Introduction to Computer-Based Education

This handbook is in two parts: Part 1 is an introduction to computer-based education. It gives a brief history of computer-based education, then describes how educators and schools use computer-based education.

This part also outlines common computer-based teaching strategies, provides a brief model of the instructional process, and describes the role of the computer, its hardware and software, and tells you how to assess costs and benefits of computer-based education.

Part 2 is technical summary of C.A.S., DIGITAL'S Courseware Authoring System. This part of the handbook contains detailed information about the Courseware Authoring System.
The information in this manual is subject to change without notice and should not be construed as a commitment by the Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Copyright© 1983, 1984 by Digital Equipment Corporation. All rights reserved.

The following are trademarks of Digital Equipment Corporation:

C.A.S.        DIGITAL
DAL           FOCAL
DEC           IAS
DECnet        PDP
DECsystem-10  RSX
DECtape       UNIBUS
DECUS         VAX
DIBOL         VMS
Contents

PREFACE

PART 1. An Introduction to Computer-Based Education

CHAPTER 1. INTRODUCTION 5

CHAPTER 2. COMPUTER-BASED EDUCATION: A BRIEF HISTORY
Formative Ideas: Teaching Machines And Programmed Instruction 9
Significant Projects And Developments 10
Commercially-Based Efforts 10
Educationally-Based Efforts 12
Microcomputing 14

CHAPTER 3. EDUCATIONAL COMPUTING AND COMPUTER-BASED EDUCATION
How Educators Use Computers 17
Research Uses 17
Administrative Uses 18
Instructional Uses 18
How Schools Use Computer-based Education 19
Managing Instruction 20
Helping Students Learn 22
Common Computer-based Teaching Strategies 23
Drill and Practice 23
Dialogue 26
Simulation 29
Games 32
Computer Assisted Testing (CAT) 34
Problem Solving 40
Discovery Learning 41
## CHAPTER 4. INCORPORATING COMPUTERS INTO INSTRUCTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Brief Model Of The Instructional Process</td>
<td>45</td>
</tr>
<tr>
<td>1: Educational Goals and Curricular Objectives</td>
<td>45</td>
</tr>
<tr>
<td>2: Content Specification</td>
<td>46</td>
</tr>
<tr>
<td>3: Content Analysis</td>
<td>46</td>
</tr>
<tr>
<td>4: Defining Instructional Objectives</td>
<td>46</td>
</tr>
<tr>
<td>5: Selecting Teaching Strategies and Modes of Delivery</td>
<td>47</td>
</tr>
<tr>
<td>6: Design and Implementation</td>
<td>47</td>
</tr>
<tr>
<td>7: Delivery</td>
<td>47</td>
</tr>
<tr>
<td>8: Evaluation</td>
<td>47</td>
</tr>
<tr>
<td>The Role Of The Computer</td>
<td>48</td>
</tr>
<tr>
<td>When to Select the Computer</td>
<td>48</td>
</tr>
<tr>
<td>Matching Computing Capabilities to Instructional Needs</td>
<td>48</td>
</tr>
<tr>
<td>Hardware and Software</td>
<td>49</td>
</tr>
<tr>
<td>Hardware Considerations</td>
<td>50</td>
</tr>
<tr>
<td>Software Considerations</td>
<td>51</td>
</tr>
<tr>
<td>Assessing Costs And Benefits</td>
<td>52</td>
</tr>
<tr>
<td>Costs</td>
<td>53</td>
</tr>
<tr>
<td>Benefits</td>
<td>54</td>
</tr>
</tbody>
</table>

## CHAPTER 5. DESIGNING AND PRODUCING COMPUTER-BASED EDUCATIONAL MATERIALS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparing for the Control of Instruction</td>
<td>59</td>
</tr>
<tr>
<td>Designing the Lesson</td>
<td>60</td>
</tr>
<tr>
<td>Program Development</td>
<td>64</td>
</tr>
<tr>
<td>Lesson Implementation</td>
<td>65</td>
</tr>
</tbody>
</table>

## CHAPTER 6. SOFTWARE TOOLS TO ASSIST YOU IN PRODUCTION

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer-Managed Instruction Systems</td>
<td>67</td>
</tr>
<tr>
<td>Authoring Languages</td>
<td>70</td>
</tr>
<tr>
<td>Editors</td>
<td>70</td>
</tr>
<tr>
<td>Utilities</td>
<td>74</td>
</tr>
</tbody>
</table>
# CHAPTER 7. TRENDS:
THE NEW INFORMATION TECHNOLOGY

- From Print to Electronic Communication 75
- The Effect of Computers 76
- The Complete Information Environment 76
- Epilogue 77

---

# PART 2: C.A.S. Technical Summary

## CHAPTER 8. INTRODUCTION TO DIGITAL’S COURSEWARE AUTHORING SYSTEM (C.A.S.)

- The User Interface 81
- C.A.S. Bookkeeping and Reporting 85
- C.A.S. Authoring Tools 86

---

## CHAPTER 9. DAL:
THE DIGITAL AUTHORING LANGUAGE

- A Simple Structure for Easy Operation 87
- A Variety of Commands 88
- Summary of DAL Instructions 91
- DAL Built-in Functions 110
- DAL Data Types 114
- Usage Characteristics of Variables 114
- DAL Operators 116
- Response-Related System Variables 117
- Graphics and Graphing System Variables 121

---

APPENDIX A 126

APPENDIX B 131

GLOSSARY 139
Preface

Computer-based Education (CBE) is difficult to define concisely for a number of reasons. First, the field lacks a single, unambiguous set of terms and descriptions. The problem is further complicated by the fact that each computer vendor whose equipment is used to support CBE efforts imposes its own set of terms on the techniques of computing.

Another reason is that no single technique or project dominates the field of CBE. Therefore, summative information about the field is not available from a single source.

For these and other reasons it is difficult for educators to find the information necessary to formulate accurate opinions and perspectives so important to the decision making process.

This handbook is designed to provide an introduction to the field of Computer-based Education and, in particular, to Computer-assisted Instruction (CAI), the process of using the computer as a tool in the teaching process, and Computer-Managed Instruction (CMI), the process of using the computer as a tool in managing the teaching process. The handbook is divided into two parts. Part 1 provides general background information about ideas and techniques related to the field. DIGITAL hopes that the background information and project descriptions will help teachers develop a reasonable initial perspective on CAI and CMI, and that the descriptions of general CAI and CMI techniques will help teachers clarify their own thinking about the ways in which computers may be used in the classroom.

Part 2 describes products and services which are available from Digital Equipment Corporation in support of educational computing. Only a clear understanding of the technical capabilities of various computer systems can help educators to select equipment well suited to their instructional needs.

The next two decades will see a critical transition in the educational process. The pressures of new information technologies and the changes which they will impose provide a challenge to educational leaders. Perhaps the greatest advantage teachers and administrators can have is a clear understanding of the principles upon which new information technologies are based and the range of educational problems to which they may be applied. This handbook is targeted toward that end.
For their support in the preparation of portions of this document, Digital Equipment Corporation gratefully acknowledges the assistance of Indiana University, Bloomington Academic Computing Services, and Dr. William H. Sanders, coordinator of the University's Workshop for Computer-based Instructional Support.
PART 1

An Introduction to Computer-Based Education

This part of the handbook contains introductory information about computer-based information. It gives a brief history of computer-based education, then describes how educators and schools use computer-based education.

This part also outlines common computer-based teaching strategies, provides a brief model of the instructional process, and describes the role of the computer, its hardware and software, and tells you how to assess costs and benefits of computer-based education.
1 Introduction

Computers are now playing an active role in the education process. In the 60's, experiments with computers indicated their value as educational tools. But in the 60's computers were expensive, difficult to use, and vaguely threatening to many educators.

Things have changed. Computers in classrooms are not only less frightening, they are more and more in demand, and at all levels of education from universities to elementary schools. The current concern is not whether to use computers in classrooms, but how to use them in classrooms.

Computers are much less expensive, much easier to use, and are becoming “smart” enough to play an active role in the student’s education. Teaching that makes of use a computer terminal, referred to generically as “computer-based education,” is not only a feasible option, but a necessity for a growing number of schools. Computers are now being used to free teachers from rote duties so they can use their time to the best possible advantage: teaching.

Advantages for Teachers

Computers help teachers by relieving them of repetitive teaching tasks. Today, colleges are forced to meet the growing need for “developmental instruction.” This need arises from lower standards in high schools; one of every three college math courses is taught at the pre-algebra level. Computers can be used to assist teachers with the drill and practice sessions required in such courses.

Testing and scoring on a computer not only makes it easier to monitor the progress of an individual, but it can aid the student in judging his own success, measured against a well-defined set of goals.

Storing student responses on the computer brings another advantage to the instructor. By checking reactions to his courses, an author can check the quality of his course. For example, a teacher can determine whether a topic in a course is already known to most students and alter the course appropriately.

Advantages for Students

Most advantages for teachers become advantages for students. For example, students profit when teachers can pay closer attention to their work; students also profit when they can work quickly through material they are already familiar with and more slowly with new material. In both situations,
the student must prove his ability with the material, but is able to pace himself through it rather than relying on a more generalized pace set for an entire class.

But computers also provide benefits for students beyond those realized by teachers. The computer provides a mechanism for involving them more closely with material being learned, thus providing a higher level of motivation. The computer also provides stimulation in the form of graphics, animation, color, and sound. A very important advantage is that the computer provides a means for tutoring individual students at their own speed.

COMPUTER-BASED EDUCATION

The advantages that have become clear in the past few years have brought many types of computers to the classroom, from single-user systems which perform the basic functions like drill and practice sessions to larger computer systems that perform a multitude of functions designed to aid the educator. With the entry of these machines and systems, a vocabulary has also been introduced.

The umbrella term for using a computer for educational purposes is “computer-based education.” This term refers to any tutoring or testing done with the assistance of a computer and can be segmented into subcategories which present material to students, evaluate the student’s comprehension, and advise a course of learning based on the student’s performance.

CMI (computer-managed instruction) is the portion of CBE that directs the student’s progress. It can be implemented in any degree of complexity, from test administration to something approaching guidance counselling.

CBI (Computer-Based Instruction) is the portion of CBE that actually involves teaching the student. The computer presents new material and assures that he understands it by means of an interactive dialogue or by formal testing.

CAI (computer-aided instruction) is an older term for CBI. The formative years of CBE were limited to simple drill and practice and not ideally suited to presentation of new material to the student. At that point, computers aided instruction; now they take a more important role in helping teachers teach.
DELIVERY DEVICES AND SYSTEMS

In the 60's, delivery devices were limited to hardcopy teletype-like devices that were fine for drill and practice, but not for much else.

An Early CBE Teletype Delivery Device

In recent years, keyboards and video display devices attached to larger computer systems bring a higher degree of control and variety to the learning process. These devices can make use of such motivating tools as graphics, sound, color, and animation by using the resources of host computers.

Only recently have machines been affordable enough for standalone microcomputers to be used as delivery devices. Even now such devices don't provide all the flexibility and function of larger systems. However, local micros attached to host systems provide a new level of flexibility, function, and convenience for education use.

Micro Delivery Devices Attached to a Host Computer

Such a system provides the convenience of the micro in the classroom for a variety of uses and the power and greater storage capacity of the host for maintenance of statistics and storage of courses. And such configurations are only a beginning when you consider networks and other computer technology which is rapidly becoming cheaper and more convenient to use.
SUMMARY

This chapter is intended to be an introduction to computer-based education (CBE); the more detailed chapters which follow in Part 1 provide information on the history of CBE, how computers are used in education, and how educators go about using computers as tools to design and implement courses.

Part 2 provides a technical summary of an advanced computer-based education system: C.A.S., DIGITAL’s Courseware Authoring System. C.A.S provides all of the functions described in this chapter and, as you’ll see, is flexible enough to grow with in the future.
Computer-Based Education: A Brief History

This chapter provides a brief overview and history of the ideas on which computer-based education has grown. Also, this chapter lists some of the significant projects which have guided computer-based education (CBE) in its development and direction.

FORMATIVE IDEAS: TEACHING MACHINES AND PROGRAMMED INSTRUCTION

Two concepts influenced the formation of ideas about Computer-based Education (CBE): the teaching machine and Programmed Instruction.

The concept of using a machine to teach began to gain credence in this country during the 1920's primarily as a result of the work of a psychologist named Sidney Pressey. Pressey developed and exhibited a device which presented multiple choice questions to students and kept track of their right and wrong answers. Its principles, consistent with the ideas of his contemporaries in educational psychology (particularly B. F. Skinner), became closely associated with the development of Programmed Instruction in the 1950's.

Programmed Instruction was, in a sense, an attempt to teach without direct teacher interaction. Textbooks were designed in such a way that they presented materials in a sequence of small segments called frames. After each frame, the student was required to answer a question and on the basis of that answer additional frames were presented. The intention was to take each student through the entire text at a speed and in a manner consistent with his abilities.

By the late 50's it was possible to identify two basic approaches to programmed instruction both of which would later be incorporated as strategies for computer-assisted instruction (CAI). The first approach, most often associated with B. F. Skinner, organized frames of material in a linear sequence. Each student was required to work the same series of frames and the sequence was designed so that the transition from one frame to another was simple and contained minimal additional information. It was reasoned that because of the simplicity of each step in the process that students would be able to complete instruction on their own.

The second approach to Programmed Instruction is typified by the work of Norman E. Crowder. Crowder's TUTORTEXTS or "scrambled textbooks" were based on a branching style of programming which allowed different students to take different paths through the material. After reading each
frame the student was asked a multiple choice question. Based upon his answer, the text directed him to the next step in the instructional program. A wrong selection might direct the student to remedial material or to a frame that clarified his error, while a correct response would move the student ahead to new information. In this style of programmed instruction, as opposed to the linear style, each student would take a completely different path through the text.

The computer is in many ways an ideal device for the implementation of these approaches to instruction; in spite of the controversy about programmed instruction, its concepts have had considerable effect on the research and development of Computer-based Education.

**SIGNIFICANT PROJECTS AND DEVELOPMENTS**

Efforts to computerize instruction began in earnest about 1960. Beginning in the early 60's, extensive projects were conducted commercially as well as privately.

For the most part, projects and experiments in computer-based education have been mutually exclusive of each other. Key innovators in universities and commercial research organizations have found ways to muster resources to build individual programs. Such programs have been based upon different computers, used different programming languages, and often reflected unique educational philosophies. While many such projects have proved to be of great value to sponsoring organizations (and often to others), no single project has become the de facto standard for computer-based education.

**Commercially-Based Efforts**

Some of the more important commercially based early experiments were conducted by International Business Machines; System Development Corporation; Bolt, Beranek, and Newman, Inc.; and Thompson Ramo Wooldridge, Inc.

**International Business Machines (IBM)**

IBM initiated one of the earliest experiments when in 1958 it used a computer program to teach binary arithmetic. In 1961, IBM researched computer-assisted instruction in the areas of stenotyping, psychological statistics, and German reading. By 1966 the company was involved in the development of a programming language specifically designed for instructional applications which required student-machine dialogue. Since then, COURSEWRITER has gone through a number of revisions but remains a key part of what IBM offers as aids to the instructional process.
IBM also made a significant effort in the area when it offered the System 1500 as a prepackaged instructional system. The system could support as many as 32 student stations each of which could be equipped with a display terminal, an image projector, and audio equipment. The COURSEWRITER language was emphasized as a source language for writing 1500 course materials. IBM supplied some course materials for the system, but the bulk of the materials were developed by individual users. Beginning in 1967, the company also hosted seminars in an attempt to provide educators with the necessary skills and information for the implementation and operation of an instructional computing center.

**Systems Development Corporation (SDC)**

SDC was an outgrowth of the Rand Corporation, a research organization which had been under contract to the Air Force for the development of pilot training simulators. Its project, Computer-based Learning for Automated School Systems (CLASS), attempted to develop a computer-based instructional system to cover the entire educational process. It attempted to provide administration, counseling, and both individual and group instruction in a single unit. Its classroom was an automated combination of computerized drill and practice, filmstrips, television, short films, and other audio-visual aids.

**Thompson, Ramo, Wooldridge, Inc. (TRW)**

TRW's project was somewhat less ambitious than that of SDC. The MENTOR Automatic Tutoring Device supposedly contained all the logic necessary for automatically scoring student responses, making decisions based upon those responses, selecting subsequent presentations and gener-
ally controlling the conditions of presentation. It also included a variety of audio-visual aids. The major difference between CLASS and MENTOR lay in the integrated approach which CLASS used and which MENTOR did not. MENTOR was really a standalone teaching machine and was not designed to interface with other aspects of the educational process.

**Bolt, Beranek, & Newman (BB&N)**
The initial project at BB&N was even more limited in scope than MENTOR. It was implemented on a small Digital Equipment Corporation PDP-1 computer and was primarily concerned with drill and practice applications with respect to paired associate items in German vocabulary.

The latter three of these projects, while having considerable effect on the relevant literature of the early sixties, were shortlived; the major reason being that the computers used in these projects were both too expensive and too limited in capability to be cost effective in an instructional setting.

### Educationally-Based Efforts

**Stanford University**
One of the earliest educationally-based projects to gain wide recognition began at Stanford University under the direction of Patrick Suppes. His efforts in elementary math and reading are documented in several published volumes. It was Suppes who first provided an elementary school teacher with a computer-based teaching device on a daily basis. During the early sixties the Stanford project, consisting primarily of drill and practice applications in arithmetic, expanded from one teacher with 41 students to a network which included a significant number of California school districts as well as elementary schools as far away as Mississippi. Suppes has since moved into the commercial market and offers courseware and instructional systems through Computer Curriculum Corporation.

**University of Illinois**
As Suppes was beginning work at Stanford the PLATO project was gearing up at the University of Illinois under the direction of Donald Bitzer. In 1960 Bitzer succeeded in connecting one interactive terminal (and later two) to the University of Illinois's ILLIAC I computer. From those modest beginnings and with considerable governmental support PLATO grew into a large computer network with extensive capabilities. Most instructional materials were developed by interested faculty and staff working alone or in small groups with support from a small pool of professionals. PLATO is being distributed commercially by Control Data Corporation for both educational and non-educational applications. Today PLATO is the most formidable example of instructionally oriented interactive computing using a single large central computer.
Brigham Young University—University of Texas
In 1971, under a grant from the National Science Foundation, the MITRE Corporation subcontracted with the University of Texas CAI Laboratory and the Research and Development Department of Brigham Young University for their participation in the design and implementation of the minicomputer based CAI system which came to be known as TICCIT. The system combines computer and television technologies and assumes the presence of a team of professional designers and programmers to produce courseware. Initial TICCIT installations involved junior colleges, but the system has since moved into other environments and has enjoyed considerable success. TICCIT is currently a commercial product of Hazeltine Corporation.

University of California at Irvine
In the late 1960's the University of California at Irvine founded the Educational Technology Center. The center, under the direction of Alfred Bork and supported by considerable outside funding, has supported the development of Computer-assisted Instructional materials in various disciplines, but with a heavy emphasis on physics and other science related areas.

The work of the center is unique in its resistance to the notion of a standardized CAI language or system. Development has taken place on various computer systems of all sizes and, for the most part, with the use of standard programming languages.

University of Iowa (CONDUIT)
The incompatibility of available computers and programming languages poses a serious barrier to the dissemination of instructional materials. With funding from the National Science Foundation, CONDUIT was formed in conjunction with the Weeg Computing Center at the University of Iowa. Since inception and with help from other participating institutions, CONDUIT has sought ways to develop CAI programs in a manner conducive to their transportability between systems. These efforts have included the specification of "minimal" versions of programming languages which contain only the most easily transported commands.

CONDUIT acts as both a clearinghouse and a publisher by providing technical support and translation services to authors. It then reviews, tests, and markets materials. Authors receive royalties on each copy sold. CONDUIT publishes “Pipeline” magazine which contains articles of interest to CAI practitioners as well as the catalog of their offerings. CONDUIT has traditionally been a reliable source of information and materials related to CAI materials.

Minnesota Educational Computing Consortium (MECC)
For many years, the state of Minnesota has provided funds to support a state-wide consortium of schools and universities to promote the develop-
ment and exchange of instructional computing materials among institutions. In many ways, MECC serves both as a model of such activity for the nation and a source of commercially available CBE material.

Originally based upon a network of mainframe and minicomputers, MECC took the then unprecedented step in the mid 1970's of announcing a shift in emphasis to stand-alone microcomputers and negotiated a contract for the purchase of a large volume of Apple II personal computers. Many of the materials formerly delivered in the MECC timeshared environment have since been converted to micro based formats and are available for purchase from the consortium in cassette tape or floppy disk format.

The MECC micro move did much to legitimize the entry of personal computers in the minds of educators.

**Microcomputing**

Since its introduction in 1975, the microcomputer has had greater impact on education than any previously available class of machine. Its low initial cost, portability, and relative ease of use make it ideal for many educational situations. Although many companies currently produce microcomputers for a wide array of applications, only a few have made significant inroads into the educational market. Those companies have been led by Tandy Corporation (TRS-80 microcomputer), Apple Computers, Inc. (Apple II and II-plus), and Commodore Business Machines (PET computer).

More recently, corporations which have traditionally sold only larger machines have begun to enter the educational microcomputer market place. The IBM personal computer has been well received. And as other major vendors with large sales and service forces, enter the market, the complexion of the personal computer market will change considerably.

School administrators are faced with increased demand for students to have experience with computers at school. These administrators have invested heavily in microcomputers because they are relatively affordable machines.
Microcomputers are in use in computer literacy programs, high school computer science curricula, programming courses, programs for gifted and talented students, and many others.

Although many schools are involved at the local level in the production of microcomputer-based CAI materials, mechanisms for wide dissemination and publication of those materials have not yet matured. Even when such mechanisms develop, the eventual role of the microcomputer in CBE vis-à-vis other classes of computers is not clear. While micro’s have obvious advantages and attractions, they also have limitations.

Since other aspects of the technology (particularly those related to data communications) have become more available, many institutions have been investing in distributed computing networks using computers of various sizes and capabilities with the intention of taking advantage both of the power and speed of larger computing machines and the small size, portability, and relative simplicity of personal microcomputers.

While predictions about the future are always dangerous, the instructional computing facility of the future will probably involve a network of various sizes and capacities serving different needs. The need for such networks and additional information is provided in Chapter 7.
Educational Computing and Computer-Based Education

This chapter provides an overview of how educators make use of computers and how schools can make use of computer-based training. The chapter outlines 7 basic instructional strategies involving computers and describes how computers can be used as tools for testing, for problem solving, and for discovery learning.

Educational uses of computers can be divided into three distinct areas: research uses, administrative uses, and instructional uses.

Educational Applications of Computers

- Research
  - About Computers
  - With Computers
- Administration
- Instruction
  - Teaching About Computers
  - Teaching With Computers

General Uses of Computers in Education

HOW EDUCATORS USE COMPUTERS

The following sections outline the ways that educators use computers for research, administration, and instruction.

Research Uses

Over the years, educators have developed the computer in two distinct ways. First, the computer has been the object of research for university based computer scientists. Second, the computer has been a tool in research efforts of educators across many disciplines.
Research ABOUT computers
Researchers in higher education have been involved with the development of computers since the first one was built at the University of Pennsylvania's Moore School of Electronics at the end of World War II. Research in the techniques of design and use of computers continues in computer science departments of universities all over the world.

Research WITH computers
More significant, perhaps, than the role of the computer as the object of research has been its role as a tool for research. Beginning in the 1950's with the introduction of the first commercial computers, the field of data processing began to grow and today the use of computers as tools for information gathering, storage, and analysis constitutes one of our nation's largest industries.

Administrative Uses
Educational administrators face the same management problems as do their counterparts in business and industry. The payroll must be processed, resources must be inventoried, and budgets must be projected. No surprise, then, that the computer has had a significant impact in education administration. In fact, prior to the microcomputer, those school districts which bought computers were much more likely to buy them for administrative purposes than for purposes of instruction. Occasionally, a machine used primarily for administration would be made available parttime for instructional uses.

Instructional Uses
In the same way that educational research uses have been divided into two subcategories, so have instructional applications. Teachers in the class room use computers essentially in two ways: as an object of instruction or as a tool in the instructional process.

Instruction ABOUT Computers
Two major activities comprise the use the computer as the object of instruction: Computer Science and Computer Literacy. The discipline of computer science teaches about the nature of computers to students who wish to pursue a computer related academic or professional career.

In recent years, as the emphasis on computers in our society has increased, schools have become concerned that they must assist students in becoming familiar with the uses and impacts of computers in daily life and on the job. Activities related to this task generally have been referred to as Computer (or Computing) Literacy.
In the rush to build Computer Literacy programs, many educational institutions have invested heavily in standalone microcomputers. The majority of activity taking place on those computers is either the running of prepackaged programs to help students gain confidence in using computers or the teaching of computer programming, usually in a language such as BASIC.

While instruction about computers is important, this handbook is intended to highlight the role of the computer as a tool in the instructional process.

**Instruction WITH Computers**
The computer can be used as a tool in instruction. The process of using a computer as a tool for teaching has come to be referred to in most circles as Computer-based Education (CBE). The balance of this chapter is dedicated to defining CBE and in identifying the ways in which educators have chosen to use computers in the teaching process over the past 25 years. In general, those uses relate to the delivery and evaluation of instruction as described in the general discussion of the instructional process in Chapter 4.

**HOW SCHOOLS USE COMPUTER-BASED EDUCATION**
Common practice divides the field of CBE into two subfields: Computer-managed Instruction (CMI) and Computer-assisted Instruction (CAI). In some circles, particularly among European educators, CAI is referred to as Computer-assisted Learning (CAL). The term Computer-based Instruction (CBI) also may crop up from time to time. Operationally, the three terms (CAI, CAL, CBI) are virtually synonymous.
Two Main Subfields of CBE

Managing Instruction
The term "Computer-managed Instruction" (CMI) refers to the use of the computer to control and manage the instructional environment. In any course of instruction, much of an instructor's time is spent in clerical and administrative activities such as planning, sequencing activities, organizing materials, and grading assignments. Under many circumstances such activities may be delegated to a computer, as illustrated below.

Activities Which Can be Delegated to a Computer
For example, assume that a chemistry professor has responsibility for 500 freshman students enrolled in a course on Introductory Organic Chemistry. The course consists of readings, films, demonstrations, laboratories, lectures, and perhaps one or two field trips.

Traditionally, at the beginning of the semester these students would be handed a course syllabus which they would all follow in lock-step fashion through 16 weeks and a final examination. Even the most conscientious instructor would be unable to take into account either the individual needs of students or the many differences in learning styles which exist among them. Even in a lock-step program, the administration of such a course is a logistical nightmare common in colleges and universities today.

In such a situation, it may be possible (with adequate preparation) to use a computer as a course manager. Assuming that a suitable CMI software package is available, a number of possibilities exist to improve the effectiveness of the course:

- The course syllabus can be made available on a computer. Students need to check the computer regularly during their course of study for new assignments and information. Any student’s syllabus can be changed to accommodate his situation.
- Films, lectures, field trips, and other non-computer-based events can be scheduled or rescheduled easily in a manner which ensures that every student will “get the word”.
- CAI materials can be arranged in modules (and sequences of modules) and delivered to students in an organized fashion.
- Quizzes and tests can be worked into the module menu so that students may “test out” of a module if they can show mastery of its concepts or demonstrate mastery after receiving instruction.
- Scores on tests can be automatically gathered and reported to the instructor. Students who are falling behind or having special difficulty can be identified.
- The computer also may be used as a communication medium so that students and instructors can exchange electronic mail and comments thereby reducing “telephone tag” and the overhead of unnecessary personal appointments. Students may be allowed to communicate with other students in this fashion and solicit assistance from their peers.

A system of this sort can go a long way toward reducing instructional management problems inherent in many class situations. As the popularity of telecourses, home study, correspondence, and other non-traditional forms of instruction continues to rise, the importance of these systems (and their availability) will continue to increase. Control systems such as these are already vital to many industrial and on-the-job training efforts where they contribute to significant reductions in instructional time and costs.
Such CMI systems tend to be more available on larger computers due to the amount of computing power required to support them. Systems such as PLATO, TICCIT, and Digital’s Computer Based Education System have strong CMI components which add to the instructional environment the flexibility, control, and power which are not possible with either stand-alone microcomputers or on machines made available for instruction on an occasional basis.

**Helping Students Learn**

Educators also use computers as a delivery medium for instruction. In that sense, the computer is not unlike other tools such as blackboards or textbooks, but important differences do exist between computers and other media. The major difference lies in the computer’s potential to provide students with an interactive involvement with instructional materials. Such involvement has a number of benefits:

- **Students may be given various degrees of control over their own learning, including selection of work time, location, pace of personal progress, and other factors.**
- **Instruction can be tailored to the learning needs of individual students, because interactions can be designed which alter lesson content based on students’ correct or incorrect responses.**
- **Student “attention level” must remain high. An interactive CAI lesson does not allow students the luxury of daydreaming in class.**
- **Information on student performance (and on effectiveness of the materials) can be gathered for each student during the lesson and stored for future reference.**

Early efforts to use computers to teach were often efforts to transfer familiar instructional strategies into a computerized form. An analogy may serve to make this point more clear. In the early twentieth century when motion picture technology was in its infancy, some movies were made by placing a movie camera in a theatre and photographing a live play. Neither movement of the camera nor the use of any cinematographic techniques familiar to us today were used. Practitioners simply did not have enough experience to exploit the medium. The same situation existed in the early days of CAI and to some degree still persists today.

During the past 25 years, practitioners have moved from the design and implementation of familiar, often simplistic, instructional strategies to highly sophisticated learning systems. They have been aided by the dramatic increase and reduced costs of available computing power as well as by the enhanced sophistication of recent software systems.
CAI is analyzed below in terms of often-used instructional strategies. Eight strategies are described with explanations, examples, and important considerations for their use. Most CAI material (often called "courseware") written during the past 20 years falls into one or more of these categories. For that reason, the categories provide not only a method of organizing the activities of the past, but also a framework within which the CAI novice may examine his own instructional objectives.

Commonly Used Strategies for Computer Assisted Instruction

- **DRILL & PRACTICE**
- **DIALOGUES**
- **GAMES**
- **TESTING**
- **SIMULATIONS**
- **PROBLEM SOLVING**
- **DISCOVERY LEARNING**

---

Basic Instructional Strategies

COMMON COMPUTER-BASED TEACHING STRATEGIES

As shown in the illustration above, there are 7 basic instructional strategies. The sections below outline them and describe their characteristics.

**Drill and Practice**

Drill and practice as a teaching strategy is familiar to all educators. In general, drill and practice promotes the acquisition of knowledge or skill through repetitive practice. In CAI, drill and practice usually refers to small tasks such as the memorization of spelling or vocabulary, or the practicing of arithmetic facts.

The most common drill and practice lessons have been designed to teach "paired associate" materials. Paired associate materials create stimulus/response situations, as explored in the research of the psychologist B. F. Skinner. Examples of paired associate tasks include French/English vocabulary drills, arithmetic drills, and music drills (matching musical sound to
appropriate notation). Some reasons for the popularity of computer based drill and practice materials include:

1. Such activities are essential to success in school, but teachers rarely have time in class to help students “overlearn” material in this manner.
2. The materials are totally quantifiable; that is, the entire learning task can be clearly specified. This helps make it a logical choice for implementation on computers.
3. Drill and practice programs that deal only with paired associate tasks are relatively easy to design and develop.

**Characteristics of Drill and Practice Programs**
Drill and practice programs basically fall into two categories:

1. Programs which present materials from stored lists of items (such as lists of vocabulary words to be matched with their foreign language counterparts).
2. Lessons which generate materials according to a formula or a pattern.

In the first case, the computer maintains lists of foreign words and their English language equivalents. The program presents the foreign word in some manner and ask the student to respond with the English equivalent. Other examples include medical terms and their meanings, states and their capitols, chemical elements and their symbols.

In the second case, programs are based upon algorithms (or methods) of creating drill items. For example, problems in arithmetic or mathematics can be generated algorithmically. In such a situation, the computer knows that it must select two numbers and ask the student to add them together. The computer can select the numbers and compute what the answer should be.

This method eliminates the need to store large numbers of specific arithmetic problems and has other advantages mentioned below. Musicians have frequently used algorithms to generate musical intervals and chords through sound synthesizers attached to computers. Students listen to the sound and respond with proper notation or in some other way indicate that they can properly identify the characteristics of the music.

**Sequential versus Random Presentation**
In their simplest forms, “stored list” programs simply read down the lists of paired associates and present them to the students, much as a parent would flip through a stack of flash cards. Unfortunately, students who must work through the program more than once may memorize the order of the stimuli.

To eliminate this problem, more sophisticated programs select stimuli from the stored lists in a random fashion. This insures that the items are presented
in a different order each time the student works through the materials. As systems have become more sophisticated and easier to use, and as educators have become more knowledgable, they have shown a distinct preference for randomly generated drills.

Some such drills have extensive capabilities for keeping track of student responses and for recycling missed items into the pool of items to be learned. For instance, one very powerful drill and practice algorithm allows teachers to construct extensive lists of paired associate items, and to specify a number of parameters concerning their presentation, including:

1. the number of items out of the large pool to be grouped together for one exercise (e.g. 20 items at a time out of a pool of 5,000),
2. the number of times each item must be answered correctly before it is removed from the pool,
3. the number of times to wait after an item has been answered correctly before presenting it again,
4. the number of times to wait after an item has been answered incorrectly before presenting it again,
5. when items are answered incorrectly, the program will search its list to see if the student’s response represents a confusion with some other item. It will then present the confusion item with the original item to help the student eliminate the confusion.

In such a mature drill program considerable flexibility can be given to the teacher to tailor the characteristics of the drill to individual students and to teachers’ perceptions of the learning situation.

Paired associate drills are often placed in the context of games. For instance, the popular TV game called Concentration requires students to uncover and match hidden stimuli on a gameboard. The game requires that students not only memorize the stimuli and responses but also their locations on the gameboard. Some games add a timing factor to help students increase speed with performance.

In drill and practice programs such as these, the same program which is used to teach the material to the students may often be used for testing. In fact, many programs include completion criteria by which students may be graded. For example, a teacher may feel it is reasonable to require students to work on a drill until they have completed 19 out of the past 20 items correctly.

Keeping records of student performance in a CAI lesson may or may not be important to the teaching mission of a given instructor. In cases where record keeping is important, programs must make allowance for storing the identity and performance of individual students, and should provide a facility whereby instructors may print out that information for their records.
Dialogue

The dialogue approach to computer-assisted instruction is perhaps used more than any other in spite of the relative difficulty of effective design and implementation. Dialogue lessons attempt to emulate a dialogue between a teacher and a student. The computer, in effect, has information which the student must learn through some type of interaction. There are two general types of dialogues: computer-controlled dialogues (called tutorials) and learner-controlled dialogues (called inquiries).

**Tutorial Dialogue**

In the early days of CAI, tutorial was nearly synonymous with programmed instruction. Early tutorials presented a frame or more of information, asked a question and selected subsequent information based on student responses. Unfortunately, entrepreneurs at that time implemented tutorials on whatever computer happened to be around, used whatever computer languages might be available, and wrote materials often with little or no knowledge of good teaching practice. (No wonder that early CAI left a bad taste in the mouths of many educators.)

These problems were recognized early on and many institutions and organizations (including corporations such as IBM) tried to develop software packages which would assist in the preparation and delivery of tutorials. Author languages (which are covered later, in Chapter 6 of the handbook) such as COURSEWRITER, PLANIT, and PILOT were developed to offer educators templates into which they could type instructional content and through which they could provide instruction to students. Such systems provided methods for entering text frames, right answer options, wrong answer options, feedback options, and mechanisms for the selection of subsequent material based on student responses.
In many cases, the prospects for preparing successful tutorial lessons are much better today, providing you choose the appropriate types of hardware and software. Certain characteristics of tutorials are listed below along with some guidelines for considering their instructional use:

- **Linear page turning.** Novice tutorial designers begin by writing frames of text which require little or no interaction with the student. Essentially, the student turns pages electronically when going through such a lesson. Such an approach does not make effective use of the computer's capabilities and might better be left to textbooks and other print media.

- **Linear with branching.** Principles of programmed instruction may well adapt to implementation on computers. The design of effective instructional frames, the use of frequent student interaction, graphics, and other techniques can improve its quality. Details on these and other items may be found in Chapter 5.

- **Menus, modules, sequences.** Tutorial materials can be very boring if they present frame after frame of text without providing the student with a mechanism for organizing learning. This problem can be attacked by providing menus as organizers, by sequencing material in reasonable modules (15 or 20 minutes worth of work), and by using other techniques for organizing learning mentioned in Chapter 5.

**Design Considerations**

The effectiveness of a tutorial relies on its content and its design. Production of good tutorial lessons requires skill in a number of areas:

1. **Appropriate message design.** That is, the ability to arrange information reasonably and attractively in a fashion which best conveys its meaning.

2. **Good questioning techniques.** The ability to design good questions, both objective and subjective ones.

3. **Response judging.** Preparation of lessons which process various types of responses effectively.

The task of judging student input is perhaps the largest single problem in tutorial preparation. Objective questions (multiple choice, matching, true/false) tend to be safer to use. Short answer questions, or questions which elicit very limited free form language response, are often manageable although somewhat more difficult. The more complicated the expected response, the greater the difficulty in having a computer make a judicious assessment about the correctness or incorrectness of the response. A wrong judgment will send the student down a wrong path in the lesson, cause him considerable frustration, and result eventually in the failure of the instruction.

Standard programming languages such as BASIC, Fortran, and Pascal do not inherently provide for such answer judging structures. Programmers must provide them manually and they require a considerable amount of
programming overhead to produce. Specialized languages such as TUTOR and DAL have such capabilities built in. The availability of such languages ensures a much greater chance of success in the design and production of tutorial style materials with minimal programming effort.

**Inquiry Dialogue**

A slight twist on the idea of tutorial lessons as described in the section above has sometimes been referred to as “inquiry”. If the tutorial mode may be thought of as a situation in which the computer has information which will be systematically bestowed upon the student, the inquiry approach implies that the computer has information which the student must extract from it through a student-initiated dialogue. That is to say that the student must inquire about information relevant to the field of study.

Most characteristics of tutorial lessons apply also to inquiry. Many of the design problems are the same and the principles of message and frame design and response judging are similar; the major difference being that instead of judging answers as in tutorial mode the computer must now analyze questions.

Such analysis can be achieved if a vocabulary for the dialogue can be clearly defined. The vocabulary must include not only distinct words and phrases, but also the syntax in which words and phrases may be presented to the program.

The technical problems of developing programs capable of such language analysis are significant, particularly when using programming languages which were not designed with such a purpose in mind. DAL (as mentioned above) offers considerable advantage in this regard.

**Characteristics of Inquiry**

Inquiry resembles the process of problem solving which is dealt with in a later section. In an inquiry situation, the student is provided with an information environment and with a set of rules for obtaining information. The objective of an inquiry episode may be goal oriented or simply experiential. The student may have a specific objective or he may not.

A chemistry lesson, for example, might require a student to identify, through the use of laboratory procedures, an unknown chemical element or substance. A lesson for medical students may describe the condition of a hypothetical patient and require that the student, through the application of routine diagnostic practices, access the patient’s condition and recommend treatment.
In general, the inquiry approach may be more motivating for students than a tutorial one, and in some tutorial situations it may be possible to take advantage of this "twist" if the content material can be adapted and if sufficient technical and programming resources are available.

**Simulation**

Simulation may be defined as the controlled representation of real world phenomena. Simulations are employed in instructional situations where real world experiences are either unavailable or undesirable. Well designed simulations may be particularly appropriate when the real experience is too costly or awkward to provide, or when the element of risk is involved. For example, first time airplane pilots or automobile drivers may prefer to practice in a simulator rather than in the air or on an open highway. Science students (and people in the immediate vicinity!) may prefer to simulate dangerous experiments rather than actually perform them. In any event, instructional simulations assist students in the acquisition of knowledge and skills through surrogate experiences, and the success of any simulation relies heavily upon the degree to which it resembles the real experience.

**Characteristics of Simulation**

Three general types of instructional simulations may be identified. Practitioners do not necessarily agree on terminology and the labels used here are somewhat arbitrary, but the three types of simulations may be referred to as Task Performance simulations, System Modeling simulations, and Experience/Encounter situations.

**Task Performance Simulations**

These simulations are designed to assist students in the acquisition of skills related to the successful completion of a specific task. Flight and driving simulations are excellent examples of the task performance type. Such simulations are generally goal oriented and designed to help students increase skill in completing the task.

**Systems Modeling Simulations**

These simulations are used to assist students in the acquisition of information, insight, and understanding about a system. In general, such learning would be difficult to achieve through other means. For example, a mathematical model of the Chesapeake Bay Ecosystem has been built to enable researchers and scientists to examine the interrelationships among various factors affecting that geographical area. Other systems which educators have deemed suitable for simulation have included economics, world populations, the solar system, human anatomy, and transportation systems.
Such simulations offer students the opportunity to manipulate various system parameters and to examine their effects. These simulations often are process oriented rather than goal oriented. That is, manipulating the system to gain understanding of the operations and processes of the modelled system may be more important than achieving any specific objective within the system.

**Experience/Encounter Simulations**
These simulations often are used to provide students with exposure to ideas and experiences otherwise unavailable. For instance, a student who cannot travel to a foreign country might be led through the process of applying for a passport, selecting personal items to pack for the trip, negotiating customs inspections, reading menus, and other activities through which he may gain an intellectual appreciation of the real experience.

Preservice teachers, working their way through a simulation of their first year of teaching, may be exposed to pressures and decision situations in a relatively painless manner. For example, would you join the teachers’ union during your first year of teaching? What would you do if Johnny, who is captain of the football team (and whose father’s company just made a substantial equipment donation to your school), fails your midterm examination four days before the big game?

While many of these situations may never occur, simulations give students the opportunity to think about and internalize facts, feelings, and ideas.

**Design Considerations**
The problems in designing task performance and system model simulations are distinctly different from those related to experience models. In the case of task performance and systems models, the factors and parameters which bear upon the situation are highly quantifiable, and may be represented by numeric values or mathematical relationships. Experience models on the other hand are more qualitative in nature and often involve opinion and subjective situational analysis. For that reason, there is always the danger of introducing bias. Responsible designers often try to use a team approach or subject their individual work to peer review.

The design of task performance and systems models involve three or four distinct steps, depending upon whether or not the simulation is goal oriented or not:

1. Defining the task or system: identifying all variables which describe the state of the task or system. In a flight simulator, for example, such variables would include altitude, air speed, angle of flaps, fuel level, etc.

2. Presenting the task or system status to the learner: defining the method by which the student may ascertain the current state of the task or system. In a laboratory experiment on fractional distillation such methods may include
thermometer readings, the level of liquids in beakers, and the status of the bunson burner. (Notice that in such a simulation, and in many others, graphics can be a significant enhancement.)

3. Presenting a method for changing the status of the task or model: In most cases students will interact with the computer through a keyboard. Designers should keep in mind that the complexity of typing skills required to execute the simulation will have an effect on student performance. On some of the more elaborate computer systems students may alter the status by touching the screen or by manipulating game paddles and joy sticks.

4. Understanding criteria for task completion for task performance simulations: For example, in flight simulations, successful completion may be described as a condition in which speed and altitude both equal zero. Of course, the rapidity with which they approached zero may be a factor!

The design of experience models requires a considerably different approach. Variables are more likely to be used to remember decisions which the student made that may have an effect on later activities. Criteria for completion may or may not be necessary. The student may or may not be aware of which aspects of the simulation are really relevant to the outcome, and in some cases the status of the simulation at any point in time will be unclear.

Experience model simulations tend to be goal oriented, even if the goal is simply defined as completion of the simulation. The primary steps in the design of such simulations are:

1. Definition of a primary path by which the student can achieve the goal with minimal difficulty or distraction.
2. Identification of critical decisions by which the student may digress from the primary path.
3. Definition of alternative paths through which the student must travel as a consequence of the decision making process.

A popular microcomputer simulation entitled Oregon Trail leads students through a covered wagon pilgrimage across early nineteenth century America. Students must stock their wagons with food, ammunition, clothing, and personal items. Their selections have considerable effect later as they must make decisions and digress from the primary path. Students on a simulated trip to Russia may regret not packing tissue paper when they find it unavailable in the USSR hotels. Students in a teaching simulation may at the end of the “school year” face the consequences of having elected to grade on an absolute scale when half of their students make scores below sixty on the final exam.

Construction of experience models often provides a considerably more flexible environment for designers than other forms of simulation. Such simulations have potential for creative, innovative, and motivating instructional experiences and are one of the most underutilized CAI strategies.
Games

Few activities are more motivating and entertaining than games, as indicated by the recent phenomenal growth in the home video and arcade game market. Many games find their way into the software libraries of schools and sometimes into classroom related activities. But what constitutes a good game and what instructional value can it have?

A game may be defined as a goal oriented activity which can successfully be completed by the skillful application of a set of rules. More specifically, a game has an objective. The rules of the game specify the process by which a player can achieve the objective. The playing of the game (that is, the process of applying the rules in different ways) often constitutes a struggle for the player, a problem solving exercise. The player is motivated to engage in this struggle by his anticipation of eventual reward: winning. In many respects, games are not unlike simulations, and in some cases no distinction can really be made between the two.

Characteristics of Games

Games may be divided into two types: games of competition, and games of cooperation. The vast majority of games are games of competition in which two or more players compete against each other individually or in groups. Regardless of whether these games have any academic instructional value, they definitely teach competitiveness, individuality, and skills related to personal achievement.

Games of cooperation tend to emphasize group problem solving, teamwork, and social skills. One such game places players together on a desert island. The island has few resources but each player possesses both skills and supplies necessary to the survival of the group. It is in the best interest of each member of the group to assist his companions to survive.

For educational purposes games also may be divided into recreational games and instructional games. While the value of recreational games in a classroom setting is questioned by many educators, their use may be effective in reducing anxiety among new computer users and in assisting those users to become familiar with manipulating the machine. As new students become familiar enough with the game to enjoy it and play it for its own sake the instructional value is reduced. Many educators would prefer to introduce new students to computers using other methods.

Instructional games are those whose content and process are closely related to some instructional objective or benefit. Word games and vocabulary games, such as the Hangman game, are obvious examples.
Two other elements of games should be mentioned: the elements of skill and chance. Victory in many of the most effective games requires both. The element of chance insures that while increased skill will bring victory more often, the game will never lose its challenge.

**Design Considerations**
A game actually constitutes a formal system similar in many respects to other formal systems which we use every day. Language is a formal system. Mathematics is a formal system, as are music, rules of social etiquette, and expected classroom behavior.

Theoretically, the rules which govern the behavior of any formal system could be used to construct a game. The game then could be used to teach the characteristics of the formal system. While this may sound easy, the design of effective games requires considerable skill and artistic creativity. A game is instructionally sound to the degree that it accurately represents the formal system being taught.

The first step in designing an instructional game is to define the constructs upon which the game will be based. Ideally, those constructs correspond as closely as possible to the content and process which is to be taught. This step is perhaps the most important.

“*How the West Was Won*”, a computer game based upon the constructs of arithmetic, serves as an excellent example. The game features a race between a stagecoach and a train engine. Success in the game depends entirely upon a player’s skill in formulating arithmetic problems and computing their solutions. The computer randomly generates three numbers which the student must combine arithmetically \((2 + 3 - 1 = 4)\). On each turn the student must decide which combination of the numbers will produce an answer equal to the optimal number of moves on the gameboard. The student must then enter both the problem and the answer to receive credit and make the move. The constructs are exactly those of arithmetic and understanding of them is needed to win the game.

“*Lunar Lander*”, a popular game on many systems just a few years ago, provides a less effective example. Lunar Lander supposedly teaches students to understand the interrelationships between forces acting upon a space capsule as it attempts to land on the surface of a planet. While the student manipulates such aspects as direction and amount of thrust, understanding constructs is not critical to victory. The game can be won through trial and error. In fact, students rarely take the time to purposefully compute the moves. The constructs are those of physics, but understanding is not necessary for victory.
The second important step in the design of a game is the definition of the criteria for victory. Victory should require understanding of the constructs on which the game is built as well as skillful manipulation of the rules.

The third step involves the development of the rules by which the game will be played. What constitutes a turn? What restrictions must be placed on a player's use of the constructs in order to maintain fairness, promote the playability of the game, and control the pace? For example, in "How the West Was Won", on any given turn a player cannot use the same arithmetical sign twice. That is, 3x2x4 is an illegal move. This is a restriction on the constructs of arithmetic in order to insure that students don't ruin the playability of the game by simply maximizing the number of moves each time. More advantage may be achieved by landing on your opponent's space to send him backwards, or landing on a town to gain a bonus turn, or landing on the entrance to a short cut.

While the design and implementation of effective instructional games is difficult, an awareness of these considerations may increase the probability of both selecting and designing appropriate instructional games.

**Computer Assisted Testing (CAT)**

As computers have moved into classrooms, more and more teachers are examining the role of the computer in testing and evaluation. At one time or another computers have been used to support all aspects of the testing cycle. A general model of the testing cycle is described briefly below followed by a description of the role of the computer in each area.

**The Testing Cycle Model**

For purposes of this discussion, a fully developed computer-assisted testing system has five major capabilities:

- Test construction
- Test delivery
- Grading and analysis
- Item analysis
- Item banking

The term “testing cycle” implies a cyclical process in which tests are constructed based on the contents of an item bank. Items selected for any test eventually are graded and evaluated in such a way that the characteristics of the item are returned to the bank at the end of the cycle along with information which effects the manner in which those items will be used in the future.
Larger computers have enough storage space and power to support all aspects of the testing cycle. In the early days of commercial computing, computers for such purposes were beyond the budget of most schools, small colleges, and some universities. Yet the need for standardized testing allowed certain testing service organizations to develop. These organizations, such as the Educational Testing Service at Princeton, could afford to purchase large computers and provide test construction, evaluation and reporting, item analysis, and item banking as services to schools around the country. Over the years, these services have collected huge amounts of information which allow them to maintain national normative statistics on many aspects of our educational setting.

Over the years locally based smaller scale testing operations in support of classroom teachers using teacher-made tests have become popular. In these local situations any or all components of the testing cycle may be involved.

**Computer-Assisted Test Construction**

In preparing a test, the traditional classroom teacher makes certain implicit decisions. How many questions should the test contain based on the length of time available in which to administer the test and upon ease of grading? How many concepts should the test cover? How many questions should be asked on each concept? How many objective questions should be asked? How many essay questions should be asked? Among objective questions, how many should be true/false, multiple choice, matching, or fill in the blanks? Having generally made such decisions, the teacher then begins to construct individual questions or perhaps a pool of questions from which test items will be selected.
Test construction generally has been a manual process for teachers and those who are less organized often repeat the entire procedure from scratch for each new test. Other teachers keep the same tests from year to year in order to reduce the amount of preparation. But today, even personal computers can go a long way toward streamlining and improving the process. Even without real item banking, questions can be stored in text files. A reasonably unsophisticated computer program is capable of reading the items from the file and printing out copies of the test which can then be distributed to the students. With a little more effort, several additional advantages may be realized:

- The computer can be used to scramble the order of items so that two, three, or more versions of the same test can be produced, thus reducing the ease of cheating.
- Once the test items are on file, more items can be added and eventually an item bank can be built up.
- Even a relatively simple system eliminates the need to type and retype test forms.

Additional comments on test construction appear below in the discussion on Item Banking.

**Computer-Assisted Test Delivery**

If sufficient terminals or systems are available, students may be asked to take tests at a terminal. A number of advantages may be realized under this approach:

- The order of questions is irrelevant, since the computer can be programmed to know which question it asked and to remember the student’s response to that question.
- On systems which have an internal clock (the computer system is aware of date and time) students may be allowed to take the test independently within a certain time frame. This is particularly feasible if equivalent forms of the same test can be given to insure that all students don’t receive the same items. (Equivalent forms will be covered in the section on item banking.)
- The support of typing and grading tests and the problems of handling papers are eliminated.
- Students may receive immediate feedback on test performance. Grades can be made available as soon as the test is over.

**Grading and Analysis**

In the case of paper delivered tests, mark-sense answer sheets can be made available on which students can record the answers to multiple choice questions. The mark-sense answer sheets may then be fed to an optical scanner which will automatically record student responses and grade the test. With the proper software, a profile of the test can be constructed for the teacher that includes the distribution of scores on the test, the identifica-
tion of questions that are too easy or difficult, or other information gained from standard statistical treatments of test information. In addition, grades may be reported on paper printout or if the testing package is connected to an electronic grade book, grades can automatically be reported to the grade book.

Such a grade book might contain the names of all students in all classes, their scores on all tests, classroom assignments, and homework, as well as other pertinent information. A grade book package may have the facility to compute students’ grades automatically at the end of the semester or to provide an instant report of the student’s work.

**Item Analysis**
Effective objective testing requires constant reevaluation of test questions. In the case of multiple choice questions, effectiveness depends upon many factors, including the construction of the question stem, the selection and construction of the right answer, and the selection and construction of the distractors.

A good test item discriminates between more knowledgeable and less knowledgeable students. For example, it is reasonable to assume that the more knowledgeable students will answer most of the questions correctly, and the less knowledgeable students will answer less questions correctly. Therefore, an item which discriminates correctly will be answered correctly more of the time by the more knowledgeable students and will be missed more of the time by the less knowledgeable students.

An item which discriminates in the wrong direction (that is, a question that is answered correctly by people who score low on the test and answered incorrectly by people who score high on the test) is a bad item. The process of determining the discrimination characteristics of test items is called “item analysis”.

In fully developed computer testing operations, the computer maintains a profile of the history of each test item. Each time a test is graded which contains that item, an analysis of the item’s discrimination characteristics is made and the history of the item is updated. In this way, over time, bad items are weeded out and new items are added to replace them. Such a process is essential to continued effective testing.

**Item Banking**
Item Banking systems require sophisticated software and considerable computing power. The item bank itself may take years to develop.

But how are all these items stored? Large testing systems maintain the information in a data base usually referred to as an item bank. A large item
bank has the following characteristics:

- Items are categorized according to the field to which they apply (e.g. mathematics, English, or music). Each pool of discipline-specific items is further subdivided into areas of the discipline. Areas may be further subdivided as many times as necessary to create pools of items within the item bank which apply to specific instructional objectives or which test specific concepts. Ideally, a pool of many items all testing the same concept should be maintained.

- Each item in the item bank consists of two parts: the item identifier and the item itself. The item identifier is something like a serial number. It consists of information that identifies the item as distinct from all the other items in the database. It may also contain the item’s usage history and performance characteristics (how many times the item has been used, how many times it was answered correctly by high scoring students and incorrectly by low scoring students, etc.).

The advantages of item banking include all of those mentioned in preceding sections plus the ability to construct equivalent forms of a given test. Equivalent forms result from the possibility of selecting for each test totally different items which over time have been shown to be statistically equivalent. That is, the items test the same objective or concept and discriminate among more knowledgeable and less knowledgeable students in a statistically equal way. Based on national norms, a student taking two or three different forms of the same test should score equally well on each.

The feasibility of computer-assisted testing depends upon many factors, including hardware and software availability, the nature of the local testing situation, and the attitudes of teachers. One word of caution: the implementation of an effective system requires considerable investment of time and effort.

In all cases, teachers should avoid placing their students at risk during the development stages of a computer testing operation. That is, until the system has proven its accuracy and value, a backup testing system should be maintained. This ensures that if the computer system or its software fails to operate correctly, the teacher has another way to fairly and accurately assess student performance without placing undue inconvenience upon the students.

The system often can be tested by treating initial efforts as exercises rather than tests. This allows the students the benefits of practicing with the computerized formats and learning the material while the system is being debugged.
When contemplating the implementation of computer-assisted test procedures, teachers should use caution and common sense. Plan to start small. Test generators are often easiest to build and, when implemented in the simpler forms described at the beginning of this section, can produce both early success and enthusiasm.

A Final Comment About the Testing Cycle
In addition to objective tests based on stored items, algorithmic test generators (based on algorithms similar to those described in the drill and practice section of this handbook) can be used. In such tests the student's ability to perform a certain type of task will be assessed. The computer algorithm will generate some number of problem items which the student must answer. Algorithms are particularly effective in mathematics, science, music, and other quantifiable formal systems where algorithmic problem generators can be designed.

In particular, mathematics and music educators who have been using CAI for a long period of time have built up a bank of algorithms which may be used both for practice and testing across many different concepts and problem types.

This discussion has emphasized only the use of objective tests using stored items. At least two other points should be made.

First, many teachers prefer to use essay questions in an effort to assess student ability to synthesize concepts and ideas. The role of the computer today in grading essay tests is severely limited. Hardware capable of reading the written word (even the type written word) are new, expensive, and do not represent viable alternatives for most educational institutions. In addition, the use of the computer to analyze free form language is in its infancy, more a topic of research in Artificial Intelligence than a viable testing option for the classroom.

Second, the manual grading of essay tests is likely to be a highly subjective process. A student's performance on a essay test will be the result of a combination of factors including not only his knowledge of the subject, but his general ability, style of writing, his teacher's subjective opinions based on class experience and the mood of the teacher while the essay is being graded. Many experts on testing agree that well written objective items can assess the same knowledges and skills as essay tests and recommend their use instead.
Problem Solving

In many academic disciplines, particularly those related to mathematics and science, the development of a student’s ability to formulate and solve problems constitutes a major instructional objective. This objective may be relatively easy to accomplish for simple problems involving lower level skills. In such situations students can represent the problem manually. That is, they can write down a formula or other symbolic representation of the problem and compute the answer with pencil and paper. Certainly in most math classes, teachers require not only the correct answer but also a statement of the process by which the student arrived at the answer. In such situations, students usually record their calculations step by step.

In more sophisticated situations, not only may it be impossible for students to write down a step by step representation of the process, but the problem may be so huge and the calculations so time consuming that a student can lose track of the problem as a whole. In such cases educators have allowed students to use computers as tools in solving certain kinds of problems. Ideally, the student analyzes the problem, formulates a representation of the problem, designs and implements (or uses) a computer program to produce the correct result. The student is both relieved of the burden of computation and freed to concentrate totally on formulating and solving problems.

In non-mathematical disciplines, variations on the technique have been used. Models or computer programs have been developed to assist in solving problems specific to those fields. For instance, Schools of Business use programs such as the Interactive Financial Planning System (IFPS) to help students master the problems associated with financial planning and the use of spread sheets. The program allows students to build a model of a business based upon aspects of budget, production, sales, and other factors. The student then manipulates the business model in an attempt to improve his ability to plan for the business’s future.

Linguistic students, when faced with the problem of identifying the unknown author of a particular literary work, may use a textual analysis package. With such a package the literary characteristics of the works of known authors can be analyzed. The profile of such an analysis might contain lists of key words, phrases, and grammatical constructs along with their frequency of use or patterns of occurrence. For example, an analysis of the Shakespeare plays might produce a statistical profile by which Shakespearean works could be identified within the limits of statistical probability. Such an analysis of an unknown work might be compared with analyses of known authors of the same period, and often a reasonable prediction can be made concerning the identity of the authors.
Problem solving with computers is not really different from problem solving through other means. The computer is simply a tool that students must learn to manipulate in the same way that they learned to manipulate other tools such as calculators, slide rules, pencils and erasers. The strategy can, however, expand the learning experience in significant new ways.

**Discovery Learning**

The old cliche says that "experience is the best teacher". Indeed, the complete education may be thought of as the sum total of all of a student's experiences. From this perspective the act of teaching may be thought of as the process of arranging, organizing, and providing structured experiences for students—in most cases, experiences which students are unlikely to encounter on their own.

Such ideas are fundamental to the concept of discovery learning, a strategy by which educators place students in structured environments and provide them with tools for exploring, analyzing, and mastering new concepts and principles.

Much pioneering work in this area has been conducted at the Massachusetts Institute for Technology by Seymour Papert and his associates. Papert has long been concerned about the problems which many children have in mastering the fundamental concepts and operations of mathematics. He has compared the learning of mathematics to the learning of language and has speculated that children learn language so easily because it is a vital part of their day to day experiences and an essential tool in dealing with the world around them.

Papert has attempted to use the computer to develop such an environment for mathematics. One product of his research has been LOGO, a computer programming language of sorts, that allows students to communicate with the computer in simple terms. LOGO is graphically oriented and students move a "turtle" around the terminal screen in accordance with the rules of the language to produce graphic shapes and images.

LOGO is constructed in such a way that many of the constructs of mathematics are inherent in the language itself. So, as students learn to manipulate the language, try new combinations of commands, and produce either anticipated or serendipitous results, they are applying the basic principles of mathematics in a highly enjoyable, motivating, and natural way.
Theoretically such computer-based learning environments, like games, could be built for any formal system. Even though such a goal is difficult to achieve in practice, elements of discovery learning can be embedded in and around other CAI strategies. When adequate software is available, menus of related materials sandwiched between pre- and post-tests may be offered to students under “learner control” (the learner essentially has free choice over the manner in which he studies materials). A student may be allowed to take a pretest and receive recommendations on lessons to study, or he may simply browse through available materials knowing that eventually he must take and pass the post-test.

Particularly when the post-test is generated from a bank of items so that repeatable tests over the same material can be given, students may be allowed to select their own materials, study in their own style, and at their own convenience until such time as they elect to take and pass the post-tests.

With some of the more sophisticated computer management packages, such as those inherent in PLATO or in the Digital Courseware Authoring System, such environments can be constructed and controlled to facilitate discovery learning. The concept is somewhat more difficult to implement on systems of lesser capability. Such learner controlled environments are much more motivating than traditional lock-stepped, teacher controlled environments and their potential use should be considered whenever instructional objectives and available resources make them possible.
Incorporating Computers Into Instruction

The instructional process is cyclical. It begins with the formulation of educational goals and global curricula objectives and ends with an evaluation used to modify and improve instruction during the next cycle. A thorough treatment of the instructional process is beyond the scope of this handbook, but a general understanding of the steps in the process should enhance awareness of the computer's potential role and the problems associated with the design of computer-based materials.

Published experts disagree on details and terminology, but at least 8 general steps in the instructional cycle are identifiable. This chapter provides a brief discussion of these steps, followed by comments on the potential of the computer in this process. This chapter also includes tables of questions you might ask when trying to make decisions about which hardware and software to use in the instructional process.

---

The Instructional Cycle

1. Societal/Educational Goals
2. Global Curricular Objectives
3. Content Selection
4. Content Analysis
5. Behavioral Objectives
6. Evaluation
7. Delivery of Instruction
8. Design of Instruction

Eight General Steps in the Instructional Cycle

A BRIEF MODEL OF THE INSTRUCTIONAL PROCESS

The sections below describe steps in a general model of the instructional process.

1: Educational Goals and Curricular Objectives

Educational goals and global objectives of curricula are determined by society at large. Such goals are seldom explicitly stated, but are inherent in the basic values of society. These values force implicit decisions by school boards, educational administrators, scholars, and teachers.
Educational goals change very slowly relative to the other steps in the instructional cycle. The process of evaluation is more likely to change content and specific instructional objectives rather than change the goals and global objectives of an educational system. Educational goals are reflected by the presence in the curriculum of broad areas of knowledge, such as mathematics, language arts, and music.

2: **Content Specification**  
What should be taught? Can areas of knowledge and skills which comprise curricula be precisely defined at an operational level?

Every discipline has a history and each year new content must be added to that history in response to new developments. For example, in the areas of math and science, explosive growth has forced frequent and drastic additions to the content and has resulted in higher and higher levels of specialization, particularly at the college level.

Curricular content specifications often are specific to geographical locations or to specific school districts. Content is defined to some degree by the textbooks which school districts make available for use by their teachers, but, in general, teachers determine what content will be taught in their own classes.

3: **Content Analysis**  
Essential to the formulation of behaviorally oriented instructional objectives is the process of content analysis. In practice, content analysis and the definition of behavioral objectives often take place concurrently. Content analysis breaks down content into two types of building blocks: concepts and tasks.

A concept is generally accepted as the smallest single object of instruction. A task, usually related to a particular concept, is a process by which a student may demonstrate competency or understanding of the concept. If an adequate job of analyzing concepts and defining tasks is done, then clear behavioral objectives can be written and sequenced. This type of analysis goes a long way in ensuring the quality of all instruction, including CBE.

4: **Defining Instructional Objectives**  
What tasks must be performed to demonstrate understanding of content and concepts?

While teachers may handle the problem differently, all teachers must decide some way in which behavior is accepted as evidence of student learning. For example, a behavioral objective in elementary arithmetic may be stated as follows:
"The student will perform 20 problems in addition with 90% accuracy."

Such behaviorally-oriented specifications of objectives make it possible to design instruction and evaluation instruments to deliver and measure learning. Good, clear behavioral objectives are critical to the design of successful computer-based instruction.

5: Selecting Teaching Strategies and Modes of Delivery
Assuming adequate concept and task analysis and the definition of specific behavioral objectives, what are the best ways to facilitate student learning?

At this point teachers must decide how to present materials to students. The tools of the traditional classroom include lectures, paper and pencil exercises, textbook readings, and verbal recitations. In more recent years other media, including films, slides, and audio tapes, have been added to the teacher's resources. Any strategy selected must be appropriate for the content, for the learner, and for the delivery device.

6: Design and Implementation
Instructional design may be defined as the art of arranging content and activities in such a way as to best facilitate learning. Over time, effective designers acquire a pool of skills and techniques which they bring to bear on instructional problems. Classroom teachers often go through this process intuitively, usually serving as their own delivery systems. During delivery the teacher may drastically modify design based upon feedback from the class.

Explicit and well-documented design is more critical to the success of materials that are not "teacher delivered" (e.g. films, laboratory procedures, or computer-assisted instruction) because those media lack the intelligent flexibility of the teacher.

7: Delivery
What is the best way for students to interact with the instructional material?

In light of the considerable skill and flexibility that a good teacher brings to steps 6 and 7, the problems inherent in using machine-based delivery systems can be more fully appreciated. Such systems, even the most "intelligent" computer-based ones, cannot match the flexibility of a teacher. Careful selection and design of such delivery systems are all the more critical.

8: Evaluation
Instructional evaluation is often divided into two types: "formative" and "summative". Formative evaluation is on-going during the instructional process and involves the acquisition of feedback concerning the instruction from colleagues, students, and other sources. Formative evaluation helps to ensure constant quality control over the instruction.
Summative evaluation usually occurs at or after the delivery stage of instruction. It is useful in modifying the instructional process for the next cycle. Summative evaluation has its greatest effect on definition of instructional objectives and the subsequent steps in the cycle. On occasion, summative evaluation may alter content specification and even educational goals.

**THE ROLE OF THE COMPUTER**

Analysis of past uses of computers in instruction indicates that the primary instructional impact of the computer has been felt in the areas of delivery and evaluation. Most of the strategies outlined in the second part of Chapter 3 have to do with delivery, except for computer-assisted testing, which may be involved both in delivery and evaluation.

As systems and software become more sophisticated, their use will extend to the design and implementation of materials. Such applications will include the use of computer programs to define and develop other computer programs or other instructional media, such as films or books.

**WHEN TO SELECT THE COMPUTER**

A conservative proponent of computer-based education once remarked, “I have spent more time talking people out of computer-based approaches than I have spent in the design and production of materials.” That statement pointedly reflects an important problem in computer-based education today: the identification of appropriate instructional uses of the computer. Just because the computer can be used to teach a particular topic does not mean that it should be.

Two sets of decisions must be made:

- Computer capability must be matched to instructional needs.
- Benefits of a computer-based approach must be weighed against instructional costs.

**Matching Computing Capabilities to Instructional Needs**

Assume that a hypothetical designer has followed the instructional development model outlined above, and that the process of concept/task analysis and instructional objective definition has produced a reasonably accurate instructional needs assessment. Assuming that the same designer has decided to pursue the feasibility of using a computer-based approach, those needs (and potential delivery strategies) must be analyzed to determine requisite computing capabilities.
The following sections include discussion questions you should ask when deciding about computer-based education. They include lists for evaluating the pros and cons of hardware and software as well as a sample formula for measuring costs and benefits.

**HARDWARE AND SOFTWARE**

Computing capabilities may be divided roughly into two categories: hardware and software. "Hardware" means equipment—the physical devices which comprise the computer, such as terminals, tape and disk storage devices and memory. "Software" refers to computer programs—the electronic instructions which cause the equipment to perform a given task. The relationship between the two is crucial to this discussion and warrants a short discussion.

Historically, computer systems have been very expensive. Their major function has been data processing, scientific research, and other functions which required similar computing styles. Organizations which purchased these machines took for granted that they had to hire teams of technicians and programmers in order to tailor systems to their needs. Under those circumstances, software costs were buried in the salaries of the staff and were an accepted part of the operating overhead.

In recent years the costs of hardware and equipment have dropped drastically, particularly in the field of microcomputing. Software costs on the other hand have actually risen in real terms and now represent the major costs in computing. Consider that the cost of programming a particular package on a multimillion dollar mainframe may well be essentially the same as the cost of programming the package on a microcomputer costing a few hundred dollars. Coupled with the exponential explosion in the number of computing machines available, this fact dramatically demonstrates the demand for the scarce commodity of quality software.

What does this mean to the teacher/designer? It means that quality design and implementation of just one lesson may well cost more than the computer on which the lesson will be delivered! Under such conditions design and development activities may not be quite so attractive.

Increasingly, instructional packages which meet instructional needs (or which come close) may be already available for specific machines, and both hardware and software can be acquired by purchase. Such may be the case for microcomputer-based applications or for applications which run on larger computers identical to those available to the school. In these cases, design and development are unnecessary and the cost issue may be reduced to the assessment of costs and benefits as outlined in the next section.
# Hardware Considerations

A thorough awareness of the technical capabilities of various hardware is essential to making decisions about the sorts of instructional materials which can be successfully delivered. Does delivery require color, graphics, sound, light pen or touch input, or other specialized equipment? Concurrently, a thorough knowledge of software, including operating systems, programming languages, and utilities is needed to determine whether the equipment can be made to function as required.

A complete technical analysis may be beyond the capabilities of the average designer or teacher. When necessary, professional assistance should be sought to ensure that questions such as those listed in the table below have been answered to everyone's satisfaction.

## Hardware: What capabilities are necessary?

<table>
<thead>
<tr>
<th><strong>Student Workstation Characteristics</strong></th>
<th>How will students enter information into the computer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Typing</td>
<td></td>
</tr>
<tr>
<td>• Touching the screen</td>
<td></td>
</tr>
<tr>
<td>• Using a “light pen”</td>
<td></td>
</tr>
<tr>
<td>• With a “joystick”</td>
<td></td>
</tr>
<tr>
<td>• Using a “digitizer”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How will information be presented to the student?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Text on a visual display terminal (VDT)</td>
</tr>
<tr>
<td>• Graphics on a VDT (at what resolution?)</td>
</tr>
<tr>
<td>• Color or black/white</td>
</tr>
<tr>
<td>• Display speed (number of characters per second)</td>
</tr>
<tr>
<td>• Hardcopy (printed out)</td>
</tr>
<tr>
<td>• Video (motion images or still frame)</td>
</tr>
<tr>
<td>• Sound or voice</td>
</tr>
<tr>
<td>— Synthesized</td>
</tr>
<tr>
<td>— Recorded</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Machine Capacity</strong></th>
<th>How much computing power is needed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Main memory (type and amount)</td>
<td></td>
</tr>
<tr>
<td>• Mass storage (type and amount)</td>
<td></td>
</tr>
<tr>
<td>• Simultaneous users (under timesharing conditions)</td>
<td></td>
</tr>
<tr>
<td>• Processor speed (machine instructions per second)</td>
<td></td>
</tr>
</tbody>
</table>

[Computing capacity should be selected that will provide user(s) with response time of 1 second or less under most conditions.]
Software Considerations

Software issues deserve a special word, particularly with respect to the relationship between CAI and CMI. With the emphasis on the development of CAI materials and the delivery of lessons, the need for control over the computer-assisted instructional process is often overlooked.

Without adequate CMI capability, accurate and dependable records cannot be kept on student progress and performance. Without such records, evaluating student performance and the effectiveness of the instruction is extremely difficult. This lack of control and its consequent lack of evaluative data prevents much CAI activity from entering the mainstream of instruction. Therefore, both CMI and CAI software capabilities must be considered.

In addition, standard programming languages (such as BASIC, PASCAL, and FORTRAN) inherently lack certain capabilities essential to good computer-assisted instruction. Perhaps the largest shortcoming is in the area of response judging. When analyzing student responses, particularly short answer or freeform, sophisticated techniques are required to allow the program to handle irrelevant information, such as unanticipated capitalization, extra words, words in the wrong order, misspelled words, and synonymous words.

In some cases, predefined vocabularies of concepts and words are essential to adequate response judging. Developing such routines within the context of standard higher level languages can be a programmer's nightmare. Any language specifically designed for CAI should include commands which inherently perform these functions. The most notable examples of such languages are TUTOR and the Digital Authoring Language.

Good instructional software should be able to trap all types of program errors. Programs need to be "unbreakable" in the sense that no action taken by the student should cause program termination without explanation.

The best CAI systems also provide certain tools (utility programs) which make the preparation of materials easier. Many of these tools are covered in detail in Chapter 6.

In the final analysis, if a particular instructional design is going to make demands upon the special capabilities of hardware, software must be available to control it. If the software is not available, then it must be developed; and each development project adds cost and complexity to the entire package. The following is a list of questions you might typically ask when looking for software for a particular instructional situation.
What Software Is Necessary?

**Commercially Available Software**
- Can the required software/courseware be purchased?
- Is it of sufficient technical and pedagogical quality?
- Is it compatible with available computer systems?
- Is adequate documentation available from the vendor?
- Is it a "one-time" purchase or a periodic license?
- Does the vendor charge a maintenance fee for support?
- What restrictions, if any, are placed on software use?

**Locally Developed Software**
- What programming language(s) is available?
  - Can the language(s) control the student workstation equipment?
  - How effectively can the language(s) analyze student responses?
  - Will response time be adequately fast under normal conditions?
- Is a CMI package available?
  - What data collection and record keeping abilities does it have?
  - What types of reports can be generated?
- What "utilities" are available to support development?
  - text editors
  - graphics editors
  - character set editors
  - file handlers
  - debuggers
  - templates
  - others?

---

**ASSESSING COSTS AND BENEFITS**

The importance of measuring benefits against costs in CAI was clearly demonstrated by the fate of many of the early CAI projects mentioned in Chapter 2. SDC's CLASS system and TRW's MENTOR are classic examples of extremely high ratios of costs expended to benefit received. Regardless of the capabilities inherent in the respective systems neither was affordable within the educational structure of the times. Cost is always a factor. A thorough assessment of both the costs and the benefits of a good CBE project will not only demonstrate its feasibility but will also contribute much to garnering support from key administrators and other sources of funding.
Costs

While all costs of any particular CBE situation would be impossible to innumerate, many of the general costs related to hardware and software issues may be inferred from tables listed above. In addition, you should also consider less direct costs such as those shown in the table describing indirect costs, below.

Truly adequate measures of instructional costs are illusive, but when calculating costs for the purposes of comparing two methods of delivering similar instruction, consistency and common sense will be strong allies. The most frequently used cost measures tend to represent cost in terms of dollars spent for a particular unit of instruction (such as a course or a seminar) or dollars spent per unit of time (e.g. cost per student contact hour, per semester, or per school year). Whether these measures or others are selected depends entirely upon the demands of the situation.

Calculations for CBE situations are similar. Again, when analyzing costs for comparison purposes, care must be taken that the measures used are appropriate for the methods being compared. Some examples of possible cost measures are provided below:

1. A CBE project director may wish to know how much his project costs per hour of instruction. He will add up all costs innumerate above and divide the costs by the number of hours which students have spent in contact with his project. He will then be able to estimate the cost per hour of instruction for comparison purposes with other methods.

2. In some cases it may be important to know the cost per student work station. In such a case total costs would be divided by the number of work stations available for student use.

3. Principals, Parent-Teacher Associations, and school boards may be more interested in the costs of instruction per student per year. In such a case the annual costs of the CBE project might be divided by the total number of students to whom the project is available.

Other similar measures are certainly possible depending upon the methods used by individual institutions for purposes of cost accounting.

In addition to hardware and software costs, certain expenses must be incurred which are not directly related to computer technology. Many of these costs, largely related to personnel and operating expenses, are listed below. They must also be included in any assessment of the cost of CAI projects and operations.
Indirect Costs

Personnel
• Initial Capital Costs
  Software/courseware design
  Programming and development
  Documentation
• Continuing Costs
  Software maintenance
  Consultants
  Lab monitors and student aids

Supplies and Expenses
• Initial Capital costs
  Laboratory or classroom furnishings
  Furniture
• Continuing Costs
  Facility and equipment maintenance
  Lighting, electrical power, etc.
  Consumables
  — Paper
  — Printer ribbons
  — Diskettes
  — Tapes
  — Others

Benefits

So far, costs have been discussed without consideration for potential instructional benefits. Costs alone are valid measures of comparison only if the methods being compared are equally effective. Unfortunately, adequate measures of instructional effectiveness are generally not available until the instruction has been fully developed and evaluated. Therefore, the presence of an experienced CBE professional will be extremely important during the planning stages of any project. The professional should be able to make reasonable estimates about the general success of a proposed project.

In computer-based education, at least four areas of potential benefit have been prominent:
• Reduction in teaching time
• Reduction in instructional costs
• Increased instructional effectiveness
• Ability to teach difficult concepts more easily
These benefits may or may not be mutually exclusive, depending upon the type of CAI program under scrutiny. A reduction in instructional time will make both students and other instructional resources available for other activities. Reduced instructional costs may result from the elimination of costly equipment or procedures. An increase in instructional effectiveness may be demonstrated by higher student scores on evaluation measures. Finally, benefit may be gained by using a computer to teach concepts that are either impractical or difficult to teach through other means.

The table below contains a hypothetical comparison of costs and benefits for two methods of instruction which may serve as a useful model.

**Analysis of Costs and Benefits**

A. If the primary benefit of the CAI will be the teaching of content which cannot be taught without a computer:
   1. Add all periodic operating costs of the instruction implicated in preceding tables to estimate emphasis (annual operating expenses).
   2. Amortize total capital cost over the period of time the materials may be expected to be in use (usually some number of years). For example:

   $\frac{\text{Total Capital Cost}}{\text{Years of Use}} = \text{Annual Capital Cost}$

   3. Add the estimates from steps A1 and A2 above to estimate total periodic expenses (e.g. annual cost).
   4. Use total periodic cost or another appropriate measure (e.g. cost per student per year, cost per course, etc.) to decide if the instruction is worth the investment.

B. If the primary benefit of the CAI is expected to be reduced instructional cost (time) and/or increased effectiveness, a comparison of alternative methods may be appropriate:
   1. Estimate the cost of each method as appropriate (using the preceding tables to assist with CBE estimates).
   2. Estimate the benefit of each method based on data available. Units of measure should be the same for each method and should be compatible with units of cost estimate, e.g. if cost per course are used, then average student achievement in the course might be appropriate. In other cases, cost/student contact hour of instruction and student achievement may be more appropriate.
   3. Compare cost versus effectiveness for the two methods. A simple graph such as the one below may be helpful if more sophisticated methods are not available.
SAMPLE COMPARISON

COST OF INSTRUCTION ($$$)

METHOD A

METHOD B

STUDENT ACHIEVEMENT (%)

The graph will quickly show the relationship between methods. In the example shown, the cost of method A is 50 higher than that of method B, but the increase in student achievement is modest. Whether the increased achievement is worth the additional expense will depend on a variety of factors.

When considering the implementation of CAI, all these factors must be considered in addition to pedagogical considerations. The potential costs of the project must be assessed and measured against expected benefits. Only with such information can a wise decision be made about the incorporation of any CAI project into the instructional mainstream of an institution. Without such information and careful planning, major projects run numerous risks including the risk that insufficient resources will be available to produce the project and make it operational or that the project will not be effective enough to justify its cost. Such a failure can cause irreparable harm to future efforts.

On the more positive side, most CAI projects taking place today are modest in scale, and are not subject to the high risks of their counterparts. The film industry offers a convenient analogy.

It takes careful planning and execution by professional filmmakers to produce blockbuster motion pictures. The same is true of computer-based instructional materials. However, many individuals own equipment and produce home movies for their own use and enjoyment. Such may be the case for CAI today. An educator should not expect curricular miracles from his initial dabblings in CAI any more than he should expect to produce "Gone With the Wind" in his backyard with an 8mm movie camera. Nevertheless, computer-based materials, whether produced commercially or "at home," are having and will continue to have positive effects, and certainly computers should not be excluded from the warehouse of resources brought to bear on instruction.
SUMMARY

In general, the largest costs of instructional computing today, will not be hardware but rather the development and acquisition of good “courseware.” The least expensive systems (microcomputers) may not be the best investment because of the severe limitations they place on development potentials. A few more dollars invested on more sophisticated equipment and development software may significantly reduce development costs and increase the likelihood that the eventual cost/benefit ratio will be favorable.
Designing and Producing Computer-Based Educational Materials

Instructional design, the process of designing and implementing lessons on a computer, is very much an art. While professionals and educators may agree on basic principles, individual models are as different as people.

This chapter identifies four general (strategy-independent) steps in the design process:

- Preparing for Control of Instruction
- Designing the Lesson
- Program Development
- Implementing the Lesson

The importance of any step or detail varies according to strategy, type and amount of production resources available, the purpose and intended use of the materials, and other factors. Materials designed in and for local institutions tend to be loosely structured and informally managed. Professionally produced products intended for commercial distribution enjoy much tighter controls. The suggestions included in this chapter are intended to help you organize and identify important issues.

PREPARING FOR THE CONTROL OF INSTRUCTION

In the traditional classroom, the teacher not only delivers instruction but is responsible for managing delivery, collecting data, and evaluating both student performance and instruction. If students and teachers are to be accountable for computer-based instructional performance, then similar functions must be performed. Access to materials must be controlled in such a manner that data on student performance can be collected and reported easily.

Providing this capability may vastly increase the complexity of any CAI project. In some cases the design and production of control mechanisms and record-keeping facilities can require more resources than the instructional materials themselves! If students are going to be held accountable for performance on the materials, and control and record-keeping mechanisms cannot be provided or developed, then arrangements must be made to assess the instruction through other means. This issue should be dealt with at the earliest stage of the design process (see the discussion of CMI in Chapter 4). If your school purchased an instructional computing system that has CMI capabilities built in, this step becomes an opportunity rather than a problem.
DESIGNING THE LESSON

This section highlights five important aspects of lesson design:

- Student records
- The student/computer interface
- Lesson flow
- Scripting
- The design document

The topics of student records and the student/computer interface deal with overall design decisions; the topics of lesson flow, scripting, and the design document provide ideas and suggestions on the production of a design document.

Student records
If records are to be kept on student performance, three key questions must be answered: “What information to keep?”, “How to keep it?”, and “How to report it?”. Useful information may include:

- Scores on pre- and post-tests
- Score on the lesson itself (or parts of lessons)
- Student responses to individual questions
- Amount of time spent in the lesson
- Amount of time spent on individual questions (“response latency”)

Additional information of a nonevaluative nature may also be desirable:

- Date and time of student use of the materials
- Student’s current location in the program (to allow him to leave and come back at a later date)
- Number of individual sessions the student used in order to complete the lesson

Building a database to store and retrieve such information is generally a part of only the more sophisticated projects.

However, prepackaged record-keeping (CMI) systems do exist. They are mostly available through larger systems designed for computer-assisted instruction. Similar (but more limited) systems also are available for some personal computers.

Systems such as the Courseware Authoring System from Digital Equipment Corporation permit the arrangement of lessons in menus which control
student access to the materials. The package also provides data gathering capabilities inside its authoring language which generate student usage and statistics and pass them to the record keeping facilities. Report generators inside the CMI part of the system allow teachers to print student records. Such systems make record-keeping a reasonable option rather than a design nightmare.

The Student/Computer Interface
The student/computer interface defines the general characteristics of the student’s interaction with the computer. The design of this interface requires that you:

- Determine the degree and nature of control which the student will have over his learning environment
- Define appropriate lesson display conventions and screen layouts

Learner control involves providing the student with a sense of awareness of the lesson and mechanisms for manipulating it. A general rule is that at any point in the lesson the student should be able to answer three questions:

- “Where am I in this lesson?”
- “Where can I go from here?”
- “How can I get there?”

Many different tools can be provided to help the student answer these three questions:

- Lesson introductions (if not too extensive) can help.
- Graphic overviews are of great value.
- Block diagrams with arrows showing the flow of the lesson are helpful.
- The use of menus within a lesson breaks up distinct portions of a lesson and offer a good mechanism for overview and control.

Diagrams and maps can be used to show students subsequent steps in the lesson whenever necessary. Movement within the lesson is made easier with keywords which initiate immediate movement to predetermined locations or special function keys to make commonly used displays available.

The overall design of screen displays is equally important. In any lesson, displays should be consistent so that students develop a legitimate set of expectations for lesson performance. For example, the top four lines of the screen might always be reserved for instructions, whereas the bottom four lines might always be reserved for documentation of function keys or keywords. The middle of the screen might be reserved for the presentation of lesson content and for interaction.
Sample Screen Layout

If a consistent screen layout is used, variations of the layout can effectively highlight important concepts and ideas. Such consistency helps students retain a sense of security, particularly when a lesson undergoes a shift in content or interaction.

Lesson Flow
Smooth lesson flow depends on adequate task and concept analysis and the definition of specific instructional objectives.

The first step in deciding lesson flow is to arrange concepts and tasks in some sort of logical order. A concept map can be very helpful in this regard. Concepts and tasks can be arranged and sequenced according to their interrelationship. A concept map (such as the one illustrated below) shows which concepts are prerequisite to others and provides guidance in sequencing materials, building menus, and division of activities. In tutorial lessons, for example, each node or concept may become a questioning episode or a small exercise.

As each episode or exercise is designed, some representation of the computer’s activities must be produced in even greater detail. This representation must enable a programmer to implement the lesson in the chosen language. Flow charts are often used for this purpose.
CONCEPT A is comprised of SUBCONCEPTS A1 - A4. A and B are prerequisites to C. CONCEPT C is prerequisite to D (which consists of D1 and D2). CONCEPTS D and E are prerequisites to F.

Concept Map

Scripting
From a complete flow chart, distinct episodes and the individual frames within them may be identified. Depending upon the situation, careful attention must be paid to methods of presenting texts, constructing questions, judging responses, and providing feedback. Issues concerning the use of graphics, alternative media, and other peripheral devices may also apply.

The Design Document
Professional organizations usually produce a formal design document which serves as the template for programmers during lesson implementation. The level of detail may vary greatly but, in addition to other items, a design document generally contains:
• Hardware and software specifications
• Concept maps
• Database structures, variables, and file specifications
• Flow charts
• Frame designs
• A complete script where appropriate
PROGRAM DEVELOPMENT

Program development generally includes programming, debugging, and documenting the project. A good design simplifies development. While a discussion of specific programming concepts and principles is beyond the scope of this book, project managers should be aware that this activity often requires more time and effort than anticipated.

Documentation, the process of recording the operation of a program and the details for its use, is essential. Two types of documentation are important: technical documentation and user documentation.

Technical documentation includes the design document along with a listing of the computer program. The program must contain notes written by the programmer to assist other people in understanding how the program operates. Such information is essential for maintaining the program and for fixing it if it ever fails to operate properly. Other parts of the documentation include complete specifications of files that the program uses, lists of program variables and their uses, and any additional relevant information.

User documentation is designed to provide the teacher and students with directions on the use of the program. For the teacher such documentation should include a statement of objectives, a definition of the target audience, an overview of the lesson content, estimated lesson completion times, and any other information which will help the teacher arrange a proper instructional environment.

For the student, user documentation might include instructions, workbooks, tables, charts, graphs, and other material necessary for proper use of the program.
LESSON IMPLEMENTATION

The process of implementing the lesson includes pilot testing, revising, and the dissemination of the materials for student use. If the materials are delivered by a CMI system, then the materials must be properly linked to the CMI software.

Ideally, new materials should be piloted “in parallel” with other instructional methods. In other words, when first trying a drill and practice with a class, the teacher would be well advised to make the new drill an optional or ungraded exercise. This will allow students to use the materials at no risk and permits collection of data on performance of materials in the classroom. After initial piloting, appropriate revisions can be made and repeat the pilot if necessary.

Eventually the materials will reach “steady state” and can be phased formally into instruction. Summary evaluation data may then be collected along with comments from students and the materials may be used for grading and may be revised as required on a periodic basis.
6 Software Tools to Assist You in Production

Designers and producers of courseware in particular have developed many ways over the years to reduce the time and cost involved in the production and delivery of materials. Two types of tools, CMI systems and CAI templates, are designed to provide control structures for computer-based education. Two other types of software—author languages and editors—help reduce the production time of CAI programs. All such tools, when available, reduce the costs of producing and delivering quality CAI.

COMPUTER-MANAGED INSTRUCTION SYSTEMS

The ideas and principles of computer-managed instruction were described at some length in Chapter III. CMI offers instructors a prepackaged control system for the computer-based environment. More sophisticated systems running on larger computers provide not only menu drivers to control the order and presentation of instructional modules and lessons, but also record collection and data base facilities for the accurate reporting of student performance. A general CMI system provides a mechanism for overcoming problems of instructional control and is an important component of the delivery process. Figure 6.1 provides an example of a student menu page from a CMI system. Figure 6.2 contains a sample student progress report.

---

Student Menu Example

<table>
<thead>
<tr>
<th>COURSEWARE</th>
<th>Student menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>What would you like to do?</td>
<td></td>
</tr>
<tr>
<td>ASSIGNMENTS</td>
<td>Do assignments</td>
</tr>
<tr>
<td>BROWSE</td>
<td>Browse through lessons</td>
</tr>
<tr>
<td>MAIL</td>
<td>Run the system mail program</td>
</tr>
<tr>
<td>EXIT</td>
<td>Exit system (same as PF4)</td>
</tr>
</tbody>
</table>

PF2 = Help, PF4 = Exit

Use ▼▲, then RETURN
Student Progress Report Example

CAI Templates

Templates offer another control mechanism. Whereas CMI provides control at the curricular or course level, templates provide control at the lesson level. Over the years, templates have been developed for many of the CAI strategies described in Chapter 3.

Drill and Practice

For example, to the extent that all paired associate drill and practice lessons are similar, a single program can be written which can provide the structure for a drill on any subject.

A typical template has three components: a delivery program (often called a driver), a file structure for storing the drill items, and an editor for manually entering and deleting items from the files.

The driver provides the format for the drill by defining the screen displays, and the algorithm for presenting items. The algorithm may be based on the sequential selection of items from the files, the random selection of items from the files, or a combination thereof. It also provides record-keeping capabilities to support the assessment of student performance.
The major advantage of such a program lies in its adaptability to any lists of paired associate items. It can be used for the memorization of arithmetic facts, medical terms, foreign language vocabulary, or any other stimulus/response material. The driver need be written only once.

The file structure is designed to store drill content in a fashion usable by the algorithm. This may include files of items, and a history of each student’s use of that item. Such information is used to make decisions about the degree to which the student has mastered any particular pair.

An editor program shows the instructor what items are in the file and allows for the addition or deletion of items. The editor allows different instructors to build different files which may be attached to the algorithm (or to the driver) to vary the content.

**Other Templates**

Templates have been written for other strategies, too. Some authoring languages (discussed below) provide templates for tutorial programs in which the specific content is stored in files.

General purpose models for systems and simulations also have been built. Such models allow instructors to define in a systematic fashion the formulas and mathematical interrelationships of formal systems. The model then uses those formulas to mimic the system itself. With such a template, a particular model may provide experience with characteristics of the solar system, as well as the physiology of the human body. The key to the use of such models lies in the ability of the instructor to formally define the system in a manner which conforms to the requirements of the template.

Templates have also been built to support efforts in computer-assisted testing. Multiple choice testing templates include a driver for the presentation of multiple choice test questions, item banks for storing the items, and editors for building questions.

The major advantage of templates lies in the fact that the template itself only needs to be built one time and then may be used to support a specific type of CAI indefinitely. Designers and instructors would do well when planning computer-assisted exercises to examine the potential for acquiring or building a template. If a template can be acquired that will perform 80% of functions needed, it may be more cost effective than designing and producing materials locally.
AUTHORING LANGUAGES

Authoring languages are designed to facilitate the development of computer-based instructional materials. Essentially two types of authoring languages exist. The template-style authoring language described in the previous section as typified by COURSEWRITER, PILOT, and PLANIT. The programming-style authoring languages involve considerably more sophistication and in many ways cannot be distinguished from other higher level languages such as FORTRAN, BASIC, or PASCAL. The major beneficial characteristic of these author languages is that they contain enhanced capabilities for accepting and judging student responses, supporting graphics, collecting and storing student data, and controlling a variety of peripheral devices. Leading examples of such languages include TUTOR and the Digital Authoring Language (DAL).

While template-style languages have been designed primarily to provide tutorial or inquiry style instruction (some have been enhanced with graphics), higher level programming-style authoring languages are not strategy-dependent. In fact, they may be used in many cases to provide not only any form of instruction, but also to devise friendly "front ends" to noninstructional applications. In this respect, they are superior to traditional languages such as FORTRAN or BASIC.

EDITORS

The preparation of screen displays is important to the effectiveness of CAI. Instructional materials need to be not only pedagogically sound, but also aesthetically appealing. The aesthetics of a CAI lesson are no less important than those of a book or a film in that they play a large role in forming the student's attitudes about the lesson. The judicious use of graphics, color, and other special effects can be helpful in this regard.

Unfortunately, on many systems, even those which may have graphic capability, the preparation of attractive displays can be time consuming. In some cases preparation of graphics requires considerable technical skill and requires knowledge of machine language programming! More sophisticated and desirable systems have editors which allow developers to interactively construct screen displays by placing text and graphics on the screen and storing them for later recall by the program.
In the ideal case, such editors actually will translate the display into the programming language or allow for the incorporation of the code into the program. The illustration below provides an example of a special purpose editor for developing graphic displays for incorporation into CAI lessons.

**V1.01 - Command:**

Commands (and default address ranges) are:

- read * <filename>
- write * <filename>
- alter @
- copy @
- delete ..., update @
- group ..$
- move @
- scale @
- tilt @
- label.
- name .
- quit
- verify *
- join 1
- extract * <filename>
- font # <filename>
- background <filename>

---

**CREATE GRAPHS WITH SYMBOLS:**

**FREIGHT TRAFFIC IN KALAMAZOO HARBOR**

<table>
<thead>
<tr>
<th></th>
<th>(In thousands of ships)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

---

Editor for Developing Graphics Displays
Text Editors
Other editors, often called text editors, may be used for general purpose applications. They simply provide ways of building files full of text. These files may contain drill and practice items, text for screen displays, or the actual source code (program commands) for the CAI program. On large systems any number of such editors may exist.

On smaller systems, such as microcomputers, such editors are usually inherent in the operating system itself if they exist at all. BASIC subsystems such as those on the Apple and the Radio Shack (and even on larger systems) have a simple editor inherent in the subsystem which provides a mechanism for entering and storing programs.

Editors for Alternate Character Sets
Character set editors may also be available. Their nature and use requires some explanation. All computers have the capability to type out the letters and numbers of the English language. Often it is advantageous in CAI to be able to print out something other than letters and numbers.

In general, a letter typed on a printer or displayed on a screen is simply a computer response to an internal code of some sort. Receipt of the internal code by a computer terminal or a display screen will cause a particular letter to appear at a specified location on the screen. In such areas as foreign language instruction, it is advantageous to have the computer type out something other than an English letter in response to a particular code. For instance, instruction in Mandarin or Hebrew will require the ability to print foreign alphabets.

Therefore, more sophisticated CAI languages support alternate character sets. That is, not only can one print out the English character set but also the Hebrew character set or an oriental character set or any other available character set.

But where do these character sets come from? They must be defined. The definition of these character sets is the role of the alternate character set editor. The illustration below provides a screen display of the output of a typical character set editor. The editor is used to define which pixels (picture elements) in the character matrix will be “on” or “off” to form the desired shape.
Example of How a Character Set Editor Works

Animation with Alternate Character Sets
Alternate character sets also may be used to draw and animate pictures. A block of characters side by side (referred to as a multiple character image or a MOSAIC) may be used to enhance graphic output. The illustration below shows a picture of a test tube created with the mosaic character technique.

Using Alternate Characters in a Mosaic to Form a Test Tube
Character sets are powerful tools in instructional computing programming and greatly enhance the potential quality of instruction. They are generally available only on more powerful and flexible systems.

**UTILITIES**

In addition to those tools described here, other utilities such as programs for manipulating the contents of files, for copying materials from one location in the computer to another, for testing and debugging programs, and many other functions may be used by CAI developers to reduce the time and effort required in the preparation of materials. The availability of such programs and utilities should be a major consideration in the selection of a CAI system. Careful consideration should be given to the developmental cost savings which can be achieved with these tools.
Trends: The New Information Technology

For nearly five centuries print has been the dominant information technology in society. Until the 20th century all storage, retrieval, and transmission of information involved the printed word. The advent of telecommunications and computing have offered modern society alternative information systems.

FROM PRINT TO ELECTRONIC COMMUNICATION . . .

While print will continue for some time to be the dominant information technology, the combination of telecommunications and computers have created drastic changes in the way society handles information on a daily basis. Indeed, the operation of today’s governmental/industrial complex is impossible without the support of a vast network of computers in constant communication with each other. Interconnected computers are critical to our national defense, to our banking system, and to most other aspects of today’s economy.

The infusion of this new information technology into society has created hundreds of thousands of new jobs in the past two decades and has altered the requirements of a significant percentage of all other jobs, particularly in the white collar sector. Some estimates suggest that 75% of all U.S. jobs will require some knowledge of computers by 1990.

JOBS AND TECHNOLOGY
This drastic shift in employment trends and other changes are placing considerable pressure on our education system. The rapidly increasing number of computers available in schools amid the rising demand for "computer literacy" provide evidence of education's attempt to respond.

TRENDS IN EDUCATION
The changes required in our educational system will likely be more than cosmetic. Education today is fully based upon print media. Books and libraries are the critical repositories of knowledge and the basis for most information interactions. The question which concerns many educators today is "what happens to education if the information base of society shifts from print to electronic media?" Such a shift is likely to demand an alteration in the very fabric of the education system.
THE EFFECT OF COMPUTERS

For education to continue to meet the needs of society the new information technologies of that society must be integral to the education process. Ultimately, computers seem destined to become to schools what books have been for 500 years. The implications are staggering.

As computers become increasingly central to information interchange both in the home and at school, the role of computer-based education will likewise increase. New uses which extend far beyond those mentioned in this handbook will be found. More and more information will be available from an ever-widening circle of resources.

Teachers’ roles, classroom practices, and the administration of education will all change. Increased individualization of instruction will supplant traditional, lock-step approaches.

Changes in the job market already are making alternative training increasingly attractive to a growing segment of the population. Junior colleges, vocational and technical schools, and industrial training programs are attracting young people seeking technical training on the way to a secure future. Adult and continuing education courses are attracting larger numbers of participants as current workers attempt to retool and expand their horizons. In government and industry the benefits of CAI in the reduction of instructional time and costs are already being exploited. Traditional education eventually must face the same issues.

THE COMPLETE INFORMATION ENVIRONMENT

The most recent milestone in the integration of print, telecommunications, and computing is the computer-controlled video disk. The video disk can store still or motion pictures, sound, and other information in computer-readable form. It therefore represents a single storage medium common to these technologies. As other innovations follow, the ability of computers to handle all forms of information will increase as will the power and flexibility of systems themselves. In the near future (perhaps not in five years but certainly no later than twenty) a complete information environment should be available to the general public through a variety of information services delivered in the home and community.
This environment is likely to be driven by what researchers today are referring to as "intelligent CAI" and/or "expert systems". Such systems will take advantages of developments in the area of artificial intelligence and information processing to organize and present large amounts of information in new and different ways.

Electronic news media are already offering attractive alternatives to the home market. Such systems, when connected to huge data bases of information, offer interesting new opportunities for the retrieval and synthesis of information. Such techniques are likely to affect schools where sophisticated learning environments may greatly expand educational opportunities.

EPILOGUE

Regardless of what the future may hold, the demand for educationally useful computer systems is increasing. As those systems become more sophisticated and as prices continue to drop larger systems are falling within the purchasing power of schools. While only a few years ago microcomputers were the only financially viable alternatives for many schools, the purchasing decision is becoming less clear. The purchase of many stand-alone microcomputers may be an attractive option to many educators. However, a minicomputer supporting a network of inexpensive micros may be more cost effective in light of the increased power and flexibility.

Educators, now more than ever, must examine their changing educational needs and match them to the broadening range of technical resources available. No other issue offers a more critical challenge to education today.
PART 2

C.A.S. Technical Summary

This part of the handbook contains programmer-oriented information about the Courseware Authoring System. The components of the system are introduced in Chapter 8 and Chapter 9 describes the language elements that comprise the Authoring Language, the heart of the system.
Introduction to Digital’s Courseware Authoring System (C.A.S.)

DIGITAL’S Courseware Authoring System (C.A.S.) is a CBE system designed to make all the components of automated instruction available from a single software interface. The illustration below shows you the components of the system.

![Diagram of Courseware Authoring System Components]

Components of the Courseware Authoring System (C.A.S.)

THE USER INTERFACE

The User Interface gives access to all the elements of lesson delivery and is entirely menu-driven. Menus are designed to accommodate users who know little or nothing about computers, but can pick from a list, or newterm-ff(menu), of things to do.

C.A.S. is an “enclosed environment” for teaching. This means that students or naive users of the system cannot wander around the computer’s operating system. Rather, menus provide paths for both students and teachers through the system. As you can see in the illustration below, the menu is operated with the terminal’s keypad and arrow keys. Help is always available and you can redraw the screen at any point in a lesson.
Courseware Authoring System (C.A.S.) Menu Controls

These controls allow students or authors to move up and down a list of options. The illustration below shows the students' menu; you can see that the student doesn’t have many options to select. Students can’t do much more than do assignments, send mail to instructors, or exit the system.

The C.A.S. Student Menu

PF2 = Help, PF4 = Exit
Use 00, then RETURN
Student Browse Lists
You can see on the sample menu that there is a selection called "Browse." When students select this menu item, they see a list of topics they can look at but will not be scored for. These topics are generally unrestricted lessons which appear on everyone’s browse list. However, authors can restrict access to lessons for just one or for several classes. For example, you might not want sophomore physics students browsing through junior courses.

As soon as a lesson is assigned to a student, that lesson is removed from his browse list until he completes the lesson. This prevents the student from practicing a lesson while he is being tested.

Types of Users

The User Interface is sensitive to the type of user using the system. There are three types of users:

- Students
- Instructors
- Authors

The User Interface recognizes the different responsibilities of each type of user and presents each with an appropriate (and different) menu. We’ve already seen how the student uses C.A.S., but the instructor and the author deserve further discussion.

The Instructor’s Role
Instructors have more responsibilities and more options than students. They register students and keep track of assignments. They also access report functions to see student scores, class averages, and so forth. For the sake of system security, instructors are associated with their classes and can look at records only for the students in those classes. (The Physics teacher probably doesn’t want to look at the English Literature grades anyway!)

The Instructor can also specify whether assignments are structured or unstructured. Structured assignments must be taken in a specific order and there are usually prerequisites which are enforced. Students may complete unstructured assignments in any order they like. The Student Status Report indicates whether assignments are complete, partially done, or not yet begun.
Instructors also decide whether their groups need passwords and whether they can access their accounts only through their own VMS account. The Instructor’s Main Menu is shown below; other Instructor Menus are shown in Appendix A.

The C.A.S. Instructor Menu

The C.A.S. Author Menu
In general, instructors register students, make assignments, and monitor the progress of students.

The Author's Role
Authors are primarily responsible for newtermf(publishing) lessons. Publishing is the process of creating, maintaining, and distributing lessons on the system. The author's menu deals mostly with these types of functions. They also have the option of looking at certain reports which deal with the maintenance of lessons.

All of the menus available in the C.A.S. system are shown in Appendix A.

BOOKKEEPING AND REPORTING

C.A.S. provides a means of “remembering” such important data as whether a student completed half of a lesson, which questions he answered correctly, and where to pick up when he returns to the lesson. C.A.S. bookkeeping functions also notify students of assignments, track due dates for assignments, and record lesson completion.

This bookkeeping function collects many types of statistics which can be accessed by means of 11 different kinds of reports. For example, one report lets the instructor see a single student’s data to monitor that student’s progress. Another type of report gives a histogram of class scores, and another prints all student responses to a particular unit in a lesson. A typical report is shown below; all the reports available are illustrated in Appendix B.

![Histogram of Scores for Lesson: Multiply]

DATE: 3/30/82 HISTOGRAM OF SCORES FOR LESSON: MULTIPLY TIME: 09:16:57

C.A.S. Report
C.A.S. AUTHORING TOOLS

All material to be learned is seen by students in the form of lessons. For example, a student could pick from a lesson on mathematics, or history, or literature. In any of these cases, the material he picks is in the form of a lesson. Likewise, assignments are given in the form of lessons.

Lessons are built with authoring tools. Authoring tools are software tools used by authors to create and maintain lessons. The programming language itself is the most important tool. It is called DAL, the Digital Authoring Language. This language is specially designed for authors of lessons and includes instructions and built-in functions and variables. The language is designed to work with C.A.S. to collect data, monitor student progress, control lessons.

Other software tools include graphics editors, character set editors, and text editors. These tools are briefly described in Chapter 6. But the most important tool is the Digital Authoring Language (DAL), which is described in the next chapter.
9 DAL: The Digital Authoring Language

At the heart of C.A.S. is the Digital Authoring Language (DAL), designed for developing computer-based instruction. DAL includes modern code structuring facilities that make lessons easy to read, modify and maintain, so that authors can learn the language quickly. DAL provides language elements for control, sequencing and structure, graphics calculations, statistics, and response checking. Authors who have experience with other authoring languages will find DAL more powerful and easy to use.

The Digital Authoring Language offers features common to other VAX/VMS languages, including access to VAX/VMS system services and standard call interfaces to routines written in other computer languages. DAL is implemented as a native mode compiler under the VAX/VMS operating system. DAL produces object files that can be linked to produce native mode executable lessons, thus providing optimal system performance during lesson delivery.

A Simple Structure for Easy Operation

Each lesson written in DAL consists of a lesson level and one or more units. Lesson level instructions define lesson variables and lesson goals and determine the sequence by which units are executed. Units usually are a sequence of presentation, questions, student response, and feedback. This separation of control and content does not prohibit one unit from calling another as a subroutine.

In general, units display data to a student, ask a question, and provide feedback based on student response. The author controls the display of feedback through branching facilities. If the student responds correctly, the system returns a “satisfied” indicator. This allows the lesson control logic to perform conditional actions based on the student’s answer. Units may also define their own variables, and perform the logic and control functions.

Lessons are independent but can communicate with each other. One lesson can call another as a subroutine; the system treats the pair of lessons as one large lesson. Authors can keep usage statistics for this type of lesson as if the statistics applied to a single learning lesson.
A Variety of Commands

Lessons in the Digital Authoring Language consist of lines of language source code. The source code is written in an easy-to-read, structured format of instructions and arguments.

Each Digital Authoring Language instruction is a single mnemonic. Instructions are grouped into six categories including: lesson structure, control logic, variable definition and calculation, response judging, scoring responses and accumulating data, and text and graphics instructions.

Lesson Structure Instructions
Lesson structure instructions provide the main structure of a lesson, defining lesson and unit names. These instructions also control the overall execution of units. Authors can use lesson structure instructions to call programs outside the lesson such as statistical packages or control routines for real-time devices.

Control Logic Instructions
Control logic instructions manage the sequence, iteration, and conditional execution of other instructions. Most of these commands are used in pairs to begin and end a code structure. Users can nest control logic structures several levels deep.

Variable Definition and Calculation Instructions
Variable definition and calculation instructions define the names and formats of all variables, and initialize the variables. They also define data types (real, string, integer and Boolean). These instructions provide the author with advanced mathematical capabilities for complex calculations and data manipulations.

Response Judging Instructions
Response judging instructions perform the vital function of judging student responses. DAL can parse all student responses with one sequence of instructions. The author defines what constitutes a right or wrong answer and the appropriate action to be taken in each case.

Authors can control whether a response may vary from, or must match exactly, an expected student response. A DAL lesson can check student responses for spelling, synonyms, punctuation, and units of measurement. Lessons can accept answers in the form of words, sentences, phrases, and mathematical notation. It can even convert student answers from one unit of measure to another (e.g. quarts to liters). Authors can also choose to ignore words extraneous to the answer.
Response Scoring and Data Accumulation Instructions
Scoring responses and accumulating data allow authors to control scoring. Through scoring instructions, answers can be weighted differently. Authors can accumulate data on unexpected responses, time required for each response, time required for the weighted differently. Authors can accumulate data on unexpected responses, time required for each response, time required for the entire lesson, and much more. DAL will file this information automatically when students take lessons.

Text and Graphics Instructions
Text and graphics instructions give every author the full range of graphics capabilities available with the GIGI terminal. These capabilities include a palette of eight foreground and background colors, the ability to select four alternate character sets, up to 16 variable text sizes, different display modes such as blink and erase, overstrike capabilities, and the absolute positioning of graphics.

The Digital Authoring Language allows authors to design text and graphics for a lesson. With text, they can select the screen address, specify the text, and include any necessary modifications like italicization, changes of character size, and mode alterations. Relocatable pictures can be drawn with any combination of dots, curves, lines, boxes, circles, or vectors. The diagrams can be shaded with characters or with predefined shading patterns.

Plotting Instructions
Instructions are available that allow plotting of an x and y axis coordinate system. It takes only one command to define and plot each axis, and one command to graph each mathematical function with relative graphics. Authors can set the origin of the coordinate system at any point on the screen; both positive and negative values can be plotted. Authors also have windowing capabilities to define exactly where graphics can be displayed and where they are restricted.

Graphics in DAL Lessons

There are two ways to include graphics in a lesson. Authors may use DAL commands to design graphics and text in a lesson. Graphics can also be created separately and included in text as needed. Separate picture files can be created with the ReGIS Graphics Editor, ReGIS Data Plotting Package, or other ReGIS-based software utilities.

With DAL, authors can place text and/or graphics anywhere on the screen. There are three screen coordinate systems that can be used singly or together: Gross, Fine, and Normalized. Gross coordinates are best for
setting text. Fine coordinates address the smallest displayable unit of the screen, and are used to create most graphics. Normalized coordinates, which specify an area that is a proportion of the total screen area, are suited to graphing arithmetical functions.

Variables

The Digital Authoring Language has both user-defined and system variables. The user-defined variables may be defined at the lesson level and then referenced by any unit. A variable may also be defined by a unit, in which case only that unit can reference the variable.

User-defined variable names can be from two to eight characters in length. The variable must be defined with either a data type or a usage characteristic. Usage characteristics determine if a student can use a variable, and whether or not the variable is permanent. Permanent variables are automatically filed between sessions. Both the variable and its most recent value are available after the lesson is completed.

DAL maintains several system variables. A lesson can read every system variable and can update many of them. There are three levels of system variables: graphics and graphing (e.g., cursor position); response related (e.g., last student response); and timing and miscellaneous (e.g., time limits).

Creating Lessons

Once an author has written a lesson, it must be compiled and linked, like all VAX/VMS programs. The Digital Authoring Language compiler generates an object file from the lesson source code. It can also produce listing files containing source code listings. The compiler is simple to operate. The user enters the compiler command, the name of the source file, and optionally, any of the four available compiler switches. If more than one source file name is included, the instructions in all source files are compiled into a lesson.

The compiler also displays error messages during execution, both at the terminal and in a listing file.

Object code produced by the compiler requires one more step before it can be executed. The VAX/VMS linker takes the compiled code and produces shareable, executable lessons by linking all specified files into a single executable image. For maximum system performance during lesson delivery, DAL’s run-time support system is shared among users.
SUMMARY OF DAL INSTRUCTIONS
The following sections list DAL instructions; the instructions are grouped functionally.

Program Control Logic Instructions

**ASSIGN**
The ASSIGN instruction assigns a value to a variable. The variable must be defined with the DEFINE instruction before a value can be assigned.

ASSIGN variable := argument

**BRANCH**
The BRANCH instruction transfers control to another point in the lesson. The transfer is conditional, depending on the arguments passed via BRANCH.

BRANCH condition,$label1[,$label2]

**DEFINE**
The DEFINE instruction defines the name, data type, and usage characteristics of variables, as well as the name and value of constants.

DEFINE name,name,..:datatype[,usage]

**DO**
The DO instruction executes the unit or the lesson named as its argument. This instruction can be used at lesson level to control the sequence of units in the entire lesson, and at unit level to control execution of one unit from another.

DO unitname[(arg__lst)]

**ELSE**
The ELSE instruction marks the beginning of a series of instructions that are executed only when the condition specified by the preceding IF instruction is false.

ELSE

ELSEIF

ENDIF

**ENDFOR**
The ENDFOR instruction marks the end of the iterative instructions begun by the preceding FOR instruction.

ENDFOR
ENDIF
The ENDIF instruction marks the end of the instructions that are conditionally executed by the preceding IF instruction.

ENDIF \[\text{RET}\]

ENDLESSON
The ENDLESSON instruction is the last instruction in a source file in DAL. It is a signal to the compiler that this is the end of the source code.

ENDLESSON \[\text{RET}\]

ENDLOOP
The ENDLOOP instruction marks the end of the instructions that are repeated as long as the condition specified by the preceding LOOP instruction is true.

ENDLOOP \[\text{RET}\]

FOR
The FOR instruction begins an iterative structure and specifies the number of times the instructions between the FOR and its associated ENDFOR are executed.

FOR counter: = initial\_value,ending\_value[,increment] \[\text{RET}\]

FUNCT
The FUNCT instruction begins a specialized subroutine that either defines a new function, which then can be called like the system functions, or redefines a system function. The DEFINE instruction must be used at lesson level to define the name of the function.

FUNCT name(argument[,argument,\ldots]:return\_type) \[\text{RET}\]

IF
The IF instruction marks the beginning of an IF structure and specifies the condition to be tested.

IF boolean expression \[\text{RET}\]

LESSON
The LESSON instruction specifies the name of the lesson. It also defines the beginning of a lesson for the compiler.

LESSON lesson\_name \[\text{RET}\]
LOOP
The LOOP instruction marks the beginning of a LOOP structure. A LOOP structure contains a series of instructions that are executed as long as the condition specified by LOOP is true.

LOOP boolean expression

OTHER
The OTHER instruction marks the beginning of a series of instructions that are executed only if none of the value instructions between the preceding TEST instruction and the OTHER instruction have caused a different set of instructions to be executed.

OTHER

.
.
.

ENDTEST

OUTLOOP
The OUTLOOP instruction is an optional instruction in a LOOP structure. The LOOP instruction specifies a condition that must be false if control is to be passed to the instruction following the ENDLOOP. The OUTLOOP instruction specifies an alternate condition that also causes control to pass to the instruction following the ENDLOOP.

OUTLOOP boolean expression

REDO
The REDO instruction reexecutes the current unit when it is used at unit level, or reexecutes the lesson when it is used at lesson level.

REDO

RELOOP
The RELOOP instruction, like the OUTLOOP instruction, provides an alternate path out of a LOOP structure. RELOOP specifies a condition that passes control to the preceding LOOP instruction before the ENDLOOP is reached.

RELOOP boolean expression

RETURN
The RETURN instruction returns control to the calling unit or lesson before the end of the unit or lesson in which it is executed.

RETURN
STOP
The STOP instruction ends execution of the lesson. The restart variables are saved, and the lesson is considered incomplete.
STOP [ABORT]  RET

UNIT
The UNIT instruction serves two functions: it specifies the unit’s name in the form needed by the DO instruction and it identifies the beginning of the unit.
UNIT unitname [(arg__lst)]  RET

VALUE
The VALUE instruction is part of a test structure and serves two purposes: it specifies a value, a series of values, or a range of values to be compared to the contents of the variable specified in the associated TEST instruction. VALUE also marks the beginning of a series of instructions that are executed only if its argument matches the value of the variable specified in the associated TEST instruction.
VALUE value,...or range__of__values  RET

Figure Drawing Instructions

BOX
The BOX instruction draws a rectangular box on the screen using the specified addresses as opposite corners of the rectangle.
BOX [corner__address]opp__corner__address[:thickness]  RET

CIRCLE
The CIRCLE instruction draws a circle whose center and radius are specified. When optional arguments are used to specify the beginning and end of an arc, CIRCLE draws only the arc.
CIRCLE x,y:r[,begarc,endarc]  RET

CURVE
The CURVE instruction fits a curve to a specified number of points on the screen.
CURVE point;point;point;...;  RET
DOT
The DOT instruction illuminates the smallest displayable screen element.

```
DOT x, y  RET
```

ERASE
The ERASE instruction erases all or part of the screen. The current background color is restored.

```
ERASE [corner_address; opposite_corner_address] RET
```

FCOLOR
The FCOLOR instruction specifies a color for writing subsequent text and graphics.

```
FCOLOR  0 = DARK
         1 = BLUE
         2 = RED
         3 = MAGENTA
         4 = GREEN
         5 = CYAN
         6 = YELLOW
         7 = WHITE
```

GBOX
The GBOX instruction is the analog of BOX using the graphing coordinates. The GBOX instruction draws a rectangular box on the screen by mapping the values used as its arguments to the screen with the graphing coordinates defined by the most recent GORIGIN, AXES, and SCALE instructions.

```
GBOX [corner_address;] opp_corner_address[:thickness] RET
```

GCIRCLE
The GCIRCLE instruction is the analog of CIRCLE using the graphing coordinates. The value specified as the center of the circle is mapped to the screen with the graphing coordinates defined by the most recent GORIGIN, AXES, and SCALE instructions. Optional arguments draw an arc.

```
GCIRCLE x, y., r[, begarc, endarc] RET
```

GCURe
The GCURVE instruction is the graphing analog of CURVE. Each argument specifying one point on the curve is mapped to the screen with the graphing coordinates defined by the most recent GORIGIN, AXES, and SCALE instructions.

```
GCURe point; point; point;...; RET
```
GDOT
The GDOT instruction is the analog of DOT in the graphing coordinate system. The value used as its argument is mapped to the screen with the graphing coordinates defined by the most recent GORIGIN, AXES, and SCALE instructions.

GDOT \( x, y \)  

GLINE
The GLINE instruction is the graphing analog of LINE. The two points specified as its arguments are mapped to the screen using the graphing coordinates defined by the most recent GORIGIN, AXES, and SCALE instructions. GLINE draws a line between these two points.

GLINE \[ \text{beg\_}x, y; \text{end\_}x, y \]  

GORIGIN
GORIGIN is one of the instructions that define the screen location and the scale for graphs. GORIGIN defines the point on the screen that is the origin of the graphing coordinate system. AXES defines the screen location of the ends of both axes. SCALEX and SCALEY define the scale as linear. LSCALEX and LSCALEY define the scale as logarithmic.

GORIGIN \( x\_\text{address}, y\_\text{address} \)  

GVVECTOR
The GVVECTOR instruction is the graphing coordinate analog of VECTOR. The values that define the head and tail of the arrow are interpreted as points in the graphing coordinate system defined by the most recent GORIGIN, AXES, and SCALEX and SCALEY or LSCALEX and LSCALEY instructions.

GVVECTOR \[ \text{tail\_}\text{address}; \text{head\_}\text{address}; \text{head\_size} \]  

HREF
The HREF instruction specifies a reference line used to shade figures drawn with the graphics instructions. The area from the reference line to the line drawn by subsequent graphics instructions is shaded in the current foreground color and the current pattern.

HREF \( y\_\text{coordinate} \)  

LINE
The LINE instruction draws a straight line from one point on the screen to another.

LINE \[ \text{beg\_}\text{screen\_address}; \text{end\_}\text{screen\_address} \]
RBOX
The RBOX instruction is the analog to the BOX instruction in the relative addressing system. The RBOX instruction draws a rectangle on the screen using the specified addresses as opposite corners of the rectangle. The addresses are interpreted as relative to the current RORIGIN.
RBOX [corner_address;]opp_corner_address[:thickness] RET

RCIRCLE
The RCIRCLE instruction is the analog of CIRCLE in the relative graphics system. The RCIRCLE instruction draws a circle whose center and radius are specified. The address of the center of the circle is interpreted relative to the current RORIGIN. Optional arguments specifying the beginning and end of an arc draw only the arc.
RCIRCLE x,y:rl,[begarc,endarc] RET

RCURVE
The RCURVE instruction is the relative address analog of the CURVE instruction. The RCURVE instruction fits a curve to a specified number of points on the screen. The address of the points are relative to the current RORIGIN.
RCURVE point;point;point;... RET

RDOT
The RDOT instruction is the analog of DOT using relative addresses. The RDOT instruction illuminates the smallest displayable screen element. The address of the dot is relative to the current RORIGIN.
RDOT x,y RET

RLINE
The RLINE instruction is the relative graphics analog to LINE. The RLINE instruction draws a straight line from one point on the screen to another. The addresses of the points are relative to the current RORIGIN.
RLINE[beg_screen_address;]end_screen_address RET
ROTATE
The ROTATE instruction specifies that the addresses used for subsequent relative graphics (R-prefix instructions) are rotated a specific number of degrees before the figure is drawn.

+ ROTATE  degrees  RET
-

RSIZE
The RSIZE instruction modifies the vertical and horizontal dimensions of relative graphics.
RSIZE x__size,y__size  RET

RVECTOR
The RVECTOR instruction is the relative graphics analog of the VECTOR instruction. The RVECTOR instruction draws an arrow between any two points. The size of the arrow head is proportional to the length of the arrow.
RVECTOR [tail__address;]head__address:head__size  RET

VECTOR
The VECTOR instruction draws an arrow between any two points. The size of the arrow head is proportional to the length of the arrow.
VECTOR [tail__address;]head__address:head__size  RET
Input/Output Instructions

CLOSE
The CLOSE instruction closes a file so that its contents are no longer available to the lesson. CLOSE can store the file on disk or delete it.

CLOSE channel_no[,DELETE] RET

GET
The GET instruction reads a record from an open file into a string variable.

GET channel_no,variable[,record_no] RET

INCLUDE
The INCLUDE instruction inserts DAL source code contained in another file into this source file at compile time.

INCLUDE "sourcefile__name" RET

INPUT
The INPUT instruction displays the prompt character and waits for a response from the keyboard.

INPUT address RET

OPEN
The OPEN instruction performs the following file functions:

• Establishes a channel number for accessing the file and for identifying it in subsequent GET, PUT, and CLOSE instructions.
• Finds the file, opens it, and verifies that the file attributes correspond to the attributes specified in the instruction. If the file cannot be found, the OPEN instruction creates a new file with the name and attributes specified.
• positions the file.

OPEN filename,chn__no,model[,RANDOM,rec__size] RET

READ
WRITE
UPDATE

PUT
The PUT instruction writes a record into a file.

PUT chn__no,variable[,rec__no] RET
Response Judging Instructions

CONVERT
Authors can specify that students must enter both a value and units of measurement in responses. The CONVERT instruction specifies the formula used to convert a value expressed in one unit of measurement to an equivalent value in another unit.

CONVERT primary_unit = secondary_unit [RET]

DELIMIT
The DELIMIT instruction specifies the character that ends a student's response. The default character is the carriage return generated by the RETURN key.

DELIMIT numeric expression [RET]

ENDQ
The ENDQ instruction marks the end of a series of response-judging instructions begun by the QUERY instruction.

ENDTEST
The ENDTEST instruction marks the end of a TEST structure.

ENDTEST [RET]

GOAL
The GOAL instruction divides units of a lesson into sections that are scored individually.

GOAL number [RET]

JUDGE
The JUDGE instruction modifies the judgment of student responses. Both the judgment that a response is right or wrong and the events that follow the judgment can be changed.

JUDGE AGAIN [RET]
CONTINUE
IGNORE
NO
OK
STOP
LOG
The LOG instruction creates a file containing information about student performance. Each time the lesson is executed, a record is appended to the file.

LOG WORDS
UNITS
RESPONSES
LATENCY
FULL
OFF

MARKUP
The MARKUP instruction writes the contents of the system variables OKWORD or NOWORD two spaces to the right of the student's response on the screen or, if there is not enough room, two lines below the prompt character.

MATCH variable,string,string,...

NOISE
The NOISE instruction specifies words to be ignored when the student uses them in a response.

+ NOISE string,string,...
-

NOWORD
The NOWORD instruction specifies the word displayed when the instruction MARKUP is used as a response-contingent instruction for an answer judged wrong.

NOWORD string

OKWORD
The OKWORD instruction specifies the word displayed when the MARKUP instruction is used as a response-contingent instruction with an answer judged OK.

OKWORD string
QUERY
The main function of the QUERY instruction is to mark the beginning of a response-judging block. The QUERY instruction also displays the prompt character and pauses until a response has been typed on the keyboard. The end of the response is indicated by the system delimit character.
QUERY screen_address RET

RIGHT
The RIGHT instruction specifies anticipated right answers to a QUERY. It can indicate the beginning of a series of response-contingent instructions that are executed only if the student's response matches a right answer.
RIGHT string1 string1... RET

RIGHTV
The RIGHTV instruction is a version of the RIGHT instruction that specifies right answers as variables or expressions. It is used when the student's response is an expression to be evaluated; when the answer has been found by evaluating expressions, as is often the case in mathematics; when files are used to store sets of questions and answers; and in other cases where the right answer is more easily specified as a variable than as a constant.
RIGHTV expression[,tolerance][,unit] RET

SCORE
The SCORE instruction enables and disables scoring. It can be used at any point in the lesson or in a unit. Scoring is on by default.
SCORE boolean expression RET

SEED
The SEED instruction generates the seed value passed to the random number generator. Without the SEED instruction, the system generates the same sequence of numbers each time.
SEED [expression] RET
SPECS
The SPECS instruction modifies the requirements for judging a student's response right or wrong by changing the default specifications for exactness of spelling, punctuation, and capitalization; for allowing extra words, the same words in different order, or student variables in expressions; or for requiring units of measurement in the response.

SPECS keyword RET
ANYORDER/NOANYORDER
CAPS/NOCAPS
EXACT/NOEXACT
EXP/NOEXP
EXTRA/NOEXTRA
PUNC/NOPUNC
UNITS/NOUNITS

SYN
The SYN instruction specifies synonyms for use in response judging by identifying a target word and a list of synonyms. When a target word is specified in the argument to a RIGHT or WRONG instruction, all of the synonyms in the list are considered equivalent in judging the student's response.

+ SYN target__word,synonym,synonym,... RET
-

TEST
The TEST instruction serves two purposes: it marks the beginning of a test structure that ends with the instruction ENDTTEST and it specifies the name of the variable that is the basis for the test.

TEST variable RET
VALUE
.
.
.
ENDTEST
**WEIGHT**
The WEIGHT instruction defines the value added to the student’s score when the student’s response to a QUERY is judged. The score is added to the SCORE system variable and to the element of the SCORES system variable indexed by the current goal.

```
WEIGHT number
```

**WHEN**
The WHEN instruction specifies that characters input from the keyboard, elapsed time, or absolute time can interrupt the current unit and cause execution of another unit.

```
WHEN keyword,condition,unitname
STRING
ELAPSED
TIME
INTERVAL
```

**WRONG**
The WRONG instruction specifies one or more anticipated wrong answers to a QUERY. The WRONG instruction can indicate the beginning of a series of response-contingent instructions that are executed only if the student’s response matches the specified answer or answers.

```
WRONG string1|string1|...
```

**WRONGV**
The WRONGV instruction is a version of the WRONG instruction that specifies wrong answers as variables or expressions. The WRONGV instruction is used when the student’s response is an expression to be evaluated; when the answer has been found by evaluating expressions, as is often the case in mathematics; when files are used to store sets of questions and answers; and in other cases where the wrong answer is more easily specified as a variable than as a constant.

```
WRONGV expression[,tolerance][,unit]
```
Plotting Instructions

AXES
The AXES instruction defines the screen location of the maximum boundaries of the axes of a graph. Its arguments specify the number of screen dots from the origin in each of the four possible axis directions.

AXES + and − x__distance, + and − y__distance

The AXES instruction is one of four instructions that determine the screen location, marking, and scale of the x and y axes of the current graph. The GORIGIN instruction defines the screen address of the origin. The AXES instruction defines the screen addresses of the maximum boundaries of the axes. The SCALEX and SCALEY instructions correlate distances on the defined axes with numeric values so data can be plotted. The MARKX and MARKY instructions define the markings and numeric labels on the axes, and draw the axes and markings on the screen.

DELTA
The DELTA instruction specifies a standard increment for the independent variable in a GRAPH instruction, thus graphing points across the screen at regular intervals.

DELTA increment

GRAPH
The GRAPH instruction repeatedly evaluates the expression used as its argument and draws a dot for each evaluation. The origin and scale of the graph are established by the related instructions GORIGIN, AXES, SCALEX and SCALEY, or LSCALEX and LSCALEY. The instruction DELTA defines the increment for the independent variable in the expression.

GRAPH x, f(x)

HBAR
The HBAR instruction draws one horizontal bar of a bar graph. The current graph origin, dimensions, and scale defined by the most recent GORIGIN, AXES, and SCALES and SCALEY or LSCALEX and LSCALEY instructions determine the placement of the bar on the screen.

HBAR x__endpoint, y__endpoint: bar__width[, shade__character]
LScaleX, LScaleY
The LScaleX and LScaleY instructions specify logarithmic scaling factors used to map graphs to the screen. The arguments for these instructions are identical. With GORIGIN and AXES, LScaleX and LScaleY provide the basis for defining the graphing coordinate system with logarithmic scales.

LScaleX x_min, x_max
LScaleY y_min, y_max

MARKX, MARKY
The MARKX instruction draws the x-axis of a graph with tic marks and numeric labels. The MARKY instruction draws the y-axis. The MARKX and MARKY instructions differ only in the graph axis they draw. The arguments for MARKX and MARKY are identical. The MARKX and MARKY instructions draw the axes of a graph at the screen locations defined by GORIGIN and AXES. The markings, both tic marks and display of numbers, are selected by the arguments to the MARKX and MARKY instructions. The actual numbers depend on the ScaleX and ScaleY (or LScaleX and LScaleY) instructions which define the scale of the axes.

MARKX major_int, minor_int, display_char[, total_chars, dec_pl]

SCALEX/SCALEY
The SCALEX and SCALEY instructions specify the scaling factors used to map graphs to the screen. SCALEX defines the scaling factors for the x-axis. SCALEY defines the scaling factors for the y-axis. The syntax of the two instructions is the same. With GORIGIN and AXES, SCALEX and SCALEY provide the basis for defining the graphing coordinate system.

SCALEX x_min, x_max

VBAR
The VBAR instruction draws one vertical bar of a bar graph. The current graph origin, dimensions, and scale defined by the most recent GORIGIN, AXES, and SCALEX and SCALEY instructions determine the placement of the bar on the screen.

VBAR x_end_point, y_end_point: bar_width[, shade_character]
### ReGIS Manipulations Instructions

- **MLOAD**
  
  The MLOAD instruction loads a macrograph into the GIGI terminal.
  
  MLOAD ident__char, filename  [RET]

- **MPLOT**
  
  The MPLOT instruction draws a macrograph loaded into the GIGI terminal by a previous MLOAD instruction.
  
  MPLOT ident__char  [RET]

### Text Graphics Instructions

- **CHAR**
  
  The CHAR instruction modifies one character in an alternate character set. The CHAR instruction specifies the dot pattern displayed on the screen.
  
  CHAR charset__name,"character",  [RET]

- **CHARSET**
  
  The CHARSET instruction specifies one of the four character sets as active. If an alternate character set is specified, the instruction also loads the alternate character set into the terminal from a file. The dot patterns in the active character set are used in writing text until another CHARSET instruction selects another character set. The standard ASCII character set is one of the four; the three alternate character sets are user defined.
  
  CHARSET charset__name  [RET]

- **ITALICS**
  
  The ITALICS instruction specifies that subsequent text is to be written in italics.
  
  ITALICS degree  [RET]

- **SIZE**
  
  The SIZE instruction alters the size of text.
  
  SIZE x__size, y__size  [RET]
  
  size  [RET]

- **TROTATE**
  
  The TROTATE instruction specifies that subsequent text is to be rotated a multiple of 45 degrees from horizontal.
  
  TROTATE degrees  [RET]
Display Control Instructions

AT
The AT instruction selects a screen address. The first character specified by the next WRITE instruction is displayed at this address. This address is also used as the current location for the M PLOT instruction.

AT screen_address

The AT instruction sets the current location for the cursor and establishes a left margin for subsequent displays. The left margin is used by default by the QUERY and INPUT instructions. WRITE instructions that write more than one line of text also use the left margin to display a block of text.

BCOLOR
The BCOLOR instruction specifies the background color of the screen. When a black-and-white monitor is used, the colors specify shades of gray from darkest to lightest.

BCOLOR 0 = DARK
1 = BLUE
2 = RED
3 = MAGENTA
4 = GREEN
5 = CYAN
6 = YELLOW
7 = WHITE

When the BCOLOR instruction is executed, the screen's background color changes to the color specified. The screen color remains the same until it is changed by another BCOLOR instruction. The BCOLOR instruction does not modify any text or graphics already displayed on the screen.

The current background color is stored in the system variable BCOLOR.

MODE
The MODE instruction specifies a mode of graphics display that affects the appearance of text and graphics on the screen as well as the way display information is stored in the GIGI bit-map memory.

MODE NORMAL/INVERSE
FIXED/BLINK
OVERLAY
REPLACE
COMPLEMENT
ERASE
PATTERN
The PATTERN instruction selects a pattern for line drawings and for shading the area specified by HREF.
PATTERN%r_expression or keyword

PAUSE
The PAUSE instruction suspends execution of a lesson for a specified amount of time, until a certain time is reached, or until a specified response is entered.
PAUSE keyword,expression
  STRING
  ELAPSED
  TIME

SLIDE
The SLIDE instruction displays a filename.PIC file containing a slide. The PIC file can be:
- Any .PIC file from the current tray
- The next sequential .PIC file in the current tray
- Any .PIC file without reference to a tray
SLIDE
  SLIDE filename.PIC
  SLIDE slide__number

TRAY
The TRAY instruction loads a tray file created by the GIGI Slide Projection System and makes it available for subsequent SLIDE instructions.
TRAY trayfile

WRITC
The WRITC instruction is a variation of the WRITE instruction. The WRITC instruction inserts a carriage-return/line-feed before the beginning of the text. When the WRITC is used, the first character of the specified text is displayed at the left margin established by the most recent AT, GAT, or RAT instruction and one line below the last text displayed.
WRITC text

WRITE
The WRITE instruction specifies the text and variables whose contents are to be displayed on the screen.
WRITE text
DAL Built-in Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td>returns the absolute value of its argument, which can be a scalar or an array.</td>
</tr>
<tr>
<td>ALT(s,s,...)</td>
<td>returns a pattern which is a set of alternative strings. For example, ALT (&quot;a&quot;, &quot;b&quot;, &quot;c&quot;) returns a pattern which matches any of the characters &quot;a,&quot; &quot;b,&quot; or &quot;c.&quot; Patterns returned by the ALT function can be used as arguments to other string functions or can be compared to string variables or string constants in Boolean expressions. Variables can be arguments to this function.</td>
</tr>
<tr>
<td>ANTILOG(x)</td>
<td>returns the antilogarithm of its argument.</td>
</tr>
<tr>
<td>ARCOS(x)</td>
<td>returns the angle whose cosine is its argument. The angle is returned as a decimal in radians.</td>
</tr>
<tr>
<td>ARCSIN(x)</td>
<td>returns the angle whose sine is its argument. The angle is returned as a decimal in radians.</td>
</tr>
<tr>
<td>ARCTAN(x)</td>
<td>returns the angle whose tangent is its argument. The angle is returned as a decimal in radians.</td>
</tr>
<tr>
<td>ARCTAND(x,y)</td>
<td>returns the angle whose tangent is x,y. If y evaluates to 0, pi/2 is returned in radians.</td>
</tr>
<tr>
<td>ASCII(x,i)</td>
<td>requires a string constant or variable in x and returns the ASCII value of the ith character in the string. The second argument defaults to 1 if it is omitted.</td>
</tr>
<tr>
<td>CHAR(x)</td>
<td>returns the character whose ASCII value is x.</td>
</tr>
<tr>
<td>CHARSETLD(x)</td>
<td>returns the Boolean value 1 if the character set x has been loaded during this execution of the lesson. The argument x must be identical to the argument to the CHARSET instruction.</td>
</tr>
<tr>
<td>COS(x)</td>
<td>returns the cosine of its argument. The argument must be in radians.</td>
</tr>
<tr>
<td>DEG(x)</td>
<td>returns the value of its argument in degrees. The argument must be in radians.</td>
</tr>
<tr>
<td>DELETE(x,y)</td>
<td>returns a subset of string x with y deleted. The string y can be a pattern-valued expression. Only the first instance of y is deleted even if y appears more than once. If the substring y is not in the string x, x is returned unchanged.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DET(array)</td>
<td>returns the determinant of the square array given as its argument.</td>
</tr>
<tr>
<td>DIMS(array)</td>
<td>returns the number of dimensions of the array named.</td>
</tr>
<tr>
<td>DIMAX(array,i)</td>
<td>returns the upper bound of the ith dimension of the array named.</td>
</tr>
<tr>
<td>EOF(channel no)</td>
<td>returns the Boolean value 1 if the file associated with the channel is at its end-of-file mark. The EOF function returns the Boolean value 0 if the file is not at its end-of-file mark. This function is used with the READ instruction. This function sets the system variable IORESULT.</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>returns a value equal to e raised to the power of the argument.</td>
</tr>
<tr>
<td>FIND(channel no, record no)</td>
<td>attempts to position the file associated with the channel number at the specified record number, and returns the Boolean value 1 if the attempt is successful. If the record number is lower than the record number at the current position, REWIND occurs automatically. This function sets the system variable IORESULT.</td>
</tr>
<tr>
<td>IDEN(x)</td>
<td>returns an identity array of order x which can be used directly in expressions, or can be assigned to a variable.</td>
</tr>
<tr>
<td>INSTRING(x,y)</td>
<td>returns a number corresponding to the character position of string y in string x. If string y contains more than one character, the position of the first character is returned. If y is not contained in x, 0 is returned.</td>
</tr>
<tr>
<td>INT(x)</td>
<td>returns the truncated integer portion of its real number argument. The sign is preserved.</td>
</tr>
<tr>
<td>INV(array)</td>
<td>returns the inverse of its argument which must be a square array.</td>
</tr>
<tr>
<td>IS</td>
<td>The metaoperator IS changes the three conversion functions NUMBER, INT, and INV to interrogative functions. ISNUMBER(x) returns 1 if x is either a real number or an integer, and 0 otherwise. ISINT(x) returns 1 if x is an integer. ISINV(x) returns 1 if x has a nonzero determinant.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LEN(x)</td>
<td>returns the length (number of characters) of its string-valued argument as an integer.</td>
</tr>
<tr>
<td>LN(x)</td>
<td>returns the logarithm (base e) of its argument.</td>
</tr>
<tr>
<td>LOG(x)</td>
<td>returns the logarithm (base 10) of its argument.</td>
</tr>
<tr>
<td>LOWER(x)</td>
<td>returns a string with all uppercase letters in its string-valued argument converted to lowercase (digits and symbols are not modified).</td>
</tr>
<tr>
<td>NUMBER(x)</td>
<td>returns the real number represented by its string-valued argument.</td>
</tr>
<tr>
<td>PICFILE(filename)</td>
<td>opens the specified file and sends the ReGIS instructions for all subsequent screen displays to this file as well as to the screen. The file is closed by the function with no argument. The file extension can be included; the default is .PIC. The PICFILE function returns a Boolean 1 if the file was successfully opened or closed.</td>
</tr>
<tr>
<td>RAD(x)</td>
<td>returns the value of its argument in radians. The argument is assumed to be in degrees and is evaluated mod 360.</td>
</tr>
<tr>
<td>RANDOMN(m,sd)</td>
<td>returns a randomly selected number given a mean and a standard deviation of a normal distribution. Default values are 0 and 1.</td>
</tr>
<tr>
<td>RANDOMP(lambda)</td>
<td>returns a randomly selected number between 0 and 1 selected from a Poisson distribution of intensity lambda. The default is 1.</td>
</tr>
<tr>
<td>RANDOMU(x,y)</td>
<td>returns a randomly selected number between x and y drawn from a uniform distribution. Default values are 0 and 1. The value of x is included in the distribution. The value of y is not.</td>
</tr>
<tr>
<td>REAL(x)</td>
<td>returns a real number equivalent to its integer argument.</td>
</tr>
<tr>
<td>REPLACE(x,y,z)</td>
<td>returns string x with the first instance of string y replaced by string z. Replacement occurs from left to right. If string y does not exist in string x, string x is returned unchanged.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>REWIND(channel no)</td>
<td>resets the file associated with the specified channel number to its first record. Returns 1 if the rewind is successful; returns 0 if it is not. This function sets the system variable IORESULT.</td>
</tr>
<tr>
<td>ROUND(x)</td>
<td>returns an integer which is equivalent to the rounded real number argument. The sign is preserved.</td>
</tr>
<tr>
<td>SIGN(x)</td>
<td>returns +1 if the value of x is a positive number and -1 if the value of x is a negative number. If the value of x is 0, 0 is returned. The argument can be a real number, an integer, or an array.</td>
</tr>
<tr>
<td>SIN(x)</td>
<td>returns the sine of its argument. The argument must be in radians.</td>
</tr>
<tr>
<td>SQRT(x)</td>
<td>returns the square root of its argument. The argument must be positive, and cannot be a table or an array.</td>
</tr>
<tr>
<td>STRING(x)</td>
<td>returns a character string equivalent to its real or integer argument.</td>
</tr>
<tr>
<td>SUBSTR(string, y, z)</td>
<td>returns the substring of string which begins at character position y and is z characters long. Returns a null string if string does not contain at least z characters starting at position y.</td>
</tr>
<tr>
<td>TAN(x)</td>
<td>returns the tangent of its argument. The argument must be in radians.</td>
</tr>
<tr>
<td>TRAN(array)</td>
<td>returns the transpose of the array specified as its argument.</td>
</tr>
<tr>
<td>UPPER(x)</td>
<td>returns a string with all lowercase letters in its string-valued argument converted to uppercase. Digits and symbols are not changed.</td>
</tr>
<tr>
<td>WORD(n, string)</td>
<td>returns the nth word in the specified string variable. If n evaluates to less than 1 or to a number greater than the number of words in the string, a null string is returned. Words are terminated by a space, a RET, a TAB, or any punctuation mark except apostrophe, hyphen, dollar sign, and underscore. Other punctuation marks are ignored, and not returned as part of the word.</td>
</tr>
</tbody>
</table>
DAL DATA TYPES

The following is a list data types which DAL uses.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Containing a whole number. Integer Values range from $+2^{32} -1$ to $-2^{32} -1$. A number with no decimal point is considered to be an integer constant.</td>
</tr>
<tr>
<td>REAL</td>
<td>Containing a number that can have both a whole and a fractional part. Positive real numbers range from $+2^{127} -1$ to $+2^{(-127)} -1$. Negative real numbers range from $-2^{127}$ to $-2^{(-127)}$. Fractional values are displayed with up to six digits. A number with a decimal point is considered to be a real constant.</td>
</tr>
<tr>
<td>STRING</td>
<td>Containing a character string. String constants are enclosed in double quotation marks.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Having two possible values, true and false. The keywords TRUE and FALSE can be used to assign values to Boolean variables or to test Boolean expressions. TRUE is equal to 1, and FALSE is equal to 0.</td>
</tr>
</tbody>
</table>

USAGE CHARACTERISTICS OF VARIABLES ASSIGNED VIA DEFINE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMANENT</td>
<td>The current value of permanent variables is stored in a file when a lesson is stopped, and is available the next time the lesson is executed by any DAL user. Each lesson has one set of permanent variables which are referenced during concurrent or consecutive executions. Permanent variables can be of any data type. Permanent string variables can contain any number of characters during execution of a lesson. Only the first 80 characters are stored when the lesson ends. Permanent variables must be defined at lesson level. Arrays and tables cannot be defined as permanent variables.</td>
</tr>
<tr>
<td>Usage</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RESTART</td>
<td>The current value of a restart variable is stored in a file when a student stops a lesson at a STOP instruction. When the student who stopped a lesson restarts it, the values of restart variables are restored from the file. Each student who executes the lesson has a separate copy of all restart variables. Some system variables are restart variables. Restart variables can be of any data type. Restart string variables can contain any number of characters during execution of the lesson. Only the first 80 characters are stored when the lesson ends. Restart variables must be defined at lesson level. Arrays and tables cannot be defined as restart variables. During execution of a lesson, restart variables have no special characteristics.</td>
</tr>
<tr>
<td>STUDENT</td>
<td>The name of a student variable can be used by students in responses. The current value of a student variable is evaluated when students enter the variable name. Student variables must be defined at lesson level.</td>
</tr>
<tr>
<td>TABLE</td>
<td>A table is a two-dimensional array. One dimension contains elements of any one data type. The second dimension contains strings used a subscripts. The subscripts of tables are enclosed in square brackets (table.__name{substring}). Tables are arbitrarily sized and grow dynamically as needed.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>An array is an n-dimensional array containing elements of any one data type. Arrays are sized when they are defined. The size values are enclosed in square brackets immediately following the array name, and subsequent references to the array require square brackets enclosing the subscripts (array.__name[1,1]).</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>A variable defined with the FUNCTION usage characteristic is the name of a user-defined function whose returned value is of a data type specified.</td>
</tr>
</tbody>
</table>
## DAL Operators

The following is a list of DAL operators, what they mean, the data type each operates on, and the data type of results of operations.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Data Type Operated On</th>
<th>Data Type of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>integer, real</td>
<td>same</td>
</tr>
<tr>
<td>+</td>
<td>concatenation</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>-</td>
<td>negative</td>
<td>integer, real</td>
<td>same</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>integer, real</td>
<td>same</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>integer, real</td>
<td>same</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>integer, real</td>
<td>same</td>
</tr>
<tr>
<td>^</td>
<td>exponent</td>
<td>integer, real, Boolean</td>
<td>string</td>
</tr>
<tr>
<td>=</td>
<td>equal</td>
<td>integer, real, Boolean</td>
<td>string</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>not equal</td>
<td>integer, real, Boolean</td>
<td>string</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>integer, real, Boolean</td>
<td>string</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or</td>
<td>integer, real, equal to</td>
<td>Boolean</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>integer, real, Boolean</td>
<td>string</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or</td>
<td>integer, real, equal to</td>
<td>Boolean</td>
</tr>
<tr>
<td>AND</td>
<td>logical AND</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>OR</td>
<td>logical OR</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>XOR</td>
<td>logical XOR</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>IMP</td>
<td>logical IMP</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>NOT</td>
<td>logical inverse</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>MOD</td>
<td>Modulo</td>
<td>integer</td>
<td>integer</td>
</tr>
</tbody>
</table>
## RESPONSE-RELATED SYSTEM VARIABLES

Response-related system variables contain information about student responses and response judging. Some are restart variables and are stored in a restart variable file if a lesson is stopped with a STOP instruction.

### Response-Related System Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Modified By</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSCNT</td>
<td>integer</td>
<td>Sequential number of RIGHT, RIGHTV, WRONG, or WRONGV matched by most recent response.</td>
<td>Cleared by QUERY. Set by first match and not reset until QUERY is encountered again. Set to 0 if no match.</td>
</tr>
<tr>
<td>GOAL</td>
<td>integer restart</td>
<td>Number of current goal.</td>
<td>GOAL instruction</td>
</tr>
<tr>
<td>LATENCY</td>
<td>integer seconds</td>
<td>Time from QUERY or INPUT until delimiter is received.</td>
<td>Updated after every response and available until prompt is displayed again.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>integer</td>
<td>Length in characters of most recent response.</td>
<td>Updated at end of each response.</td>
</tr>
<tr>
<td>NNO restart</td>
<td>integer</td>
<td>Number of responses judged NO since beginning of lesson. Unless default response judging is overridden, this variable is 0.</td>
<td>Incremented when unit containing QUERY returns with answer judged NO.</td>
</tr>
<tr>
<td>NOK restart</td>
<td>integer</td>
<td>Number of responses judged OK since beginning of lesson.</td>
<td>Incremented by a right answer. Updated when unit containing QUERY returns.</td>
</tr>
<tr>
<td>NOKFIRST</td>
<td>integer</td>
<td>Number of first responses judged OK since beginning of lesson.</td>
<td>Incremented by a right answer on the first attempt. Updated when unit containing QUERY returns.</td>
</tr>
</tbody>
</table>

---

117
<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Modified By</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMTRIES</td>
<td>integer</td>
<td>Number of times the student has responded to this QUERY.</td>
<td>Incremented at each response. Cleared at end of unit.</td>
</tr>
<tr>
<td>QELAPSED</td>
<td>integer</td>
<td>Time that query is available. 0 indicates no time limit. QELAPSED counts down as time elapses. When it reaches 0, the variable TIMEOUT is set.</td>
<td>Set to 0 at beginning of lesson. Changed only if new value assigned.</td>
</tr>
<tr>
<td>QLENGTH</td>
<td>integer</td>
<td>Number of characters allowed in response. If this number is reached, response is terminated and judging begins.</td>
<td>Set to 0 (no limit) at beginning of lesson. Reset only if new value assigned.</td>
</tr>
<tr>
<td>QUERIES</td>
<td>integer</td>
<td>Number of QUERY instructions since beginning of lesson. QUERIES is not incremented when prompt is redisplayed after a wrong response. It is incremented if any instruction (REDO, LOOP, FOR, BRANCH) returns control to a point preceding the QUERY.</td>
<td>QUERY</td>
</tr>
<tr>
<td>Name</td>
<td>Data Type</td>
<td>Description</td>
<td>Modified By</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>string maximum</td>
<td>Exact response.</td>
<td>New value assigned at QUERY or INPUT New response is terminated by DELIMIT character or value of QLENGTH. Characters entered later are read by next INPUT or QUERY.</td>
</tr>
<tr>
<td></td>
<td>500 chars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESPONSEV</td>
<td>real integer</td>
<td>String in RESPONSE evaluated as expression of some data type as arguments to</td>
<td>Evaluated when RIGHTV or WRONGV is executed.</td>
</tr>
<tr>
<td></td>
<td>Boolean</td>
<td>RESPONSE or evaluated as expression of some data type as arguments to RIGHTV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>or WRONGV.</td>
<td></td>
</tr>
<tr>
<td>SATISFIED</td>
<td>Boolean</td>
<td>TRUE if the query is judged OK; FALSE if the query is judged NO. SATISFIED</td>
<td>Updated when unit containing QUERY returns.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is TRUE unless default response judging is overridden.</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>string 20 chars,</td>
<td>User's real name entered at time of registration in group.</td>
<td>Read from DIMENSION files. When lessons are executed from VAX/VMS command level, this variable is null.</td>
</tr>
<tr>
<td></td>
<td>uppercase, left</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>justified, blank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>string HH:</td>
<td>Current time according to machine clock.</td>
<td>VAX/VMS system</td>
</tr>
<tr>
<td></td>
<td>MM:SS.HH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Data Type</td>
<td>Description</td>
<td>Modified By</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>IORESUL</td>
<td>integer</td>
<td>A numeric code which reports the status of the most recent IO request.</td>
<td>CLOSE instruction, GET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = normal return</td>
<td>instruction, OPEN instruction,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = invalid channel number</td>
<td>PUT instruction, TRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = insufficient virtual memory</td>
<td>instruction, SLIDE instruction,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = channel not open</td>
<td>CHARSET instruction, MLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 = file is read only</td>
<td>instruction, REWIND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = file is sequential only</td>
<td>function, FIND function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 = file is write only</td>
<td>EOF function, PICFILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 = new file was created with OPEN instruction</td>
<td>function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 = tray not opened (SLIDE expects previously opened tray file) a 5- or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-digit VAX/VMS error code (see note at end of table).</td>
<td></td>
</tr>
</tbody>
</table>
## GRAPHICS AND GRAPHING SYSTEM VARIABLES

All graphics and graphing variables which define the current cursor position in the different coordinate systems are updated by all instructions which display text or graphics on the screen.

### Graphics and Graphing System Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Modified By</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCOLOR</td>
<td>integer 0 to 7</td>
<td>Value of current background color.</td>
<td>BCOLOR instruction</td>
</tr>
<tr>
<td>CHARSET</td>
<td>string character set</td>
<td>Name of last instruction specified.</td>
<td>CHARSET</td>
</tr>
<tr>
<td>FCOLOR</td>
<td>integer color.</td>
<td>Value of current foreground</td>
<td>FCOLOR instruction</td>
</tr>
<tr>
<td>GORIGINX</td>
<td>real</td>
<td>Value of x-coordinate last specified.</td>
<td>GORIGIN instruction</td>
</tr>
<tr>
<td>GORIGINY</td>
<td>real</td>
<td>Value of y-coordinate last specified.</td>
<td>GORIGIN instruction</td>
</tr>
<tr>
<td>GWHEREX</td>
<td>real</td>
<td>X-coordinate of the last point displayed expressed in current graph scale.</td>
<td>Updated by all graphics instruc-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If text was displayed last, this point is the upper left corner of the char-</td>
<td>tions. GORIGIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>acter cell.</td>
<td>resets. SCALE resets.</td>
</tr>
<tr>
<td>GWHEREY</td>
<td>real</td>
<td>Y-coordinate of the last point displayed expressed in current graph scale.</td>
<td>Updated by all graphics instruc-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If text was displayed last, this point is the is the upper left corner of</td>
<td>tions. GORIGIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the character cell.</td>
<td>resets. SCALE resets.</td>
</tr>
<tr>
<td>ITALICS</td>
<td>integer</td>
<td>Value last specified.</td>
<td>ITALICS instruction</td>
</tr>
<tr>
<td>RORIGINX</td>
<td>integer fine co-</td>
<td>X-coordinated of point last specified.</td>
<td>RORIGINX instruction</td>
</tr>
<tr>
<td></td>
<td>ordinates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Data Type</td>
<td>Description</td>
<td>Modified By</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>RORIGINY</td>
<td>integer</td>
<td>Y-coordinate of point last specified.</td>
<td>RORIGIN instruction</td>
</tr>
<tr>
<td></td>
<td>fine coordinates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSIZEX</td>
<td>integer</td>
<td>X size coefficient.</td>
<td>RSIZE instruction</td>
</tr>
<tr>
<td>RISZEN</td>
<td>integer</td>
<td>Y size coefficient.</td>
<td>RSIZE instruction</td>
</tr>
<tr>
<td>RWHEREX</td>
<td>integer</td>
<td>X-coordinate of point last drawn relative to RORIGIN. If text was drawn last, this point is the upper left corner of the character cell</td>
<td>Updated by all graphics instructions. Reset by RORIGIN.</td>
</tr>
<tr>
<td></td>
<td>fine coordinates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RWHEREX</td>
<td>integer</td>
<td>Y-coordinate of point last drawn relative to RORIGIN. If text was drawn last, this point is the upper left corner of the text cell.</td>
<td>Updated by all graphics instructions. Reset by RORIGIN.</td>
</tr>
<tr>
<td></td>
<td>fine coordinates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZEN</td>
<td>integer</td>
<td>X size last specified or size when SIZE has only one argument.</td>
<td>SIZE instruction</td>
</tr>
<tr>
<td>SIZEN</td>
<td>integer</td>
<td>Y size last specified. This variable is not changed when the SIZE instruction has only one argument.</td>
<td>SIZE instruction</td>
</tr>
<tr>
<td>Name</td>
<td>Data Type</td>
<td>Description</td>
<td>Modified By</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>WHERE</td>
<td>integer row and column coordinates</td>
<td>Row and column address of last point displayed. After text is displayed, this point is one point to the right of the upper right corner of the character cell. After line graphics are displayed, the variable contains the address of the upper left corner of the text cell in which the last dot was drawn.</td>
<td>Updated by all graphics instructions. Reset to 0 by ERASE with no arguments.</td>
</tr>
<tr>
<td>WHEREX</td>
<td>integer fine coordinates</td>
<td>X-coordinate of last point displayed. After text is displayed, this point is one point to the right of the upper right corner of the character cell.</td>
<td>Updated by all graphics instructions. Reset to 0 by ERASE with no arguments.</td>
</tr>
<tr>
<td>WHEREY</td>
<td>integer fine coordinates</td>
<td>Y-coordinate of last point point displayed. After text is displayed, this point is one point to the right of the upper right corner of the character cell.</td>
<td>Updated by all graphics instructions. Reset to 0 by ERASE with no arguments.</td>
</tr>
</tbody>
</table>
Appendix A
C.A.S. Menus

This appendix contains menus available to most types of users of DIGITAL's Courseware Authoring System. All of the menus available are not included; these offer a representative sample.

---

**Student Menu**

What would you like to do?

- **ASSIGNMENTS**
  - Do assignments
  - Browse through lessons

- **MAIL**
  - Run the system mail program

- **EXIT**
  - Exit system

PF2 = Help, PF4 = Exit

Use & & , then RETURN

---

**Assignment List**

What would you like to do?

- **ANALOGIES**
  - WORD RELATIONSHIPS
  - Due date: 27-MARCH-82

- **COMPLEX**
  - THE COMPLEX SENTENCE
  - Due date: 5-APRIL-82

- **GRAMMAR**
  - GRAMMAR AND USAGE
  - Due date: 7-MARCH-82

- **MODIFIER**
  - MISPLACED MODIFIERS
  - Due date: 30-MARCH-82

- **PARTS**
  - PARTS OF SPEECH
  - Due date: 15-MARCH-82

- **PREFIX**
  - WORD PREFIXES
  - Due date: 12-MARCH-82

- **PRONOUN**
  - STUDY OF PRONOUNS
  - Due date: 12-MARCH-82

- **SIMPLE**
  - THE SIMPLE SENTENCE

PF2 = Help, PF4 = Exit

Use & & , then RETURN

---

**Student Assignment Menu**
Student Browse Menu

Author Menu
Author Browse Menu

What would you like to do?

- Analogies
- Complex
- Grammar
- Modifier
- Parts
- Phonics
- Prefix
- Pronoun
- Sentence
- Simple
- Spell
- Suffix
- Usage
- Verb

PF2 = Help, PF4 = Exit

Use F0, then RETURN

Author Reports Menu

What would you like to do?

- 02: Available Lesson Reports
- 03: Lessons Published by Author (screen output only)
- 04: Responses for a Unit in a Lesson (screen output only)
- 06: Unit Analysis of Unit within Lesson within Group
- 09: Formatted dump of .DLG

EXIT = Exit

PF2 = Help, PF4 = Exit

Use F0, then RETURN
Appendix A

**Group Instructor Menu**

What would you like to do?

- Group Update
- Reports menu
- Assignment edit
- Browse in available lessons
- Run the system mail program
- Exit system

PF2 = Help, PFA = Exit

Use 0 0, then RETURN

---

**Group Edit Menu**

What would you like to do?

- Edit group information
- Change user information
- Delete user
- List of users
- Register a student
- Register a group instructor
- Register an author
- Exit

PF2 = Help, PFA = Exit

Use 0 0, then RETURN

---

Group Edit Menu
COURSEWARE

AUTHORING SYSTEM

Reports Menu

What would you like to do?

01 Student Status Report
02 Available Lesson Reports
05 Histogram of Scores for a Lesson (screen output only)
06 Unit Analysis of Unit within Lesson within Group
07 Lessons Assigned to Group
08 Student Scores for a given Lesson
10 List of users in group
11 Dump of the information (.INF) file

EXIT Exit

PF2 = Help, PF4 = Exit Use 0 0 , then RETURN

MR-S-2057-82

Group Reports Menu

COURSEWARE

AUTHORING SYSTEM

Group Assignment Menu

What would you like to do?

ADD Add new assignments
LIST List of current assignments
PUBLISHED List of published lessons
DELETE Delete an assignment

EXIT Exit menu

PF2 = Help, PF4 = Exit Use 0 0 , then RETURN

MR-S-2056-82

Group Assignment Menu
What would you like to do?

- WORD RELATIONSHIP
- THE COMPLEX SENTENCE
- GRAMMAR AND USAGE
- MISPLACED MODIFIERS
- PARTS OF SPEECH
- INTRO TO PHONICS
- WORD PREFIXES
- STUDY OF PRONOUNS
- SENTENCE COMPLETION
- THE SIMPLE SENTENCE
- SPELLING DRILL
- WORD SUFFIXES
- GLOSSARY OF USAGE
- STUDY OF VERBS

PF2 = Help, PF4 = Exit

Use , then RETURN

Group Browse Menu
Appendix B
C.A.S. Reports

This appendix contains reports available to most types of users of DIGITAL's Courseware Authoring System. All of the reports available are not included; these offer a representative sample.

Student Status Report

Available Lessons
Histogram of Scores for a Lesson
### Unit Analysis of Unit Within Lesson Within Group

<table>
<thead>
<tr>
<th>STUDENT NAME</th>
<th># TRIES</th>
<th>RESPONSE TIME</th>
<th>AVG RESPONSE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>JULIE</td>
<td>8</td>
<td>1:24</td>
<td>0:11</td>
</tr>
<tr>
<td>MIKE</td>
<td>15</td>
<td>0:41</td>
<td>0:03</td>
</tr>
</tbody>
</table>

**Overall Average**

<table>
<thead>
<tr>
<th>NUMBER OF TRIES</th>
<th>TOTAL RESPONSE TIME</th>
<th>AVERAGE RESPONSE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>01:02</td>
<td>00:05</td>
</tr>
</tbody>
</table>

### Lessons Assigned to Group

<table>
<thead>
<tr>
<th>LESSON NAME</th>
<th>LESSON DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTS</td>
<td>PARTS OF SPEECH</td>
</tr>
<tr>
<td>GRAMMAR</td>
<td>GRAMMAR AND USAGE</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>THE SIMPLE SENTENCE</td>
</tr>
<tr>
<td>PRONOUN</td>
<td>STUDY OF PRONUNES</td>
</tr>
<tr>
<td>VERB</td>
<td>STUDY OF VERBS</td>
</tr>
<tr>
<td>PREFIX</td>
<td>WORD PREFIXES</td>
</tr>
<tr>
<td>SUFFIX</td>
<td>WORD SUFFIXES</td>
</tr>
<tr>
<td>ANALOGIES</td>
<td>WORD RELATIONSHIPS</td>
</tr>
<tr>
<td>MODIFIER</td>
<td>MISPLACED MODIFIERS</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>THE COMPLEX SENTENCE</td>
</tr>
<tr>
<td>SENTENCE</td>
<td>SENTENCE COMPLETION</td>
</tr>
<tr>
<td>SPELL</td>
<td>SPELLING DRILL</td>
</tr>
</tbody>
</table>

* * The lessons for this group must be taken in the sequence listed.
List of Users in a Group

Lessons Published by Author
Responses for a Unit in a Lesson
### Student Scores for a Given Lesson

**RPT08**  
**DATE:** 4/02/92  
**GROUP:** ENGI  
**TIME:** 00:34:32

<table>
<thead>
<tr>
<th>DIMENSION</th>
<th>NAME</th>
<th>STUDENT NAME</th>
<th>RIGHT/ASKED</th>
<th>% SCORE</th>
<th>RANK</th>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BARBARA</td>
<td>BARBARA WHITE</td>
<td>25/28</td>
<td>93.37%</td>
<td>1</td>
<td>00:02:47</td>
</tr>
<tr>
<td></td>
<td>BARRY</td>
<td>BARRY BROWN</td>
<td>25/29</td>
<td>92.46%</td>
<td>1</td>
<td>00:02:48</td>
</tr>
<tr>
<td></td>
<td>CAROLYN</td>
<td>CAROLYN LYNN BARRON</td>
<td>25/28</td>
<td>89.29%</td>
<td>2</td>
<td>00:02:10</td>
</tr>
<tr>
<td></td>
<td>DAVE</td>
<td>DAVE GREEN</td>
<td>15/23</td>
<td>53.57%</td>
<td>9</td>
<td>00:02:08</td>
</tr>
<tr>
<td></td>
<td>ELIZABETH</td>
<td>ELIZABETH</td>
<td>21/23</td>
<td>75.00%</td>
<td>7</td>
<td>00:01:40</td>
</tr>
<tr>
<td></td>
<td>GARY</td>
<td>GARY SWIFT</td>
<td>13/27</td>
<td>65.0%</td>
<td>9</td>
<td>00:01:31</td>
</tr>
<tr>
<td></td>
<td>JANE</td>
<td>JANE STONE</td>
<td>47/56</td>
<td>87.0%</td>
<td>3</td>
<td>00:03:03</td>
</tr>
<tr>
<td></td>
<td>JEFFREY</td>
<td>JEFFREY</td>
<td>3/23</td>
<td>24.00%</td>
<td>13</td>
<td>00:01:07</td>
</tr>
<tr>
<td></td>
<td>DOC</td>
<td>JIM SMITH</td>
<td>25/23</td>
<td>93.29%</td>
<td>3</td>
<td>00:01:57</td>
</tr>
<tr>
<td></td>
<td>JOHN</td>
<td>JOHN CORE</td>
<td>13/29</td>
<td>46.43%</td>
<td>10</td>
<td>00:01:06</td>
</tr>
<tr>
<td></td>
<td>JULIE</td>
<td>JULIE NIELSEN</td>
<td>25/33</td>
<td>70.57%</td>
<td>5</td>
<td>00:02:12</td>
</tr>
<tr>
<td></td>
<td>MARY</td>
<td>MARY POTTER</td>
<td>4/23</td>
<td>14.29%</td>
<td>14</td>
<td>00:01:09</td>
</tr>
<tr>
<td></td>
<td>MIKE</td>
<td>MIKE BROWN</td>
<td>24/23</td>
<td>95.71%</td>
<td>4</td>
<td>00:01:55</td>
</tr>
<tr>
<td></td>
<td>REGINALD</td>
<td>REGINALD GOOD</td>
<td>22/23</td>
<td>95.57%</td>
<td>6</td>
<td>00:01:24</td>
</tr>
<tr>
<td></td>
<td>RICH</td>
<td>RICH</td>
<td>22/23</td>
<td>95.57%</td>
<td>6</td>
<td>00:01:41</td>
</tr>
<tr>
<td></td>
<td>SUE</td>
<td>SUE BLACK</td>
<td>21/23</td>
<td>75.30%</td>
<td>7</td>
<td>00:01:19</td>
</tr>
<tr>
<td></td>
<td>TONY</td>
<td>TONY DICE</td>
<td>12/23</td>
<td>42.86%</td>
<td>11</td>
<td>00:01:15</td>
</tr>
</tbody>
</table>

**MEAN SCORES:** 65.42  
**TOTAL:** 00:34:32  
**AVERAGE:** 00:01:49

---

**RPT08**  
**DATE:** 4/02/92  
**GROUP:** ENGI  
**TIME:** 09:32:15

<table>
<thead>
<tr>
<th>SUMMARY SHEET</th>
<th>STUDENTS COMPLETING LESSON</th>
<th>MEAN SCORE</th>
<th>STANDARD DEVIATION</th>
<th>VARIANCE</th>
<th>TOTAL RESPONSE TIME</th>
<th>AVERAGE RESPONSE TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>019</td>
<td>65.42</td>
<td>105.99</td>
<td>214.38</td>
<td>00:34:32</td>
<td>00:01:49</td>
</tr>
</tbody>
</table>

---

**Student Scores for a Given Lesson**

---
Appendix B

Formatted Dump of .DLG File
# Glossary

## A

**adjunct CAI:** CAI used to support or augment instruction that is primarily delivered through another means.

**author:** one who designs and develops courseware.

**authoring language:** a programming language designed to simplify the development of CAI materials. Most often associated with tutorial style instruction.

## C

**CAI:** Computer-assisted Instruction or Computer-aided Instruction

**CAL:** Computer-assisted Learning

**cathode ray tube:** a television-type display screen used in most computer terminals today.

**CBE:** Computer-based Education

**CBI:** Computer-based Instruction

**CBT:** Computer-based Training

**computer literacy:** the general ability to understand and use computers to do useful work.

**Computer-aided Instruction:** see Computer-assisted Instruction

**Computer-assisted Instruction:** In general, the use of the computer as a teaching tool. Synonomous with Computer-based Instruction, Computer-assisted Learning, Computer-based Training, and Computer-aided Instruction.

**Computer-assisted Learning:** see Computer-assisted Instruction

**Computer-based Education:** the use of the computer in support of the educational process.

**Computer-based Instruction:** see Computer-assisted Instruction

**Computer-based Training:** see Computer-assisted Instruction

**Computer-Managed Instruction:** the use of the computer to manage the instructional process, includes maintaining records on student performance, controlling the availability and timing of instructional events, and providing progress reports to instructors and students.
courseware: computer programs developed specifically to teach educational content or to train people in particular skills.

cursor: a visible mark denoting a location on a CRT.

F

frame: a term from Programmed Instruction. Usually refers to one screen display, concept, or interaction.

G

graphics: non-textual screen displays including pictures, line drawings, and animation.

H

hardware: the physical equipment of a computer system, including processor, memory, input/output devices and and other components.

higher level language: English-like languages for programming computers. Computer systems must eventually translate higher level languages into machine language before the higher level program must be executed. Fortran, Basic, and DAL are all examples of higher level languages.

I

intelligent CAI: refers to courseware designed to alter its activities, presentation, and style based on interaction with the individual learner.

J

joy stick: a gearshift-like device used to move a cursor around a CRT.

L

learner control: an aspect of the instructional process concerned with a learner’s ability to affect the manner in which he is taught. Most often refers to the level of flexibility which a student has in manipulating the performance of courseware.

light pen: a device for touching the CRT screen. When the pen touches the screen, the location of the point of contact is transmitted to the computer.
M

machine language: a representation of the electronic impulses which actually control a computer. Machine language usually represents the presence and absence of electronic impulses using 1's and 0's.

mainframe: a relative term usually referring to a large centralized computer system often performing many different tasks simultaneously.

microcomputer: a complete computer system based upon a microprocessor.

microprocessor: generally refers to a single silicon “chip” which contains all the logic and control functions around which to build a microcomputer system.

minicomputer: a relative term referring to a “midsize” computer. Larger than a microcomputer, yet smaller than a mainframe.

P

paddles: knobs or levers (similar in function to the joy stick) which alter the location of the cursor on the CRT screen.

Primary CAI: CAI when it represents the main method of instruction (as opposed to “adjunct CAI”).

Programmed Instruction: a type of instruction based upon the work of psychologist B. F. Skinner in which instructional content is broken down into small concepts and ideas and presented in sequence to the student. Ideally, each “frame” is so simple and follows logically from the last so that most students can follow the instruction with little or no trouble.

S

software: computer programs. Languages or symbols used to control the operation of the system.

stand-alone: a complete computer system not attached to any other resource. Usually refers to small, single user computers.

T

terminal: a single work station. Usually consists of a CRT and typewriter-like keyboard, but often supported by additional devices.
Glossary

touch (or touch input): communicating with a computer by touching its CRT rather than by typing.

V

visual display terminal: a device used to display visual information to students. The most prevalent display device today is the CRT.

voice synthesis: the mechanical reproduction of speech. Voice synthesizers produce the phonemes of human speech through electronic means.