERALL

This command clears all procedures from RAM. Check with POTS. You should get no reply because no procedures are in RAM.

14.3 Loading A File From Your Working Diskette

You have returned to the computer after a week's rest. You want to use the procedures in your file SHAPES. How do you get it from your diskette into RAM?

Your first step would be to load Logo into the computer (see Section 3 for instructions). Once you have the ? prompt on your screen you are ready to begin. The command DIR will list the files on your diskette and should be used to check the correct spelling of your file names. Now type

LOAD "SHAPES

Check with POTS. You should see a list of all the procedures saved in the file SHAPES and now in RAM.

TO CIRCLE :N
TO TRIANGLE :N

TO SQUARE :N
TO INITIALS

14.4 Editing (Changing) a File

Let's add a new procedure to our file SHAPES.

1. Define a procedure called TEE with the following instructions:

```
FD 50
RT 90
FD 25
BK 50
```

Refer to Section 12.1 - Defining Procedures in the Logo Editor,

2. Check with POTS to see what procedures are currently in RAM. Remember when you save, you save all the procedures listed by the POTS command. So be sure you have exactly what you want.

3. Type `SAVE "SHAPES`

Oops! What happened? You got an error message:

FILE SHAPES ALREADY EXISTS

This is your old version. You don't need it anymore because you have a new up-to-date edited version. So, let's get rid of it.

`ERASEFILE "SHAPES`

The command ERASEFILE removes unwanted files from your own diskette. Now check with DIR. You should get no reply because there are no Logo files on your diskette. But don't panic! Your work is still in RAM. To reassure yourself, check with POTS.

4. Now save the procedures listed by POTS. Type

`SAVE "SHAPES`

Check with DIR to be sure the file is on your diskette.

Practice

1. Clear RAM so that the command POTS gives no reply. Now save a file on your diskette with only the procedures INITIALS and TEE in it. Call the file LETTERS. If you get the error message N HAS NO VALUE, ignore it.
2. Change your file SHAPES by removing the procedures INITIALS and TEE. You want only the procedures to draw a circle, triangle, and square in the file SHAPES.

Feedback

Here are the commands you should have used. Be sure you know what each does. If you are not sure, refer to the relevant section (use the Table of Contents to find the section you need).

1. ERALL
2. POTS (You should get no reply)
3. LOAD "SHAPES
4. POTS (You should get a list of these procedures: CIRCLE :N, TRIANGLE :N, SQUARE :N, INITIALS, TEE)
5. ERASE "CIRCLE :N

Here is where you should have gotten the error message N HAS NO VALUE. It can be ignored ONLY when erasing procedures with a variable. In fact, it is not necessary to include the variable :N.

6. ERASE "TRIANGLE
7. ERASE "SQUARE
8. POTS (You should see only the two procedures: INITIALS and TEE.)
9. SAVE "LETTERS
10. DIR (You should see SHAPES and LETTERS)

2. ERALL
LOAD "SHAPES
POTS
ERASE "INITIALS
ERASE "TEE
POTS
BRASEFILE "SHAPES
SAVE "SHAPES
DIR

15. Subprocedures

Understanding procedures is a very important aspect of using Logo. To refresh your memory, reread Section 12. Procedures can be used as building blocks to make longer more complicated programs.

```
*****
*
*           When one procedure is used as part
*
* of another procedure, it's called a subprocedure.
*
*****
```

You have already defined a procedure SQUARE :N (check with POTS). When you use SQUARE :N as part of another procedure, it becomes a subprocedure. Look at this example. Type

```
ED "FLAG
FD 25
SQUARE 25 (remember to give a value to :N)
END
```

You now have a new procedure FLAG which uses SQUARE :N as a subprocedure. Type FLAG and see what happens.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
X
X           Did you get this error message:
X
X           I DON'T KNOW HOW TO SQUARE
X
X           Check that SQUARE is in RAM.
X
X           If not, load your file SHAPES.
X
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Now with FLAG as a subprocedure, you can make a more complex procedure called CROSS.

```
ED "CROSS
REPEAT 4 [FLAG RT 90]
END
```

If you tried to draw CROSS without using FLAG and SQUARE as subprocedures, it would be a long and complicated process. Using subprocedures makes it easier to do and to understand. Since each subprocedure exists as a separate entity, the debugging process is also easier.

Practice

Study the following uses of procedures and subprocedures. Be sure you understand how they work.

```
TO FLAGBACK
FLAG
BK 25
END
```

```
TO FLAGS
REPEAT 4 [FLAGBACK RT 90]
END
```

```
TO MANYFLAGS
FLAGS
RT 45
FLAGS
END
```

15.1 Subprocedures - a Problem-solving Technique in Action

Subprocedures become extremely important when you have to solve complex problems. In Unit 1 you learned the technique of stepwise refinement: break a complex problem into simpler, more manageable parts. Logo allows you to do this using subprocedures.

Let's apply this technique to solve a complex problem such as: draw a face. Remember to approach this problem systematically.

**** What exactly do you want?**

Since you're very artistic, you decide to draw a face that looks like this:



Now break the problem down into more manageable parts. You would probably draw first the head, then the eyes, nose and mouth. With Logo, each of these parts can be drawn by a separate procedure. As a final step you use these as sub-procedures in the main procedure FACE. Let's see how it's done.

Below are the commands to draw each part of the face. Using these commands, define the following procedures: HEAD, LTEYE, RTEYE, NOSE, MOUTH. Note that some of these use the previously defined procedures SQUARE, TRIANGLE, CIRCLE as subprocedures. Make sure you have them in RAM.

```
(the head)
LT 90 PU
FD 50 PD
RT 90
HT
CIRCLE 1
```

```
(the right eye)
PU
RT 90 FD 55
LT 90 PD
SQUARE 15
```

```
(the left eye)
PU
FD 5 RT 90
FD 20 LT 90
PD
SQUARE 15
```

```
(the nose)
PU
LT 180 FD 15
RT 90 FD 10
RT 90 PD
TRIANGLE 15.
```

```
(the mouth)
PU
BK 20 LT 90
PD
FD 30 BK 50
RT 90
```

Using these procedures now as subprocedures, define the main procedure FACE:

```
ED "FACE
CS
HEAD
LTEYE
RTEYE
NOSE
MOUTH
END
```

You have only to type FACE. Try it.

Note that the order in which you place the subprocedures is extremely important. Here's why. Define the following main procedure and see the difference.

```
ED "FACE2
CS
LTEYE
RTEYE
NOSE
MOUTH
HEAD
END
```

The thing to remember is that the turtle's state at the end of one subprocedure must be the same as its state at the beginning of the next subprocedure. That's why FACE2 has a bug. The turtle's state at the end of MOUTH is not the same as it's state at the beginning of HEAD.

** check your work. Does your procedure FACE still have a bug? If so, check that your subprocedures SQUARE, TRIANGLE, and CIRCLE are the same as those defined in Section 13.

Practice

1. a) Below is a series of commands. Divide them into subprocedures. Define each subprocedure and then define the main procedure HOUSE.

```
REPEAT 4 [FD 50 LT 90]
FD 50
LT 90
REPEAT 3 [FD 50 RT 120]
RT 90 BK 50
LT 90 FD 20
RT 90
REPEAT 3 [FD 15 LT 90]
FD 35
LT 90
```

- b) Why can't you use your procedure SQUARE :N?
- c) Why is it much easier to debug the procedure HOUSE than to debug the list of commands shown above?

Feedback

1. The main procedure could be defined as.

```

      TO HOUSE
      CS
      BODY
      ROOF
      DOOR
      FINISH
      END

```

The subprocedures could be defined as

```

      TO BODY
      REPEAT 4 [FD 50 LT 90]
      END

```

```

      TO ROOF
      FD 50
      TRIANGLE 50
      END

```

```

      TO DOOR
      BK 50
      LT 90 FD 20
      RT 90
      REPEAT 3 [FD 15 LT 90]
      END

```

```

      TO FINISH
      FD 35
      LT 90
      END

```

- b) Your SQUARE :N procedure turns RT 90 not left. You could define a new subprocedure to draw a square turning left:

```

      TO SQUAREL :N
      REPEAT 4 [FD :N LT 90]
      END

```

- c) Using only subprocedures and a main procedure makes debugging much easier. If there is a bug in your main procedure, you have only to run each subprocedure one at a time starting with BODY to verify that each one is correct. Once the subprocedure with the bug is located, it is a much simpler process to correct that subprocedure than to correct a long and unwieldy string of commands.

N.B. You do NOT need to save the above procedures for FLAG, FACE, and HOUSE on your diskette (Unless, of course, you want them for yourself.)

16. A Final Review

Before you begin work on your final project, review the summary sheet on the next page. If you are unclear about any of the terms or commands, review the relevant section in this module. If you are still having difficulty, borrow the IBM tutorial Logo: Programming with Turtle Graphics from the reserve booth. Work through chapters one, two, three, eight, and nine. These chapters will give you added practice in using the commands explained in this module.

16.1 Summary

You should be able to explain the following terms:

Logo	Turtle Graphics
the turtle's state	the turtle's position
the turtle's heading	the total turtle trip theorem
procedures	subprocedures
defining a procedure	Logo editor
error messages	variable

You should know how to use the following commands:

FORWARD	FD	CLEARSCREEN	CS
BACK	BK	SHOWTURTLE	ST
RIGHT	RT	HIDETURTLE	HT
LEFT	LT	PENUP	PU
REPEAT		PENDOWN	PD
EDIT "procedure name		PENERASE	PE
ERASE "procedure name		FULLSCREEN	FS
TO		MIXEDSCREEN	M
END		TEXTSCREEN	TS
SAVE "file name		PO "procedure name	
LOAD "file name		POTS	
ERASEFILE "file name		DIR	
ERALL		CTRL-BREAK	

You should also be able to draw squares, triangles, and circles of varying sizes anywhere on the screen, as well as define, erase, save, and load procedures.

17. Recommended Reading

For those of you who have been bitten by the programming bug and want to learn more about programming in Logo, the following books are recommended.

Abelson, H. and diSessa, A. (1981). Turtle Geometry: The computer as a medium for exploring mathematics. Cambridge: MIT Press.

This book illustrates the use of turtle graphics to explain advanced mathematical concepts at the university level.

Papert, Seymour (1980). Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Papert outlines his vision of computers as tools to learn with. The origin and philosophy behind Logo are explained.

Watts, Daniel (1984). Learning with Apple Logo. McGraw Hill Book Co.

An excellent book for learning how to program in Logo on any machine not just the Apple.

18. Final Project

1. Write a program using Logo turtle graphics to draw a means of transportation; for example, a car, a boat, a truck, a bicycle, etc..

- ** your program should consist of one main procedure and several subprocedures as in the FACE example in Section 15.1.

- ** it should have NO bugs.

- ** it should be saved in a separate file on your own diskette. Use your last name as the file name. (The extension .lf is automatically added so you know it's a Logo file.)

Your drawing need not be complicated. You will be evaluated strictly on the above four points. *

2. Write a brief description of the problem-solving strategy you used to arrive at your solution.

- ** how did you attack the problem? Did you work from a written plan?

- ** did you write an algorithm first before you used the computer?

- ** how did you use the computer: to debug your algorithm? to determine angles and distances on a trial and error basis? to immediately draw your shape?

- ** did you use any of the techniques described in Unit 1 - Problem Solving?

There is no one correct answer. The objective of this exercise is to get you thinking about how you solve programming problems so that you become more conscious of the process.

When you have completed your final project, have the lab attendant copy your file onto the hard disk. Hand in your description at the reserve booth. Then do the quiz.

The quiz for Lesson 3 is on both Unit 1 - Problem Solving, and Unit 2 - Programming with Logo. To review, study the summary sheets at the end of each unit. The completed quiz should be returned to the reserve booth.

```
*****
*
*           Lesson 3 is done when you have           *
*
*           1. finished your Logo program,            *
*
*           2. handed in your written description, and *
*
*           3. completed the quiz on Units 1 and 2.    *
*
*****
```

Cursor Movement Keys



Backspace

*Erases character to left of cursor.



Cursor Left

*Moves cursor one space to the left.



Cursor Right

*Moves cursor one space to the right.



Shift

* Moves cursor to beginning of current line.



* Moves cursor to end of current line.



Cursor Up

Moves cursor up to previous line in editor.



Cursor Down

Moves cursor down to next line in editor.



Enter

*Carriage return.



Erases everything on current line to right of cursor.



*Inserts last line deleted.

Ctrl-Break

1. Outside editor, interrupts and stops a running procedure.
2. Inside editor, aborts editing.



Interrupts whatever is running. Typing any key resumes running.



Moves cursor to end of the edit buffer.



Moves cursor to top of the edit buffer.



*Deletes character at cursor position



Moves cursor to end of current page in editor.



Leaves editor, reading buffer as if it was typed in.



Outside editor, corresponds to TEXTSCREEN.



Outside editor, corresponds to MIXEDSCREEN.



Inserts a copy of the last line typed.



Outside editor, corresponds to FULLSCREEN.



Outside editor, corresponds to PAUSE.



Moves cursor to top of current page in editor.

*Indicates a key function that works both inside and outside the editor.