

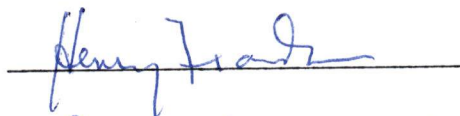
To the Graduate Council:

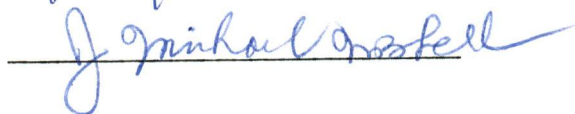
I am submitting herewith a thesis written by Stephen John Widner entitled "A Computer-Assisted Instruction System for Teaching Elementary FORTRAN Programming Concepts." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



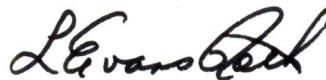
Robert M. Aiken, Major Professor

We have read this thesis and  
recommend its acceptance:





Accepted for the Council:



Vice Chancellor  
Graduate Studies and Research

Thesis  
79  
.W535  
cop. 2

A COMPUTER-ASSISTED INSTRUCTION SYSTEM FOR TEACHING  
ELEMENTARY FORTRAN PROGRAMMING CONCEPTS

A Thesis  
Presented for the  
Master of Science  
Degree  
The University of Tennessee, Knoxville

Stephen John Widner

June 1979

1390086

## ACKNOWLEDGEMENTS

The writer would like to express his appreciation to his committee, Dr. Robert Aiken, Dr. Michael Moshell, and Dr. Henry Frandsen for their guidance and encouragement in this study.

Also due thanks are the students who participated in the study and their instructors, Rebecca Meacham, Harry Nayak, and Mukesh Sundaram, without whose assistance the study would not have been possible.

The writer also wishes to express his gratitude to his immediate family. For their help in preparing typed material and the manuscript he thanks his father, John Widner, and his brother and sister, David Widner and Mrs. Elizabeth Jackson. For her encouragement when hope for finishing seemed dim, he thanks his mother, Mrs. Nelle Widner.

## ABSTRACT

A system of computer programs for providing computer-assisted instruction to students in elementary FORTRAN programming courses is presented in this thesis. The system consists of seven lesson modules, written in the COURSEWRITER III language for an IBM 370/148, and a short handout associated with each module. The first lesson introduces the student to the cathode ray tube terminals and the remaining lessons focus on programming. The handouts are written in a humorous manner in order to promote interest. The modules likewise maintain a conversational tone so that the students will feel at ease with the computer.

The design and operation of the system are described and examples of the handouts and terminal sessions are included in the appendices. The system was tested in two sections of Computer Science 1510 at the University of Tennessee during Winter Quarter 1979. In this study the lessons were used as a supplement to the normal classroom instruction received by Computer Science 1510 students. Volunteers from the two classes were divided into a control and an experimental group, with the experimental group participating in the program. A posttest was given to both groups near the end of the quarter. Results of this test and student comments are described in the thesis. The posttest and evaluation questionnaire appear in the appendices. The system was determined to be a useful technique for presenting elementary programming concepts to novice programmers.

## TABLE OF CONTENTS

| CHAPTER                                                    | PAGE |
|------------------------------------------------------------|------|
| I. REVIEW OF RELATED LITERATURE AND RESEARCH . . . . .     | 1    |
| Introduction . . . . .                                     | 1    |
| Brief History of CAI . . . . .                             | 2    |
| Information science . . . . .                              | 3    |
| Engineering . . . . .                                      | 3    |
| Behavioral science . . . . .                               | 3    |
| Educational technology . . . . .                           | 4    |
| Curriculum approach . . . . .                              | 5    |
| PLATO and TICCIT . . . . .                                 | 5    |
| Motivation for Using CAI . . . . .                         | 6    |
| Related Projects and Studies . . . . .                     | 8    |
| IPAS . . . . .                                             | 8    |
| BATS . . . . .                                             | 9    |
| U. S. Naval Weapons Laboratory CAI system . . . . .        | 10   |
| BIP . . . . .                                              | 10   |
| TeachEASY . . . . .                                        | 11   |
| Temple flow diagram system . . . . .                       | 13   |
| SPIDO . . . . .                                            | 14   |
| MATHCALC . . . . .                                         | 15   |
| II. OVERVIEW OF PROJECT . . . . .                          | 18   |
| The Problem . . . . .                                      | 18   |
| Statement of the problem . . . . .                         | 18   |
| Hypotheses . . . . .                                       | 18   |
| Need for the study . . . . .                               | 18   |
| Procedures . . . . .                                       | 20   |
| Description of the program . . . . .                       | 20   |
| Assignment of subjects . . . . .                           | 26   |
| Administration of treatments . . . . .                     | 27   |
| III. PRESENTATION AND INTERPRETATION OF DATA . . . . .     | 28   |
| CSCAII Statistics . . . . .                                | 28   |
| The Experiment . . . . .                                   | 30   |
| Experimental subjects . . . . .                            | 30   |
| Analysis of data . . . . .                                 | 30   |
| Interpretation of Data . . . . .                           | 33   |
| IV. EVALUATION AND SUGGESTION FOR FUTURE STUDIES . . . . . | 36   |
| Student Reaction . . . . .                                 | 36   |
| Usage of CSCAII . . . . .                                  | 36   |

| CHAPTER                                                                | PAGE |
|------------------------------------------------------------------------|------|
| Benefits of CSCAI1 . . . . .                                           | 36   |
| The informal approach . . . . .                                        | 37   |
| Errors and possible improvements . . . . .                             | 37   |
| Obstacles to completion of CSCAI1 . . . . .                            | 39   |
| Value of CSCAI1 in introductory programming<br>courses . . . . .       | 39   |
| Computer-assisted instruction as a part of the<br>curriculum . . . . . | 40   |
| Recommendations and Future for CAI . . . . .                           | 40   |
| Enhancements to CSCAI1 . . . . .                                       | 40   |
| Advances in hardware and software . . . . .                            | 41   |
| Interdepartmental computer-assisted instruction . . . . .              | 43   |
| Conclusions . . . . .                                                  | 43   |
| BIBLIOGRAPHY . . . . .                                                 | 45   |
| APPENDICES . . . . .                                                   | 48   |
| Appendix A . . . . .                                                   | 49   |
| Appendix B . . . . .                                                   | 67   |
| Appendix C . . . . .                                                   | 71   |
| Appendix D . . . . .                                                   | 73   |
| Appendix E . . . . .                                                   | 74   |
| VITA . . . . .                                                         | 79   |

## CHAPTER I

### REVIEW OF RELATED LITERATURE AND RESEARCH

#### I. INTRODUCTION

Although computer-assisted instruction projects have been prevalent in the United States and other countries since the late 1950's, relatively few have dealt with teaching programming techniques (19). Perhaps, the lack of research in the area stems from the belief that programming is an art which cannot be left under control of a machine. In the early days of computing, when all programming was done at or slightly above machine level, artistry was dominant. The best programmers were those who could cleverly manipulate binary digits (bits) to produce efficient code.

The advent of high-level languages relieved the programmer of much of this bit manipulation and allowed him to concentrate more of his energy on the problem at hand. Programs written in high-level languages tend to be more standardized and readable than their machine-language counterparts. Teaching a high-level language via a computer then seems a much more manageable project than teaching machine language.

Even when using high-level languages, most programmers preferred to retain an identity, and developed a distinctive style. Tricks and clever techniques remained the rule rather than the exception. Through the efforts of Dijkstra, and others, a standardized programming style, known as structured programming, has evolved (14). This standardization

has made computer-assisted teaching of programming even more feasible than before. Research in computer-assisted instruction in general and in teaching programming in particular will be discussed in the remainder of this chapter. The relationship of this research to the topic of this thesis, a FORTRAN teaching system called CSCAII, will also be discussed.

## II. BRIEF HISTORY OF CAI

Research in CAI first took place in the late 1950's. Throughout the 1960's, pockets of CAI study formed across the country (19). The increase in CAT research paralleled the development of a compatible teaching technique, programmed instruction. As computer memory became cheaper, central processing unit speed and power increased exponentially; thus, the cost of CAI research dropped sharply. Federal spending on education continued to rise, and several grants were awarded to CAI researchers. By the late 1960's students at all levels of education were receiving instruction under computer control. Centers of CAI activity in the United States included the Universities at Illinois, Brigham Young, Florida State, Texas, Stanford, California-Irvine, and many other sites (3,4,7,11,13).

Five major approaches to CAI research, described by Karl Zinn of the University of Michigan, have been tried in the United States with varying degrees of success (19). These can be classified as the information science, engineering, behavioral science, educational technology, and curriculum approaches. Researchers in each field have provided data which should be useful in the design of future CAI systems.

### Information Science

Applications from artificial intelligence, natural language processing, and speech recognition are among the contributions of information science. As a rule, information scientists are less interested in educational techniques and practices than in their research topics, but their results have been useful. The techniques developed by information scientists such as Jurg Nievergelt (19) have led to some of the most significant curriculum materials currently available at universities.

### Engineering

The engineering approach combines elements of computer science, educational technology, and common sense. A major strength of this approach is shown in projects such as PLATO which respond to the needs of authors and students. Systems developed in this manner are generally usable, balancing economics and capability, and convenience of predefined strategies with flexibility through user definition of strategies. A notable weakness in the engineering approach is that it rarely incorporates basic principles from psychology or computer science, but finds a strategy that works. A prime example of the engineering approach is the PLATO system, discussed in more detail later in this chapter (7).

### Behavioral Science

Theoretical foundations, analysis of learning data, and building of mathematical models for learning are the focus of behavioral

science studies. Research on instruction has led to new techniques for development of instructional materials. Thus far most of the theories and models have not been sufficiently developed to make extensive use of the capabilities of the computer. The TICCIT project, a joint effort between the MITRE Corporation and Brigham Young University, is based on instructional materials prepared according to a set of rules of effective instruction developed by Victor Bunderson (13).

### Educational Technology

The source of much of the curriculum development activity has been educational technology. Courses developed using this approach feature multimedia presentations and programmed instruction books. The author uses standard strategies into which he may insert questions, anticipated answers, and practice exercises. Since the information is available only through computer-controlled media, this method of presentation will remain expensive as long as computers remain costly. One such project is located at Stanford University where their computing system disseminates instructional computing to centers throughout the country. Included are materials for young children, learners with special difficulties, and university courses (3,19).

An example of the Stanford activity is the CAI course developed for the Slavic languages department (19). Several specialized courses have such low attendance that the department cannot afford to staff them. Tutoring, drills, and exercises are offered, thus allowing the professors to handle a larger number of students and a greater variety of courses than possible previously (19).

### Curriculum Approach

Emphasized in the curriculum approach is subject expertise in which the teacher/author takes a central role as subject expert in creating materials and determining their uses. Mobilization of resources has been accomplished through an Annual Conference on Computers in the Undergraduate Curriculum. Student use of the computer as a tool in simulation, modeling, and problem solving is favored (19).

### PLATO and TICCIT

The two most extensively funded, most sophisticated, and most intensely scrutinized CAI projects to date are PLATO and TICCIT. Results from the projects will be of interest to designers of future CAI systems because of the differences in the design philosophies of PLATO and TICCIT.

In 1959 the University of Illinois received a 5 million dollar grant from the National Science Foundation, matched by the State of Illinois, to develop a CAI system. Four generations of PLATO have followed in the intervening 20 years. PLATO is designed to be easily used by teachers to develop their own lessons (courses). The programming (author) language, called TUTOR, can be learned quickly by people who are not computer-oriented. Although the rudiments of TUTOR are simple and an author can begin to write courses after a short orientation period, complex lessons require that the author be familiar with a substantial portion of the language (7).

In contrast to the PLATO philosophy, the courseware of the TICCIT system is designed for the teacher by professional programmers (13). The TICCIT program was begun in 1968 by the MITRE Corporation with the idea that coupling color television displays to a time-sharing computer system would result in a CAI system economical enough to permit individualized instruction to a large number of students. The courses are developed by MITRE within a rigid framework, not tailored to the needs of a particular school or business. Richard P. Morton of the MITRE Corporation explains the TICCIT philosophy.

The TICCIT courses are mainline courses. That is, most students receive all of their instruction from the TICCIT system. The role of the teacher is changed to that of a tutor, advisor, diagnostician and problem solver for individual students whenever the need arises.

Another major difference between PLATO and TICCIT is in the economics of the two projects. Where PLATO began as an experiment without consideration to cost effectiveness, TICCIT was designed with emphasis on economy. The difference in philosophy results from the fact that TICCIT was started as a commercial effort of a corporation and PLATO grew from experimentation in an educational environment.

### III. MOTIVATION FOR USING CAI

Some of the questions many educators are asking are, "What is computer-assisted instruction, and can the computer really teach? What is the best way to present material to students to insure learning and retention? How does a student learn?"

According to Peter M. Dean of International Business Machines, learning requires participation by the student (6).

Educational theory suggests that learning does not take place unless the learner himself is overtly involved in some interaction with that which he is attempting to learn. A covert interaction which only involves recognizing or assuming that one knows an answer does not result in learning.

Dean claims that the important thing about tutorial interaction is not the cleverly designed question but the presence of an external person (or computer) who forces the student to study the subject matter or answer the questions.

The study pattern of most college students is to procrastinate for most of the term and to study very diligently immediately before the final examination. Many teachers attempt to force the student to study earlier by testing more frequently. Most find that testing is necessary to keep the students motivated.

A good CAI system then is one which meets the following criterion: provides individualized instruction to large groups of people; breaks material into small, manageable units on which the student can be tested; allows the student to proceed at his own pace through the course; promotes learning by compelling the student to spread his study over a long period of time rather than feverishly studying before a test; and provides positive rather than negative feedback to the student having trouble.

#### IV. RELATED PROJECTS AND STUDIES

Research leading to the development of CAI systems to teach programming has taken two main directions. On the one hand there are the language-specific projects, those intended to provide instruction in a particular programming language. On the other hand there are those which teach flow charting or structured design, with the choice of languages of secondary importance. Combining these approaches are hybrid systems which contain elements of both approaches. CSCAI1, the system described in this thesis, falls into the hybrid category.

##### IPAS

The Interactive Program Advising System of the University of Illinois is designed to help the student understand FORTRAN (10). Its goals are to eliminate conceptual errors held by the student, increase the student's understanding of language constructs, and increase program readability and efficiency. The system is unaware of the problem the student is attempting to solve and does not attempt to direct the student toward writing a correct program. Instead it comments on the constructs the student has used while coding his program.

The basic tool of IPAS is a set of equivalence transformations which convert the student's statement (or series of statements) into an equivalent set. IPAS asks the student whether or not the equivalent set has the meaning he intended. If not, the student is requested to correct whatever error is involved.

IPAS is used primarily as an alternative to the overcrowded consulting laboratory at the University of Illinois. When a student asks for assistance, a human consultant often suggests an approach radically different from the student's algorithm, thus imposing the consultant's style on the student. IPAS works on code the student has already written, pointing out potential errors but leaving the basic algorithm intact. Using this approach the student benefits from being able to develop his own programming style.

#### BATS

Unlike IPAS, the Basic Automated Teaching System of Pontificia Universidade Católica in Rio de Janeiro is a self-contained course, teaching the elements of Portuguese BASIC (8). Its structure is much like that of CSCAII although students using BATS receive all their instruction from the textbook and the terminal instead of attending lectures.

The BATS system is divided into a series of lessons consisting of an introductory unit, the lesson core, the lesson summary, the lesson review, the quiz, and the evaluation questionnaire. The course is offered as a predetermined sequence of lessons because of the limited memory of the host minicomputer. The student can break out of the rigid framework only by requesting a summary or review of a previous lesson.

The main challenge of a programming course comes from teaching the translation of the solution of a problem into an algorithm. BATS uses a structured programming approach, with several of its lessons

dedicated to problem resolution. As in CSCAII, examples are given showing how to solve a problem using a systematic way of thinking. BATS also offers other exercises similar to those in CSCAII: (1) predicted PBASIC response—the student analyzes a set of commands and predicts the outcome; (2) constructed PBASIC commands; (3) several types of multiple choice exercises. The student prepares his algorithm off-line and submits the program to PBASIC, which is under the control of BATS.

#### U. S. Naval Weapons Laboratory CAI System

Dennis Conti reports on an interactive system to teach FORTRAN (5). An existing FORTRAN interpreter, DISPLAYTRAN, is incorporated into a CAI environment to provide a system for tutoring students in FORTRAN. The student receives drill-and-practice exercises similar to those in the analysis section of CSCAII; that is, free-form responses are accepted. In most cases the responses are FORTRAN statements which are checked for correctness by DISPLAYTRAN. A record is kept showing the concepts with which the student had trouble so that he may review them the next time he signs on. Whenever he wishes, the student may ask for a review of any section of the course. At appropriate places during the course, the system will automatically cause a review to take place for sections which troubled the student previously.

#### BIP

The BASIC Instructional Program of Stanford University is a stand-alone, self-contained course in BASIC programming at the high school/college level (3). It consists of a monitored BASIC

interpreter, a curriculum of 100 programming problems of varying difficulty, a HINT system which gives textual and graphic aid in problem solving, individualized task selection based on a model of the student's skill, and a student manual.

The primary emphasis of BIP is placed on the solution of programming problems, called tasks. The student is guided through the selected task by BIP, which first suggests prerequisite reading in the student manual. Hints may be requested at any time when additional information is needed. Subtasks isolate a part of the main problem as a smaller programming problem, helping the student reduce the main task to smaller, more manageable parts.

The error detection facilities of BIP include four different capabilities: syntax checking; execution monitoring; detection of program structure errors; and detection of errors related to the current task (those which make the program an unacceptable solution to the problem). BIP also contains two tracing facilities. TRACE allows the student to follow the execution of his program. As each statement is executed, its line number appears on the CRT screen. FLOW allows line-by-line execution under control of the carriage return key. When an IF or GOTO is executed, an arrow is drawn on the screen to indicate the transfer of control.

### TeachEASY

The TeachEASY system of the University of Iowa is a tutorial program which teaches the basics of the locally developed Easy

Assembler System (17). The implementation uses a variety of approaches interwoven into a short course including a prerequisite test, a student handbook, review request, assignment of student programming, step-by-step analysis of one program, and acceptance of student programs for the EASY system.

The tutorial concentrates on concepts relating to computer organization: general machine structure, instructions, their codes, and the action they produce; operations; arithmetic, branching, indexing; input and output; and data representation. When the student has completed the tutorial section, he is allowed to sign on in the interactive mode and enter his EASY program.

Prior to the experimental period, a quiz was given over previous unrelated material and another quiz which indicated a universal lack of knowledge concerning the concepts to be covered in the course segment. After the experiment another test was given relating to EASY concepts taught during the experiment. The results indicate that the students in the experimental group performed as well or better than those in the control group.

In contrast to the previously discussed projects are three which teach fundamental skills required for programming without concentrating on a particular programming language. They are more concerned with teaching structured programming style and understanding of algorithm formulation than with the syntax and capabilities of a language. The authors of these courses maintain that the student who is well-grounded in the fundamentals can translate his flow diagrams into the programming language of his choice.

### Temple Flow Diagram System

The computer-aided flow diagram teaching system discussed by Elliot B. Koffman and Frank L. Friedman of Temple University ascribes to this philosophy (12). The goal of the system is "to assist students in developing a systematic approach towards using the computer to solve problems."

Koffman and Friedman describe the process which should be followed by students prior to coding their programs, a systematic approach which produces programs with fewer errors.

1. Understand the problem
2. Identify the problem (program) inputs and outputs
3. Describe all program variables beforehand in a Data Table. Specify their attributes, use, and mode of definition in the program
4. Write a careful, precise, description of the steps to be performed by the program (algorithm)
5. Flow Diagram, this algorithm (4 and 5 can be combined)
6. Redefine this flow diagram in successive stages until its conversion to code becomes "transparent."

Unfortunately, most students ignore this procedure and leap ahead into coding. The flow diagram teaching system forces the student to perform the six preceding steps before attempting to write the program. The system acts as a "knowledgeable assistant helping to organize one's program rather than as a teaching system per se." It asks questions which are intended to force the student to think clearly about his solution and eliminate sources of difficulty before they turn into program bugs.

A student/machine dialogue allows the student to input his flow diagram in a top-down manner, refining each step as is necessary.

The steps are numbered initially in increments of 10 and the refinements are defined in steps of 1. For instance, the first level of a flow diagram may contain a box saying COMPUTE MEAN. The computer will tell the student that this box needs refinement, and the student will instruct the computer in the refinement.

When the flow diagram has been refined as much as possible, the student is allowed to convert his diagram into a program. In this case, the language is a version of structured FORTRAN. If the program does not produce the desired result, the programmer has the option of redefining the flow diagram and rerunning the program. In the future, implementers will modify the system to permit automatic generation of the program from the flow diagram. Thus the student could modify his program simply by changing its flow diagram.

### SPIDO

The SPIDO system, described by Christian Pellegrini of the University of Geneva, resembles in design the Temple flow diagram system (14). The student designs his program in flowchart form and enters it to the system, which translates it into the PASCAL language.

SPIDO operates on a set of symbols separated into two subsets, simple and composite. Simple symbols are used for the following purposes: sequence of instructions; reference for execution; and reference for insertion. Composite symbols are subdivided into two groups—repetitive statements, including repetition and loop, and selective statements, including conditional action and multiple choice

action. By using this set of symbols and attaching comments and instructions to them, the system can produce a compilable program.

Writing a program with the aid of SPIDO is done in two independent parts, development and production. In the development part the user groups the symbols to build flowcharts of the modules constituting the program. The production part consists of interpretation of the flowchart and generation of a compilable program by SPIDO.

The development part is subdivided into two phases, creation and update. In the creation phase the user creates a new module which will be added to those already existing in his program. All the data concerning the structure, both the intrinsic and associated information pertaining to each symbol in the flowchart, are stored in central memory for quick accessibility. The user can catalogue the module when it is complete.

The update phase allows modifications or subsequent development of an existing module. The user retrieves the module from the catalogue, makes his modifications, and recatalogues the module. When he is satisfied that the flowchart is complete, he releases it to the program generator.

#### MATHCALC

Alfred Schmitt of the University of Karlsruhe in West Germany authored a CAI program called MATHCALC, which teaches elementary programming concepts (15,16). MATHCALC is described as an interactive programming language which reflects mathematical structure and mathematical reasoning as far as possible.

MATHCALC simulates the methods used by the mathematician—start with definitions, build a theorem from the definitions and previously defined theorems. The student enters into a dialogue with the machine. During this dialogue he can enter function definitions and ask for an evaluation of the functions. Once he has defined a function, he may use it in the definition of a more complex function. Recursive definition of functions is encouraged because of its mathematical elegance. The student learns MATHCALC by working with it interactively, not by being taught as in the traditional CAI system.

CSCAII contains elements of many of the studies discussed in this chapter but also investigates some previously unexplored territory. Its structure is linear, presenting material in the same sequence it is studied in class. A similar approach is followed by BATS, which also contains many programming examples like those in CSCAII.

After finishing a lesson in CSCAII, the student is allowed to review the remedial section, the analysis section, or both. Other studies, particularly the U. S. Naval Weapons Laboratory and BATS projects, allow review of the important concepts which have just been presented.

CSCAII is slanted toward FORTRAN programming, but general programming techniques and style are also stressed. In that respect CSCAII resembles the TeachEASY, SPIDO, Temple flow diagram, and MATHCALC systems, where computer organization is stressed as much as the programming language being taught.

Unlike many of the other studies, CSCAII includes special printed material which accompanies the terminal sessions. Some of the other projects require manual reading in conjunction with the terminal activity but none of the manuals takes an approach as unconventional as that in CSCAII. The activities of the fictitious Computer Science class of Mr. Byte should amuse the student who is experiencing many of the same problems as the mythical students. At the same time, the student may find that he can better understand the material by reading the handouts and proceeding through these CAI modules.

## CHAPTER II

### OVERVIEW OF PROJECT

#### I. THE PROBLEM

##### Statement of the Problem

The purpose of this study is to explore the usefulness of a sequence of CAI modules as an aid in teaching FORTRAN to students in Introductory Computer Science Courses.

##### Hypotheses

The following null hypotheses were set forth, to be tested at the 5 percent level of confidence:

1. There is no significant difference in understanding and retention of FORTRAN-related and programming-related concepts between the experimental group and the control group.
2. There is no significant difference in performance (as measured by the final course ranking) between the experimental group and the control group.

##### Need for the Study

The traditional introductory programming course at most colleges and universities consists of two or three lectures per week in which the fundamentals of the language are taught. Several programming problems are assigned in order to reinforce the concepts taught in class. Such an arrangement is not totally satisfactory because

programming is an activity learned best in a hands-on environment. Students often have difficulty in transferring the ideas and techniques discussed in class into workable computer programs. Consulting laboratories attempt to bridge the gap by providing assistance to students who experience programming problems.

The introductory courses in Computer Science at the University of Tennessee are structured in the traditional manner, with two or three lectures per week and four to six programming assignments. The final grade for the course depends on the student's performance on the programming assignments and on two or three in-class examinations. Usually, the examinations are weighted more heavily than the programs in computing the grade, even though the number of hours spent preparing the programs often exceeds the number spent studying the concepts and preparing for the examinations.

Since the most effective way to learn to program is by programming, it is expected that the student who completes his assignments will also perform well on the examinations. Such is not always the case. The overworked tutors in the consulting laboratory sometimes give too much help to students who seek their assistance. Students who complete their programs in this manner lose much of the benefit of writing a program and thus usually perform poorly on the examinations.

The computer-assisted instruction approach attempts to give additional individual attention by allowing the student to proceed at his own pace through the course. Since the Computing Center provides a large number of terminals on campus for up to 19 hours per day, the

student has ample opportunity to schedule terminal sessions at his convenience.

The computer guides the student through material he has already seen in class in an attempt to reinforce his learning, quizzing him on both general programming concepts and constructions specific to FORTRAN. If the student demonstrates that he understands the concepts, he is allowed to proceed quickly through the lesson. If he needs some review, the computer provides him with appropriate remedial material. It is hoped that the individual instruction a student receives will help bridge the gap between the classroom and the programming laboratory.

## II. PROCEDURES

### Description of the Program

A seven-section course has been developed by the author to teach FORTRAN. Each section (lesson) of the course consists of a short handout (5 to 15 pages) and a corresponding module written in COURSEWRITER III. Each of the modules is written in a conversational tone in order to motivate the students as well as help them better understand the concepts being discussed.

The first lesson is devoted to familiarizing the student with the cathode ray tube terminals on which the course is presented, and the remaining lessons teach six programming concepts. Figure 1 describes the student's path through the course, which has been given the name CSCA11. The lessons in CSCA11 are presented in the same sequence as they are presented in class so that the student can go to the terminal as soon as he has attended the appropriate lecture.

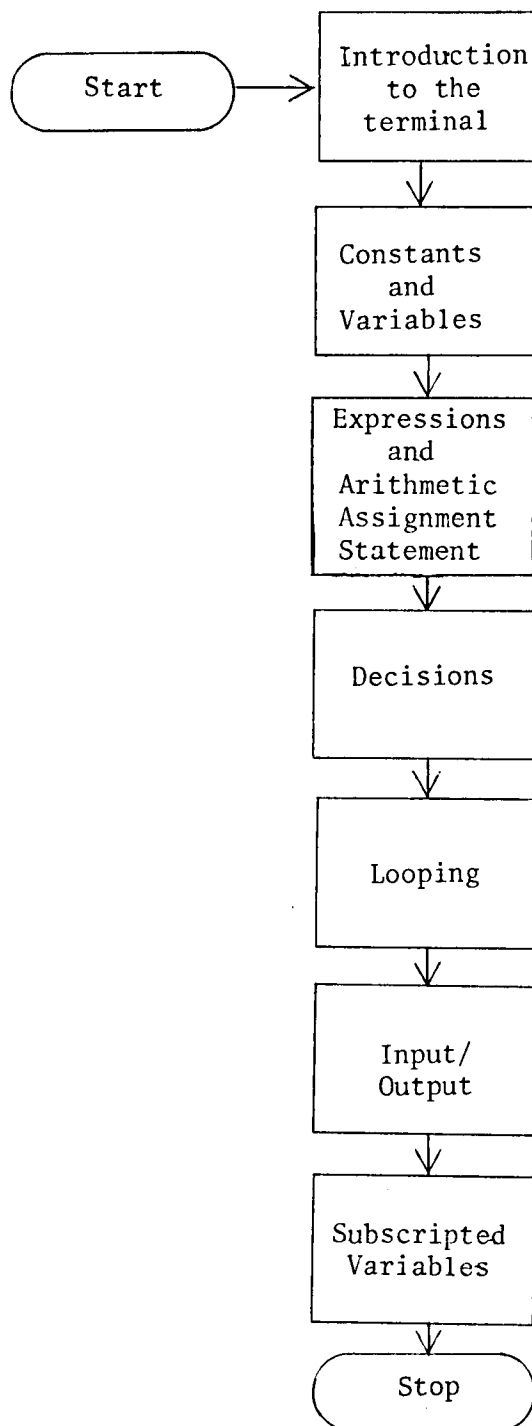


Figure 1. Flowchart of CSCAI1

CSCA11 is intended to be used as a means of reviewing the broad concepts discussed in class. For example, the instructor usually introduces FORTRAN constants and variables during the first week of the quarter. For best results, the student should promptly read the handout on constants and variables and go to the terminal to work through the module. When he has finished the lesson, the program instructs him to sign off. When he signs back on, he will be restarted at the beginning of the next lesson.

Typical introductory programming texts are basically manuals, describing the syntax of the language and giving examples of syntactically correct statements. They explain the rules for writing correct programs, and generally include program segments which serve as models of good programming style. The handouts for CSCA11 are a departure from the manual approach. Prepared in a light-hearted vein, they attempt to attract the attention of the student by relating programming concepts to situations in everyday life.

The handouts are centered around the Computer Science class of graduate student Jasper M. Byte. In each handout a student in that class attempts to relate the programming concept being studied to some totally unrelated experience in his life. For example, in "George Meets the Loop: the Story of a Daydreaming Student," George routinely daydreams in class and incorporates the subject of the lecture into his dream (see Appendix A). Mr. Byte's lecture on loops sends George into a reverie about his unsuccessful fishing experiences of the previous summer. By representing his week-long

fishing expedition in flowchart form, George relates the subject of looping to an incident from his experience.

Each of the handouts (excluding the introduction to the terminal) contains a similar situation. It is hoped that such an approach will stimulate interest in what is often the most boring part of a programming course, the text. Since each module refers extensively to its associated handout, the student is expected to read it prior to signing on and to refer to it during the terminal session.

The first lesson, "Introduction to the Terminal," demonstrates the operation of the cathode ray tube (CRT) terminals which are distributed across the campus. The most prominent CRT types, DECSCOPE of Digital Equipment Corporation and ELITE 1520 of Data Media, are online to the Digital Equipment Corporation PDP-10. COURSEWRITER III, which runs on the IBM 370/148, is available through the interface between the PDP-10 and the 370/148.

Since many students have had no previous experience with a CRT and may be apprehensive about using it, the introduction attempts to relieve some of their fears. It describes the facilities of the terminal necessary to complete CSCAI1 without giving an exhaustive demonstration of the terminal's capabilities. If the student lacks confidence after completing the introduction, he may repeat it until he feels ready to continue.

The conversational tone of CSCAI1 is informal, in keeping with the approach followed in the handouts. The machine/student dialogue takes the form of a friendly computer whose sole purpose is to teach

FORTRAN to its new protege. When explaining a concept, the computer often slips into the traditional teacher/student mode but quickly returns to more informal conversation when the point has been made. The student who is apprehensive about "talking" to a machine before beginning CSCAII should be put at ease by the friendly conversation.

When the student feels comfortable with the CRT, he is ready to proceed to the six lessons teaching programming style and FORTRAN fundamentals. Each of these lessons is structured as in Figure 2.

The pretest and posttest are 10 question multiple-choice tests constructed randomly from a pool of at least 20 questions. The purpose of the tests is to monitor the student's understanding of the concepts presented in the lesson, both before and after the presentation. The COURSEWRITER recordkeeping facility allows storage of the student's performance on the test and the amount of time spent answering the questions.

In the remedial section the computer assumes a tutorial role and reviews the basics of the appropriate concepts. Included are a detailed review of the syntax of the FORTRAN statements being studied and several examples of their use.

The analysis section is presented in tutorial and drill-and-practice modes. In the early lessons this section consists mainly of recognizing and coding valid statements. Later lessons ask the student to analyze program segments, checking for logic errors. CSCAII helps the student to localize the error by simulating the step-by-step thought process followed by professional programmers in debugging their

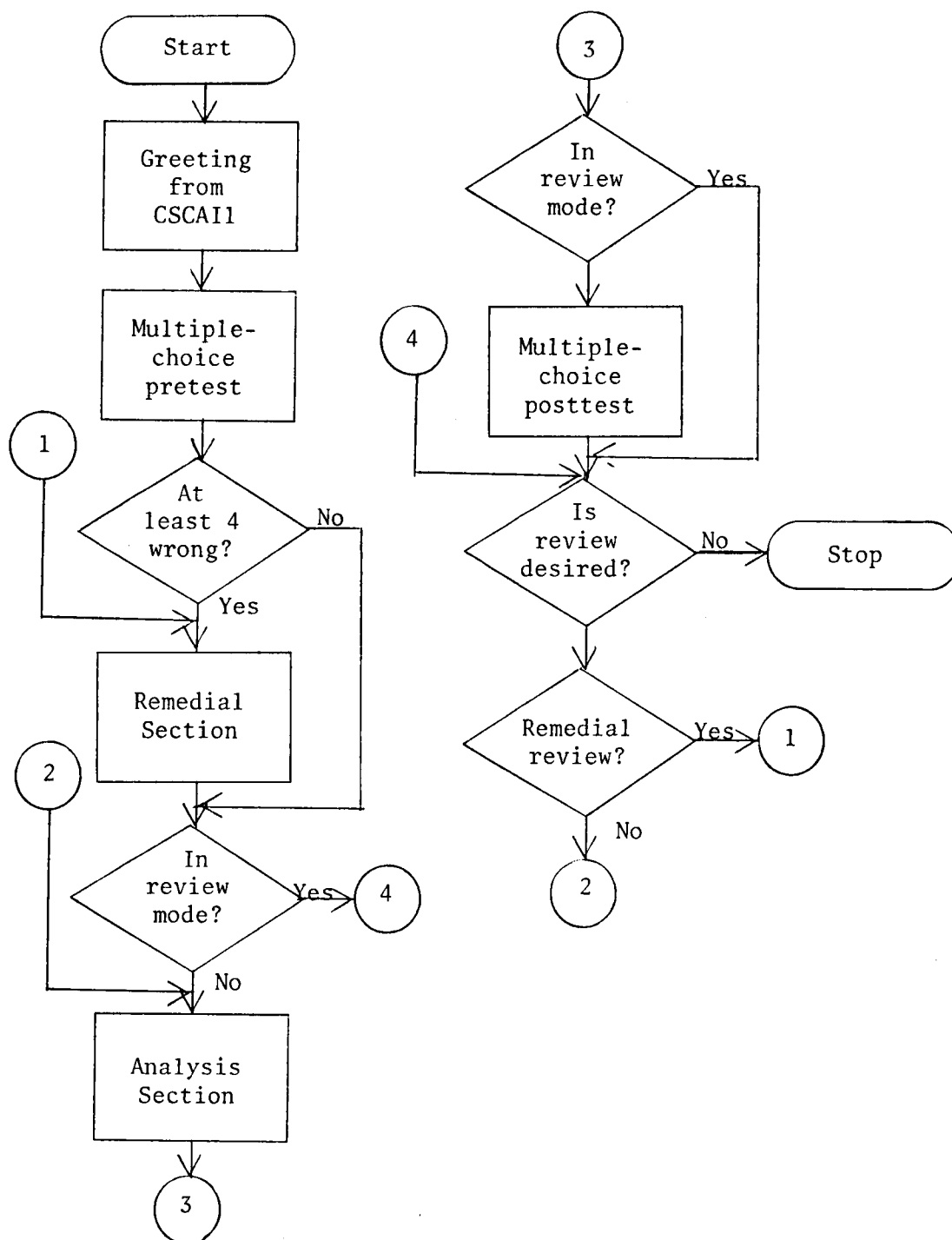


Figure 2. Flowchart of CSCAI1 Lesson Design

own programs. If the student cannot determine the location of the error after the simulation, CSCA11 points it out and explains how to correct it.

If the student desires more review after the posttest, he is allowed to return to earlier portions of the lesson. He may repeat the remedial section, analysis section, or both as many times as he thinks is necessary.

### Assignment of Subjects

Computer Science 1510, oriented toward liberal arts students, serves as the first course for Computer Science majors. Students who have chosen to take 1510 are generally highly motivated. Algorithm formulation and computer organization are emphasized as strongly as the fundamentals of FORTRAN.

Students in two sections of Computer Science 1510 were chosen as possible subjects for this study during Winter Quarter of 1979. The author solicited volunteers from the two sections, cautioning the students not to volunteer unless they felt certain that they would complete CSCA11. A total of 40 agreed to be subjects. The pool of volunteers was randomly divided into two groups, one control group of 20 students and one experimental group of 19. The experimental group originally contained 20 students, but one dropped the course. Only volunteers were included in the study because they could be considered a homogeneous group.

### Administration of Treatments

Both the control group and the experimental group received regular classroom instruction in FORTRAN and performed the programming assignments.

Although they were advised of the general time frame in which to complete each lesson of CSCAII, the students in the experimental group had flexibility in scheduling terminal sessions. The University of Tennessee Computing Center facilities were available from 9 a.m. until 4 a.m., Monday through Saturday, and 12 noon to 7 p.m. on Sunday. Except for brief periods of hardware or software failure, CSCAII was available to the students during those hours.

Students in the experimental group were assigned student numbers which were registered to CSCAII. Registration of a student gave him access to CSCAII and set aside a unique area on disk for storage of student data. This area contained a pointer to the student's current position in the course and kept a record of his performance on the pretests and posttests in CSCAII.

## CHAPTER III

### PRESENTATION AND INTERPRETATION OF DATA

#### I. CSCAII STATISTICS

The COURSEWRITER recordkeeping facility allows storage and subsequent retrieval of data pertaining to student performance in a course. CSCAII kept the following statistics on the experimental group: amount of time logged on; score on the pretest and posttest for each section of the course; and time spent on each test. Results are shown in Table 1.

In each section of CSCAII the mean score on the posttest was higher than on the pretest. Since the tests were randomly generated, the average student should have received a mix of simple and difficult questions. Thus, the consistently better performance on the posttests indicates that learning has occurred.

Not only did the students answer more questions correctly on the posttests than the pretests but they also completed the posttests more quickly. The difference is probably attributable to several factors. One such factor involves familiarity with the operation of the terminals. As the student works through a CAI module his speed should increase with experience. Another reason for increased speed involves the composition of the tests. The pool of questions from which the tests are formed is small enough so that some questions will appear on both tests. The student should remember these questions

Table 1. CSCAI1 Internal Test Results

| Subject Area:                            | Pretest Averages   |                   | Posttest Averages |        |
|------------------------------------------|--------------------|-------------------|-------------------|--------|
|                                          | Score <sup>a</sup> | Time <sup>b</sup> | Score             | Time   |
| Constants and Variables                  | 9.00               | 220.76            | 9.65              | 141.47 |
| Expressions and<br>Assignment Statements | 7.88               | 243.82            | 9.24              | 171.12 |
| Decisions                                | 8.12               | 248.29            | 9.14              | 141.71 |
| Looping                                  | 8.12               | 236.71            | 9.47              | 177.24 |
| Input/Output                             | 9.41               | 190.06            | 9.82              | 105.47 |
| Subscripted Variables                    | 7.20               | 209.30            | 8.38              | 142.86 |

<sup>a</sup>Out of a possible 10.

<sup>b</sup>In seconds.

and thus answer them more quickly than he answers new questions. A final reason is that learning may have occurred. The student may have better understood the concepts discussed in the module he has just taken, and thus answers the questions more quickly than before.

Assuming that learning has occurred, it is important to determine whether or not the student retains this knowledge over a period of time. If it can be demonstrated that the average student retains this information, then CSCAII can be termed successful. The experiment discussed in the next section of this chapter attempts to make this determination.

The average student in the experimental group spent approximately five hours logged on to CSCAII and an additional two and one-half hours reading the handouts. The times varied only slightly from student to student so that the group can be considered fairly homogeneous.

## II. THE EXPERIMENT

### Experimental Subjects

The subjects consisted of 39 students enrolled in two classes of Computer Science 1510 at the University of Tennessee during the Winter Quarter of 1979.

### Analysis of Data

The subjects were divided randomly into an experimental group of 19 students and a control group of 20 students. Originally, each group contained 20 students, but one member of the experimental group dropped the course. Each group attended lectures, performed programming

assignments, and took examinations on the concepts discussed in class. Additionally, students in the experimental group participated in CSCA11.

Following the experiment, all students were given a posttest designed to measure their understanding of basic programming concepts (see Appendix E). General topics as well as those specific to FORTRAN were tested. The test consisted of 35 questions of which 10 were multiple-choice, 15 were true-false, and 10 were short answer. Students were allowed 30 minutes to complete the test. Most students were able to finish within the time limit.

The results of the test were analyzed at the 5 percent level of confidence. In addition, the final grade in Computer Science 1510 of each subject was made available for analysis at the 5 percent level of confidence.

For the posttest and the final grade, the students were ranked on the basis of their scores. A two-sample t test was performed to determine the validity of the two null hypotheses (see Appendix C). The formula for this test is:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - \delta}{\sqrt{(n_1 - 1) s_1^2 + (n_2 - 1) s_2^2}} \sqrt{\frac{n_1 n_2 (n_1 + n_2 - 2)}{n_1 + n_2}}$$

For the null hypotheses being tested in this thesis,  $\delta = 0$ .

The results of the analysis are shown in Table 2 and Table 3. In this experiment, a t statistic of absolute value greater than 1.96 indicated a significant difference not attributable to random chance at the 5 percent level of confidence.

Table 2. Test of Hypothesis 1

| Group:                                   | Control      | Experimental |
|------------------------------------------|--------------|--------------|
| Number of<br>Participants - $n_1, n_2$ : | 20           | 19           |
| Mean Scores - $x_1, x_2^a$ :             | 25.40        | 27.05        |
| Variances - $s_1^2, s_2^2$ :             | <u>11.73</u> | <u>9.72</u>  |
| $t^b$ :                                  | 1.57         |              |

<sup>a</sup>A perfect score is 35.

<sup>b</sup>Not significant at 5 percent level of confidence with 37 degrees of freedom.

Table 3. Test of Hypothesis 2

| Group:                                   | Control     | Experimental |
|------------------------------------------|-------------|--------------|
| Number of<br>Participants - $n_1, n_2$ : | 20          | 19           |
| Mean Scores - $x_1, x_2^a$ :             | 3.10        | 3.63         |
| Variances - $s_1^2, s_2^2$ :             | <u>1.25</u> | <u>0.69</u>  |
| $t^b$ :                                  | 1.68        |              |

<sup>a</sup>On a 4.0 scale.

<sup>b</sup>Not significant at 5 percent level of confidence with 37 degrees of freedom.

The first hypothesis concerned the performance of the two groups on the 35 question posttest. The experimental group scored better on the average than the control group but the resulting t score of 1.57 did not indicate that the difference was significant.

The second hypothesis concerned the performance of the two groups in Computer Science 1510. Grading is done on a 4.0 scale with quality points assigned in the following manner: A = 4.0; B = 3.0; C = 2.0; D = 1.0; and F = 0.0. Again the experimental group proved superior to the control group but the difference was not great enough to be considered significant. For Hypothesis 2 the test statistic was 1.68.

### III. INTERPRETATION OF DATA

The results of this experiment do not strongly suggest that computer-assisted instruction is an effective tool for reinforcing learning in elementary programming courses. Although neither of the null hypotheses could be rejected, the experimental group was slightly favored in each case. The main purpose of a CAI project of this type is to improve student comprehension of the subject matter and to promote its retention over a period of time. Consistently superior performance on the CSCAI1 posttests indicates improvement in comprehension for the short term. The experimental results suggest that students participating in CSCAI1 retained at least some information presented in the modules over a span of several weeks.

Several important limitations to this study should be noted. The study reflects only the effect of a particular computer-assisted

instruction program on a relatively small group of volunteers—39 students enrolled in an elementary FORTRAN programming course at the University of Tennessee. The findings would not necessarily be applicable to other CAI exercises with similar purposes.

CSCAII was designed to reinforce learning of the concepts of elementary programming. It is not a replacement for any classroom instruction but only a supplement. If CSCAII had been more comprehensive, taking the place of some or all classroom instruction, a better comparison could have been made than that performed in this study.

Complete replacement of classroom instruction by an expanded version of CSCAII would create a much better test of the value of CAI in teaching programming. The CAI program would then be forced to stand on its merits against traditional classroom instruction. Any statistical differences could be attributed to differences in the methods of instruction. In the present study, the origin of the slight difference in favor of the experimental group is not clear. Possibly, participation in CSCAII forced these students to spend more time reviewing the material than they would have otherwise spent. Perhaps, the drill-and-practice exercises served to remind the students of their weaknesses, prompting them to review their manuals. The novelty of using a cathode ray tube for the first time and talking to a computer may have enticed the students to work harder. Since the difference between the two groups is not statistically significant, however, the difference may be completely explained by chance.

In addition, the hypotheses were not subjected to evaluation because no such evaluation was known to exist. Findings of this study are subject to restrictions resulting from the use of unvalidated methods of evaluation.

The final examination given to all students participating in the experiment was designed as a general indicator of understanding of elementary FORTRAN programming concepts. It was compiled with little input from the instructors of the participating classes. Since each instructor presents the material from a different perspective, stressing what he or she considers to be important, performance on the test may vary because of the difference in instructors. Students of the instructor whose emphasis most closely matches that of the test designer should perform well on the test. Thus, Hypothesis 1 depends upon the performance of the individual instructor as well as his students.

Likewise, Hypothesis 2 depends upon the instructor. No two instructors assign grades in the same manner. Since the number of volunteers in one class exceeded the number in the other class, too much weight may have been given to the scores of the students from the larger pool. If all participants had been selected from one class, the results from Hypothesis 2 would have been more meaningful.

## CHAPTER IV

### EVALUATION AND SUGGESTION FOR FUTURE STUDIES

#### I. STUDENT REACTION

Students who participated in CSCA11 were given the opportunity to voice their opinions about the course in the short questionnaire they received along with the posttest (see Appendix D). Most of the participants responded anonymously although a few signed their names to the questionnaire. In general, response to CSCA11 was favorable.

##### Usage of CSCA11

Most of the students who volunteered for CSCA11 had finished the course by the time they received the questionnaire. The few who had not finished complained that Computer Science 1510 is an extremely time-consuming course even without the additional burden of CSCA11. Even so, these students lacked only one or two sections completing the course. On the average, students who completed CSCA11 spent two and one-half hours reading the preliminary handouts and five hours at the terminal.

##### Benefits of CSCA11

The volunteers were asked to comment on any benefits they derived from participating in CSCA11. They cited a variety of benefits from the course. The most prevalent response was the CSCA11 helped to reemphasize material studied in class and in some cases covered topics in greater detail. One student noted that in the traditional classroom

environment, testing carries with it a penalty for incorrect answers. CSCA11 tested his understanding of programming concepts without penalty.

Other students felt that some of the teaching techniques in CSCA11 which are not commonly used in the classroom helped them to learn. They contended that being forced to consider intentionally incorrect examples improved their program debugging skills.

Two participants considered the humanizing effect of computer-assisted instruction to be its most important benefit. They had received no previous direct exposure to computers and were put at ease by the informal dialogue.

#### The Informal Approach

Students were asked their opinions of the informal approach taken by the CSCA11 handouts and dialogues. Their comments reflected both satisfaction and dissatisfaction with the approach. Some students considered the lessons to be an entertaining change of pace, helping to keep their interest and to make interaction with the computer enjoyable. Others felt that the approach sometimes led to oversimplification of the material. One considered some of the handouts to be a waste of time.

#### Errors and Possible Improvements

Since some of the students participating in the experiment plan to major in Computer Science, it is important that they learn to evaluate existing programs and systems. They were asked to note obvious errors and suggest means of improving the program. One

deficiency that many students mentioned involved correct answer matching. In the remedial and analysis sections of each module students are requested to answer questions in their own words. COURSEWRITER allows a great deal of flexibility in answer matching, such as partial answer processing, acceptance of multiple correct responses, and editing for spaces and carriage control characters. More careful definition of anticipated responses and the design of unambiguous questions could eliminate many of these problems.

Another complaint with CSCAII was that when a student signed off in the middle of a lesson, he was restarted at the beginning of that lesson when he signed on again. In COURSEWRITER the restart point after sign off is the option of the author. If it appears that most students want to restart at the sign off point, the author can easily redefine the method.

Other students felt that CSCAII sometimes presented more material than the screen could hold and referred to material which had already disappeared from the screen. The program could have been more effectively designed for the screen if more time had been available.

Some students expressed dissatisfaction with the lengths of various sections, with the topics of looping and input/output singled out as too lengthy. One participant suggested that these sections be broken up into two or more subsections on which the student is tested.

Others felt that the user of CSCAII should be provided an index of the topics so that he can review only the material in which he is deficient. Similarly, another student suggested that the program

should recognize deficiencies and refer the student to a section of the course designed to remedy the problem.

#### Obstacles to Completion of CSCAI1

Students found few stumbling blocks to completion of the course. Terminal usage normally was not so great as to cause long waiting periods. Most of the complaints involved the terminals and the computer system supporting them. Students sometimes found the IBM 370 or PDP-10 down when they signed on. At other times they received the message "No CAI lines available," meaning that COURSEWRITER was unavailable. The PDP-10 system crashed a few times during terminal sessions, thus forcing the student to restart the lesson from its beginning. A final complaint was that during periods of heavy usage, response from the terminal degraded.

#### Value of CSCAI1 in Introductory Programming Courses

The participants were asked if computer-assisted instruction would be valuable for introductory programming courses. Most agreed that CAI is an effective technique for presenting rudimentary material. The consensus was that CAI could be useful as a complement to classroom instruction and as a means of reviewing material in more detail than in class. The CAI approach also could be effective for presenting remedial material to students who need individual attention.

### Computer-Assisted Instruction As a Part of the Curriculum

Almost unanimously, the students disapproved of the addition of computer-assisted instruction to introductory programming courses as they currently exist. These courses are among the most time-consuming of all undergraduate courses. One student suggested that CAI would be feasible as a replacement of one hour per week of lecture. Others felt it should not be required but be available as an option for remedial reasons. Another believed that making CAI compulsory would remove the incentive for self-help. CAI would then be just another requirement of the course and become a measure of understanding rather than an aid to understanding.

## II. RECOMMENDATIONS AND FUTURE FOR CAI

Possible enhancements to the existing CSCAI1 system and design considerations for similar future CAI projects are discussed in this section. Included are discussions of improved programming techniques, advances in hardware and software, and cooperation among the departments on the University of Tennessee campus.

### Enhancements to CSCAI1

Several possible improvements to CSCAI1 have been noted in the preceding student comments. CSCAI1 could also be enhanced by incorporating some of the techniques employed in the studies mentioned in Chapter I.

One of the most meaningful enhancements would be to interface CSCAII with a FORTRAN compiler or interpreter and allow online entry of short programs similar to those in the Stanford BASIC Instructional Program (3). Such a technique would probably require a complete rewrite of CSCAII in a language compatible with FORTRAN because COURSEWRITER is a self-contained language with few interfacing capabilities. Even so, the effort should be worthwhile because the immediate feedback on incorrectly coded statements promotes faster learning.

The availability of a FORTRAN compiler would allow other possibilities for improvement. A technique similar to one employed by the Illinois Interactive Program Advising System for determining the validity of sections of code could be implemented (10). A set of equivalence transformations could be developed to convert the section of code into an equivalent set. The student can then determine whether or not he has coded the statements he intended.

#### Advances in Hardware and Software

Future CAI projects can benefit from the acquisition of improved hardware and software, some of which is already on the market. The studies in Chapter I have shown that CAI systems can be developed on both large and small computers. Future acquisitions by the University of Tennessee Computing Center and the Computer Science Department will determine the direction of these projects.

For the large computer, terminals with advanced capabilities could be purchased. Light pens could allow the choice of an answer with a

simple touch to the screen. Terminals with paging capabilities would allow the author to format his text on the screen into discrete pages which could be reset on the screen by student control. Some terminals can be interfaced with audio or audio-visual equipment, thus allowing a wider range of presentations. For instance, program segments or entire programs could be displayed on a video screen at the same time as the explanatory text is displayed on the terminal.

Various departments across the campus, including Computer Science, own or lease minicomputers. Most systems include graphics terminals and magnetic disk storage devices. If demand for CAI becomes great enough, the Computing Center or Computer Science Department might purchase or lease a minicomputer to be dedicated to CAI research. Several successful CAI systems have been developed for minicomputers, including the BASIC Automated Teaching System (8).

New CAI software can come from two sources, in-house development and purchase or lease. Many packaged special-purpose courses are available from IBM and various universities. By obtaining the packaged software, the user can avoid costly duplication of effort. For some areas of research no such packages exist or their cost is prohibitive. In these cases in-house development is the answer. Currently, COURSEWRITER is the only CAI language available at the University of Tennessee. Perhaps more CAI languages will appear on the market in the future so that the prospective author will have more options.

### Interdepartmental Computer-Assisted Instruction

The development of a campus-wide CAI laboratory would be a possible future project. Such a large-scale undertaking would require promotion by departments using CAI and cooperation among departments concerning program development and utilization.

The laboratory could involve programs written for the IBM 370 and the various departmental minicomputers. A library of programs could be collected to provide remedial work in a variety of subject areas. The programs may be a mix of locally written programs and those purchased or leased. The key ingredient to success of the CAI laboratory is interdepartmental cooperation.

### Conclusions

A computer-assisted instruction program was developed to supplement classroom instruction in introductory FORTRAN programming courses. The system consisted of short handouts written with an informal, often humorous, attitude and seven computer modules teaching the elements of FORTRAN.

The present study was an attempt to determine whether or not the computer program, called CSCAI1, has a significant impact on the performance of students in an introductory FORTRAN course. Thirty-nine students from the University of Tennessee in Knoxville were used in this experiment.

The students were randomly divided into two groups. Both groups received classroom lectures and completed FORTRAN programming

assignments. The experimental group additionally participated in CSCA11. Both groups were then given a 35 question posttest. The scores of each group were tabulated and the two-sample t test was used to analyze the results. The same t test was used to compare the final grades of the students in the two groups.

The results of the experiment and the comments of the participants indicate that computer-assisted instruction can be a valuable tool for promoting understanding of elementary programming concepts. The experimental group scored better on the average than did the control group on the posttest, but the difference was not statistically significant. Likewise, the final grades of the experimental group were not significantly higher than those of the control group.

## BIBLIOGRAPHY

## BIBLIOGRAPHY

1. Allen, Michael W. "Computer Managed Instruction: A Definitive Design," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 115-122.
2. Ballaben, G. and P. Ercoli. "Computer Aided Teaching of Assembler Programming," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 217-221.
3. Barr, Avron and Marian Beard. "An Instructional Interpreter for BASIC," SIGCSE Bulletin, Vol. 8, No. 1, February 1976, pp. 325-335.
4. Bork, Alfred. "Learning with Computers—Today and Tomorrow," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 17-22.
5. Conti, Dennis M. "An Interactive System to Teach FORTRAN," SIGCAI Interface, Vol. 4, No. 2, April 1970, pp. 18-20.
6. Dean, Peter M. "Why CBI? An Examination of the Case for Computer-Based Instruction," Journal of Computer Based Instruction, Vol. 4, No. 1, August 1977, pp. 1-7.
7. Denenberg, Stewart A. "A Personal Evaluation of the PLATO System," SIGCSE Bulletin—Computer Uses in Education, No. 12, April 1978, pp. 3-10.
8. dos Santos, Sueli M. and Marilia R. Millan. "A System for Teaching Programming by Means of a Brazilian Minicomputer," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 211-217.
9. Gazzaniga, G. and L. Ironi. "Some Applications of Computer Aided Instruction to Numerical Analysis and Programming Languages," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 627-629.
10. Gillett, Will. "An Interactive Program Advising System," SIGCSE Bulletin, Vol. 8, No. 1, February 1976, pp. 335-350.
11. Homeyer, Fred C. "A Computer-Assisted Instruction Course to Teach the ELASTIC Programming Language," The University of Texas Computer-Assisted Instruction Laboratory, 1967-1968, pp. 46-48.
12. Koffman, Elliot B. and Frank L. Friedman. "A Computer-Aided Flow Diagram Teaching System," SIGCSE Bulletin, Vol. 8, No. 1, February 1976, pp. 350-354.

13. Morton, Richard P. "A Variety of TICCIT Systems—An Overview," SIGCSE Bulletin, Vol. 8, No. 1, February 1976, pp. 144-148.
14. Pellegrini, Christian. "Un Systeme Interactif d'aide a la Conception de Programmes," Ph.D. Thesis No. 1724, Universite de Geneve, 1975.
15. Schmitt, Alfred. "Simulative Transfer of Mathematical Concepts by Interactive Programming," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 811-815.
16. Schmitt, Alfred. Teaching of Elementary Programming Concepts with a Flexible CAI System. Technical Report, Karlsruhe, West Germany, 1976.
17. Sjoerdsma, Ted. "An Interactive Pseudo-Assembler for Introductory Computer Science," SIGCSE Bulletin, Vol. 8, No. 1, February 1976, pp. 342-349.
18. Su, Stanley Y. W. and Ahmed E. Emam. "Teaching Software Systems on a Minicomputer: A CAI Approach," IFIP Second World Conference on Computers in Education, Marseilles, France, September 1975, pp. 223-230.
19. Zinn, Karl L. Computer Aided Instruction in the United States. Technical Report, Ann Arbor, Michigan, 1976.

## APPENDICES

## APPENDIX A

### Introduction to the Terminal

Congratulations! You are one of the fortunate Computer Science 3159 students who will participate in our brand-spanking new supplementary course at the cathode ray tube terminal. Our terminal course will help you learn to program and have a few laughs on the side.

Seriously, folks, there is a little more to the terminal course than having some fun. You, a typical Computer Science 3159 student, are participating in an experiment which, if successful, may become a part of the curriculum. The entire course consists of a series of seven mini-courses—the introduction and six others focusing on the topics covered by your classroom instructor. Each mini-course consists of a handout and a session at the terminal. You should participate in all sessions in order to derive full benefit from the course. Participation does not guarantee a good grade but you probably will understand programming concepts better than most 3159 students.

Many of you have absolutely no experience with the mysterious black box known as the computer. You, therefore, probably have quite a few misconceptions about what a computer can do. Movie computers, such as HAL in "2001: A Space Odyssey," seem to possess human qualities like fear and self-preservation. Frankly, the Hollywood image of the computer is unrealistic. In truth, the computer is an obedient slave which performs exactly the tasks it is given by its human masters. The computer is very simple-minded. It cannot converse with people

in English, German or Spanish. Special-purpose languages with very limited vocabularies and syntax have been devised for man/machine conversations. In Computer Science 3159, you will be learning a significant portion of one of these languages—FORTRAN. The earliest version of FORTRAN appeared in the middle 1950s. Over the next two decades, FORTRAN has evolved into a much more powerful and flexible language than the original version.

The computer can directly execute only statements in a very simple language known as machine language. Therefore, the FORTRAN program you enter into the computer must be translated into machine language before it is executed. This requires that your FORTRAN program pass through another program, called a compiler, which translates it into machine language. The compiler we use in Computer Science 3159 is known as WATFIV, a compiler developed at the University of Waterloo in Canada. WATFIV was specially devised for students who are just learning FORTRAN. WATFIV gives more extensive error messages than most compilers and some extensions to FORTRAN which make programming easier.

The handout you are now reading and the associated course on the terminal are designed to give you a basic understanding of the cathode ray tube terminal operation. Take this handout to the terminal with you for reference.

These terminals can be found in various locations on campus, with the largest number in Glocker 67 and Ayres 101. The terminals in Glocker are the DECSCOPE model manufactured by Digital Equipment Corporation. Those in Ayres Hall are Elite 1520 terminals by Data Media. You can use either type in our computer-assisted instruction course.

Here are the steps you need to follow to sign on the terminal:

1. Turn on power—there is a switch on the right side of both types of terminals.
2. (for Data Media only) Press the LOCK button to force the terminal into uppercase.
3. Type PW (do not press RETURN) and the cursor will jump down a line.
4. Type in the COURSEWRITER number assigned to you by your instructor and press RETURN. The computer will respond—TYPE COURSE NAME.
5. Enter the name of the Computer-Assisted Instruction course, also given to you by your teacher and press RETURN.
6. You should then be signed on, and the computer will direct your actions from then on.

Now you are ready to sign on the terminal and learn what you need to know about terminal operation. Good luck!

#### George Meets the Loop: The Story of a Daydreaming Student

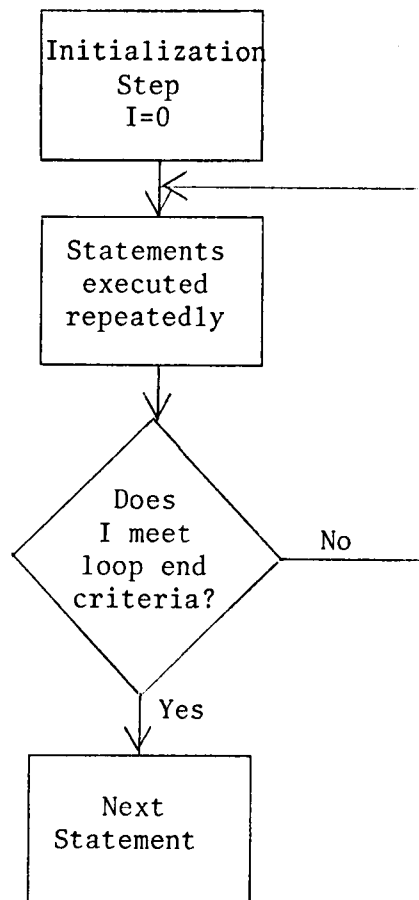
Meet George, a typical student taking Computer Science 3159. He is taking the course for the same reason that many other students are taking it. CS 3159 is required for engineering majors. Like most students, George is an accomplished daydreamer. Put George in a classroom and it is as if somebody flipped a switch in his head. He is able to hear and subconsciously absorb some of what the teacher is saying. The only problem is that what he absorbs becomes a part of his daydream. That is all right in interesting courses, but can you imagine what boring daydreams he has in Graphics? It's enough to put a person to sleep.

Computer Science 3159 is no different from any other course as far as George is concerned. George sat and daydreamed through the lessons on constants, variables, and expressions. Ho-hum! Today, the

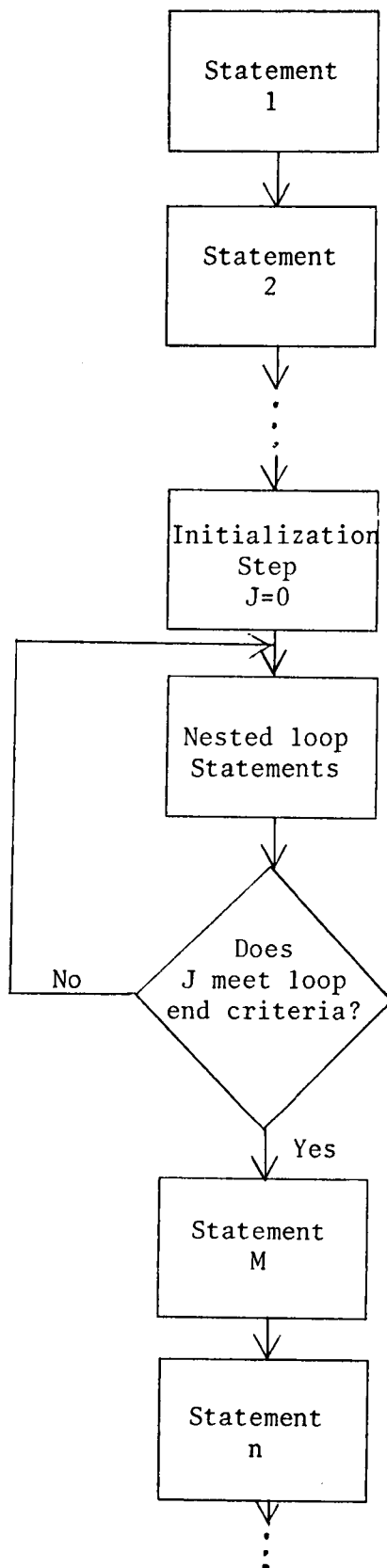
topic is loops. George's subconscious sits up and takes notice. Finally, a subject that will give it some exercise.

George's instructor, Mr. Byte, begins to explain the concept of looping. "Up to this point we have talked about statements which are executed serially. But a program consisting of only arithmetic assignment and input/output statements is not very powerful. You execute each statement once, then you reach the end of the program. You have no power to repeatedly execute these statements. The technique of looping allows you to do this. A loop consists of the following elements: (1) initialization step—generally a variable, often called the index, is given an initial value; (2) execution of the statements contained in the loop—usually included among these statements is one which alters the value of the variable you initialized; (3) test step—in this step, the index is tested to see if it meets a certain condition—if it does, the statements contained in loop are reexecuted—if not, the statement following the test statement is executed next."

Mr. Byte continues by putting a flowchart for a simple loop on the board.

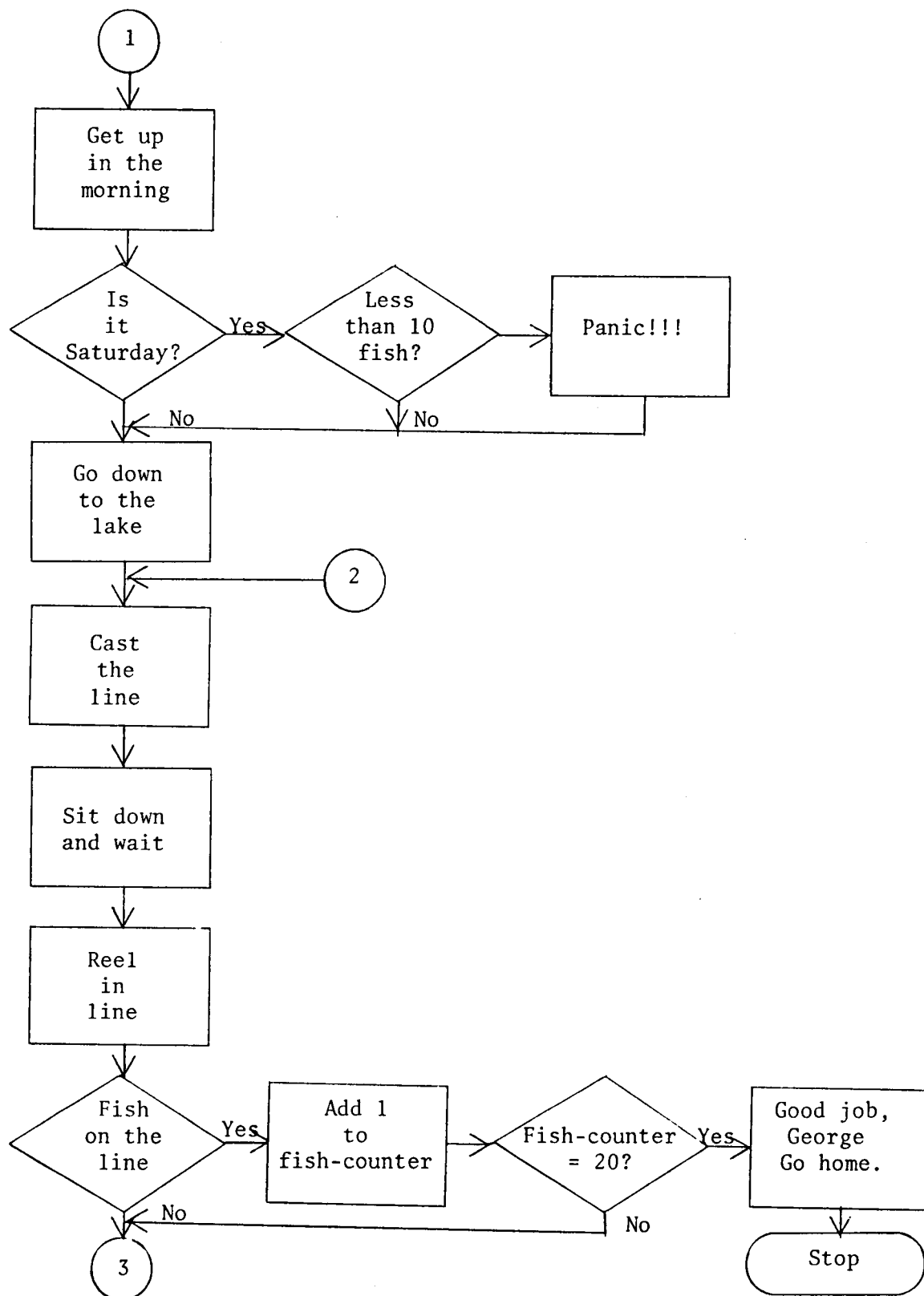


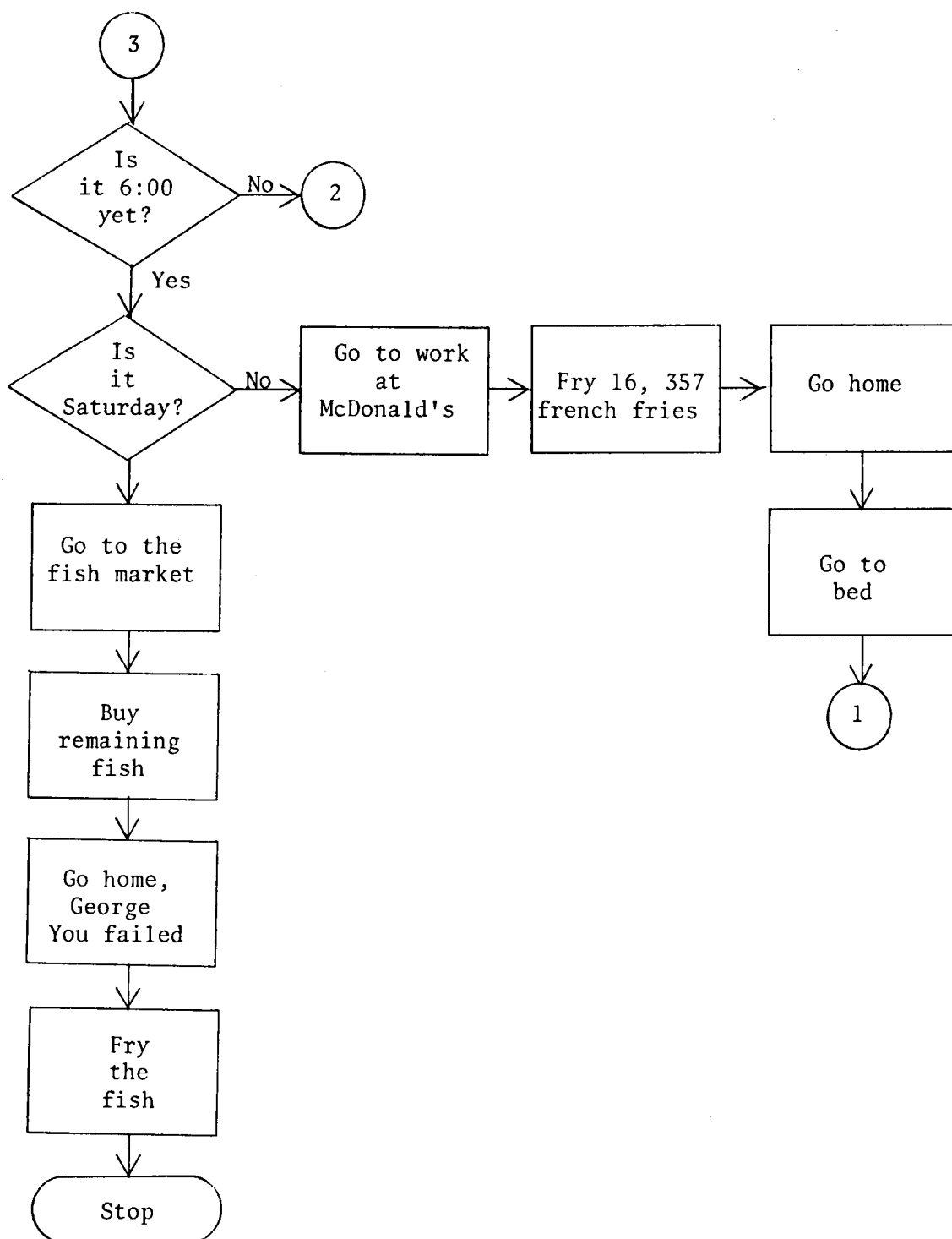
"A loop may be contained within a loop. This is called nesting. For example, the box 'Statements executed repeatedly' can be expanded like this."



By this time, George is deeply involved in a daydream about fishing. You see, George is a good old boy from East Tennessee who likes to go out on the lake, chew some Red Man, toss in a line and lie back and wait for the fish to do the work. He is remembering the time last summer that he promised a fish fry to a group of his friends. Since he had invited nine friends and since he figured that he would need two average-sized fish per person, George had to catch twenty fish by Saturday, the day of the fish fry. Since it was just Monday night, he wasn't worried. Surely, he could average four fish a day for the next five days. Mr. Byte's discussion of looping intrudes on George's daydream so that he begins thinking of that experience as a loop. The following flowchart describes his thoughts.

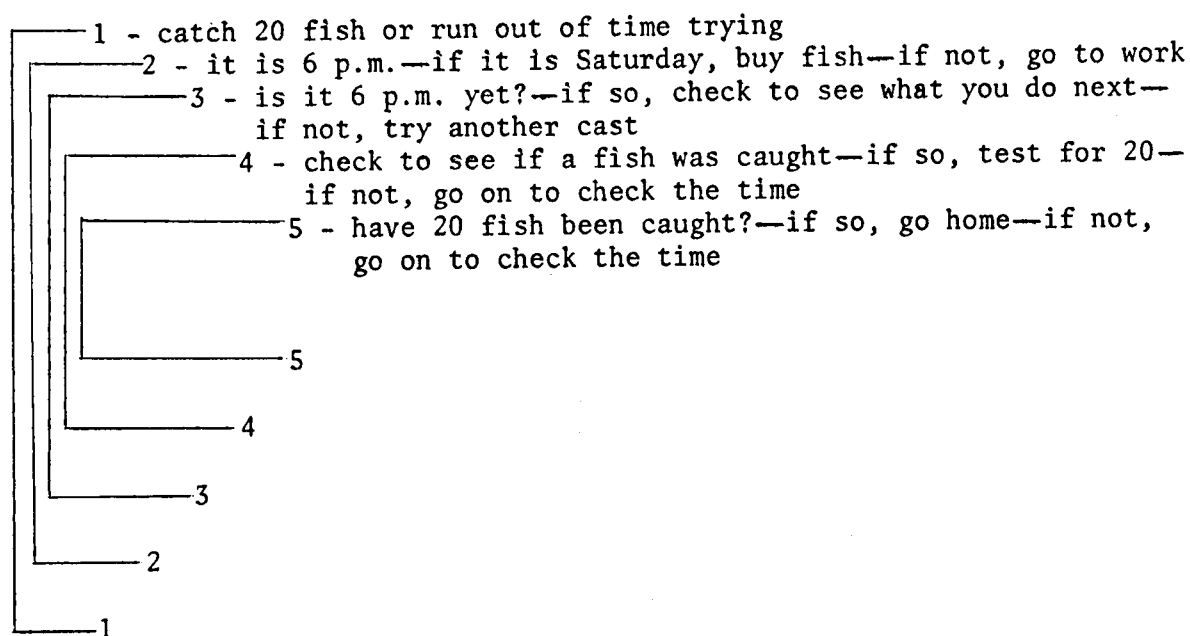
FLOWCHART IS ON THE NEXT TWO PAGES





George diagrams all the possibilities in his head. What path did he actually take? Well, he spent \$10 for fish that Saturday afternoon.

In George's dream we find several loops. The whole dream is a large loop, which is terminated in one of two ways. Either George catches 20 fish or he has not caught 20 fish when 6 p.m. on Saturday arrives. Within the large loop are some other loops. Look at their hierarchy.



As you can see, there are five loops—one all-inclusive outer loop and within each of the levels 1-4 another loop is embedded. Each loop has an entry and an escape clause. The escape clause is very important in looping. Without one, a loop would never end; that is, it would be an infinite loop. It seems obvious in this example what the escape clauses are. Loop 1 will continue to be executed until one of two conditions occurs—George catches 20 fish or 6 p.m. Saturday has arrived. Loop 2 also has two escapes. Assuming that George has not caught 20 fish and

that it is 6 p.m., George must check the calendar portion of his watch for the day of the week. If it is not yet Saturday, George will go to work, come home after work, go to bed and start a new day in the same way he began the preceding one. If it is Saturday, George knows that he has run out of time and must buy some fish or cancel the fish fry. Loop 3 is the loop in which George must decide whether or not to quit fishing. If it is 6 p.m., he then goes through the Loop 2 decision process. If not, he continues fishing. Loop 4 is the loop in which George actually does the fishing. He checks at the end of each cycle (cast, wait and reel in) to see whether or not he has caught a fish. If he has, he checks for 20. If not, he drops down to check the time of day. Finally, Loop 5 is executed only when the following conditions are true: (1) George has just caught a fish; (2) it is not yet 6 p.m.; (3) it is either Saturday or some day in the week before Saturday.

When they are just learning to program, many students neglect to include an escape clause in their loops or they may code the escapes incorrectly. This is the most common error that beginning programmers make in the logical organization of their programs. Your Computer Science 3159 instructor has probably already urged you to write flowcharts describing the logical progression of your program. Doing this can help you to organize your thoughts before you actually code them in FORTRAN. Believe it or not, a competent programmer can take a well-written flowchart (even if it were written by someone else) and translate it into the programming language of his choice. Flowcharts need not be language-specific.

Mr. Byte has just finished his discussion of flowcharting and he pulls out his FORTRAN manual. The girl sitting behind George sneezes loudly and George snaps to attention. He is glad that his daydream is over. He looks around at the other students poring over the chapter on looping in their manuals and he sheepishly opens his. Mr. Byte is talking a mile a minute.

"There are several statements in FORTRAN which can be used together to form loops. The ones I plan to talk about today are the GO TO, the computed GO TO, the Logical IF, and the Arithmetic IF. The GO TO is the most simple. When you execute the statment 'GO TO n,' control is passed to the statement numbered n. By itself, the GO TO is fairly useless, but in conjunction with the others it can be powerful."

RRRRRIIIIIINNNNNGGGG. "Saved by the bell," sighs George. As they walk out of class, George and his friend Nancy talk about how they wish that CS 3159 were not a required course.

"I don't see why we need to know anything about computers anyway," Nancy moans. "When we get out in the 'real world,' we'll have programmers to do that kind of thing for us. The faculty is so out of touch with the 'real world.' If they would come down out of their ivory towers for a while, they might learn a few things."

"See you later, Nancy," mumbles George as he heads for his dorm. All that talk about the real world has sent him into his dream world again.

He sees an old professor with a worried look sitting up in a tall ivory tower. "I am old and gray now," the professor tells his nurse.

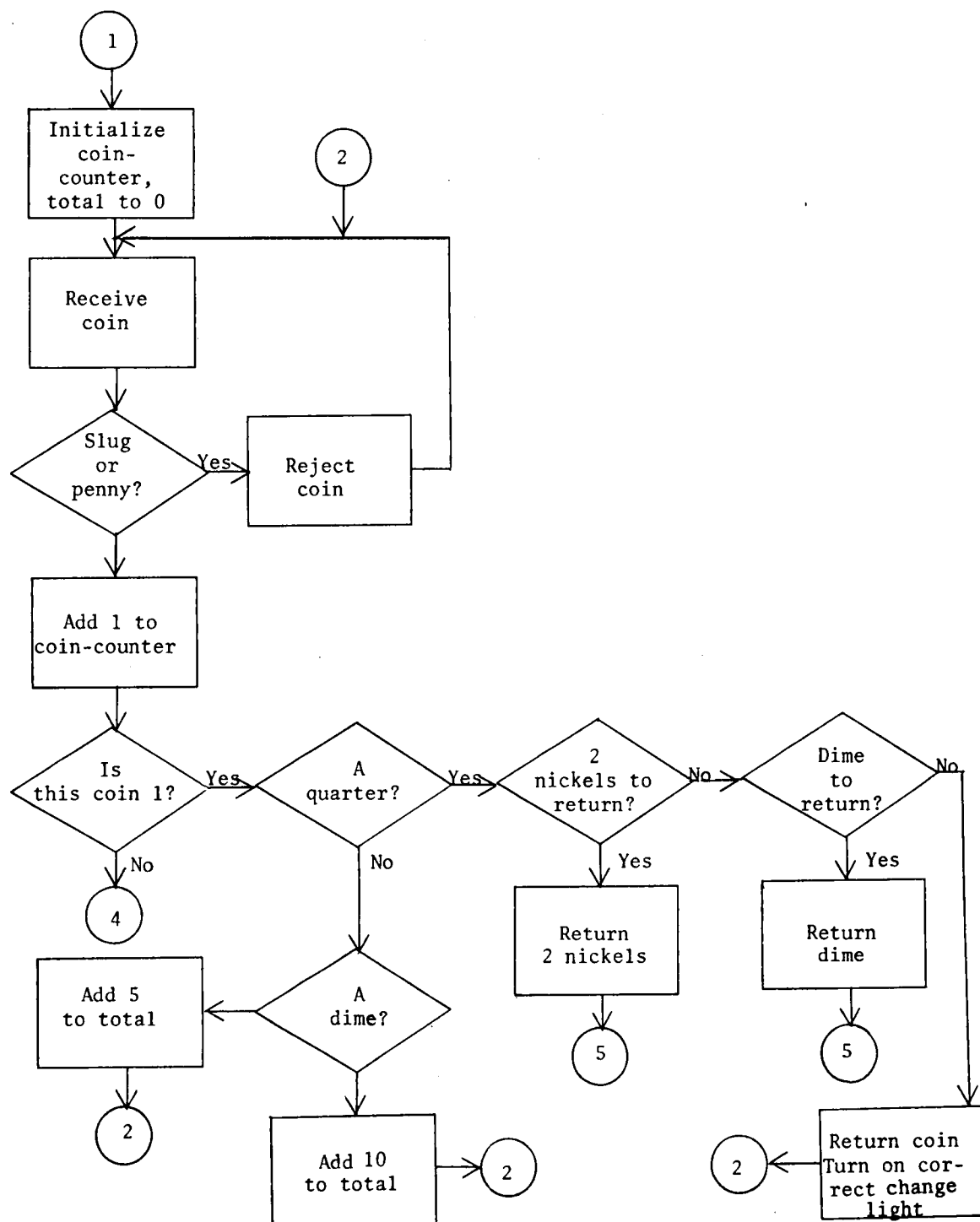
"And I have all my life given nothing of practical value to the world."

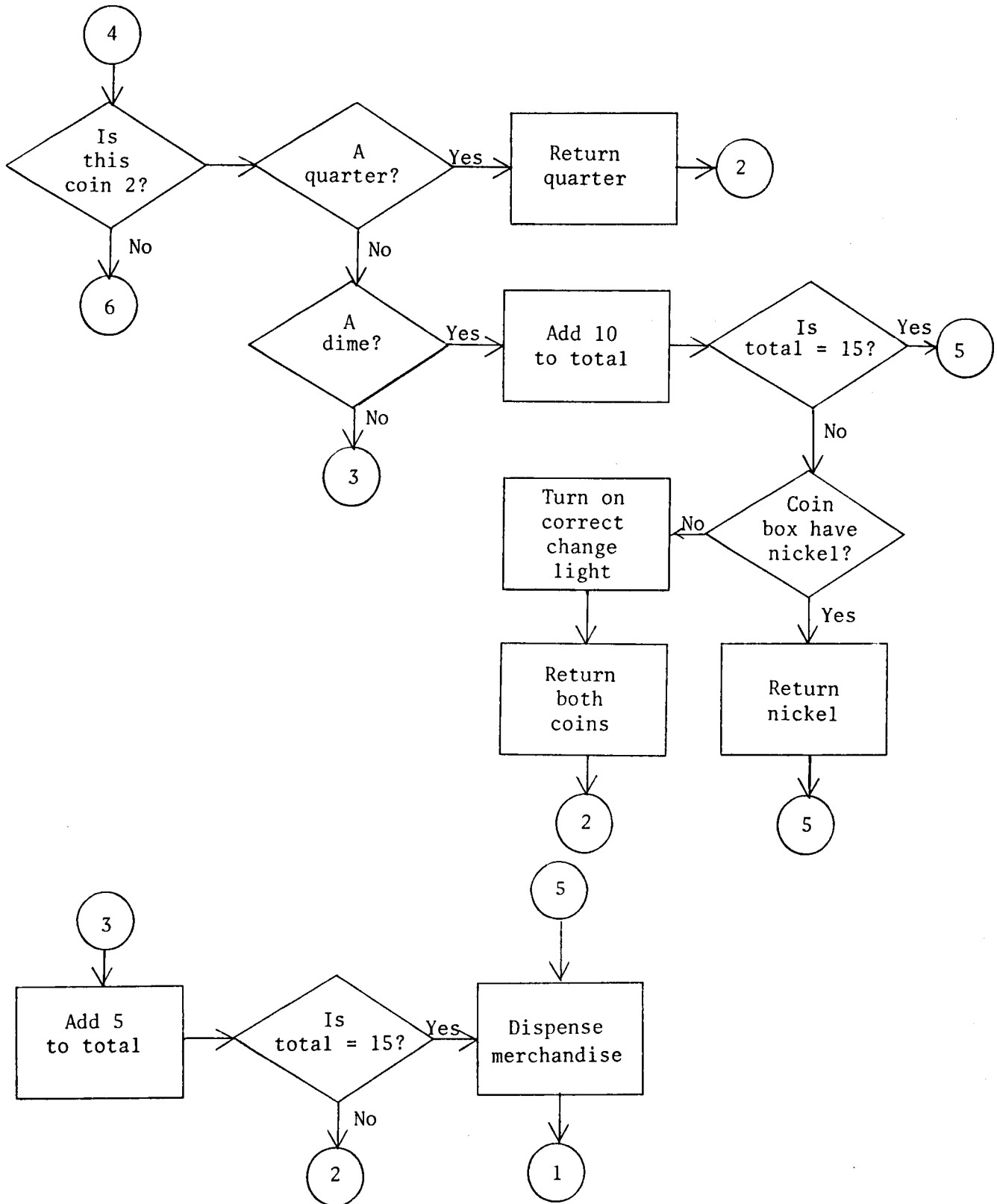
"Oh, you are much too modest," exclaims the nurse. "How can you forget the electric suit you invented? A lot of men really get a charge out of that."

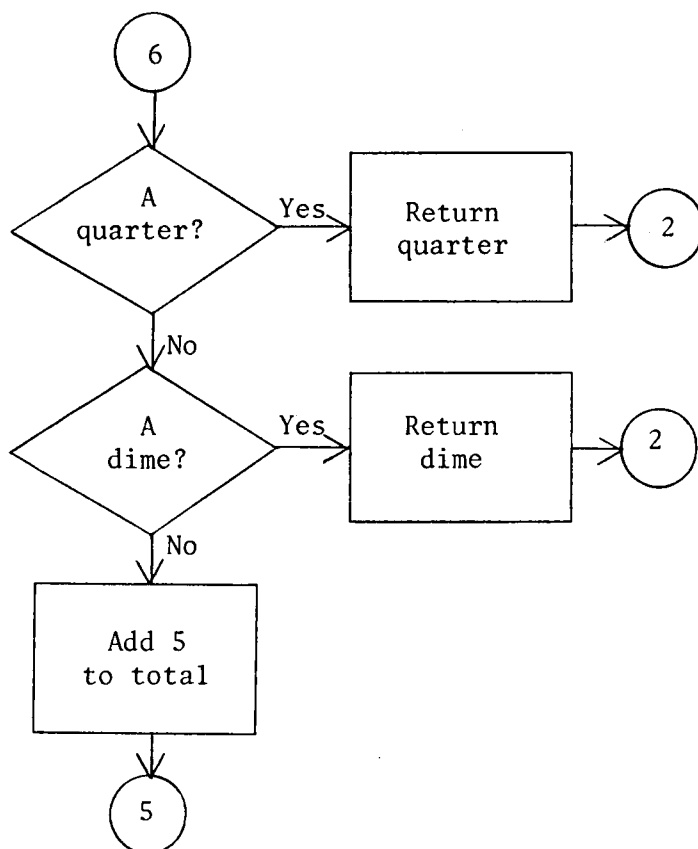
"But it was just a passing fad. I want to invent something permanent, something that will still be used long after I die," he cries. "I've got it. I'll invent a machine that makes change. Well, think of the possibilities. Automated dispensing of goods will become a reality."

The nurse cannot bear to tell the old man that automatic vending machines have been in use for many years, so she goes along with his scheme. The first thing he does is to flowchart the possibilities. He assumes, for simplicity, that each item in the machine costs 15¢ and that the machine will accept only nickels, dimes, and quarters. The inventor takes stock of the situation. A minimum of one coin (a quarter) will cause a sale to be made. The maximum number of coins for a sale is three (three nickels). The machine must accept multiple coins in more than one sequence (nickel and dime or dime and nickel are valid combinations). Here is the flowchart he devised.

•







George awakes to realize that the Coke machine in Greve Hall had just confiscated his last quarter. He kicks the machine for a while, then shrugs his shoulders and heads for his room.

Later, George decides that he had better go down to the Computing Center and punch up Program 2. After all, it is due the next afternoon. He punches it and runs it a few times to get rid of the syntax errors. Then the fun really begins. You will get to observe George's predicaments when you go through the course at the terminal.

The next two pages contain Program 2, which is discussed in the module.

## Program 2—CS 3159

You are to write a program to process orders for parts made by Ajax Manufacturing Company, your employer. The orders are taken by the order entry department and coded on forms for keypunching.

Each order consists of the following cards:

- (1) Header card (one per order)
- (2) One item card for each part ordered
- (3) Trailer card—with cash total (one per order)

The format of the header card is

| <u>Column</u> | <u>Description</u> | <u>Type</u>  |
|---------------|--------------------|--------------|
| 1             | Record Code        | Integer      |
| 2-21          | Customer Name      | Alphanumeric |
| 22-41         | Street Address     | Alphanumeric |
| 42-61         | City, State, Zip   | Alphanumeric |
| 62-66         | No. of Item        |              |
|               | Records in Order   | Integer      |
| 67-80         | Blank              |              |

The format of the item record is:

| <u>Column</u> | <u>Description</u> | <u>Type</u>          |
|---------------|--------------------|----------------------|
| 1             | Record Code        | Integer              |
| 2-16          | Part Number        | Alphanumeric         |
| 17-21         | Quantity Ordered   | Real (0 dec. places) |
| 22-31         | Unit Price         | Real (2 dec. places) |
| 32-80         | Blank              |                      |

The format of the trailer record is:

| <u>Column</u> | <u>Description</u>        | <u>Type</u>          |
|---------------|---------------------------|----------------------|
| 1             | Record Code               | Integer              |
| 2-8           | Quantity for entire order | Real (0 dec. places) |

The trailer record contains the sum of the quantities of the individual parts. This value is computed by Order Entry personnel on an adding machine as an added check against keypunch errors.

The program you write will produce a report for each customer you process. The following sample shows how your output should look:

LOBERTINI FORKLIFTS  
6250 RIVER ROAD  
LOUDON, TN 37774

| PART NUMBER | QUANTITY  | UNIT PRICE | EXTENDED PRICE |
|-------------|-----------|------------|----------------|
| CH077       | 100       | 4.53       | 453.00         |
| TI-8564     | 50        | 2.00       | 100.00         |
| BU069       | 10        | 15.00      | 150.00         |
| TX3280-2    | 20        | 17.50      | 350.00         |
| TOTALS      | 180 PARTS |            | 1053.00        |

If the quantity stored in the trailer record is different from the sum of the quantities per item, do not print a total line. Instead, print a comment, such as "WARNING—TRAILER TOTAL DIFFERS FROM ITEM TOTAL." This will allow Order Entry to check the keypunched quantities against the ones they coded.

There will be an unknown number of orders per run of your program with no trip card in the data (remember the END= option on your READ statement).

## APPENDIX B

### Examples of Student/Computer Dialogue

#### Introduction

Computer: Hi, Steve. You have reached a milestone, the last section of our CAI course. Variable subscripting is the subject. Have you read about David and his economics test analysis yet?

Student: Yes.

Computer: Like Mary from the muffler factory, David has shown his ingenuity by applying the training he received in FORTRAN class in solving a real-life problem. In his case, it is an analysis of the test questions in an elementary economics class. Take note, and follow his example. To continue, hit return.

#### Sample Pretest Dialogue

Computer: What is wrong with the following statement?

$X = (A(3)*R) + B*A/3.0$

- (A) Nothing
- (B) Unmatched parentheses
- (C) Subscripted variable A appears without a subscript
- (D) Mixed mode

Student: C.

Computer: No doubt about it.

. . . . .

Computer: What are the allowable subscript forms?

- (A) Integer constant, integer variable, integer expression
- (B) Integer constant, real constant
- (C) Integer constant, integer variable
- (D) Integer constant

Student: B.

Computer: I'm afraid you missed it.  
Take a look at the question and the answers.

What are the allowable subscript forms?

- (A) Integer constant, integer variable, integer expression
- (B) Integer constant, real constant
- (C) Integer constant, integer variable
- (D) Integer constant

Student: A.

Computer: Yes, all three are allowed.

. . . . .

Computer: What is the main advantage of the implied DO?  
(A) No advantage at all  
(B) It allows you to control the order in which the subscripts of a multi-subscripted array vary  
(C) It allows you to extend your arrays past their bounds  
(D) It saves memory

Student: B.

Computer: All right.

### Program Segment Analysis

Computer: Let's now look at some fairly simple and other fairly complicated problems involving subscripted variables. I will ask you some questions about them and you should answer them in "your own words." Then I will try to analyze "your own words."  
To continue, hit return.

. . . . .

Computer: Write a DIMENSION statement to describe three-dimensional array P, which is 8 by 10 by 6.

Student: DIMENSION A###.

Computer: Please repeat.

Student: DIMENSION P(8,10,6).

Computer: Absolutely right.

Is there anything wrong with this program segment?

```

      DIMENSION XP(5,5)
      .
      .
      .
      READ(5,25)XP
25   FORMAT(5F15.0)

```

Student: No.

Computer: You are right. A subscript is not required in an I/O statement.

The result of the two I/O statements is to read 5 data cards. The data on card 1 fills column 1 of the array (XP(1,1), XP(2,1), . . . , XP(5,1)). The data on each succeeding card fills another column. This is true because elements of a two-dimensional array in FORTRAN are stored in column order.  
To continue, hit return.

Tell me the value which will be stored in A(5) after the execution of this program segment.

```

      DIMENSION A(25)
      .
      .
      .
      A(1)=1.0
      DO 78 I=2,25
78   A(I)=A(I-1)*FLOAT(I)

```

Student: 120.

Computer: You are sharp today.

The aim of this segment is to calculate I! (I factorial) for I=1,25 and store the results in array A. The value of 5! is  $5*4*3*2*1 = 120$ .  
To continue, press return.

### Request for Review

Computer: Now you have the opportunity to return to either the remedial section or the analysis section for review by typing

GO TO X

where X is

SBSTRT - Remedial Material

SBANL - Analysis Material

Otherwise, hit return.

This is your last chance. If you want to review some material, type GO TO SBSTRT or GO TO SBANL. Otherwise, hit return.

To continue, hit return.

## APPENDIX C

### The Two-Sample T Test of Significance

The two-sample t test is intended for use in making decisions about the relative values of two population means. It assumes two normal populations of equal variance and has the advantage that the groups being tested need not have the same number of members. The formula for t is:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - \delta}{\sqrt{(n_1 - 1) s_1^2 + (n_2 - 1) s_2^2}} \sqrt{\frac{n_1 n_2 (n_1 + n_2 - 2)}{n_1 + n_2}}$$

where  $\bar{x}_1$  and  $\bar{x}_2$  are the observed means of the experimental and control groups,  $n_1$  and  $n_2$  are the number of members of the two groups,  $s_1^2$  and  $s_2^2$  are the observed variances of the two groups, and  $\delta$  is the difference between the means being tested in the null hypothesis.

In this study the test may be written as follows:

$$H_0: \mu_1 - \mu_2 = 0$$

$$H_1: \mu_1 - \mu_2 \neq 0$$

where  $H_0$  is the null hypothesis and  $H_1$  is the alternate hypothesis, and  $\mu_1$  and  $\mu_2$  are the means being tested. We reject the null hypothesis if for  $(n_1 + n_2 - 2)$  degrees of freedom,  $t < -t_{\alpha/2}$  or  $t > t_{\alpha/2}$ . Since the test is being made at the 5 percent level of

confidence,  $\alpha = .05$ , and thus  $\alpha/2 = .025$ . If the value of  $t$  falls in the range  $-t_{\alpha/2} \leq t \leq t_{\alpha/2}$ , the null hypothesis cannot be rejected.

The two-sample  $t$  test requires the assumption that the variances of the two populations are equal. In order to test the validity of that assumption, the following test statistic is used:

$$F = \frac{s_M^2}{s_m^2}$$

where  $s_M^2$  is the larger of the two sample variances and  $s_m^2$  the smaller of the two. The test for this study may be written:

$$\begin{aligned} H_0: \sigma_1^2 &= \sigma_2^2 \\ H_1: \sigma_1^2 &\neq \sigma_2^2 \end{aligned}$$

where  $\sigma_1^2$  and  $\sigma_2^2$  are the actual variances of the populations. The null hypothesis will be rejected at the 5 percent level of confidence if  $F > F_{\alpha/2}(n_M - 1, n_m - 1)$ , the  $F$  distribution with  $n_M - 1$  and  $n_m - 1$  degrees of freedom.

In both tests in this study the null hypothesis could not be rejected. Thus, the two-sample  $t$  test may be used.

## APPENDIX D

### Questionnaire for CSCAII Users

1. Did you complete the computer-assisted instruction program named CSCAII? If not, why not?
2. Approximately how much time did you spend on CSCAII, both in reading the handouts and in terminal activity? How many terminal sessions did you take?
3. Do you feel that the computer-assisted instruction program in which you participated was beneficial to you in understanding the material in this course? Please elaborate.
4. What are your general impressions of the informal approach taken by the handouts and terminal sessions?
5. Did you find some obvious mistakes in the program? If so, please note them here.
6. How could the program be improved? Any suggestions are appreciated.
7. Did you encounter any major obstacles preventing completion of the seven-session course, such as unavailability of terminals? If so, what were they?
8. Do you feel that computer-assisted instruction in some form would be worthwhile for students in an introductory FORTRAN course? If so, why? If not, why not?
9. Do you recommend that computer-assisted instruction become a part of the curriculum in introductory programming courses? If so, why? If not, why not?
10. You may make further comments in the space provided below.

## APPENDIX E

### Final Exam for CSCA11

#### Multiple Choice

1. Why is the following expression syntactically incorrect?  
 $(A+B)*C+(D*A(2))$   
  
(A) Unmatched parentheses  
(B) Array referenced without a subscript  
(C) Mixed mode  
(D) Missing operator
2. What, if anything, is wrong with the following statement?  
 $X=IRP**2+LECTURN \ -4$   
  
(A) Mixed mode  
(B) Nothing  
(C) A variable contains too many characters  
(D) Ambiguous statement
3. What is the effect of the following statements?  
 $IF(A.LT.B)GO \ TO \ 30$   
 $IF(C.LT.D)GO \ TO \ 30$   
  
(A) GO TO 30 if A is less than B and C is less than D  
(B) GO TO 30 if A is less than B or C is less than D  
(C) GO TO 30 if A is less than B or C is not greater than D  
(D) GO TO 30 if A is not greater than B and C is not greater than D
4. How many times is the range of this DO executed?  
 $DO \ 60 \ I=1,50,2$   
 $A(I)=I$   
 $A(I+1)=1**2$   
 $60 \ CONTINUE$   
  
(A) 50  
(B) 51  
(C) 25  
(D) 24

5. What is the difference between the DOWHILE construction and DO loop?
- (A) No difference
  - (B) The range of the DOWHILE is always executed at least once while the range of the DO is executed only if its condition is true
  - (C) The DO may contain more statements in its range than the DOWHILE
  - (D) The range of the DO is always executed at least once while the range of the DOWHILE is executed only if its condition is true
6. The DIMENSION statement has what effect?
- (A) Assigns values to array elements
  - (B) Allocates storage locations which can be referenced by one variable name
  - (C) Allocates space and assigns values
  - (D) Assigns a type to arrays
7. How many cards should be read by the following READ statement in order to fill array X?
- ```
DIMENSION X(10,10)  
READ(5,25,END=600)X  
25 FORMAT(5F15.3)
```
- (A) 100
  - (B) 10
  - (C) 20
  - (D) 50
8. What is the effect of the following statement?
- ```
CHARACTER*80 A(50),B,C*132
```
- (A) Define 50 element array A and variable B as character variables of length 80, variable C length 132
  - (B) Define variable A as character length 50, B length 80, and C length 132
  - (C) Define 50 element array A and variables B and C character length 132
  - (D) Define A, B, and C character length 80
9. The logical IF statement allows a \_\_\_\_\_ way branch.
- (A) 3
  - (B) n—where n is the quantity of parenthesized statement numbers
  - (C) 2
  - (D) 6

10. What are the three components of a loop?

- (A) GO TO, check of loop variable, execution of loop statements
- (B) Decision, execution of loop statements, initialization of loop variable
- (C) Initialization of loop variable, execution of loop statements, check of loop variable
- (D) READ, WRITE, GO TO

### True/False

- 11. A FORMAT statement may be referenced by only one READ or WRITE. \_\_\_\_\_
- 12. The Arithmetic IF allows a three-way branch. \_\_\_\_\_
- 13. An array may be referenced without a subscript in Input/Output statements. \_\_\_\_\_
- 14. The DATA statement is an executable statement. \_\_\_\_\_
- 15. The following sequence of statements computes the sum of the squares of the first 25 integers and stores them in ISUMSQ. \_\_\_\_\_  

```
ISUMSQ=0
DO 50 I=1,25
ISUMSQ=I**2
50 CONTINUE
```
- 16. Is it permissible to branch from a statement in the range of a DO loop to a point outside the loop. \_\_\_\_\_
- 17. A statement function definition can itself reference another previously defined statement function. \_\_\_\_\_
- 18. The DO CASE statement is analogous to a conventional FORTRAN statement, the COMPUTED GO TO. \_\_\_\_\_
- 19. In the field specification F10.2, the sign and decimal point are included in the 10 characters. \_\_\_\_\_
- 20. The purpose of the / in a FORMAT statement is to allow skipping of lines on output and records on input. \_\_\_\_\_
- 21. According to the predefined conventions of variable naming, an integer variable is 1-6 characters in length with the first character in the range A-H, O-Z and the remaining characters alphanumeric. \_\_\_\_\_

22. Advocates of structured programming disdain the use of a certain statement, the Logical IF. \_\_\_\_\_
23. In a flowchart, a diamond with one line entering and two lines exiting denotes a decision. \_\_\_\_\_
24. It is permissible to branch into the range of a DO loop from outside the range. \_\_\_\_\_
25. Assume that  $X=7.0$ ,  $Y=3.0$ ,  $Z=6.0$ . The value stored in I by the following statement is 4 \_\_\_\_\_  

$$I=X*Y/Z$$

### Short Answer

26. You may skip to the top of a new page by using \_\_\_\_\_ as the carriage control character.
27. The IF-THEN-ELSE, DOWHILE, and DO CASE allow the programmer to code \_\_\_\_\_ programs in FORTRAN.
28. What value is stored in X after execution of this program segment if the value read for N is 5?  

```

      READ(5,10)N
10  FORMAT (I3)
      X=1.0
      DO 20 I=1,N
      X=X*FLOAT(I)
20  CONTINUE

```
29. How many storage locations are allocated by this DIMENSION statement? DIMENSION A(10,5,5) \_\_\_\_\_
30. Suppose that variable X has been assigned a value of 16.75. What will be printed by this WRITE statement? \_\_\_\_\_  

```

      WRITE (6,50)X
50  FORMAT(1H0,F10.0)

```
31. What is wrong with the following expression?  $X*(I/2)-Z/Q$   
 \_\_\_\_\_
32. What, if anything, is wrong with the following WRITE/FORMAT combination?  

```

      WRITE(6,10)A,B,I
10  FORMAT(1H ,3F15.5)

```

33. Suppose that I is given the value 5 by the READ statement. Circle the statement which will be executed next after the COMPUTED GO TO.

```
      READ(5,20) I
20    FORMAT(I10)
      GO TO (30,40,50), I
25    X=I
      GO TO 60
30    X=I**2
      GO TO 60
40    X=I**3
      GO TO 60
50    X=I**4
60    Y=X**2
```

34. Write a CHARACTER specification statement to allocate a 50 element array with each element 80 bytes long.

35. What value does X get if Y has the value 9? \_\_\_\_\_

```
      IF((Y**2).LT.80.0) THEN DO
        X=1.0
      ELSE DO
        X=2.0
      END IF
```

## VITA

Stephen John Widner was born in Knoxville, Tennessee, on April 7, 1951. He attended elementary and secondary schools in Alcoa, Tennessee, and graduated first in his class from Alcoa High School in 1969. The following September he entered the University of Tennessee at Knoxville, and in June 1973, he received a Bachelor of Science degree in Mathematics.

In 1973 he began graduate work in Computer Science at the University of Tennessee. From that time until August 1975, he served as a graduate assistant programmer/consultant at the University Computing Center.

In September 1975 he was employed by Nashville Gas Company in Nashville, Tennessee, as a programmer/analyst. In June 1976 he joined Maremont Corporation in Loudon, Tennessee, as an applications programmer. Presently, he holds the title Senior Programmer at Maremont.

While employed at Maremont, he reentered graduate school and worked toward a Master of Science degree in Computer Science. He received that degree in June 1979.