COMPUTER ASSISTED INSTRUCTION FOR THE "C" PROGRAMMING LANGUAGE ON THE ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Frank W. DeMarco
Captain, USAF

AFIT/GCS/MA/85D-2

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio
COMPUTER ASSISTED INSTRUCTION FOR THE
"C" PROGRAMMING LANGUAGE ON THE
ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Frank W. DeMarco
Captain, USAF

APIT/GCS/MA/85D-2

Approved for public release; distribution unlimited
COMPUTER ASSISTED INSTRUCTION FOR THE "C" PROGRAMMING LANGUAGE ON THE ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Presented to the Faculty of the School of Engineering of the Air Force Institute of Technology Air University
In Partial Fulfillment of the Requirements for the Degree of Master of Science

Frank W. DeMarco, B.S.
Captain, USAF

December 1985

Approved for public release; distribution unlimited
The purpose of this study was to develop a computer assisted instruction (CAI) program package for use on the Zenith Z-100 microcomputer system. The package is designed to give programming students introductory information on the "C" programming language. This programming package is to be used in the training programs managed by the Computer Assisted Instruction Plans Branch of the 3300 Technical Training Wing at Keesler AFB, Mississippi.

I would like to express my sincere thanks to my thesis advisor, Dr. Henry B. Potoczny, who gave me guidance and encouragement throughout my thesis effort. Thanks is also extended to Captain Patricia Lawlis, who as my thesis reader provided many constructive comments on improving this thesis. Grateful appreciation is also extended to the sponsor of my thesis, the CAI Plans Branch at Keesler AFB, and in particular, Captain Glen A. Miller and Technical Sergeant Charles T. Neal, who provided help in the verification and validation of the programs and course material I developed.

Finally, I want to express special gratitude to my wife Anne and my children Crystal and Bryan. They have forfeited countless hours of time with me in order that I could complete my graduate work here at AFIT. Their patience, understanding, and devoted love gave me the strength I needed to overcome the many obstacles I encountered. I owe them a debt that will take a lifetime to repay.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>Abstract</td>
<td>vi</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>Background</td>
<td>1-1</td>
</tr>
<tr>
<td>Statement of Problem</td>
<td>1-2</td>
</tr>
<tr>
<td>Scope</td>
<td>1-3</td>
</tr>
<tr>
<td>Assumptions</td>
<td>1-3</td>
</tr>
<tr>
<td>General Approach</td>
<td>1-4</td>
</tr>
<tr>
<td>II. Methodology</td>
<td>2-1</td>
</tr>
<tr>
<td>The Aim of CAI</td>
<td>2-1</td>
</tr>
<tr>
<td>Advantages of CAI</td>
<td>2-1</td>
</tr>
<tr>
<td>Disadvantages of CAI</td>
<td>2-3</td>
</tr>
<tr>
<td>Development Considerations</td>
<td>2-4</td>
</tr>
<tr>
<td>Course Development Approach</td>
<td>2-5</td>
</tr>
<tr>
<td>III. Design Specification</td>
<td>3-1</td>
</tr>
<tr>
<td>General Description</td>
<td>3-1</td>
</tr>
<tr>
<td>CAI Program</td>
<td>3-1</td>
</tr>
<tr>
<td>Status Program</td>
<td>3-2</td>
</tr>
<tr>
<td>Statistics Program</td>
<td>3-3</td>
</tr>
<tr>
<td>IV. System Implementation</td>
<td>4-1</td>
</tr>
<tr>
<td>General Description</td>
<td>4-1</td>
</tr>
<tr>
<td>&quot;C&quot; Lessons Descriptions</td>
<td>4-1</td>
</tr>
<tr>
<td>CAI Program</td>
<td>4-3</td>
</tr>
<tr>
<td>Status Program</td>
<td>4-13</td>
</tr>
<tr>
<td>Statistics Program</td>
<td>4-15</td>
</tr>
<tr>
<td>V. Conclusions and Recommendations</td>
<td>5-1</td>
</tr>
<tr>
<td>General Comments</td>
<td>5-1</td>
</tr>
<tr>
<td>Suggestions for Further Study</td>
<td>5-1</td>
</tr>
<tr>
<td>Appendix A: Users Guide</td>
<td>A-1</td>
</tr>
<tr>
<td>Using Program &quot;CAI&quot;</td>
<td>A-1</td>
</tr>
<tr>
<td>Using Program &quot;Student Status&quot;</td>
<td>A-1</td>
</tr>
<tr>
<td>Using Program &quot;CAI_Statistics&quot;</td>
<td>A-2</td>
</tr>
</tbody>
</table>
Appendix B: Program Listings ............................................ B-1
  Program "CAI" ......................................................... B-1
  Program "STUDENT_STATUS" ................................. B-34
  Program "CAI_STATISTICS" ................................. B-41

Appendix C: Files Used by Program "CAI" .................. C-1
  File "INTRO" ....................................................... C-1
  File "MENU" ........................................................ C-2
  File "LESSON1" .................................................... C-3
  File "LESSON2" ................................................... C-28
  File "LESSON3" ................................................... C-51
  File "LESSON4" ................................................... C-72
  File "LESSON5" ................................................... C-92
  File "LESSON6" ................................................... C-109
  File "EXIT" .......................................................... C-128

Vita ................................................................. V-1
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>CAI - Main</td>
<td>4-5</td>
</tr>
<tr>
<td>4.2</td>
<td>CAI - Query</td>
<td>4-6</td>
</tr>
<tr>
<td>4.3</td>
<td>CAI - StartLesson</td>
<td>4-7</td>
</tr>
<tr>
<td>4.4</td>
<td>CAI - ShowTopic</td>
<td>4-8</td>
</tr>
<tr>
<td>4.5</td>
<td>CAI - Tframe</td>
<td>4-9</td>
</tr>
<tr>
<td>4.6</td>
<td>CAI - Qframe</td>
<td>4-10</td>
</tr>
<tr>
<td>4.7</td>
<td>CAI - Mquestion</td>
<td>4-11</td>
</tr>
<tr>
<td>4.8</td>
<td>CAI - Pquestion</td>
<td>4-12</td>
</tr>
<tr>
<td>4.9</td>
<td>Student_Status - Main</td>
<td>4-14</td>
</tr>
<tr>
<td>4.10</td>
<td>CAI_Statistics - Main</td>
<td>4-16</td>
</tr>
<tr>
<td>4.11</td>
<td>CAI_Statistics - Display</td>
<td>4-17</td>
</tr>
<tr>
<td>4.12</td>
<td>CAI_Statistics - ShowStats</td>
<td>4-18</td>
</tr>
</tbody>
</table>
Abstract

The field known as "computer assisted instruction" or "CAI" as it is commonly called, has gained considerable interest and support since the advent of the microcomputer. More and more people, including those in supervisory positions are beginning to see the advantages, both cost and time, in having training available in the workplace. This study developed a training package for use on the Zenith Z-100 microcomputer. The package consists of six lessons and three programs. The six lessons cover various topics dealing with the "C" programming language. The objective of these lessons is to present an introduction to the "C" programming language. The three programs are written in the Pascal programming language and are used for the following functions:

1. Provide a means of displaying the lesson material.
2. Provide a means of checking student progress.
3. Provide a means of displaying course statistics.
Background

The use of Computer Assisted Instruction (CAI) to help in the training needs of the Services has increased with the introduction of microcomputer systems into the workplace. The development of CAI courses for use on these computer systems has been lagging behind the need for training on the new systems. The CAI development process involves a working knowledge of the system to present the material as well as a knowledge of the subject to be presented. The presentation of the developed CAI course is usually controlled by means of some type of presentation program. Manpower and time constraints may prohibit development of such a program and indicate the need to utilize a commercially developed authoring/presentation system.

The use of a commercial authoring system requires that a coursewriter learn that specific authoring system for use on a specific microcomputer system. The coursewriter can then devote his attention to the development of the course subject material. The subject topics that typically are identified as of primary importance include: word processing, data base management, spreadsheets, operating systems, and programming languages.
Statement of Problem

The problem to be solved is as follows: How can a computer assisted instruction (CAI) course be written and implemented to teach the "C" programming language on the Zenith Z-100 microcomputer system without the use of a commercial authoring/presentation system? The course will be of sufficient length to instruct the beginning student to a level that will allow him/her to program using the "C" programming language. The course subject will be broken into lesson topics which are made up of subsections of the lesson topic.

Each lesson will:

1. Give the student the ability to select between being shown the complete lesson or only reviewing certain parts.
2. Have the ability to sample student comprehension during lesson presentation by means of questions.
3. Have the ability to branch, at appropriate times, to other parts of the lesson.
4. Give the student a chance to review subsections before being tested on the lesson material.
5. Have the ability to test the student on the presented material after lesson completion.
6. Have the ability to allow for review of subsections before retesting (in the case of lesson failure).

In addition to the above, a record will be kept of student responses, both during the presentation of the lessons and the tests, for later statistical analysis and
display. This is done in order to be able to identify areas of the course that perhaps are not teaching the material as intended and/or are causing the student difficulties.

Scope

The scope of this thesis effort is to design, implement, test, and validate a CAI course for presenting information on the "C" programming language. The design phase will incorporate top down structured programming techniques.

Although it is not the primary purpose of this thesis, a method for developing the textual material and presenting that material will necessarily be created. This added benefit arises from the fact that no commercially available authoring/presentation software will be used. This opens up the possibility of developing other courses using programs written during this thesis effort.

The end result of this thesis is to develop a CAI course that will be acceptable to the sponsor at Keesler AFB, who will then distribute the course to all interested training managers throughout the Services.

Assumptions

It is assumed that the students who will use this CAI course will have a working knowledge of the operation of the Zenith Z-100 microcomputer system that this course is designed to run on. This Z-100 system is the standard system purchased by government contract through Zenith Data
Systems, namely, the 192K byte, two 5.25 inch disk drive system. Although it would be of some benefit, there isn't any requirement that students taking this CAI course have access to a "C" compiler.

General Approach

The first step in this endeavor is to do research into the techniques of teaching with a computer. The purpose here is to broaden the teaching base from which to build the overall course presentation. Once the methods of presenting the material are well in hand, the course material will be researched to establish a firm background from which to teach. The next step is to write the individual lessons and develop the program to present them. A program will then be developed to do the statistical analysis and display.

Following the research and development phase will be the implementation of the system. This phase will consist of putting all the pieces into a cohesive package that will accomplish the goal of the study, namely, use the Zenith Z-100 computer system to present a CAI course on the "C" programming language.

In order to ensure the development and implementation of a quality product, an extensive testing and validation system will be incorporated throughout the study. The ultimate test will come when the sponsor at Keesler AFB tests the course against their well-established standards.
II. Methodology

The Aim of CAI

The overall aim of Computer Assisted Instruction (CAI) is, naturally, to use a computer system to assist in the training of individuals in a given subject.

In its most common form, CAI is very similar to a programmed text. The subject material is presented to the student, questions are asked of the student, answers are evaluated, and a decision is made as to what material is shown next. If the questioning indicates that the student understands the material, new material is shown. If the student seems to be having trouble with a particular part of the lesson, a branch can be made to supplemental material to help the student understand. Other forms of CAI use simulation and/or emulation techniques. These methods of instruction are very useful when teaching a specific performance process but not for such things as computer programming.

Since it is the intent of this thesis effort to teach a programming language, the method used will closely resemble that of programmed text.

Advantages of CAI

There are many advantages to CAI, the following are but a few of the more important ones: Standardization, time efficiency, availability, flexibility, modularity, and cost efficiency.
Each of these advantages contribute to the overall attractiveness of using CAI as a method of training. Standardization is accomplished by programming the subject material into the computer. In this way, each and every student who takes a given course will receive the same information. The unfortunate human flaw of a human teacher forgetting to mention some important detail is thus avoided.

The use of CAI can save time by allowing students to progress at their own rate. This is opposed to the alternative of locking them into a classroom setting and controlling the pace of the class as a whole. This leads us to another of the advantages, that of availability. Since the CAI course is conducted on a microcomputer system, the course is virtually available at all times. This means that many training requirements can be accomplished without the student having to leave his/her work area. Hence you have flexibility (another advantage) in scheduling training. Since the course is available at all times, training can be scheduled around work requirements.

The actual construction of a well developed CAI package should allow for the accessing of information in a rapid way. This usually calls for the development of sections of the course in small modules. In this way the student doesn't necessarily need to complete an entire course to get at the information that pertains to his/her job requirements. The CAI course developed in this thesis follows this modular course concept. The course is broken into lessons.
which are each broken into topics.

The last advantage mentioned is that of cost effectiveness. By taking the previous advantages into consideration it is easy to see how using CAI can achieve a cost effective training program. Training can be conducted whenever workload requirements allow the student enough free time to take CAI lessons. The need for the student to travel to some other location for needed training can also be reduced.

Disadvantages of CAI

There are of course drawbacks to everything, and CAI is no exception. A few of the disadvantages follow: System availability, Uni-directional training, acceptability.

The availability of the computer system for training purposes can restrict the usefulness of the CAI system. The primary reasons for system nonavailability are: Operational requirements and maintenance downtime. An organization's operational requirements may be such that a computer system cannot be spared in order to accomplish a training requirement in a timely manner. Obviously, if a computer system is down for maintenance, training cannot be accomplished using that system.

The second disadvantage involves the inability of the students to ask questions of their trainer. This requires the student to concentrate hard on the presented material in order to ensure the required understanding. Since the student doesn't (usually) have the ability to query the compu-

2 - 3
ter on a point that may be causing him/her problems, the student must seek out someone who knows the subject in order to receive clarification. This of course is not all bad, since this will lead to better communication in the workplace.

Lastly, CAI is not completely accepted by management personnel as an alternative to classroom instruction. Formal classroom instruction has been used for so long that many believe it to be the only effective means of accomplishing required training.

Development Considerations

There are several design considerations to take into account when coming up with a methodology for CAI course development. The first of these and perhaps the most important is the objective of what is to be taught. As stated before, the objective of this study is to create a means of presenting information on the "C" programming language on the Z-100 microcomputer system. The second consideration is the resources available for training. The necessary resources for taking the course developed in this study is any microcomputer system that runs under the MS-dos operating system. The primary system will be the Zenith Z-100. The third consideration is the teaching technique to be used. As stated earlier, this will most closely resemble a programmed text presentation. The last major development consideration is course validation. In addition to initial
development validation, the CAI package will provide a means of recording student progress and provide for statistical collection of student responses to all questions throughout the course. These capabilities will be described in detail later in the study.

Course Development Approach

The general approach to developing this CAI was to write a program which would keep track of as many as twenty students as well as present the material to the student in short topic sessions. The student has total control over which topics he/she views. The presentation program is written in the PASCAL programming language and is easy to modify in the case of any future enhancements. Two additional programs, both written in PASCAL, have been included in the package. The purpose of the first is to produce a report of the current student status for each registered student on a given student disk. The purpose of the second is to produce a report of the statistics collected on the student responses during course presentation. More detailed information can be found in Chapter 3.
III. Design Specification

General Description

The purpose of this computer assisted instruction (CAI) package is to provide a means of presenting introductory information on the "C" programming language. The material to be presented is to be stored in separate lesson files on one five and one quarter inch floppy disk that has been formatted using the MS-DOS "format" program. The lesson files are to be created using any text editor that will run under MS-DOS. The lessons are to be broken into topic sections that the student can complete in a relatively short period of time. Three programs will be provided with the CAI package and a description of these follows.

CAI Program

The main program contained in the CAI package is the one that will present the course material to the student. This program will read and display several files automatically in addition to reading and displaying the student's chosen subject material. This program will keep a record of student progress through the course as well as write to a statistical collection file to be used for future course validation and improvements.

Lesson Files. The main CAI program will have access to six lesson files which contain the course material. Each of these lesson files will contain introductory information for its particular lesson material. An associated menu is also
included for display, allowing the student to choose the topic material to be shown.

**Other Files.** In addition to the lesson files discussed above, there are five other files to be used by the main CAI program. First, there is to be a file that contains introductory comments to the student. This file will be kept short since it is to be seen each and every time the program is executed. Second, there is to be a main menu file that will allow the student to choose which of the six lessons they want to enter. This file will be restricted to one screen in size. Third, there is to be a file that contains program conclusion comments. This file will also be kept short and is only to serve as a means of assuring the student that they have correctly terminated the program. Fourth, there is to be a file that will store data on as many as twenty students. This file is where each student's progress through the course is to be kept and will be keyed on a unique student identification number. Lastly, there is to be a file that will store data for each question displayed during course presentation. This file will be used for ongoing course improvements.

**Status Program**

The status program will be provided for use by the training monitor. Its whole purpose is to provide a means of viewing each student's record in order to determine their status in the course. A report is to be generated giving
the unique student identification number and listing all lessons that have been successfully passed. The program will allow for the possibility of the training monitor having merged several student record files into one file. The program will also format the report for either screen display or hardcopy printout.

"STUDENT" File. The file to be read by the status program is to contain student identification numbers, student names, as well as lesson and topic status data. An active student file will contain at most twenty unique student records. As mentioned earlier, several student files may be merged into one student file prior to running the status program.

Statistics Program

A statistics program will be provided for use by the office of primary responsibility (OPR) at Keesler AFb. This program is to provide a means of analyzing the data collected on questions presented during each training session. A report is to be generated giving information such as lesson number, frame number, number of responses for each of the valid responses, number of right responses, number of wrong responses, percent of right responses, and percent of wrong responses. The results of the statistical analysis is to be displayed in either of two formats: screen display or hardcopy printout.
"STATS" File. The file to be read by the statistics program is to contain such items as lesson number, topic number, frame number, correct answer, and the student's response. Again, several of these files may be merged into one file prior to running the statistical program.
IV. System Implementation

General Description

The implementation of this computer assisted instruction (CAI) training package involved the development of lesson material covering six major subject areas. The development of these lessons was accomplished in conjunction with the development and validation of the three programs specified in chapter three. This chapter presents a brief description of each major component of the CAI training package. Copies of these components are provided in appendixes B and C.

"C" Lessons Descriptions

The following is a breakdown of the subject material as presented in the "C" CAI course:

Lesson One. Lesson one contains introductory information on the course and some general information on "C" programming. The lesson is broken into four subtopics and a lesson test. The first subtopic gives a short introduction to the overall course structure and some of the particulars used in the course. The second subtopic discusses the overall organization and structure of a typical C program. The third subtopic gives a description of the overall C programming environment covering such items as "compiling" and "linking". The forth subtopic states a problem to be solved and presents a solution to help introduce the student to C program statements.
**Lesson Two.** Lesson two contains information on variables, constants, operators, and expressions used in C programming. The lesson is broken into four subtopics and a lesson test. The first and second subtopics cover the declaration and use of variables and constants. The third and forth subtopics cover the use of the different operators and expressions in C programming.

**Lesson Three.** Lesson three contains information on program control statements used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic gives descriptions of the structure and use of the "if" and "if-else" control statements and how to "nest" these statements along with a description of the "switch" control statement. The second subtopic discusses the structure and use of loop statements (while, for, and do-while). The third subtopic gives a description of the "break" and "continue" statements and how they are used. The forth subtopic gives a description of the "goto" statement and the use of "labels" within a C program.

**Lesson Four.** Lesson four contains information on arrays, pointers, and address arithmetic used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic introduces the declaration, initialization, and use of arrays. The second subtopic introduces the declaration and use of pointers. The third and forth subtopics cover how to work with pointers and includes topics such as how pointers are passed to functions, how pointers
are used in conjunction with arrays, and how to use address arithmetic.

Lesson Five. Lesson five contains information on structures that are used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic introduces the idea of structures and two methods of their declaration. The second subtopic describes the use of structures within structures and arrays of structures. The third subtopic describes how to use pointers in conjunction with structures. The forth subtopic describes how structures are passed between functions.

Lesson Six. Lesson six contains introductory information on input and output capabilities of the C language. The lesson is broken into four subtopics and a lesson test. The first subtopic gives a description of the use of the standard I/O functions "getchar" and "putchar". The second subtopic gives a description of the use of the standard input function "getline". The third subtopic gives a description and examples of the standard input function "scanf". The forth subtopic gives a description and examples of the standard output function "printf".

CAI Program

Program CAI is the program that is used to present the lesson material to the student. The program is designed to present any lesson material that is in the same format as the lessons developed in this thesis. Therefore, additional
courses may be written for presentation on the Zenith Z-100 by this program. The following is a breakdown and brief description of the program.

**Structure Charts.** The program is broken into a main program and 17 procedures, all of which are written in the Pascal programming language. Structure charts of this program are presented in Figures 4.1 thru 4.8 of this chapter.

**Flow Description.** The flow of this program follows a very logical structured path. The program begins by presenting an introductory message from file "INTRO". The student is next queried for their unique student identification number. A search is then made of file "STUDENT" (which has been read into memory) and if no match is found (a new student) the student is queried for their name and unique student identification number they wish to use from this point on. Next, the student is presented a menu of lessons from which to choose (file "MENU"). Once a selection is made, introductory information for the chosen lesson is displayed and another menu is presented giving the student a choice of lesson subtopics. Once the student chooses a lesson subtopic, the topic is read into memory and topic presentation begins. When the topic is completed, an update of the CAI statistical collection file as well as the students progress record file is made. The student is then returned to the subtopic selection menu, where if the student wishes, he/she may exit to the lesson selection menu, where if the student wishes, he/she may exit the program.

4 - 4
Figure 4.3 CAI - StartLesson
Figure 4.8 CAI - Question
Status Program

Program Student_Status is the program that is used to present the current student status for all students recorded in file "STUDENT". The program is designed to accept and present any number of student records. This provides for the merging of several student files prior to running the program. There are two output formats for this program, "screen" and "hardcopy". The following is a breakdown and brief description of the program.

Structure Chart. The program is broken into a main program and five procedures, all of which are written in the Pascal programming language. A structure chart of this program is presented in Figure 4.9 of this chapter.

Flow Description. The flow of this program follows a straightforward path. The program begins by asking the user for the preferred method of report format, choices are either "screen" or "hardcopy". A header is then displayed and is followed by the student progress information. The structure of the report is in the format of "student identification number" followed by the word "passed" for every lesson that the student has successfully completed.
Figure 4.9 Student Status - Main

1: choice
2: character
3: count
4: advance
Statistics Program

Program CAI_Statistics is the program that is used to present the statistics collected on all questions asked during all course presentation sessions. The purpose of the program is to provide a means for the office of primary responsibility (OPR) at Keesler AFB to verify course content and effectiveness. The program is designed to accept and present statistics on as many as 150 different question frames. This restriction can be overcome by changing one line of source code (a constant value), if it becomes necessary. Several "STATS" files can be combined (and should be) before running this program. There are two output formats for this program, "screen" and "hardcopy". The following is a breakdown and brief description of the program.

Structure Charts. The program is broken into a main program and eight procedures, all of which are written in the Pascal programming language. Structure charts of this program are presented in Figures 4.10 thru 4.12 of this chapter.

Flow Description. The flow of this program follows a straightforward path. The program begins by asking the user for the preferred method of report format, choices are either "screen" or "hardcopy". A header is then displayed and is followed by internal reading and sorting routines. The output report is displayed in columns, giving all the needed statistics to the user. Items such as percent right and percent wrong help to validate questions.
Figure 4.10 CAI Statistics - Main
V. Conclusions and Recommendations

General Comments

The computer assisted instruction (CAI) package developed, tested, and implemented in this thesis effort presents an introduction to the "C" programming language. Although it does not get deep into fancy "C" language usage, it does serve its primary purpose of providing a strong base from which the student can build his/her "C" programming expertise. With a little initiative, the student will soon have the full power of the language at their disposal.

As was mentioned in chapter one, the primary goal of this study was to develop a course on the "C" programming language to be presented on the Zenith Z-100 microcomputer system. In order to achieve the stated goal a secondary goal had to be met, that of developing a software presentation system for the developed course material. This secondary goal provides the possibility of producing other courses for presentation on the Z-100 system.

Suggestions for Further Study

The existing presentation program is a good one as it stands, but certain enhancements would make it better. One such enhancement would be to add logic to allow for the asking of "fill in the blank" type questions. Another would be to allow the student to backup to a previously seen frame. One improvement in program control would be to read in the first frame of a topic, display it, and then read in
the rest of the topic while the student is reading the first frame. Currently, the student must wait nearly one minute before any topic material is displayed after they have chosen the topic from the topic menu.

Finally, the overall "C" course can be improved in several ways. Two of these are: provide for more branching to supplemental material and cover more of the capabilities of the "C" programming language. The course material and the programs used in conjunction with its use can be an effective means of getting introduced to the wonders of "C" programming.
Using Program "CAI"

Program CAI is the main program of this computer assisted instruction (CAI) package. The executable program is stored on "Disk 1" under the filename CAI.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Remove the operating system disk from drive A, place "Disk 1" in drive A and "Disk 2" in drive B. Disk 2 contains the six lesson files of the C CAI course.

Once the disks are in place, type CAI in response to the A> prompt. The main CAI program will begin to execute and will prompt you for any further needed responses. One important item that deserves special mention is the student identification number that you will be prompted for during initial startup. This number is used to keep track of an individual's progress through the course. In order for it to be an effective feature of the package, the same sequence of characters must be entered each time you enter the CAI program.

Using Program "STUDENT STATUS"

Program Student_Status is designed for system training monitors. It is not for use by the students taking the course. This program will produce a report giving the current student status for each student recorded in file
"STUDENT" on "Disk 1". The executable program is stored on "Disk 1" under the filename STATUS.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Replace the operating system disk in drive A with "Disk 1" of the CAI package. Once the disk is in place, type STATUS in response to the A> prompt. The Student_Status program will begin to execute and will prompt you for any further needed responses.

Using Program "CAI STATISTICS"

Program CAI_Statistics is designed for the office of primary responsibility (OPR) at Keesler AFB. It is not for use by the students taking the course. This program will produce a report giving statistics on all the C CAI course questions recorded in file "STATS" on "Disk 1". The executable program is stored on "Disk 1" under the filename VALIDATE.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Replace the operating system disk in drive A with "Disk 1" of the CAI package. Once the disk is in place, type VALIDATE in response to the A> prompt. The CAI_Statistics program will begin to execute and will prompt you for any further needed responses.
Appendix B

Program Listings

Program "CAI"

(**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS ****)

Date: 8/1/85
Version: 1.0

Title: Program CAI
Filename: CAI.FAS
Coordinator: Capt Frank W. DeMarco
Project: Masters Thesis
Operating System: MS-DOS
Language: Pascal

Use: Compile and link with PASCAL.LIB using MS-Pascal compiler and linker.
Contents: Program CAI - Main Driver.

Procedure ClearScreen - Clears Z-100 terminal screen.
Procedure RegStu - Registers a first time student.
Procedure Query - Reads in "STUDENT" file, prompts student for student identification number, and checks the ID number against current student list.
Procedure StartEnd - Reads and displays files "INTRO" at start of program and "EXIT" at end of program.
Procedure Select - Reads and displays file "MENU", prompts the student for choice of lesson to be shown.
Procedure ShowTopic - Driver of procedures that display topic material.
Procedure BlankLines - Initializes area where topic material is stored to blanks.
Procedure Readlines - Reads in topic that the student chose to view.
Procedure StorePositions - Builds an array of line positions where frames begin within the topic.
Procedure FrameHeader - Displays a frame header for a frame.
Procedure Tframe - Displays a text type frame.
Procedure Qframe - Driver for the procedures that display and handle question type frames.
Procedure Mquestion - Displays and handles multiple choice type question frames.
Procedure Pquestion - Displays and handles pick type question frames (true/false and yes/no).
Procedure RecordStats - Reads file "STATS" and adds statistical data from current session.
Procedure StuFec - Writes updated student course progress data to file "STUDENT".
Procedure StartLesson - Displays topic choices for a lesson, prompts student for choice of topic to be shown. Driver of procedures that display lesson material and update
statistical & student progress files.

* Function: The purpose of this program is to present material on the "C"
programming language. It is intended for use by the 3300 Technical
Training Wing in support of its mission. The office of
primary responsibility for this course is the CAI Plans Branch
(3300 TCHTW/TTGXZ) at Keesler AFB, MS 39534

*******************************************************************************
* Date: 9/1/85
* Version: 1.0
* Name: program CAI
* Module number: 1.0
* Description: Main driver of program
* Passed Variables: None
* Returns: None
* Global Variables Used: studentcount, choice
* Global Variables Changed: None
* Files Read: None
* Files Written: None
* Modules Called: ClearScreen, StartEnd, Query, Select, StartLesson
* Calling modules: None
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 8/1/85 - input original code
*******************************************************************************

program CAI (input, output):

const
MAXSTUDENTS = 20;
MAXLESSONS = 6;
MAXTOPICS = 5;
MAXLESSON = '6';
MAXTOPIC = '5';
ALINEP1 = 'X

BLANKS = ' ';
BLANKS = '

type
  file = TEXT;
  roll = record
    studentnumber : packed array [1..11] of char;
    studentname : packed array [1..28] of char;
    lessons : packed array [1..MAXLESSONS] of char;
    topics : array [1..MAXLESSONS, 1..MAXTOPICS] of char;
  end;

  roster = array [1..MAXSTUDENTS] of roll;

R = 2
lstat = array [1..MAXLESSONS] of char;
displayln = packed array [1..80] of char;
lessonlines = array [1..500, 1..80] of char;
menulines = array [1..22, 1..80] of char;
tstat = array [1..MAXTOPICS] of char;

var
  iomessage, student, statfile, menu, lesson, temp1, temp2 : iofile;
  advance, linecount, studentcount : integer;
  choice, lchoice : char;
  npupil : roll;
  rpupil : roster;
  lessonstat : lstat;
  println : displayln;
  lessonln : lessonlines;
  menuin : menulines;
  topicstat : tstat;

procedure ClearScreen;
begin
  write(chr(27),'H',chr(27),'J',chr(27),'w')
end;

procedure ClearScreen;
begin
  write(chr(27),'H',chr(27),'J',chr(27),'w')
end:
procedure RegStu;

var
  i, j: integer;

begin (* Procedure RegStu *)

  for j := 1 to 11 do
    npupil.studentnumber[j] := ' '
  for j := 1 to 28 do
    npupil.studentname[j] := ' '
  for j := 1 to MAXLESSONS do
    npupil.lessons[j] := ' '
  for i := 1 to MAXLESSONS do
    begin
      for j := 1 to MAXTOPICS do
        npupil.topics[i, j] := ' ';
    end;

  writeln;
  writeln('Since this is your first time into this course, I have a few ');  
  writeln('administrative matters to take care of.');
  writeln;
  writeln('Please enter your first name: ');  
  write('(Max. of 10 characters) >>>> ');  

  i := 1;
  while not (eoln) and (i < 11) do
    begin
      read (npupil.studentname[i]);
      i := i + 1
    end;
  if (eoln) and (i < 11) then
    begin
      for i := 1 to 10 do
        npupil.studentname[i] := '*';
readln
else
  readln;
writeln;
writeln;
writeln('Please enter your middle initial: ');writeln('>><><><><><> ');
i := 11;
while not (eoln) and (i < 12) do
  begin
    read (nPupil.studentname[i]);
    i := i + 1
  end;
if (nPupil.studentname[11] in ['a'..'z','A'..'Z']) then
  begin
    nPupil.studentname[12] := '>';readln
  end
else
  begin
    nPupil.studentname[12] := '*';
    readln
  end;
writeln;
writeln;
writeln('Please enter your last name: ');writeln('>><><><><><> ');
i := 13;
while not (eoln) and (i < 29) do
  begin
    read (nPupil.studentname[i]);
    i := i + 1
  end;
if (eoln) and (i < 29) then
  begin
    for i := 1 to 28 do
      nPupil.studentname[i] := '*';
    readln
  end
else
  readln;
writeln;
writeln;
writeln('Now for the most important part.');
writeln('Please enter your unique, personal student identification number: ');
write('(Max. of 11 characters) >>>>>> ');

i := 1;
while not (eoln) and (i < 12) do
begin
read (npupil.studentnumber[i]);
i := i + 1
end;
if (eoln) and (i < 12) then
begin
for i := i to 11 do
npupil.studentnumber[i] := '*';
readln
end
else
readln;

for i := 1 to MAXLESSONS do
begin
npupillessons[i] := '-';
for j := 1 to MAXTOPICS do
npupil.topics[i,j] := '-';
end;

studentcount := studentcount + 1
end;

(* Procedure RegStu *)

(* Date: 8/1/85
* Version: 1.0
* Name: procedure Query
* Module number: 4.0
* Description: Reads in "STUDENT" file, prompts student for student identification number, and checks the ID number against current student list.
* Passed Variables: None
* Returns: None
* Global Variables Used: rpupil, studentcount, npupil
* Global Variables Changed: rpupil, studentcount, npupil
* Files Read: student
* Files Written: None
* Modules Called: Clear Screen, RegStu
* Calling modules: program CAI
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 8/1/85 - input original code

************************************************************************************************
$ Global Variables Chanoed: rpupil, studentcount, npupil
$ Passed Variables: None
$ Returns: None
$ Files Read: student
$ Files Written: None
$ Modules Called: Clear Screen, RegStu
$ Calling modules: program CAI
$ Author: Capt Frank W. DeMarco
$ History:
* 1.0 Frank W. DeMarco 8/1/85 - input original code
*************************************************************************************************
procedure Query;

var
  i, ii, j : integer;
  qfound : boolean;
  character : char;

begin (* Procedure Query *)

  assign (student,'student');
  reset (student);
  character := ' ';
  if not (eof(student)) then
    read (student,character);

  for i := 1 to MAXSTUDENTS do
    begin
      for j := 1 to 11 do
        rpupil[i].studentnumber[j] := ' ';
      for j := 1 to 28 do
        rpupil[i].studentname[j] := ' ';
      for j := 1 to MAXLESSONS do
        rpupil[i].lessons[j] := ' ';
      for ii := 1 to MAXLESSONS do
        begin
          for j := 1 to MAXTOPICS do
            rpupil[i].topics[ii,j] := ' ';
        end;
    end;

  i := 1;
  studentcount := 0;
  while (character = '>') and not (eof(student)) do
    begin
      studentcount := studentcount + 1;
      while not (eofn(student)) do
        begin
          for j := 1 to 11 do
            read (student,rpupil[i].studentnumber[j]);
          for j := 1 to 28 do
            read (student,rpupil[i].studentname[j]);
          for j := 1 to MAXLESSONS do
            read (student,rpupil[i].lessons[j]);
          for ii := 1 to MAXLESSONS do
            begin
              for j := 1 to MAXTOPICS do
                read (student,rpupil[i].topics[ii,j])
              end;
          i := i + 1;
        end;
      if not (eof(student)) then
        readln (student);
    end;

end.
if not (eof(student)) then
  read (student, character);
end;

ClearScreen:

write('Please enter your student identification number: ');
write('(Max. of 11 characters) >>>>> ');

i := 1;
while not (eoln) and (i < 12) do
  begin
    read (npupil.studentnumber[i]);
    i := i + 1
  end;
if (eoln) and (i < 12) then
  begin
    for i := 1 to 11 do
      npupil.studentnumber[i] := '*'
  end;
readln;

i := 1;
qfound := false;
while (i < 21) do
  begin
    if (npupil.studentnumber = rpupil[i].studentnumber) then
      begin
        qfound := true;
        npupil.studentname := rpupil[i].studentname;
        npupil.lessons := rpupil[i].lessons;
        for ii := 1 to MAXLESSONS do
          begin
            for j := 1 to MAXTOPICS do
              npupil.topics[ii, j] := rpupil[i].topics[ii, j]
          end;
        i := i + 1
      end;
if not (qfound) and (studentcount < MAXSTUDENTS) then
  begin
    ClearScreen;
    writeln ('NO MATCH FOUND');
    RegStu;
    ClearScreen
  end
else
  if not (qfound) and (studentcount = MAXSTUDENTS) then
    begin
      ClearScreen;
      studentcount := studentcount + 1;
      writeln ('Sorry, but my class roster shows a "Full" class.');
    end
writeln('Please see your training monitor for a new student disk.');
writeln;
writeln('END OF PROGRAM')
end;

close (student)
end:  (* Procedure Query *)

procedure StartEnd(code : char);

var
    character : char;
    : : integer;

begin  (* Procedure StartEnd *)

    case code of
        'S' : assign (iomessage,'intro');
        'E' : assign (iomessage,'exit')
    end:

    reset (iomessage);
    read (iomessage,character);
    linecount := 0;

    repeat
        while (character = '#') and not (eof(iomessage)) do begin
            for i := 1 to 80 do
procedure Select

begin
    lessonstat := linecount;
    linecount := 0;
    for i := 1 to advance do
        writeln;
    for i := 1 to 27 do
        write ('');
    write ('Press RETURN to continue.');
    readln;
    if not (eof(iomessage)) then
        begin
            readln (iomessage);
            if not (eof(iomessage)) then
                read (iomessage, character)
        end;
    ClearScreen
    end;
end;

until (eof(iomessage));

close (iomessage);

end: (* Procedure StartEnd *)

******************************************************************************
* Date: 9/1/85
* Version: 1.0
*
* Name: procedure Select
* Module number: 5.0
* Description: Reads and displays file "MENU", prompts the student for
* choice of lesson to be shown.
* Passed Variables: None
* Returns: None
* Global Variables Used: lessonstat, println, choice
* Global Variables Changed: lessonstat, println, choice
* Files Read: menu
* Files Written: None
* Modules Called: ClearScreen
* Calling modules: program CAI
*
* Author: Cast Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 9/1/85 - input original code
******************************************************************************
procedure Select;

var
  character : char;
  i, j : integer;

begin (* Procedure Select *)

  assign (menu,'menu');

  for i := 1 to MAXLESSONS do
    lessonstat[i] := nbupil.lessons[i];
  repeat
  reset (menu);
  read (menu,character);

  j := 0;
  while (character in ['*','@']) and not (eof(menu)) do
    begin
      readln (menu,println);
      if (character = '*') then
        begin
          write ('*');
          for i := 2 to 78 do
            write (println[i]);
          writeln (println[79])
        end
      else
        begin
          j := j + 1;
          write ('*');
          for i := 2 to 8 do
            write (println[i]);
          write (lessonstat[j]);
          for i := 10 to 78 do
            write (println[i]);
          writeln (println[79])
        end;

      if not (eof(menu)) then
        read (menu,character)
      end;

  writeln;
  write ('ENTER THE NUMBER OF YOUR CHOICE OR "X" TO EXIT THE CAI PROGRAM: ');
  readln (choice);
  if (choice in ['1','2',...,'LESSON',',','X']) then
    ClearScreen
  else

begin
  ClearScreen;
  writeln ('Sorry, ', choice, ' is not a valid response. Please try again.');
end;

until (choice in ['I'..'VLESSON','x','X']);

if (choice in ['I'..'VLESSON]) then
  writeln ('You have chosen lesson number ', choice, '. Thank you.');
else
  writeln ('OK, I will now return you to the operating system.');
end.
close (menu);
end;  (* Procedure Select *)

begin  (* Procedure Select *)

(***************************************************************************
* Date: 8/1/95                                      *
* Version: 1.0                                      *
* Name: procedure ShowTopic                               *
* Module number: 6.1                                    *
* Description: Driver of procedures that display topic material.  *
* Passed Variables: None                                *
* Returns: None                                         *
* Global Variables Used: lessonln, npupil, topicstat    *
* Global Variables Changed: npupil, topicstat           *
* Files Read: None                                      *
* Files Written: None                                   *
* Modules Called: ClearScreen, BlankLines, ReadLines, StorePositions,  *
*                 Tframe, Oframe                        *
* Calling modules: StartLesson                          *
* Author: Capt Frank W. DeMarco                        *
*> History:                                               *
* 1.0 Frank W. DeMarco 8/1/95 - input original code    *
***************************************************************************)

procedure ShowTopic:

const
  MINSCORE = 70.0;

type
  position = record
    framenum : integer;
    value : integer;
  end;

  topictitle = packed array [1..30] of char;

var
  lplace : array [1..50] of position;
  tname : topictitle;
procedure BlankLines:

var index1, index2 : integer;

begin (* Procedure BlankLines *)

for index1 := 1 to 500 do
begin
  for index2 := 1 to 80 do
    lessonln[index1,index2] := ' ';
end;

for index1 := 1 to 70 do
  tname[index1] := ' ';

for index1 := 1 to 50 do
begin
  lplace[index1].framenum := 0;
  lplace[index1].value := 0;
end;

end: (* Procedure BlankLines *)
Name: procedure ReadLines
Module number: 6.1.2
Description: Reads in topic that the student chose to view.
Passed Variables: None
Returns: None
Global Variables Used: lessonln, lchoice, tname
Global Variables Changed: lessonln, tname
Files Read: lesson
Files Written: None
Modules Called: None
Calling modules: ShowTopic

Author: Capt Frank W. DeMarco
History:
1.0 Frank W. DeMarco 8/1/85 - input original code

***********************************************************************

procedure ReadLines;

var
  rfound : boolean;
  jvalue, :value : integer;

begin (* Procedure ReadLines *)

  writeln;
  writeln ('One moment please...');

  rfound := false;
  reset (lesson);

  repeat
    read (lesson,lessonln[1,1]);
    if (lessonln[1,1] = lchoice) then
      rfound := true
    else
      readln('lesson');
    until (rfound);

  j := 2;
  while not (eoln('lesson')) do
    begin
      read ('lesson,lessonln[1,j]);
      j := j + 1
    end;

  readln('lesson');
  i := 2;
  read ('lesson,lessonln[i,1]);
  j := 2;
while (lessonln[i, j] = 1choice) and not (eof(lesson)) do
begin
while not (eoln(lesson)) do
begin
    read (lesson, lessonln[i, j]);
    j := j + 1
end;
readln (lesson);
i := i + 1;
j := 1;
if not (eof(lesson)) then
    read (lesson, lessonln[i, j]);
j := j + 1
end;

ivalue := 1;
for ,value := 16 to 45 do
begin
    tname[i, value] := lessonln[i, j, value];
    ivalue := ivalue + 1;
end:
end: (* Procedure ReadLines *)

(*--------------------------------------------------------------------------*)
(* Date: 9/1/95  *)
(* Version: 1.0  *)
(*--------------------------------------------------------------------------*)
(* Name: procedure StorePositions *)
(* Module number: 6.1.3 *)
(* Description: Builds an array of line positions where frames begin within *)
(* the topic. *)
(* Passed Variables: None *)
(* Returns: None *)
(* Global Variables Used: lessonln, i, j, k, lplace, lchoice *)
(* Global Variables Changed: i, j, k, lplace *)
(* Files Read: None *)
(* Files Written: None *)
(* Modules Called: None *)
(* Calling modules: ShowTopic *)
(* Author: Capt Frank W. DeMarco *)
(* History: *)
(* 1.0 Frank W. DeMarco 9/1/95 - input original code *)
(*--------------------------------------------------------------------------*)

procedure StorePositions;
var
    'number, *value : integer;
begin (* Procedure StorePositions *)
i := 1;
j := 2;
k := 0;

repeat
  if (lessonln[i,j] = '1') then
    begin
      k := k + 1;
      fnumber := 0;
      for jval := 9 to 11 do
        fnumber := (10 * fnumber) + ((ord(lessonln[i,jval])) - ord('0'));
      lplace[k].framenum := fnumber;
      lplace[k].value := i;
      i := i + 1
    end
  else
    i := i + 1;
  until (lessonln[i,j] <> lchoice);

k := k + 1;  (* This marks the end *)
lplace[k].framenum := -1;  (* of the array *)

end;  (* Procedure StorePositions *)

******************************************************************************
* Date: 8/1/95
* Version: 1.0
* *
* Name: procedure FrameHeader
* Module number: 7.0
* Description: Displays a frame header for a frame.
* Passed Variables: None
* Returns: None
* Global Variables Used: None
* Global Variables Changed: None
* Files Read: None
* Files Written: None
* Modules Called: ClearScreen
* Calling modules: Tframe, Muestion, Puestion
*
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 8/1/95 - input original code
******************************************************************************

procedure FrameHeader;
begin  (* Procedure FrameHeader *)
  ClearScreen;
  writeln (ALINE1,ALINEPC):

B - 16
writeln ('* Lesson #',choice,' * Topic #',ichoice,' * Title: ',tname,' * Frame: ',frame,' *');
writeln (ALINEP1,ALINEP2)
end;  (* Procedure FrameHeader *)

****************************************************************************
* Date: 8/1/85                           *
* Version: 1.0                           *
* Name: procedure Tframe                *
* Module number: 6.1.4                   *
* Description: Displays a text type frame. *
* Passed Variables: istart               *
* Returns: None                          *
* Global Variables Used: advance, linecount, lessonLn, nextframe *
* Global Variables Changed: advance, linecount, nextframe          *
* Files Read: None                      *
* Files Written: None                    *
* Modules Called: FrameHeader            *
* Calling modules: ShowTopic             *
* Author: Capt Frank W. DeMarco         *
* History:                               *
* 1.0 Frank W. DeMarco 8/1/85 - input original code                   *
****************************************************************************

procedure Tframe(istart : integer);
var
  jnum : integer;
begin  (* Procedure Tframe *)
  advance := 0;
  linecount := 0;
  istart := istart + 1;
  FrameHeader;
  writeln (ABLANKS,BLANKSA);
repeat
  while (lessonLn[istart,j] = 'I' and (linecount < 17)) do
    begin
      writeln ('* ');
      for jnum := 1 to 79 do
        write (lessonLn[istart,jnum]);
      writeln (' *');
      istart := istart + 1;
      linecount := linecount + 1;
    end:
  writeln (ABLANKS,BLANKSA);
  advance := 27 - (linecount + 5);
end:
for i_num := 1 to (advance - 1) do
    writeln (ABLANKS, BLANKSA);
    writeln (ALINEP1, ALINEP2);
for i_num := 1 to 27 do
  write ('Press RETURN to continue.');
readln;
linecount := 0;
if (lessonIn[istart, 2] = '2') then begin
  FrameHeader;
  writeln (ALINEPs, ALINEP);
end;
until (lessonIn[istart, 2] = '3');
if (lessonIn[istart, 2] = '3') then begin
  if (lessonIn[istart, 3] = '8') then begin
    nextframe := 0;
    for i_num := 5 to 7 do
      nextframe := (10 * nextframe) +
      ((ord(lessonIn[istart, i_num]) - ord('0'))
    end
  else
    nextframe := -1;
  end;
end; (* Procedure Tframe *)

(* Procedure Tframe *)

******************************************************************************
# Date: 9/1/95
# Version: 1.0
#
# Name: procedure Tframe
# Module number: 6.1.5
# Description: Driver for the procedures that display and handle question
# type frames.
# Passed Variables: istart
# Returns: None
# Global Variables Used: lessonIn
# Global Variables Changed: None
# Files Read: None
# Files Written: None
# Modules Called: ClearScreen, Mquestion, Fquestion
# Calling modules: ShowTopic
#
# Author: Cael Frank W. DeMarco
# History:
# 1.0 Frank W. DeMarco 9/1/95 - input original code
******************************************************************************

procedure Tframe(istart : integer);
**Procedure MQuestion**

```pascal
var
  type : char;
  valq : integer;

(*-----------------------------------------------------------------------*)
(* Date: 9/1/85 *)
(* Version: 1.0 *)
(* Name: procedure Mquestion *)
(* Module number: 6.1.5.1 *)
(* Description: Displays and handles multiple choice type question frames. *)
(* Passed Variables: istart *)
(* Returns: None *)
(* Global Variables Used: lessonln, test, choice, lchoice, frame, numright, *)
(* nextframe *)
(* Global Variables Changed: numright, nextframe *)
(* Files Read: None *)
(* Files Written: templ *)
(* Modules Called: FrameHeader *)
(* Calling modules: Iframe *)
(* Author: Capt Frank W. DeMarco *)
(* History: *)
(* 1.0 Frank W. DeMarco 8/1/85 - input original code *)
(*-----------------------------------------------------------------------*)

procedure MQuestion(istart : integer);

var
  jnum : integer;
  response, correct, groupnum : char;
  $found : boolean;

begin (* Procedure Mquestion *)

  istart := istart + 1;

  FrameHeader;

  writeln;
  writeln;

  while (lessonln[istart,2] = '2') do
    begin
      for jnum := 7 to 80 do
        write (lessonln[istart,jnum]);
      writeln;
      istart := istart + 1
    end;

  writeln;

  while (lessonln[istart,2] = '7') do
```
begin
    if (lessonIn[iStart,4] = '+') then
        correct := lessonIn[iStart,3];
    write ('','lessonIn[iStart,3],');
    for jnum := 6 to 80 do
        write (lessonIn[iStart,jnum]);
    writeln;
    istart := istart + 1
end;

repeat
    writeln;
    write (' Enter your choice here == > ');
    readln (response);
    if (response in ['.A..','E','a..','e']) then
        writeln
    else
        writeln (' Sorry, that is not a valid response. Please try again.');
    until (response in ['.A..','E','a..','e']);

case response of
    '.a.' , 'A' : response := 'A';
    '.b.' , 'B' : response := 'B';
    '.c.' , 'C' : response := 'C';
    '.d.' , 'D' : response := 'D';
    '.e.' , 'E' : response := 'E'
end;

writeln (templ.choice,choice,frame:3,correct,response);

if (response = correct) then
begin
    if (test) then
        numright := numright + 1.0;
        groupnum := '4'
    else
        groupnum := '5';
    mfound := false;
    repeat
        if (lessonIn[iStart,2] = groupnum) then
            mfound := true
        else
            istart := istart + 1;
        until (mfound);
    if (groupnum = '4') then (* Start Group '4' Logic *)
begin
    while (lessonIn[iStart,4] <> 'B') or (lessonIn[iStart,5] <> 'i') do
begin
    for jnum := 4 to 80 do
        write (lessonIn[iStart,jnum]);

P = 79
writeln;
istart := istart + 1
end;
if (lessonln[istart,4] = 'B') and (lessonln[istart,5] = ':') then
begin
nextframe := 0;
for jnum := 6 to 8 do
nextframe := (10 * nextframe) +
((ord(lessonln[istart,jnum])) - ord('0'))
end
else
nextframe := -1
end: (* End Group '4' Logic *)

if (groupnum = 'E') then (* Start Group '5' Logic *)
begin
if (lessonln[istart,3] = response) and
(lessonln[istart,4] = ':') then
begin
while (lessonln[istart,4] <> 'B') or
(lessonln[istart,5] <> ':') do
begin
for jnum := 5 to 80 do
write (lessonln[istart,jnum]);
writeln:
istart := istart + 1
end;
end
else
begin
mfound := false;
jnum := 3;
repeat
while (lessonln[istart,jnum] <> ':') and not (mfound) do
begin
if (lessonln[istart,jnum] = response) then
mfound := true;
jnum := jnum + 1
end;
if not (mfound) then
begin
istart := istart + 1;
jnum := 3
end
else
begin
while ('lessonln[istart,jnum] <> ':') do
jnum := jnum + 1
end;
until (mfound) or (lessonln[istart,2] <> '5');
if (mfound) then
begin

end
while (lessonIn[istart,4] <> 'B') or
    (lessonIn[istart,5] <> ':') do
begin
    for jnum := jnum to 80 do
        writeln;
        jnum := 4;
    istart := istart + 1
end;
if (lessonIn[istart,4] = 'B') and (lessonIn[istart,5] = ':') then
begin
    nextframe := 0;
    for ,num := 5 to 9 do
        nextframe := (10 * nextframe) +
            (ord(lessonIn[istart,jnum]) - ord('0'))
end
else
    nextframe := -1
end
else
    writeln ('SOMETHING IS AWRY! LET'S TRY THAT AGAIN."
end;
end; (* Procedure Mquestion *)

Date: 9/1/95
Version: 1.0

Name: procedure Mquestion
Module number: 5.1.5.2
Description: Displays and handles pick type question frames (true/false and yes/no).
Passed Variables: istart
Returns: None
Global Variables Used: lessonIn, choice, lchoice, frame, test, numright, nextframe
Global Variables Changed: numright, nextframe
Files Read: None
Files Written: temp!
Modules Called: FrameHeader
Calling modules: Oframe

Author: Capt Frank W. DeMarco
History:
1.0 Frank W. DeMarco 8/1/85 - input original code

procedure Mquestion(istart : integer);
type
   responses = packed array [1..5] of char;

8 - 22
var
    jnum, index : integer;
    answer : sresponses;
    correct, response, groupnum : char;
    pfound, ptrue : boolean;

begin (* Procedure Pquestion *)

    i start := i start + 1;
    FrameHeader;
    writeln;

    while (lessonln[i start,2] = '2') do
        begin
            for jnum := 3 to 80 do
                write (lessonln[i start, jnum]);
            writeln;
            i start := i start + 1
        end;

    writeln;

    if (lessonln[i start,2] = '3') then
    begin
        correct := lessonln[i start,3];
        i start := i start + 1
    end;

    repeat
        index := 1;
        write ('Enter your choice here ==>

        while not (eoln) and (index < 6) do
        begin
            read (answer[index]);
            index := index + 1
        end;

        if (eoln) and (i < 6) then
        begin
            for index := index to 5 do
                answer[index] := ' 
        readln;
        end
        else
            readln;
        ptrue := false;

        if ('answer[1] in ['t','T','f','F','y','Y','n','N']) then
            ptrue := true
        else
            writeln ('Sorry, that is not a valid response. Please try again.

        until (ptrue);
case answer[1] of
   't', 'T', 'y', 'Y' : response := 'Y';
   'f', 'F', 'n', 'N' : response := 'N'  
end;

writeln (temp1,choice,lchoice,frame3,correct,response);

if (response = correct) then
   begin
      if (test) then
         numright := numright + 1.0;
         groupnum := '4'
      end
   else
      groupnum := '5';

pfound := false;

repeat
   if (lessonIn[istart,2] = groupnum) then
      pfound := true
   else
      istart := istart + 1;
   until (pfound);

while (lessonIn[istart,4] <> 'B') or (lessonIn[istart,5] <> ':') do
   begin
      writeln;
      for jnum := 4 to 90 do
         write (lessonIn[istart,jnum]);
      istart := istart + 1
   end;

if (lessonIn[istart,4] = 'B') and (lessonIn[istart,5] = ':') then
   begin
      nextframe := 0;
      for i : 5 to 9 do
         nextframe := (10 * nextframe) +
                      ((ord(lessonIn[istart,i]) - ord('0'))
      end
   else
      nextframe := -1
end;  (* Procedure Question *)

(******************************************************************************)
(* Start of main part of procedure: 0frame  *)
(******************************************************************************)

begin  (* Procedure 0frame *)

if (lessonIn[istart,14] = 'M') then
   stype := 'M'

B - 24
else
   qtype := 'P';

case qtype of
   'M' : Mquestion(istart);
   'P' : Pquestion(istart)
end;

writeln;
for ivalid := 1 to 27 do
   write ('');
write ('Press RETURN to continue.');
readln;
end: (* Procedure Oframe *)

(* Procedure ShowTopic *)

begin (* Procedure ShowTopic *)

BlankLines;
ReadLines;
StorePositions;

numasked := 0.0;
numright := 0.0;
score := 0.0;
test := false;
istart := 1;

frame := 0;
for j := 9 to 11 do
   frame := (10 * frame) + ((ord(lessonIn[istart,j])) - ord('O'));
if (lessonIn[istart,13] = 'T') and (lessonIn[istart,14] = 'T') then
test := true;
repeat
   if (lessonIn[istart,2] = '1') then
      begin
         ftype := lessonIn[istart,13];
         if (test) and ('ftype = 'O') then
            numasked := numasked + 1.0;
         case ftype of
            'T' : Tframe(istart);
            'O' : Oframe(istart)
         end;
         k := 0;
sfound := false;
   end
end: (* Procedure ShowTopic *)
repeat
k := k + 1;
if (lplace[k].framenum = nextframe) then
begin
  istart := lplace[k].ival;
  frame := lplace[k].framenum;
  sfound := true
end;
until (sfound) or (lplace[k].framenum = -1);
end;

until not (sfound) or (lplace[k].framenum = -1);
i := ord(choice) - ord('0');
j := ord(lchaoice) - ord('0');

if (test) then
begin
  ClearScreen;
  score := numright/numasked;
  score := (score * 100.0);
  writeln ('Your test score = ',score:1:2,'%');
  writeln;
  if (score >= MINSCORE) then
begin
    writeln ('CONGRATULATIONS! YOU HAVE PASSED THE LESSON TEST.');
    npupil.topics[i,j] := '+';
    npupil.lessons[i] := '+';
    topicstat[j] := '+'
end
else
begin
  writeln ('Sorry, but you missed too many questions to pass the test.');
  writeln;
  write ('I suggest that you review at least one topic before you ');
  writeln ('retake the lesson test.');
  npupil.topics[i,j] := '-';
  npupil.lessons[i] := '-';
  topicstat[j] := '-'
end
end
else
begin
  ClearScreen;
  npupil.topics[i,j] := 'x';
  topicstat[j] := 'x'
end;
end:  (* Procedure ShowTopic *)

(*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Date: 9/1/95
* Version: 1.0

8 - 26 *)
Name: procedure RecordStats
Module number: 6.2
Description: Reads file "STATS" and adds statistical data from current session.
Passed Variables: None
Returns: None
Global Variables Used: println
Global Variables Changed: println
Files Read: stats, templ
Files Written: temp2, stats
Modules Called: None
Calling modules: StartLesson

Author: Capt Frank W. DeMarco
History:
1.0 Frank W. DeMarco 8/1/85 - input original code

*******************************************************************************

procedure RecordStats;
var
i : integer;

begin (* Procedure RecordStats *)

writeln;
writeln ('One moment please, while I update my records.');
assign (temp2,'t2');
assign ('statfile','stats');
reset ('temp1');
rewrite ('temp2');
reset ('statfile');

repeat
while not (eof ('statfile')) do
begin
  for i := 1 to 80 do
    println[i] := ' ';
  writeln ('statfile',println);
  writeln ('statfile',println)
end;
until (eof ('statfile'));

repeat
while not (eof ('temp1')) do
begin
  for i := 1 to 80 do
    println[i] := ' ';
  writeln ('temp1',println);
  writeln ('temp1',println)
end;
until (eof ('temp1'));

B = 27
reset (temp2);
rewrite (statfile);

repeat
    while not (eof(temp2)) do
        begin
            for i := 1 to 80 do
                println[i] := ' ';
                readln (temp2,println);
                writeln (statfile,println)
        end;
    until (eof(temp2));

rewrite (temp1);
rewrite (temp2);
close (temp1);
close (temp2);
close (statfile);
end;  (* Procedure RecordStats *)

*****************************************************************************
* Date: 9/1/85  
* Version: 1.0  
*   
* Name: procedure StuRec  
* Module number: 6.3  
* Description: Writes updated student course progress data to file  
* "STUDENT".  
* Passed Variables: None  
* Returns: None  
* Global Variables Used: studentcount, rpupil, npupil  
* Global Variables Changed: none  
* Files Read: None  
* Files Written: student  
* Modules Called: None  
* Calling modules: StartLesson  
*   
* Author: Capt Frank W. DeMarco  
* History:  
*  1.0 Frank W. DeMarco 8/1/85 - input original code  
*****************************************************************************

procedure StuRec;

var
    i, ii, j : integer;

begin (* Procedure StuRec *)
    rewrite ('student');
for i := 1 to studentcount do
if (rpupil[i].studentnumber <> npupil.studentnumber) and
(rpupil[i].studentnumber <> '') then
begin
  write('Student:', '');
  for j := 1 to 11 do
    write(rpupil[i].studentnumber[j]);
  for j := 1 to 28 do
    write(rpupil[i].studentname[j]);
  for j := 1 to MAXLESSONS do
    write(rpupil[i].lessons[j]);
  for ii := 1 to MAXLESSONS do
    begin
      if (ii < MAXLESSONS) then
        for j := 1 to MAXTOPICS do
          write(rpupil[ii].topics[ii,j]);
      else
        begin
          for j := 1 to (MAXTOPICS-1) do
            write(rpupil[ii].topics[ii,j]);
          writeln(rpupil[ii].topics[ii,MATOPICS]);
        end;
    end;
  end;
end;

write('Student:', '');
for i := 1 to 11 do
  write(student.npupil.studentnumber[i]);
for i := 1 to 28 do
  write(student.npupil.studentname[i]);
for i := 1 to MAXLESSONS do
  write(student.npupil.lessons[i]);
for i := 1 to MAXLESSONS do
  begin
    for j := 1 to MAXTOPICS do
      write(student.npupil.topics[i,j]);
  end;
end;
close(student)
end; (* Procedure StuRec *)

(***********************************************************************)
* Date: 8/1/85
* Version: 1.0
* *
* Name: procedure StartLesson
* Module number: 6.0
* *
* Description: Displays topic choices for a lesson, prompts student for
* choice of topic to be shown. Driver of procedures that
* display lesson material and update statistical & student
* progress files.
* *
* Passed Variables: None
procedure StartLesson;

var
i, j, k, index : integer;
character : char;

begin (* Procedure StartLesson *)

case choice of
'1': assign (lesson,'b:lesson1');
'2': assign (lesson,'b:lesson2');
'3': assign (lesson,'b:lesson3');
'4': assign (lesson,'b:lesson4');
'5': assign (lesson,'b:lesson5');
'6': assign (lesson,'b:lesson6');
end:

index := ord(choice) - ord('0');
for i := 1 to MAXTOPICS do
  topicstat[i] := npupil.topics[index,i];

reset (lesson);
read (lesson,character);
linecount := 0;

repeat

  while (character = '#') and (linecount < 23) do
    begin
      for i := 1 to 80 do
        println[i] := ' ';
      readln (lesson,println);
      linecount := linecount + 1;
      writeln (println);
      read (lesson,character)
    end:

  repeat

end;
if (character = '!') then
  begin
    advance := 23 - linecount;
    linecount := 0;
    for i := 1 to advance do
      writeln;
    for i := 1 to 27 do
      write (' ');
    write ('Press RETURN to continue.');
    readln;
    readln (lesson);
    read (lesson, character);
    ClearScreen
  end;

until not (character in ['#', '!']);

for i := 1 to 22 do
  begin
    for j := 1 to 80 do
      menuin[i, j] := '
  end:

menuin[1, 1] := character;
for j := 2 to 79 do
  read (lesson, menuin[1, j]);
readln (lesson, menuin[1, 80]);

for i := 2 to 22 do
  begin
    for j := 1 to 79 do
      read (lesson, menuin[i, j]);
    readln (lesson, menuin[i, 80])
  end;

repeat
  assign (temp1, 't1');
  rewrite (temp1);
  repeat
    k := 0;
    i := 1;
    while (menuin[1, 1] in ['*', '@']) do
      begin
        if (menuin[1, 1] = '*') then
          begin
            for j := 1 to 79 do
              write (menuin[i, j]);
            writeln (menuin[1, 80])
          end
        else
          begin
            k := k + 1;
            write ('#');
          end
        i := k + 1;
      end
  until (menuin[1, 1] in ['*', '@'])
for j := 2 to 8 do
  write (menuin[i,j]);
write (topicstat[k]);
for j := 10 to 79 do
  write (menuin[i,j]);
writeln (menuin[i,80])
end;
i := i + 1
end;

writeln:
write ('ENTER THE TOPIC NUMBER OF YOUR CHOICE OR "X" TO EXIT THIS LESSON: ');
readin (lchoice);
if (lchoice in ['1'..VTOPIC,'x','X']) then
  ClearScreen
else
  begin
    ClearScreen;
    writeln ('Sorry, ',lchoice, ' is not a valid response. Please try again.')
  end;
until (lchoice in ['1'..VTOPIC,'x','X']);

if (lchoice in ['1'..VTOPIC]) then
  writeln ('You have chosen topic number ',lchoice, '. Thank you.')
else
  writeln ('OK, I will now return you to the lesson selection menu.');

if (lchoice in ['1'..VTOPIC]) then
  begin
    ShowTopic;
    RecordStats;
    StuRec;
    ClearScreen
  end;
until (lchoice in ['x','X']);
close (temp1);
close (lesson)
end;  (* Procedure StartLesson *)

{**************************************************************************************************}
(* Start of main driver: Program CAI *)
{**************************************************************************************************}

begin  (* Program CAI *)

ClearScreen;
StartEnd('S');
Query:
if (studentcount <= MAXSTUDENTS) then
begin
repeat
begin
Select:
if (choice in ['1'..'VLESSON]) then
StartLesson
end
until (choice in ['x','X']);
StartEnd('E')
end
end.  (* Program CAI *)
Program "STUDENT_STATUS"

**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS ****

Date: 8/1/85
Version: 1.0

Title: Program Student_Status
Filename: STATUS.PAS
Coordinator: Capt Frank W. DeMarco
Project: Masters Thesis
Operating System: MS-DOS
Language: Pascal

User Compile and link with PASCAL.LIB using MS-Pascal compiler and linker.

Contents: Program Student_Status - Main Driver.
Procedure ClearScreen - Clears Z-100 terminal screen.
Procedure QueryUser - Determines the users preferred method of program output (screen or hardcopy).
Procedure Header - Produces the program report header.
Procedure Display - Produces the status report for all students in file "STUDENTS".
Procedure EndScreen - Completes the screen display format for the screen method of program output.

Function: The purpose of this program is to provide a means for training managers, as well as personnel of the CAI Plans Branch (3300 TCHTN at Hansel AFB, to check student progress in the C CAI course.

Date: 8/1/85
Version: 1.0

Name: program Student_Status
Module number: 1.0
Description: Main driver of program
Passed Variables: None
Returns: None
Global Variables Used: studentcount, character, advance, choice
Global Variables Changed: studentcount, character, advance
Files Read: student
Files Written: None
Modules Called: ClearScreen, QueryUser, Header, Display, EndScreen
Calling modules: None

Author: Capt Frank W. DeMarco
History:
1.0 Frank W. DeMarco 8/1/85 - input original code
program Student_Status (input, output);

const
ALINEF1 = '******************************************************************************
ALINEF2 = '******************************************************************************
ADASH = '-------------------------------------------
DASHA~ =-------------------------------------------------------

NUMLESSONS = 6;
NUMTOPICS = 70;

type
iofile = TEXT;

var
infile : iofile;

choice, character : char;
advance, i, studentcount : integer;

(*-------------------------------------------------------------
 * Date: 9/1/85
 * Version: 1.0
 *
 * Name: procedure ClearScreen
 * Module number: 1.1
 * Description: Clears Z-100 terminal screen and sets "no-wrap" on EOL.
 * Passed Variables: None
 * Returns: None
 * Global Variables Used: None
 * Global Variables Changed: None
 * Files Read: None
 * Files Written: None
 * Modules Called: None
 * Calling modules: program Student_Status
 * Author: Capt Frank W. DeMarco
 * History:
 * 1.0 Frank W. DeMarco 9/1/85 - input original code
 *---------------------------------------------------------------*)

procedure ClearScreen:

begin (* Procedure ClearScreen *)

write (chr(27), 'H', chr(27), 'J', chr(27), 'W')

end: (* Procedure ClearScreen *)

(*-------------------------------------------------------------
 * Date: 9/1/85
 * Version: 1.0
 *
 * Name: procedure QueryUser
 *-------------------------------------------------------------*)
procedure QueryUser(var choice: char);

begin (* Procedure QueryUser *)

write ('The output of this program can be put into two (2) different');
writeln ('formats.');
writeln ('If you plan on getting a hard copy; type H .');
writeln ('If you only want a screen display ; type S .');

repeat
writeln;
write ('Enter your choice here ==> ');
readln (choice);
until (choice in ['H','H','S','S']);

if (choice in ['H','H']) then
begin
writeln;
writeln ('Press CP (CONTROL P) and then RETURN to get printout');
readln
end

end; (* Procedure QueryUser *)

(* ********************************************************************** *)

(* Date: 3/1/85 *)
(* Version: 1.0 *)

(* Procedure Header *)
(* Module number: 1.0 *)
(* Description: Produces the program report header. *)
(* Passed Variables: None *)
(* Returns: None *)
(* Global Variables Used: None *)
(* Global Variables Changed: None *)
procedure Header;
begin  (* Procedure Header *)

writeln (ALINEP1,ALINEP2);
write ('* THE FOLLOWING IS THE PRESENT STUDENT STATUS FOR STUDENTS ON');
writeln (' THIS DISK. *');
writeln (ALINEP1,ALINEP2);
write ('* STUDENT | LESSON | LESSON | LESSON | LESSON | *');
writeln ('* LESSON | LESSON *');
write ('* 1D # | 1#1 | 2#2 | 3#3 | 4#4 | *');
writeln ('* 5#5 | 6#6 *');
writeln (ADASH,DASHA);
end:  (* Procedure Header *)

*****************************************************************************

(* Date: 9/1/85
* Version: 1.0
* *
* Name: procedure Display
* Module number: 1.4
* Description: Produces the status report for all students in file
* "STUDENTS".
* Passed Variables: character, studentcount
* Returns: character, studentcount
* Global Variables Used: None
* Global Variables Changed: None
* Files Read: student
* Files Written: None
* Modules Called: None
* Calling modules: program Student_Status
* *
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 9/1/85 - input original code
*****************************************************************************

procedure Display(var char: char;
var studentcount: integer);

type
roll = record

B - 37
studentnumber : packed array [1..11] of char;
studentname : packed array [1..29] of char;
lessons : packed array [1..NUMLESSONS] of char;
topics : packed array [1..NUMTOPICS] of char;
end;

var
pupil : roll;

begin (* Procedure Display *)
    studentcount := studentcount + 1;
    while not (eof(infile)) do
        begin
            for i := 1 to 11 do
                read (infile,pupil.studentnumber[i]);
            for i := 1 to 28 do
                read (infile,pupil.studentname[i]);
            for i := 1 to NUMLESSONS do
                read (infile,pupil.lessons[i]);
            for i := 1 to NUMTOPICS do
                read (infile,pupil.topics[i]);
            end;
            write ('* ');
            for i := 1 to 11 do
                begin
                    if (pupil.studentnumber[i] = '*') then
                        write (' ')
                    else
                        write (pupil.studentnumber[i])
                end;
            write (' '):
            for i := 1 to NUMLESSONS do
                begin
                    if (pupil.lessons[i] = '*') then
                        write (' Passed ')
                    else
                        begin
                            if (i < NUMLESSONS) then
                                write (' ')
                            else
                                write (' ')
                        end;
                    if (i < NUMLESSONS) then
                        write (' ')
                    else
                        writeln (' ');% 
                end;
            if not (eof(infile)) then
                readln (infile);
        end;
if not (eof(infile)) then
    read (infile, character);
end; (* Procedure Display *)

******************************************************************************
* Date: 9/1/85
* Version: 1.0
* *
* Name: procedure EndScreen
* Module number: 1.5
* Description: Completes the screen display format for the screen method of
*   program output.
* Passed Variables: advance
* Returns: None
* Global Variables Used: None
* Global Variables Changed: None
* Files Read: None
* Files Written: None
* Modules Called: ClearScreen
* Calling modules: program Student_Status
* *
* Author: Capt Frank W. DeMarco
* History:
*  1.0 Frank W. DeMarco 9/1/85 - input original code
******************************************************************************

procedure EndScreen(advance: integer);
begin (* Procedure EndScreen *)
  for i := 1 to (advance - 1) do
      begin
          write ('*#
          write ('-
      end;
  writeln (ALINE1,ALINE2);
  for i := 1 to 24 do
      write ('',
  if not (eof(infile)) then
      begin
          write ('Press RETURN to continue.');
          readln;
          ClearScreen;
          Header
      end
  else
      begin
          write ('Press RETURN to end program.');
          readln
      end
end; (* Procedure EndScreen *)
begin (* Program Student_Status *)

ClearScreen;
QueryUser(choice);
ClearScreen;
Header;

assign (infile,'student');
reset (infile);
read (infile,character);

repeat
  studentcount := 0;
  while (character = '>') and (studentcount <= 16) and not (eof(infile)) do
    display(character,studentcount);
  advance := 21 - (studentcount + 6);
  if (choice in ['s','S']) then
    EndScreen(advance);
  until (eof(infile));

  if (choice in ['s','S']) then
    ClearScreen
  else
    writeln (ALINEP1,ALINEP2)

end. (* Program Student_Status *)
Program "CAI STATISTICS"

(**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS ****)

Date: 10/15/85
Version: 1.0

Title: Program CAI_Statistics
Filename: VALIDATE.PAS
Coordinator: Prof. Frank W. DeMarco
Project: Masters Thesis
Operating System: MS-DOS
Language: Pascal

Use: Compile and link with PASCAL.LIB using MS-Pascal compiler and linker.

Contents: PROGRAM CAI_Statistics - Main Driver
Procedure ClearScreen - Clears Z-100 terminal screen.
Procedure QueryUser - Determines the user's preferred method of program output (screen or hardcopy).
Procedure Header - Produces the program report header.
Procedure Init - Initializes array and two linked lists used in the program as well as opening file "STATS".
Procedure ReadStats - Reads file "STATS" into a linked list of frame records as well as builds an array of unique frame identifiers.
Procedure Sort - Sorts the frame identifier array into numeric order.
Procedure Display - Driver for the procedures that display the statistics for each unique question frame.
Procedure FinalScreen - Wraps up the screen display after all statistics have been processed.
Procedure BuildFrameLL - Constructs a linked list of frame records that are of the same frame.
Procedure InitDisplay - Initializes variables used in statistical analysis.
Procedure ShowStats - Analyzes and displays statistical data stored in the linked list of frame records (of the same frame).
Procedure EndScreen - Wraps up the screen display after there has been a full screen displayed.

Function: The purpose of this program is to provide a means for the OPR at Keeler Air Force Base to validate course material and teaching effectiveness by analyzing questions asked during lesson and test course presentation.
program CAL_Statistics (input, output);

const
ALINE1 = '******************************************************************************';
ALINE2 = '******************************************************************************';
ADASH = '------------------------------+----------------------------------';
BASHA = '------------------------------+----------------------------------';
Y_LESSONS = 'Y';

type
iofile = TEXT;
stastics = record
  1frame_num : integer4;
  c_answer : char;
  s_response : char;
end;
stats_array = array [1..150] of integer;
frameclone = ^framerecord;
framerecord = record
  1tf_num : integer4;
  c_ans : char;
  s_ans : char;
  next : frameclone
end;

var
infol : iofile;
stats : stats_array;
temptuff : statistics;
filenode, framehead, node, filenode, frameclone : frameclone;
character, choice : char;
advance, linecount, dindx, tot_r, tot_w : integer;
num_A, num_B, num_C, num_D, num_E, num_Y, num_N : integer;
num_sen, num_right, num_wrong, percent_right, percent_wrong : real;
procedure ClearScreen;
begin  (* Procedure ClearScreen *)
  write (chr(27),'H',chr(27),'J',chr(27),'W');
end;  (* Procedure ClearScreen *)
procedure QueryUser(var choice: char);
begin (* Procedure QueryUser *)
    write ('The output of this program can be put into two (2) different');
    writeln (' formats. ');
    writeln ('If you plan on getting a hard copy; type H . ');
    writeln ('If you only want a screen display; type S . ');
    repeat
        writeln;
        write ('Enter your choice here == > ');
        readln (choice);
        until (choice in ['h', 'H', 's', 'S']);
    if (choice in ['h', 'H']) then
        begin
            writeln;
            writeln ('Press ^P (CONTROL P) and then RETURN to get printout');
            readln
        end;
    end; (* Procedure QueryUser *)

{**********************************************************************
* Date: 10/15/85
* Version: 1.0
* Name: Header
* Module number: 1.3
* Description: Produces the program report header.
* Passed Variables: None
* Returns: None
* Global Variables Used: None
* Global Variables Changed: None
* Files Read: None
* Files Written: None
* Modules Called: None
* Calling modules: program CAI_Statistics, EndScreen
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Author: Cast Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 10/15/85 - input original code
**********************************************************************}

procedure Header;
begin (* Procedure Header *)
    writeln ('ALINEF1,ALINEP2';
    writeln ('THE FOLLOWING IS A STATISTICAL VALIDATION REPORT FOR THE C ');
    writeln ('CAI COURSE. ');}
writeln (ALINEP1,ALINEP2);
write ('* L # : F # : # A : # B : # C : # D : # E : # Y : # N *');
writeln ('* R : # W : % R : % W *');
writeln (ADASH,DASHA)
end;  (* Procedure Header *)

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* Date: 10/15/85
* Version: 1.0
* Name: Init
* Module number: 1.4
* Description: Initializes array and two link lists used in the program as well as opening file "STATS".
* Passed Variables: None
* Returns: None
* Global Variables Used: stats, filehead, filenode, node, framehead, frameno, character
* Global Variables Changed: stats, filehead, filenode, framehead, frameno, character
* Files Read: stats
* Files Written: None
* Modules Called: None
* Calling modules: program CAI_Statistics
* Author: Capt Frank W. Demarco
* History:
* 1.0 Frank W. Demarco 10/15/85 - input original code

******************************************************************************

procedure Init;

var
  i : integer;

begin  (* Procedure Init *)
  for i := 1 to 150 do
    stats[i] := 0;

  filehead := nil:
  new (node);
  filenode := node:
  filehead := filenode;

  framehead := nil:
  new (node);
  frameno := node:
  framehead := frameno;

  assign (infile,'stats');
  reset (infile);
  read (infile,character);
end;  (* Procedure Init *)
procedure ReadStats:

var
  i : integer;
  number : integer4;
  inchar : char;
  found : boolean;

begin  (* Procedure ReadStats *)

  number := 0;
  number := (10 * number) + ((ord(character) - ord('0')));
  for i := 1 to 4 do
    begin
      read (infile,inchar);
      number := (10 * number) + ((ord(inchar) - ord('0')));
      end;

  tempbuff.1tframe_num := number;
  read (infile,tempbuff.e_answer);
  read (infile,tempbuff.e_response);
  i := 1;
  found := false;
  while (stats[1] <> 0) do
    begin
      if (stats[i] = tempbuff.1tframe_num) then
        found := true;
      i := i + 1
      end;
if not (found) then
    stats[i] := tempbuff.ltframe_num;
new (node);
    filenode.next := node;
filenode := node;
    filenode.lt_num := tempbuff.ltframe_num;
    filenode.c_ans := tempbuff.c_answer;
    filenode.s_ans := tempbuff.s_response;
    filenode.next := nil;
if not (eof(infile)) then
    readln (infile);
if not (eof(infile)) then
    read (infile,character);
end; (* Procedure ReadStats *)

/***************************************************************************/
* Date: 10/15/85
* Version: 1.0
* * Name: Sort
* * Module number: 1.6
* * Description: Sorts the frame identifier array into numeric order.
* * Passed Variables: stats
* * Returns: stats
* * Global Variables Used: None
* * Global Variables Changed: None
* * Files Read: None
* * Files Written: None
* * Modules Called: None
* * Calling modules: program CAI_Statistics
* * * Author: Capt Frank W. DeMarco
* * * History:
* * 1.0 Frank W. DeMarco 10/15/85 - input original code
***************************************************************************/

procedure Sort(var stats; stats_array);

var
    temp : integer4;
    sindx,ival,imax : integer;

begin (* Procedure Sort *)
    imax := 0;
    sindx := 1;
    while (stats[sindx] <> 0) do
begin
    imax := imax + 1;
    sindx := sindx + 1
end:
repeat
  sindx := 1;
  for i := 1 to (imax-1) do
    begin
      temp := stats[sindx];
      if (temp > stats[sindx+1]) then
        begin
          stats[sindx] := stats[sindx+1];
          stats[sindx+1] := temp
          end;
      sindx := sindx + 1
    end;
  imax := (sindx-1);
until (imax = 0);
end;  (* Procedure Sort *)

*******************************************************************************
* Date: 10/15/85
* Version: 1.0
*
* Name: Display
* Module number: 1.7
* Description: Driver for the procedures that display the statistics for
*               each unique question frame.
* Passed Variables: None
* Returns: None
* Global Variables Used: filenode, filehead, framenode, framehead
* Global Variables Changed: filenode, framenode
* Files Read: None
* Files Written: None
* Modules Called: BuildFrameLL, InitDisplay, ShowStats
* Calling modules: program CAI_Statistics
*
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 10/15/85 - input original code
*******************************************************************************

procedure Display;

*******************************************************************************
* Date: 10/15/85
* Version: 1.0
*
* Name: BuildFrameLL
* Module number: 1.7.1
* Description: Constructs a linked list of frame records that are of the
*               same frame.
* Passed Variables: None
* Returns: None
* Global Variables Used: filenode, stats, node, filenode
* Global Variables Changed: framenode, filenode

B - 48
procedure BuildFrameLL;
begin (* Procedure BuildFrameLL *)
if (filenode^.ltf_num = stats[dindx]) then
begin
  new (node);
  framenode^.next := node;
  framenode := node;
  framenode^.ltf_num := filenode^.ltf_num;
  framenode^.c_ans := filenode^.c_ans;
  framenode^.s_ans := filenode^.s_ans;
  framenode^.next := nil
end;
filenode := filenode^.next;
end; (* Procedure BuildFrameLL *)

*****************************************************************************
* Date: 10/15/85
* Version: 1.0
* Name: InitDisplay
* Module number: 1.7.2
* Description: Initializes variables used in statistical analysis.
* Passed Variables: None
* Returns: None
* Global Variables Used: advance, num_seen, num_right, num_wrong,
  percent_right, percent_wrong, num_A, num_B, num_C
  num_D, num_E, num_Y, num_N
* Global Variables Changed: advance, num_seen, num_right, num_wrong,
  percent_right, percent_wrong, num_A, num_B, num_C
  num_D, num_E, num_Y, num_N
* Files Read: None
* Files Written: None
* Modules Called: None
* Calling modules: Display
* Author: Capt Frank W. DeMarco
* History:
  1.0 Frank W. DeMarco 10/15/85 - input original code
*****************************************************************************
procedure InitDisplay;

begin (* Procedure InitDisplay *)

advance := 0;
num_seen := 0.0;
num_right := 0.0;
num_wrong := 0.0;
percent_right := 0.0;
percent_wrong := 0.0;
num_A := 0;
num_B := 0;
num_C := 0;
num_D := 0;
num_E := 0;
num_Y := 0;
num_N := 0;

end; (* Procedure InitDisplay *)

procedure ShowStats;

var

lesson_number, frame_number : integer4;

Date: 10/15/915
Version: 1.0

Name: ShowStats
Module number: 1.7.3
Description: Analyzes and displays statistical data stored in the linked list of frame records (of the same frame).
Passed Variables: None
Returns: None

Global Variables Used: framenode, num_right, num_wrong, num_seen, num_A
num_B, num_C, num_D, num_E, num_Y, num_N, tot_r
tot_w, percent_right, percent_wrong, linecount
advance

Global Variables Changed: framenode, num_right, num_wrong, num_seen, num_A
num_B, num_C, num_D, num_E, num_Y, num_N, tot_r
num _W, percent_right, percent_wrong, linecount
advance

Cases Read: None
Files Written: None
Modules Called: EndScreen
Calling modules: Display

Author: Capt Frank W. DeMarco
History:
1.0 Frank W. DeMarco 10/15/915 - input original code

*******************************************************************************
procedure EndScreen(advance: integer);

var
  i : integer;

begin (* Procedure EndScreen *)
  for i := 1 to (advance - 1) do
    begin
      write ('*   |
      writeln ('|   *');
      writeln (ALINEP1,ALINEP2);
      for i := 1 to 26 do
        write (' '); 
      write ('Press RETURN to continue.'); 
      readln;
      ClearScreen;
      Header;
      linecount := 0
    end; (* Procedure EndScreen *)

(* Start of main part of procedure: ShowStats *)

begin (* Procedure ShowStats *)
  while (framencode <> nil) do
    begin

B - 51
lesson_number := (framenode^,ltf_num div 10000);
frame_number := (framenode^,ltf_num div 1000);
frame_number := (frame_number * 1000);
frame_number := (framenode^,ltf_num - frame_number);
if (framenode^,s_ans = framenode^,c_ans) then
  num_right := num_right + 1.0
else
  num_wrong := num_wrong + 1.0;
num_seen := num_seen + 1.0;
case framenode^,s_ans of
  'A' : num_A := num_A + 1;
  'B' : num_B := num_B + 1;
  'C' : num_C := num_C + 1;
  'D' : num_D := num_D + 1;
  'E' : num_E := num_E + 1;
  'Y' : num_Y := num_Y + 1;
  'N' : num_N := num_N + 1;
end;
framenode := framenode^,next
end;

percent_right := ((num_right / num_seen) * 100.0);
percent_wrong := ((num_wrong / num_seen) * 100.0);
tot_r := trunc(num_right);
tot_w := trunc(num_wrong);
if (linecount < 17) or (choice in ['h', 'H']) then
  begin
    write ('# ',',lesson_number:1,' | ',frame_number:3,' | ');
    write (num_A:2,' | ',num_B:2,' | ',num_C:2,' | ',num_D:2,' | ');
    write (num_E:2,' | ',num_Y:2,' | ',num_N:2,' | ');
    write (tot_r:2,' | ',tot_w:2,' | ');
    write (percent_right:5:1,' | ',percent_wrong:5:1,' | ');
    writeln;
    linecount := linecount + 1
  end
else
  begin
    advance := 23 - (linecount + 5);
    EndScreen(advance)
  end;
end: (* Procedure ShowStats *)

(*START OF MAIN PART OF PROCEDURE: Display *)

begin (* Procedure Display *)
filenode := filehead^,next;
framemod := framehead;

repeat
    BuildFrameLL;
    until (framemod = nil);
    InitDisplay;
    framemod := framemod^.next;
    ShowStats;
end;  (* Procedure Display */

(* Name: FinalScreen  *  Module number: 1.8  *
 * Description: Wraps up the screen display after all statistics have been *
 * processed.  *
 * Passed Variables: None  *
 * Returns: None  *
 * Global Variables Used: advance, linecount, dindx  *
 * Global Variables Changed: advance, dindx  *
 * Files Read: None  *
 * Files Written: None  *
 * Modules Called: ClearScreen  *
 * Calling modules: program CAI_Statistics  *
 * Author: Capt Frank W. DeMarco  *
 * History:  *
 * 1.0 Frank W. DeMarco 10/15/85 - input original code  *

procedure FinalScreen;
begin  (* Procedure FinalScreen *)

    advance := 2J + (linecount + 5);
    for dindx := 1 to (advance - 1) do
begin
        write ('* ');
        writeln ('*');
    end;
    writeln (ALINEP1, ALINEP2);
    for dindx := 1 to 26 do
        write ('
');
    write ('Press RETURN to end program.');
    readln;
    ClearScreen
end;  (* Procedure FinalScreen *)

K = 50
COMPUTER ASSISTED INSTRUCTION FOR THE 'C' PROGRAMMING LANGUAGE ON THE ZEN. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB ON SCHOOL OF ENGI. F W DEMARCO UNCLASSIFIED DEC 85 AFIT/GCS/MA/85D-2 F/G 9/2 NL
begin (* Program CAI_Statistics *)

ClearScreen;
QueryUser(choice);
ClearScreen;
Init;

writeln ('One moment please... reading statistical collection file.');
while (character in ['1'..'V_LESSONS']) and not (eof(infile)) do
  ReadStats;
Sort(stats);
Header;

linecount := 0;
dind := 1;
repeat
  Display;
  dind := dind + 1;
  until (stats[dind] = 0);

if (choice in ['s','S']) then
  FinalScreen
else
  writeln (ALINEP1,ALINEP2)
end. (* Program CAI_Statistics *)
Appendix C

Files Used by Program "CAI"

File "INTRO"

```
# WW WW EEEEEEE LL CCCCCC O000000 MMM MMM EEEEEEE
# WW WW WW EE LL CC 00 00 MMM MMM EE
# WW WW WW EEEE LL CC 00 00 MMM MMM EEEE
# WW WWW EE LL CC 00 00 MMM MMM EE
# WWW WWW EEEEEEE LLLLLLL CCCCCC 000000 MMM MMM EEEEEEE

TTTTTTTTTT 00000000
   TT 00 00
   TT 00 00
   TT 00 00
   TT 00000000

""" CCCCCCCCCCCCCCCC """
""" CCCCCCCCCCCCCCCC """
   CCC
   CCC
   CCC
   CCC
   CCCCCCCCCCCCCC
   CCCCCCCCCCCCC
```

THE COURSE YOU ARE ABOUT TO TAKE WAS WRITTEN BY CAPT FRANK DEMARCO
IN PARTIAL FULFILLMENT OF HIS MASTERS DEGREE IN INFORMATION SYSTEMS.

THIS COURSE IS DESIGNED AS AN INTRODUCTORY LEVEL COURSE FOR THE "C"
PROGRAMMING LANGUAGE. THE OBJECTIVE OF THE COURSE IS TO PROVIDE
ENOUGH INFORMATION TO THE STUDENT SO THAT IT MAY BE POSSIBLE FOR
THE STUDENT TO BEGIN USING THE "C" LANGUAGE FOR HIS/HER PROGRAMMING
NEEDS.

THE COURSE, AS IT CURRENTLY EXISTS, CONSISTS OF SIX LESSONS.
File "MENU"

<table>
<thead>
<tr>
<th>STATUS</th>
<th>LESSON #</th>
<th>LESSON TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>1</td>
<td>GETTING STARTED WITH C</td>
</tr>
<tr>
<td>@</td>
<td>2</td>
<td>VARIABLES, CONSTANTS, OPERATORS, EXPRESSIONS</td>
</tr>
<tr>
<td>@</td>
<td>3</td>
<td>PROGRAM CONTROL STATEMENTS</td>
</tr>
<tr>
<td>@</td>
<td>4</td>
<td>POINTERS AND ARRAYS</td>
</tr>
<tr>
<td>@</td>
<td>5</td>
<td>STRUCTURES</td>
</tr>
<tr>
<td>@</td>
<td>6</td>
<td>INPUT AND OUTPUT</td>
</tr>
</tbody>
</table>

NOTE: A "STATUS" OF "+" INDICATES LESSON SUCCESSFULLY COMPLETED.
The Lesson you are about to take contains introductory information on the course and some general information on C programming.

The lesson currently consists of five topics.

The Lesson Breakdown is as follows:

Topic 1: Introduction to C CAI course - This topic gives a short introduction to the overall course structure and some of the particulars used in the course. (Approx. time = 5 min.)

Topic 2: C Program Organization - This topic discusses the overall organization and structure of a typical C program. (Approx. time = 15 min.)

Topic 3: C Program Environment - This topic gives a description of the overall C programming environment covering such items as "compiling", and "linking". (Approx. time = 10 min.)

Lesson Breakdown Continued:

Topic 4: Your First C Program - This topic states a problem to be solved and presents a solution for you to help familiarize you with C program statements. (Approx. time = 10 min.)
Topic 5: Lesson 1 Test - This is the lesson test over items that have
been presented in the previous four lesson topics.
(Approx. time = 5 min.)

TOTAL LESSON TIME IS APPROXIMATELY 45 MINUTES.

I hope that you enjoy it!

SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Introduction to C CAI Course</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>C Program Organization</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>C Program Environment</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Your First C Program</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Test Over Lesson 1</td>
</tr>
</tbody>
</table>

NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.

Frame 100 T  INTRODUCTION TO C CAI COURSE
12 As a first topic subject I will talk a little about the C programming
12 language computer assisted instruction course as a whole.
12 C is considered a low-level general purpose programming language.
12 Its classification as a low-level language does not do it justice
12 though. The language does not provide for, among other things,
12 implicit input or output or for direct file access, but these
12 capabilities can be performed by the use of explicitly called
12 functions (procedures).
12 The C language is a small, straightforward, easy language to learn.
12 Let's take a look at what we will be covering in this course.
138:105
11Frame 105 T
12 This course is broken up into six major subject areas. Each of
these six areas are further broken into small topic areas. The goal in organizing the course in this way is to make it easier to understand as well as speed up the process of subject review.

The following is a lesson breakdown of the course:

<table>
<thead>
<tr>
<th>LESSON #</th>
<th>LESSON TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Getting Started With C</td>
</tr>
<tr>
<td>2</td>
<td>Variables, Constants, Operators, Expressions</td>
</tr>
<tr>
<td>3</td>
<td>Program Control Statements</td>
</tr>
<tr>
<td>4</td>
<td>Pointers and Arrays</td>
</tr>
<tr>
<td>5</td>
<td>Structures</td>
</tr>
<tr>
<td>6</td>
<td>Input and Output</td>
</tr>
</tbody>
</table>

Let's see if you have been paying attention. How many lessons did I say are in this course?

A: Four
B: Five
C: Six
D: Seven

Very good! You are paying attention.

No. The correct answer is Six lessons ("C").

I'm sorry, "E" was not one of your choices.

What you just saw was an example of one of three types of questions I can ask during the presentation of this course. The other types are True/False and Yes/No questions. The responses that I can recognize are as follows:

<table>
<thead>
<tr>
<th>Question Type</th>
<th>Valid Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Choice</td>
<td>A, B, C, D, E</td>
</tr>
<tr>
<td>True or False</td>
<td>True, False, T, F</td>
</tr>
<tr>
<td>Yes or No</td>
<td>Yes, No, Y, N</td>
</tr>
</tbody>
</table>

Note: For True/False or Yes/No questions I will only look at the first letter of your response, so to save time it's best to enter only T, F, Y, or N. (Answers may be in lower case.)
Let's give it a try.

This is an example of a True/False type question. (True or False)

You are absolutely correct.

Wrong! Are you yanking my electrons?

As you will no doubt notice, there is a test at the end of each lesson. In order for you to receive credit for taking this course you must successfully pass each of these tests.

There is no set lesson order in taking this course, nor is there a requirement to view each topic before taking a lesson test. It is suggested that you do take the course in the order established for reasons of material continuity and in order to enhance understanding.

It is very important that you do not interrupt the CAI program once it has been started. Your progress is only recorded at the end of each lesson topic. Please exit the program by answering "X" at the topic selection menu and the lesson selection menu.

You can check your lesson and topic progress at the selection menus by observing the "status" column displayed.

While taking this course you can be an invaluable aid in making it better by taking note of errors in the course material. If you should notice an error or believe something to be in error, just make a note of where the error appears.

To make this task easier, I display for you all the necessary information. Just record the lesson number, topic number, and of course the frame number of the frame where the error appears. Recording only the frame number will be of little help since each lesson could have a frame with the same number.

I just have one more thing to mention to you before I return you to the topic selection menu.

A word about the lesson tests.

The last lesson topic for each lesson is a test over the material covered in the lesson. As I mentioned before, these tests must be passed in order to receive credit for taking this course.

When you take a test, you will be given information during the test which will help you in locating the material that gave you problems on the test. The way in which this is done is by reference to the
This topic will discuss the overall organization of a typical C program. For ease of understanding, I will restrict the discussion to a program that is contained in one source file.

The organization of the program file would look like the following:

1. Preprocessor Statements Section
2. Global Variable Declarations Section
3. Function(s)
4. Main Program Driver

Each of the above will be discussed in this topic section.

Through the use of a preprocessor, the C compiler has the capability of: file inclusion, token substitution, and conditional compilation.

Preprocessor statement lines are defined in the C program by the use of a # as the first character on a line. These lines may appear anywhere in the program, but it is a good programming practice to place them at the beginning.

We'll take a quick look at each of these preprocessor capabilities.

Which one of the following is "not" a capability of the C preprocessor?

A Conditional Compilation
B Token Substitution
C Function Definition
D File Inclusion

Very good, you're so right.
The preprocessor control line of the form: 
#include "filename"
will include the contents of the file specified in the source
program file. (Note: The shown quotation marks are needed.)
In addition, a control line of the form: 
#include <filename>
will include the contents of the “system” file specified.
For example: 
#include <stdio.h> is the usual statement for
including the file that contains the standard I/O functions
for use with C. We will see more of this later.
One more thing: An included file may also have files included.
This of course should be done cautiously to avoid confusion.

#include file.dat is a valid C preprocessor "file inclusion" statement.
(True or False)
That’s right. You need to have quotation marks around the file name.
Wrong. Quotation marks are needed around the file name.
The correct form of the statement is 
#include "file.dat".

The preprocessor control line of the form:
#define token-name token-replacement
will substitute the value of the token-replacement for each occurrence
of the token-name throughout the program.
For example: If you have a value that might change with time, such
as a mortgage rate, you could use the #define to make future
program changes easier like this ==> 
#define interest .11
It is easy to see that this capability can be a real time saver. Not
only will it make future program changes easier but it will, with wise
token-name choices, produce an easier to read and maintain program.
Note: #undef token-name is used to cancel the token-replacement.
Which of the following is a valid C preprocessor "token substitution" statement?

A: `#define pay_grade = 11`

B: `#declare pay_grade 11`

C: `#define pay_grade 11`

D: `#declare pay_grade = 11`

Correct. Keep up the good work.

No. There is no "=" in the valid form of the statement. Answer "C" is the correct response.

No. I think you missed something. Let's look at that again.

I'm sorry, "E" was not one of your choices.

You can cause the compiler to skip sections of your source code by using the conditional preprocessor control statements of: 
- `#if` constant-expression will evaluate to "true" if the constant-expression is a non-zero value.
- `#ifdef` identifier will evaluate to "true" if the identifier had previously been defined using the `#define`.
- `#ifndef` identifier will evaluate to "true" if the identifier had not been previously defined using the `#define`.

Following the above statements would be statements that you would want to be executed based on the outcome of the statement test.

The statement `#else` would be used to identify an alternate section of code to be executed if the outcome of the `#if`.. `#else` statement test evaluates to false.

The statement `#endif` is used to terminate an `#if`.. `#else` structure.

Example: `#ifdef employed /* check to see if "employed" #define(d) */
{
   executable statements;
}
#else
   /* else "employed" is not #define(d) */
alternate executable statements;


to use the same variable in different portions of your program,
then it may be desirable to declare the variable as being global.

When you declare a global variable, remember that its "scope" is
only those functions (procedures) that physically follow it in the
program. (Note: An exception to this involves the "extern" decla-
ration statement which I will cover later.)

Let's look at an example...

#include <stdio.h>
int sum; /* Global Variable "sum" */
main() {
    sum = 100;
    add();
    add();
    printf("%d",sum);
}
add() {
    sum = sum + 100;
}

This program would print out the value 300.

Don't worry if you don't follow everything in this example. You will.

The "scope" of a global variable refers to ...
the extent the global variable is used in the program.
the area preceding the global variable declaration.
the area where a declared global variable is recognized.
the mouthwash of the global variable.
Right.
ABC No. The correct answer is "D".
B:365
E I'm sorry, variables don't have mouths.
B:360

Following global variable declarations in our typical program example
is the area where we define our functions. The structure of a function
looks like this:

return-type function-name(arguments, if any)
argument declarations, if any
{
  declarations
  executable statements
  return statement, if any
}

A function has certain required parts. Here's an example of a function
that fills the requirement:

function1() {}

The previous example of a function is an example of a dummy function.
The function doesn't actually do anything, but does qualify as a
function.

Let's look at each of the parts of the function structure.

The "return-type" in front of the function name identifies the type
of result the function will return to the function that called it.
If this return-type is not explicitly named, then it defaults to an
integer type. If a function returns a value other than an integer
and it is physically located after the calling function then it is
also necessary to declare the "function" as that return type in the
calling function.
For example: float numval, function1

The above statement would be in the declaration section of the function that calls function1.

The function declaration: float function1(numval)

would be used to identify the called function "function1" and state that the value to be returned by this function is of type float.

Note: We will discuss "float" later.

That brings us to the "function-name" part of the function structure.

The function-name can be of any length but must start with a letter.

Note: The character _ (underscore) is considered a letter in C.

The function-name can consist of any combination of letters and digits as long as it starts with a letter and does not spell a C keyword.

Note: Keywords will be discussed later.

Following the function-name is a required set of parentheses (). Inside the parentheses is where the list of passed arguments goes. Each argument is separated by a comma and appears in the order in which the calling function lists them in its calling statement.

Next in our function structure is the area for argument declaration. This is the area where we identify the "types" of the passed arguments.

For example: float function1(x,y)

float x, n; <=> argument declaration

Which of the following is required to follow the function-name in a function declaration statement?

A. " "
B. +
C. / /
D. # #

Very good.
If an argument is not explicitly declared it defaults to type integer.

Following the argument declarations is a required set of braces (). Inside the braces is where the function's declarations, executable statements, and return statement goes. Each statement in this area, as in the argument declaration area, is terminated by the use of a semi-colon.

We'll look at each of the three areas between the braces.

If an argument passed in to a function is not explicitly declared its "type" defaults to an integer. (True or False)

You are absolutely correct.

No. That statement is correct.

Here are the three areas between the () braces.

1. A function will usually have a need to have local parameters and variables defined in order to do its job in the program. The function's declaration section within the braces is where these declarations take place.

2. Following the local declarations is the function's executable statements. These are the statements to be executed by the function prior to returning control to the calling function.

3. The "return" statement is where you identify the variable that is to be passed back to the calling function. The "return" statement can be a bit confusing. There are three forms in which the statement can appear.

The most common form of "return" is: return(expression);

The "expression" can be any valid expression, such as value 2 or just value. In either case the final value will be passed back to the calling function as the value of the function-name. Remember, it is an integer unless explicitly declared otherwise.

Another form of the "return" is: return expression;

The elimination of the parentheses also eliminates the confusion of whether or not "return" is a function (which it isn't).
The last valid form of "return" is: return;
This case has the same effect as leaving out the return statement.
In either case, no value is returned to the calling function and only global variables used by the function would be changed as a result of the called function being executed.

Important Note: **Do Not Use** return(); as this will cause a compile error since "return" is not a function.

We only have one short area left to cover. But first a question.

Which of the following is "not" a valid return statement?

A return(expression);
B return expression;
C+ return();
D return;

Very good!

Wrong. That is a valid return statement. "C" is the invalid one.

"E" was not a given choice. Please try again.

*** Main Program Driver ***
This area of the program is usually located at the end of the source program file. It is the required function that starts and ends the program's execution. There must be a function by the name of main() somewhere in your C program.

The organization of the function "main()" is the same as for the functions we just covered. I bet that makes you happy!

Well that about does it for this topic. Let's take one more look at the overall construction of a typical C program before returning to the topic menu.

*** Review of C Program Organization ***

C - 14
The organization of the program file would look like the following:

1. Preprocessor Statements Section
2. Global Variable Declarations Section
3. Function(s)
4. Main Program Driver

*** This concludes this topic area. ***

END

This topic will discuss the overall C programming environment.

We will follow the complete process of creating a C program from the writing of code to the execution of the resultant executable program.

Although this may sound like a lot to cover, it really isn't.

To get us started let's take a look at the process as a whole.

The following is an outline of the steps we will cover:

1. Create Source File
2. Compile Source File
3. Error Correction
4. Link Object Code Files
5. Run Executable Code File

Let's get started ...

The most important aspect of computer programming in any language is the ability to put your thoughts into computer code. Many experienced programmers feel that the best way to write clear, concise, effective code is to write in plain English "what" it is that needs to be done.

Once the "what" has been identified you can start working on the "how" do I do it question. This brings us to a controversial topic, that of where can I do my best program development? Do I do it on paper, or do I sit at a computer terminal and "create" as I go. Well it all depends on who you talk to as to which way is better, but the person who wrote the program your using now prefers to "create" his programs at the computer terminal. Of course it is not always up to you where you do your programming. Computer time costs money after all, and you and/or your boss should be concerned about such factors.
Whichever way you finally decide to do it, you are going to need a way to put the code you have written into a source file for use on the computer. This calls for the use of a text editor. The more familiar you are with the text editor the easier and faster you can input your code into a source file. Remember, chances are you will have to make error corrections or update your program at some point.

So, learn your text editor and use it often.

Once you have created your C program source file using a text editor, it is time to compile it.

The C compiler is actually a program that performs three basic functions using three distinct programs.

1. The C Preprocessor
2. The C Compiler
3. The C Assembler

The C compiler is actually three programs in one. (True or False).

That's right. A preprocessor, compiler, and assembler all in one.

No. Are you sure you read that last frame? Let's see it again.

First, the C "preprocessor" scans the source code for preprocessor statements (# statements) and preforms all indicated actions.

Second, the C "compiler" translates the C language statements into computer assembly language statements.

Last, the C "assembler" translates the assembly language statements into the object code and places it in an object file.
This last step occurs IF you have not made any C syntax errors!

When you compile your C program, it is possible that you may have made one or two syntax errors. Don't feel bad, it can happen to even the best programmer (once in a while). If this unfortunate occurrence takes place, you can rest assured that the C compiler will let you know.

The C compiler will report any syntax errors that it encounters while compiling your source code. In order to achieve the goal of syntax error free object code, it may be necessary to go through several iterations of "compile & correct".

This process will require changes to your source file, which is a reason why you should know how to use your text editor program inside and out.

Once you have successfully produced an "object code" file, it is time to move on to creating an executable program file.

The C "linker" is a program that is used to link together object files into an executable "machine code" file. The C linker will take all specified object files as well as any needed C library functions and create for you one executable program file.

This feature allows for the creation of user functions that can be used in a variety of programs by merely linking them into the new program. These functions can then be called by the program when needed. This will save you many hours of redundant work.

What now? You ask. Well you'll see, but first a question.

Which one of the following programs will create an executable program file from one or more object files?

A Compiler
B Chainer
C C Linker
D Preprocessor
E Assembler
Correct.

Correct.
Wrong. The "Linker" creates the "executable" machine code program file.

Now that you have the executable program file you can sit back and start the seemingly long process of "logic" testing your program.

That's right! It's run time!

At this point, all syntax errors have been corrected and you have successfully created an executable program file. Now you can test your program to your heart's content and make all those changes and/or enhancements to your pride-and-joy (your C program).

Before I return you to the topic selection menu, I would like to give you a picture of the process described in this topic area.

I made no mention of "how" to do the steps only "what" steps needed to be done. The commands to "compile" and "link" differ from system to system, but are similar enough to show an example.

The following is an example of the steps needed to create and run an executable program file.

1. Create Source File ===> proname.c (using a text editor)
2. Compile Source File ===> cc proname.c (using C Compiler)
3. Error Correction ===> (As needed) (using a text editor)
4. Link Object Code Files ===> clink proname (using C Linker)
5. Run Executable Code File ===> proname (type program name)
6. Refine Program Execution ===> (As needed) (using a text editor)

This topic will develop an actual working C program for you to examine.

In order to provide a problem solving structure, here is an outline of the steps that I will be discussing:

1. Problem Definition
2. English Language Problem Solution
3. English Language - C Language Translation

C - 18
42 4. C Language Problem Solution
43B:705
41Frame 705 T
42 *** Problem Definition ***
42
42 The first thing that needs to be done is to define the problem.
42
42 The program I want to develop is one that will:
42
42 1. Take a one line input from the keyboard, and
42
42 2. Display the input line "centered" on the terminal screen.
42
42 I will show one way to accomplish this and then allow you to choose
42 whether you want to view an alternate solution.
43B:710
41Frame 710 T
42 *** English Language Problem Solution ***
42
42 After thinking out the problem, I arrived at the following five step
42 solution:
42
42 1. Define and initialize the storage area for one line of input.
42
42 2. Prompt user for one line of input.
42
42 3. Read in the one line of input keeping track of number of
42 characters read.
42
42 4. Calculate number of spaces to precede line for "centered" output.
42
42 5. Print out the input line "centered" on screen.
43B:715
41Frame 715 T
42 *** English Language - C Language Translation ***
42
42 Changing the english problem solution into C language statements we get:
42
42 1. #define CHARIN 80
42 char input_line[CHARIN];
42 for (i=0; i < CHARIN; i++)
42 input_line[i] = ' ';
42 2. printf("Please enter one (1) line of text to be centered.\n");
42 3. while ((c = getchar()) != ' \n') {
42 4. advance = (40 - (i / 2));
42 5. for (ival=0; ival < advance; ival++)
42 6. putchar(' ');
42 printf("%s", input_line);
### C Language Problem Solution

Now that we have the C statements needed for problem solution, all we need to do is declare the variables we used and put the code into a function called "main".

The following topic frame gives one complete solution.

```c
#define CHARIN 80

main() {
    char c, input_line[CHARIN];
    int i, ival, advance;
    for (i=0; i < CHARIN; i++)
        input_line[i] = ' ';  
    printf("Please enter one (1) line of text to be centered.\n");
    i = 0;
    while ((c = getchar()) != '\n') {
        input_line[i] = c;
        i++;
    }
    advance = (40 - (i / 2));
    for (ival=0; ival < advance; ival++)
        putchar(' ');
    printf("%s", input_line);  
}
```

At this stage of the course, I don’t feel that I should take the time to explain each C statement used in this example program. Please rest assured that I plan to explain all the statements used here as well as a multitude of others later in this course.

The example solution is by no means the only solution to the stated problem, it is only one of many "correct" solutions. It also is not a fool proof solution (Input of > 80 characters is not checked for.).

The rest of this topic area contains an alternate solution to this same problem. Other C statements are shown, but again no explanation is given.

Do you wish to see another example solution? (Yes or No)

```c
/* Do you wish to see another example */
```

Great! Let's take a look at one.

Alright. I will honor your decision.
In the solution we just finished with, the whole solution was contained in the "main" function. This practice is not a good one to get into the habit of. A better way to solve programming problems is to break the problem solution into small "modules" or, in the case of C, functions.

Earlier I identified several steps to be accomplished in order to solve the example problem. Each main step could be done by its own separate function or we could combine two or more steps into one function. Let's see what we end up with if we use this latter approach.

The first step was "Define and initialize the storage area for one line of input."

This of course can be accomplished using a global variable called "CHARIN" having value 80, an array declaration in function "main" of the form: char input_line[CHARIN], and a statement that "blanks" out the array using a "for" statement for control.

These three statements look like the following:

(1)  (2)  (3)

#define CHARIN 80  char input_line[CHARIN];  for (i=0; i < 80; i++)
               input_line[i] = ' ';

The second step of "Prompt user for one line of input" and the third step of "Read in the one line of input keeping track of number of characters read" can be combined into one.

For this we can define a function called "task1" that would look like:

task1(input_line, i)

char input_line[];
int i,
{  char c;
   printf("Enter one line of text to be centered.\n");
   while ((c = getchar()) != ' \n')
      {  input_line[i] = c;
         i++;
      }
   return(i);
}
The fourth step of "Calculate number of spaces to precede line for 'centered' output" and the fifth step of "Print out the input line 'centered' on screen" can be combined into one.

For this we can define a function called "task2" that would look like:

```c
int advance, ival;
advance = (40 - (i / 2));
for (ival=0; ival < advance; ival++)
  putchar(' ');
printf("%s",input_line);
```

That just leaves one function to write. That being function "main".

I will show two different "main" functions that do the same thing.

```c
main()
{
  char input_line[CHARIN];
  int i;
  for (i=0; i < CHARIN; i++)
    input_line[i] = ' ';
  i = 0;
i = task1(input_line, i);
task2(input_line, i);
}
```

### A Variation To The Problem ###

Now that we have solved the example problem two different ways, how about making a slight improvement to it. What if we wanted to clear the screen before we displayed the "centered" line on the screen?

Well, this can be done fairly easily with the following function:

```c
task3()
{
  putchar('\033');
  putchar('H');
  putchar('\033');
  putchar('J');
}
```

Now all we need to do is call "task3" from "task2" before the loop that does the "putchar(' ')".

C - 22
This is what the "task2" function would then look like:

```c
int task2(char input_line[], int i)
{
    int advance, ival;

    advance = (40 - (i / 2));
    task3();
    for (ival=0; ival < advance; ival++)
        putchar(' ');
    printf("%s", input_line); }
```

Of course, "task3" would be located ahead of "task2" in the program source file.

Let's see what our program source file would look like if we use this alternate program solution with the "clear screen" function.

```c
#define CHARIN 80

++-> task3()
;
++++-----> +---- task2(input_line, i)
;
; +++++-----> task1(input_line, i)
;
; ++++++------ main

The way to read this is: "main" calls "task1", then "main" calls "task2", then "task2" calls "task3".
```

Well, that about does it for lesson number one. If you have seen the four subject topics in this lesson, you should now be ready to take the final test. If you feel that you don't understand something well enough to pass the test, please retake the topic that is giving you problems.

Topic 1 gave an introduction to the C CAI course structure.
Topic 2 gave a description of the C program organization.
Topic 3 gave a description of the C program environment.
Topic 4 presented a programming example for your inspection.
Welcome to the final test of lesson one. This test consists of ten questions over material presented in the previous four topic areas.

In order to successfully complete this lesson, you must achieve a minimum score of 70% (seven out of ten questions correct).

If you miss a question, the correct answer will not be shown. It is up to you to research the correct answer.

Well, enough said. Let's get on with it. Good luck!

After answering a test question in this course, a reference is shown to you so that you can find the place in the lesson where the question originated. The reference is in the format of (#,@) where ...

# = lesson number and @ = frame number

# = lesson topic number and @ = topic frame number

# = lesson topic number and @ = lesson line number

# = lesson number and @ = lesson topic number

Right. (1,135)
45B:910
51Frame 910 QM
521. After answering a test question in this course, a reference is shown to you so that you can find the place in the lesson where the question originated. The reference is in the format of (#,@) where ...

52A # = lesson number and @ = frame number

52B # = lesson topic number and @ = topic frame number

52C # = lesson topic number and @ = lesson line number

52D # = lesson number and @ = lesson topic number

Right. (1,135)
54B:910

Wrong. (1,135)
55B:910

"E" was not one of your choices.

55B:905
51Frame 910 QP
522. The three capabilities of the C preprocessor are: file inclusion, token substitution, and conditional compilation. (True or False)

Right. (2,305)
54B:915

Wrong. (2,305)
55B:915
51Frame 915 QM
523. Which of the following is a valid C preprocessor "token substitution" statement?

Right. (2,315)
54B:920

C - 24
Which of the following is "not" a valid C preprocessor "conditional compilation" statement?

A) #if
B) ifdef
C) #else
D) #for
E) ifndef

Right. (2,320)

The function name in a function declaration can consist of any combination of letter, digits, or characters on the keyboard.
(True or False)

Right. (2,355)

Which of the following is the required function in a C program that usually starts and ends execution of the program?

A) start()
B) begin()
C) main()
D) driver()

Right. (2,320)

Which of the following is a list of the three programs contained in the C compiler?

A) Preprocessor, Compiler, Linker
B) Preprocessor, Compiler, and Assembler

C - 25
53C. Compiler, Assembler, and Linker
53D. Editor, Preprocessor, Assembler
54 Right. (3,515)
54 B:940
55ACD Wrong. (3,515)
55 B:940
55E "E" was not one of your choices.
55 B:935
51Frame 940 OP
528. The C "Linker" is a program that is used to link together one or more
52object files into an executable "machine code" file. (True or False)
53Y
54 Right. (3,530)
54 B:945
55 Wrong. (3,530)
55 B:945
51Frame 945 OP
529. Which of the following is a "Compiler" program that translates the
52C language statements into assembly language statements?
53A. Editor
53B. Preprocessor
53C. Compiler
53D. Assembler
53E. Linker
54 Right. (3,520)
54 B:950
55ABDE Wrong. (3,520)
55 B:950
51Frame 950 OP
5210. What is the first thing that needs to be done when solving a computer
52programming problem?
53A. Define the problem to be solved.
53B. Write the "english" language solution.
53C. Do an "english" to "C" language translation.
53D. Write the "C" language solution.
54 Right. (4,700)
54 B:955
55BCD Wrong. (4,700)
55 B:955
55E "E" was not one of your choices.
55 B:950
This marks the end of lesson number one. I hope that it was of some benefit to you. I am looking forward to seeing you in lesson number two. I hope that you didn't have too much trouble with the material presented in this lesson. If you did, please voice your comments to your training monitor who will in turn contact the CAI Plans Branch at Keesler AFB, MS.

Well, let's take a look at how you did with the test ...
THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON VARIABLES, CONSTANTS, OPERATORS, AND EXPRESSIONS USED IN C PROGRAMMING.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Variables & Constants I - This topic is the first of two that covers the declaration and use of variables and constants in C programming. (Approx. time = 10 min.)

Topic 2: Variables & Constants II - This topic is the second of two that covers the declaration and use of variables and constants in C programming. (Approx. time = 5 min.)

Topic 3: Operators & Expressions I - This topic is the first of two that covers the use of the different operators and expressions in C programming. (Approx. time = 15 min.)

Lesson Breakdown Continued:

Topic 4: Operators & Expressions II - This topic is the second of two that covers the use of the different operators and expressions in C programming. (Approx. time = 10 min.)
Lesson 2 Test - This is the lesson test over items that have been presented in the previous four lesson topics.
(Approx. time = 5 min.)

TOTAL LESSON TIME IS APPROXIMATELY 45 MINUTES.

I hope that you enjoy it!

**SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:**

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Variables &amp; Constants I</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Variables &amp; Constants II</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Operators &amp; Expressions I</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Operators &amp; Expressions II</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Test Over Lesson 2</td>
</tr>
</tbody>
</table>

**NOTE:** A "STATUS" OF "*" INDICATES TOPIC SUCCESSFULLY COMPLETED.

**Frame 100 T VARIABLES & CONSTANTS I**

*** Data Types ***

In C there are four sets of basic data types that can be used. These four are: Character, Integer, Floating point, and Double-precision floating point.

We will cover the character and integer data types in this topic area, and leave floating point and double-precision floating point for the next topic area.

I will be discussing the declaration and use of both variables and constants within the context of data type usage.

The flow of this topic area will follow the following outline:

1. Character Constants
2. Character Variables
3. Integer Constants
4. Integer Variables
Before we get too far into this area, we need to set up some rules for naming any variables that we use in our programming.

1. Variable names must begin with a letter.
2. Variable names are composed of letters and digits.
3. Variable names must not be C keywords.

In C, a "letter" is any character in the set (a..z,A..Z,_), that's all lower and upper case letters as well as the "underline" character. A "digit" is any character in the set (0..9). A "keyword" is any word in the set:
(auto, break, case, char, continue, default, do, double, else, entry, extern, float, for, goto, if, int, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, while)

A few additional facts need to be mentioned about variable names.

1. Upper and lower case names are different. This means that the variable names: answer, Answer, and ANSWER are all different variable names.
2. Only the first eight characters of a variable name are significant. This means that insert_A1 and insert_A2 are the same variable name.
3. The number of significant characters may be less than eight for external variables and function names (system dependent).

Which of the following is "not" a valid variable name?

A. X123
B. first_num
C. 2nd_in_line
D. _OUT_

Very good!

ABD No. The correct answer is "C". Variable names must start with a letter.

E I'm sorry, "E" was not one of your choices.

C - 30
A character constant is symbolized as a single character enclosed within single quotation marks.

For example: 'a'

The value of a character constant is actually the numeric equivalent of the character as defined by the computer system's character set. Thus, arithmetic operations using characters is possible but the most common use for character constants is for comparative purposes.

All this may seem confusing, but it really isn't. We will look at an example of the usage of character constants after we take a look at character variables.

**Character Variables**

A character variable is declared by the use of the keyword: char

For example: char in_char;

The character variable "in_char" will now be assigned a one byte storage location in the computer's memory. The value that will be stored in this location depends on the usage of the variable in the program. Let's look at a couple examples that should help you understand both character variables as well as character constants.

The statement: in_char = 'a'; assigns the ASCII value 97 (decimal) to the character variable location identified by "in_char" in memory.

Note: ASCII values range from 0 thru 127 (decimal) and can be found in most good programming books.

**Character Variables Continued**

Every character on the keyboard has an ASCII numeric equivalent. (By the way, ASCII stands for American Standard Code for Information Interchange.) There are, however, several characters that are hidden. These characters can be represented by character constants by using character escape sequences that start with a backslash (\).

Some of the more common character escape sequences follow:

`\b` (backspace)
`\n` (new line)
`\f` (form feed)
`\r` (carriage return)
`\\` (backslash)
`\'` (single quotation)
`\###` (### = an octal value)
An example of how you would declare a character variable using one of the special character escape sequences as a character constant is as follows:

```c
char back_space = '\b';
```

This statement assigns the ASCII value 8 (decimal) to the character variable location identified by "back_space" in memory.

An equivalent way to declare the variable "back_space" is as follows:

```c
char back_space = '\010'; OR char back_space = '\10';
```

In both statements, the character variable location "back_space" is assigned the value 10 (octal) which is equivalent to 8 (decimal).

In the statement: `char input_char = 't';`

`input_char` is called a character variable and 't' is called a character constant. (True or False)

That's right.

That is a true statement.

Which one of the following characters is used to identify a special character escape sequence?

A. $  
B. @  
C. &  
D. \  
E. #  

Very good, you're so right.

No. Answer "D" is the correct answer.

So far in this topic area you have seen rules over selecting variable names, a description and examples of character constants, a description of character variables, and a description and examples of...
how to declare special characters.
In the remainder of this topic area we will look at integer constants and integer variables.
So let's get to it ...

### Integer Size ###
The first thing we need to cover when talking about integers is the size of a number that can be used. In C we can normally use integers in the range: -32,768 thru +32,767
If it is necessary to use a number outside this range, C provides a way to accomplish this.
The use of an "unsigned" integer provides for use of numbers in the range: 0 thru 65,535
The use of a "long" integer will provide for use of numbers in the range: -2,000,000,000 thru +2,000,000,000
The way to identify which size you are using will be explained.

### Integer Constants ###
An integer constant can be expressed in one of three ways. It can be decimal, octal, or hexadecimal. Also, each of these can be either a "short" or "long" integer.
Decimal integer constants are represented by such numbers as: 238, 45920, and -72. Note that embedded commas are not used. 45,920 would be wrong.
Octal integer constants are represented by such numbers as: 089, 0150, and 014. Note that "octal" numbers all have a leading "zero".
Hexadecimal integer constants are represented by such numbers as: 0x8f, 0x2a, and 0x7b. Note that lower case and upper case can be used and "hexadecimal" numbers all have a leading "zero x".

### Integer Constants Continued ###
As I mentioned, integer constants can also be either a "short" or "long" integer. An integer will be stored as a "short" integer unless you indicate otherwise. There is, of course, exceptions to the rule. For example, if you specify an integer that is larger than 32767, then it will be stored as a long integer.
The way to indicate that an integer is to be stored as a "long" integer is to follow the number with the letter "L". Here are a few examples:

Decimal: 5987L, and 367L
Octal: 04689L, and 0824L
Hexadecimal: 0X2A5F4L, and 0x6FDAL

Note: A lowercase letter "l" may be used, but may be very confusing.

Which of the following best describes the integer constant 073564L?

A. Decimal
B. Long Decimal
C. Octal
D. Long Octal
E. Hexadecimal

Right.

No. The "0" (zero) in front makes it an "octal" and the "L" makes it a "long" integer. I think you need to review this material.

No. The "0" (zero) in front makes it an "octal" number.

No. The "L" after the number makes it a "long" octal number.

*** Integer Variables ***

An integer variable is declared by the use of the keyword: int

For example: int index;

The integer variable index will now be assigned a 16 bit storage location in the computer's memory.

Let's look at a couple examples that should help you understand both integer variables as well as integer constants.

The statement: number_in = 212; assigns the integer constant value 212 (decimal) to the integer variable location identified by "number_in" in memory.
The way to indicate which of these an integer variable will be is by using the keywords short, long, or unsigned. Here are some examples:

- short int index_1; or Just short index_1;
- long int index_2; or Just long index_2;
- unsigned int index_3; or Just unsigned index_3;

Note: When using these keywords, use of "int" is optional.

When declaring an integer variable, you only have the option of specifying the variable as being either "short" or "long". (True or False)

That's right. You can also specify it as being "unsigned".

Wrong. You can also specify it as being "unsigned".

One last word on integers.

Although C has the capability of specifying different size storage locations, this capability is limited by the specific compiler and system you are using. Please check to see if your compiler and system treat integers as described here.

In this topic area we have looked at a description and examples of character constants, character variables, integer constants, and integer variables. Also we covered variable names and special characters.

In the next topic area we will continue to discuss constants and variables by looking at floating point and double-precision floating point data types.

This concludes this topic area.
so we will cover floating point and double-precision floating point in this topic area.

I will be discussing the declaration and use of both variables and constants within the context of data type usage.

The flow of this topic area will follow the following outline:

1. Floating Point and Double-precision Floating Point Constants
2. Floating Point Double-precision Floating Point Variables

Floating point numbers are just numbers that have two parts instead of one, as in the case of an integer. You can think of a floating point number as having an integer, or whole part, and a fractional part.

These two parts are separated by a decimal point.

Examples of floating point numbers: 67.32, 2583.1, and 2.4592

How "precise" a number is has an effect on calculations performed using a stored number. Thus, the precision of a number may be very important within your program. C stores all floating point constants as double precision. This means that a large number of significant digits are stored to represent the number and hence, gives better precision in any calculations performed involving the number.

Another way of representing floating point numbers is through the use of "scientific notation". The following are examples of the use of scientific notation for floating point constants:

4.6E3 <or> 4.6e3 = 4670.0
.9834E2 = 98.34
345.0e6 = 345000000.0
-2.8473E5 = -284730.0

4.67E-3 <or> 4.67e-3 = .00467
.9834E-2 = .009834
345.0e-6 = .000345
-2.8473E-5 = -.000028473

Note: The "E" can be upper or lower case ("e").
In C, floating point variables are declared using the keyword "float", and double precision floating point numbers are declared using the keyword "double".

Here are some examples:

```c
float var_1;  // double var_1;
float var_2, var_3;  // double var_2, var_3;
```

The following illustrates the use of floating point constants and variables.

```c
float var_1 = 451.29  // float var_1 = 4.5129E2
double var_2 = 23975.5619  // double var_2 = 2.397535619e4
```

To reiterate, the use of "double" allows for the storing of a greater number of significant digits to represent a given number. Thus, more precision is gained in calculations involving the number.

Another way of achieving the precision of a double precision variable is with the keyword "long". The following two statements have the same effect:

```c
double var_1;  // >>>>>> OR >>>>>>> long float var_1;
```

In the statement `double var_one = 419.9253;` the keyword "double" is used to indicate that variable "var_one" is to be stored as a "double precision floating point" number. (True or False)

Very good.

That statement is true.
In this topic area we have looked at a description and examples of floating point and double precision floating point constants, and floating point and double precision floating point variables.

In the next topic area we will begin a discussion of operators and expressions and their use in C.

*** This concludes this topic area. ***
after a division operation. For example, in the statement:

Answer = 15 % 2;  the value stored in "Answer" would be 1, since 15
divided by 2 is 7 with a remainder of 1. Likewise:

Result = 150 % 15;  produces a value of 0 in "Result", since 15 divides
150 evenly.

The unary minus operator is used to change the sign of the operand it
operates on.

The action performed by this operator is one of returning the negative
of the value of the operand. For example, in the statement:

Answer = -x_value;  the value stored in "Answer" would be the negative
of the value stored in "x_value". For instance:

If the value stored in "x_value" is 385, then the value stored in the
variable "Answer" would be -385. Likewise, if the value in "x_value"
were -952, then "Answer" would contain the value 952.

Note: C does not have a unary plus operator.

Which of the following is the value that will be assigned to the variable
"Answer" after execution of the statement: Answer = 27 % 12; ?

A 2.25
B 2
C .25
D 2

Very good.

No. The modulus operator returns the "remainder" of "integer" division,
therefore answer "B" is correct.

I'm sorry, "E" was not one of your choices.

### Increment & Decrement Operators ###

The increment and decrement operators are represented by the following:

Increment (++) and Decrement (--)  

These two operators can be used in either "prefix" or "postfix" notation.
"Prefix" notation results in the variable being incremented or decremented before its value is taken. Whereas, "postfix" notation results in taking the variable value before it is incremented or decremented.

Let's take a look at each of these operators and see how "prefix" and "postfix" affects them.

**Increment Operator**

In the statement: \( x_{\text{value}} = x_{\text{value}} + 1; \) the value of \( x_{\text{value}} \) is incremented by 1 and restored in the memory location identified by the variable "\( x_{\text{value}} \)". This is a valid statement in C, but C also allows a shorthand way of doing the same thing. In this shorthand notation, the statement would be written as \( x_{\text{value}}++; \) Thus:

\[ x_{\text{value}} = x_{\text{value}} + 1; \text{ and } x_{\text{value}}++; \text{ are equivalent statements.} \]

The above example also demonstrates the use of "postfix" notation. The same result could have been obtained by using "prefix" notation. If you were to use the statement: \( ++x_{\text{value}}; \) the stored value of "\( x_{\text{value}} \)" would have been incremented by 1 as it was using the other two statements. Where's the difference then? Well, let's look at another example and see if it becomes clearer.

**Increment Operator Continued**

If we assign the value of 10 to the variable "\( x_{\text{value}} \)" using the statement: \( x_{\text{value}} = 10; \) and then perform some arithmetic operation using the variable "\( x_{\text{value}} \)" and the increment operator, what would be the result?

Well, it all depends on whether you use "prefix" or "postfix" notation.

If we perform the statement: \( \text{Result} = ++x_{\text{value}}; \) then the value stored in "\( \text{Result} \)" is 11, and "\( x_{\text{value}} \)" is incremented to 11, but if we perform the statement: \( \text{Result} = x_{\text{value}}++; \) then the value stored in "\( \text{Result} \)" is 10, and "\( x_{\text{value}} \)" is incremented to 11.

As you can see, this can be confusing until you get used to the idea.

The placement of the "increment" operator (either before or after the variable) has no effect on the outcome of statement execution.

(True or false)

Right.

Wrong. \( \text{variable}++ \) and \( ++\text{variable} \) will produce different results depending on how and when they are used.
Decrement Operator *

In the statement: y_value = y_value - 1; the value of y_value is decremented by 1 and restored in the memory location identified by the variable "y_value". This is a valid statement in C, but again C has a shorthand way of doing the same thing. In this shorthand notation, the statement would be written as y_value--; Thus:

y_value = y_value - 1; and y_value--; are equivalent statements.

The above example again demonstrates the use of "postfix" notation. The same result could have been obtained by using "prefix" notation. If you were to use the statement: --y_value; the stored value of "y_value" would have been decremented by 1 as it was using the other two statements. Let's again look at an example showing the difference between using "prefix" and "postfix" notation.

Decrement Operator Continued *

If we assign the value of 15 to the variable "y_value" using the statement: y_value = 15; and then perform some arithmetic operation using the variable "y_value" and the decrement operator, what would be the result?

Well, again it all depends on whether you use "prefix" or "postfix" notation.

If we perform the statement: Answer = --y_value; then the value stored in "Answer" is 14, and "y_value" is decremented to 14, but if we perform the statement: Answer = y_value--; then the value stored in "Answer" is 15, and "y_value" is decremented to 14.

Remember: prefix - value taken second, postfix - value taken first.

Which of the following represents the contents of variables "Answer" and "y_value" after execution of the statement: Answer = 25 + (--y_value);
given the initial value of "y_value" is 10?

A) Answer = 16 and y_value = 9
B) Answer = 15 and y_value = 9
C) Answer = 16 and y_value = 10
D) Answer = 15 and y_value = 10

You are correct.
Assignment Operators

The assignment operators are represented by the following:

Equal (=), and Operation equal (op=), where the "operation" is one of the binary operators.

We have already seen how the first assignment operator is used. As an example we have a statement such as: Answer = 25;

In this example, the equal assignment operator is used to place the value of 25 in the memory location represented by the variable "Answer". This assignment is done "right to left", so it is possible to make several assignments using one statement. For example:

a_val = b_val = c_val = 0; will set all the named variables to zero.

Assignment Operators Continued

The "operation equal" operators are really nothing more than a short-hand method of writing a statement that involves doing some operation on a variable and storing the result back into that variable's memory location. For example, in the statement:

x_value = x_value + 25; 25 is added to the value of x_value and the result is stored in the memory location represented by x_value.

C provides a way to accomplish this in a shorter statement (although the one above is also valid). An equivalent C statement would be:

x_value += 25;

All operations on the right will be done before the operation identified in front of the "=" sign.

Assignment Operators Continued

That last statement is an important one. For example, in the statement

a_val %= b_val + c_val; you will get a different result if the statement were evaluated as: a_val = (a_val % b_val) + c_val;

To eliminate this problem, C will evaluate the statement according to the following rule:

left_variable = (left_variable) "op" (right_expression);

Our example will be evaluated as: a_val = (a_val) % (b_val + c_val);
The result stored in "x_value" after execution of the statement:
\[ x_value = 35 + 20; \] given an initial value for "x_value" of 100,
would be 45. (True or False)

Right.

The result stored in "x_value" after execution of the statement:
\[ x_value -= 35 + 20; \] given an initial value for "x_value" of 100,
would be 45. (True or False)

Right.

In this topic area we have looked at a description and examples of
arithmetic, increment & decrement, and assignment operators.

In the next topic area we will continue our discussion of operators
and expressions and their use in C.

This concludes this topic area.

In this topic area we will continue discussing the use of operators
in expressions that we started in the last topic area.

This second topic area on this subject will cover the following:

1. Relational Operators
2. Logical Operators
3. Bitwise Logical Operators
4. Negation Operator
5. Conditional Operator

Let's get started ...

Relational operators are used within a program in order to compare
one or more data values. The relational operators are represented
by the following:

Equality (==), Inequality (!=), Greater than (>), Greater than or
equal to (>=), Less than (<), and Less than or equal to (<=).

Expressions involving these operators are evaluated as being either
"true" or "false". If an expression is "true" then the expression
has a value of 1 (one), if it is "false" then the value is 0 (zero).
Let's take a look at an example to help make this clear.

For our example let's compare two variables:

\[ \text{var}_1 \geq \text{var}_2 \] is an expression that has a value of either "true" or "false". If \( \text{var}_1 \) is indeed greater than or equal to \( \text{var}_2 \), then the expression is "true" and has a value of 1 (one). Likewise, if \( \text{var}_1 \) is not greater than or equal to \( \text{var}_2 \), then the expression is "false" and has a value of 0 (zero).

In order to give this evaluation meaning, it must be somehow used in a valid C statement. An easy to understand example is:

\[ \text{var\_flag} = (\text{var}_1 \geq \text{var}_2); \]

"\text{var\_flag}" will be assigned either 1 or 0 depending on the evaluation of the expression: \( \text{var}_1 \geq \text{var}_2 \)

The example we looked at was one in which a comparison was made between two variables. This is not the only way to use the relational operators as you can well imagine.

Some of the more common situations that relational operators are used in include: comparing array values, checking for "end of file", controlling function calls, and controlling statement execution.

Relational operators have a lower precedence than arithmetic operators, and assignment operators have lower precedence than relational operators thus, the statement: \( \text{val\_one} = \text{val}_2 \neq \text{val}_3 - 5; \) will be evaluated as: \( \text{val\_one} = (\text{val}_2 \neq (\text{val}_3 - 5)); \) The final value of either 1 or 0 will eventually be stored in the memory location represented by the variable "val\_one".

Which of the following is "not" a relational operator?

\[ \begin{align*}
\text{A} & \Rightarrow \text{=} \\
\text{B} & \Rightarrow \text{!} \\
\text{C} & \Rightarrow \text{\geq} \\
\text{D} & \Rightarrow < \\
\text{E} & \Rightarrow ++
\end{align*} \]

Right.
Logical operators are also called logical connectives and they are used to combine expressions being used for comparison. The logical operators are represented by the following:

- Logical AND (&&)
- Logical OR (||)

Expressions involving these operators are evaluated as being either "true" or "false". If an expression is "true" then the expression has a value of 1 (one), if it is "false" then the value is 0 (zero).

Let's take a look at an example.

In the expression: in_char == 'y' || in_char == 'n', the value of the expression can once again have a value of either 1 or 0 depending on whether the expression is "true" or "false".

In order to give this evaluation meaning, it must be somehow used in a valid C statement. An example is:

valid_resp = (in_char == 'y' || in_char == 'n');

"valid_resp" will be assigned either 1 or 0 depending on the evaluation of the expression: in_char == 'y' || in_char == 'n'.

In other words, valid_resp will be 1 if in_char equals either y OR n, or 0 if in_char equals anything else.

Logical operators have a lower precedence than relational operators, and assignment operators have lower precedence than logical operators thus, the statement:

result_val = val_1 < val_2 && val_3 != val_4;

is evaluated as: result_val = ((val_1 < val_2) && (val_3 != val_4));

The final value of either 1 or 0 will eventually be stored in the memory location represented by the variable "result_val".
the value of "True_Response" will be set to 1, only if both "choice == 't'" and "choice == 'T'" are true. (True or False)

That's right.

Wrong. Only one of the expressions has to be true when using the "OR" logical operator.

### Bitwise Logical Operators ###

The use of bitwise logical operators is beyond the scope of this course. However, I feel that their existence should be mentioned.

Bitwise logical operators are represented by the following:

- Bitwise AND: &
- Bitwise inclusive OR: |
- Bitwise exclusive OR: ^
- Left shift: <<
- Right shift: >>
- Unary one's complement: ~

### Negation Operator ###

The negation operator is a unary NOT operator. It is used to convert or reverse the value of the operand it appears in front of.

The exclamation point (!) is used for this operator.

For example, in the expression: !(val_one < 30) if the value of the inner expression is "true" then the value of the entire expression is "false", and vice versa.

The parentheses, in this case, are necessary since the negation operator has a higher precedence than relational operators.

A statement using the negation operator would look something like this:

```c
control_flag = ! found_flag;
```

### Conditional Operator ###

The conditional operator is what is called a "ternary" operator.

What this means is that the operator acts upon three operands. The effect it has is very similar to an "if-else" control statement.

The conditional operator is represented by a question mark and colon.
### Conditional Operator Continued

For example, in the statement:

```plaintext
new_val = val_1 == 1 ? new_val = 25 : new_val = 30;
```

The final value of "new_val" depends on the value of "val_1". If
val_1 is equal to 1, then the expression "val_1 == 1" is true and
the expression "new_val = 25" will be executed, else if "val_1 == 1"
is false, then the expression "new_val = 30" will be executed. The
value of new_val will be stored in the memory location represented
by the variable "new_val" because of the assignment operator "="
after the variable "new_val" on the left side of the statement.

In the statement:
```plaintext
val_1 = test_val ? val_1 = 1 : val_1 = 0;
```
if test_val equals 1, then the value of val_1 will be 1. (True or False)

Right.

Wrong. "val_1 = 1" would be executed, thus setting "val_1" equal to 1.

### Lesson Two Summary

Well, that about does it for lesson number two. If you have seen the
four subject topics in this lesson, you should now be ready to take
the final test. If you feel that you don't understand something well
eough to pass the test, please retake the topic that is giving you
problems.

<table>
<thead>
<tr>
<th>Topic #</th>
<th>Subject covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Character &amp; interger constants and variables.</td>
</tr>
<tr>
<td>2</td>
<td>Real &amp; double precision real constants and variables.</td>
</tr>
<tr>
<td>3</td>
<td>Arithmetic, increment &amp; decrement, and assignment operators.</td>
</tr>
<tr>
<td>4</td>
<td>Relational, logical, negation, and conditional operators.</td>
</tr>
</tbody>
</table>

Welcome to the final test of lesson two. This test consists of ten
questions over material presented in the previous four topic areas.

In order to successfully complete this lesson, you must achieve a
minimum score of 70% (seven out of ten questions correct).

If you miss a question, the correct answer will not be shown. It is
up to you to research the correct answer.
Well, enough said. Let's get on with it. Good luck!

Which of the following is not a valid variable name?

A  \_IN_
B  \+ int
C  var
D  xl1y2

Which one of the following characters is used to identify a special character escape sequence?

A  #
B  +
C  \ 
D  %

In the statement: double var\_one = 358.8204; the keyword "double" is used to indicate that variable "var\_one" is to be doubled in value before being stored in the memory location represented by "var\_one".

(True or False)
525. Which of the following is "not" an arithmetic operator?

A. +
B. *
C. /
D. =

54 Right. (3,505)
55 ABC Wrong. (3,505)
55 B:930
55E "E" was not one of your choices.
55 B:925
51Frame 930 QM

526. Which of the following is the value that will be assigned to the variable "Answer" after execution of the statement: Answer = 22 % 5;

A. +
B. 2
C. 4.4
D. .4

54 Right. (3,510)
55 B:935
55BCD Wrong. (3,510)
55 B:935
55E "E" was not one of your choices.
55 B:930
51Frame 935 QP

527. The placement of the "increment" (++) or decrement (--) operators, with respect to the variable they operate on, never has an effect on the outcome of statement execution.

528. The result stored in "Answer" after execution of the statement: Answer *= 10 + 10; given an initial value for "Answer" of 10, would be 220. (True or False)

529. Which of the following is "not" a relational operator?

A. >=
B. 
C. 
D. 

C - 49
Which of the following represent the logical operators "OR" and "AND"?

53A: || and ~

53B: ~ and &&

53C: ## and ::

53D: :: and &&

53E: @@ and ++

This marks the end of lesson number two. I hope that it was of some benefit to you. I am looking forward to seeing you in lesson number three. I hope that you didn’t have too much trouble with the material presented in this lesson. If you did, please voice your comments to your training monitor who will in turn contact the CAI Plans Branch at Keesler AFB, MS.

Well, let’s take a look at how you did with the test ...
THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON PROGRAM CONTROL STATEMENTS USED IN THE C PROGRAMMING LANGUAGE.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: If, If-Else, Nesting, and Switch - This topic gives descriptions of the structure and use of the If and If-Else control statements and how to "nest" these statements. Also covered in this topic is the Switch control structure. (Approx. time = 15 min.)

Topic 2: Loops (While, For, and Do-While) - This topic discusses the structure and use of loop statements. (Approx. time = 15 min.)

Topic 3: Break and Continue Statements - This topic gives a description of the Break and Continue statements and how and when they are used. (Approx. time = 10 min.)

Lesson Breakdown Continued:

Topic 4: Goto statement and Labels - This topic gives a description of the Goto statement and the use of labels within a C program. (Approx. time = 5 min.)
Topic 5: Lesson 3 Test - This is the lesson test over items that have
been presented in the previous four lesson topics.
(Approx. time = 5 min.)

TOTAL LESSON TIME IS APPROXIMATELY 50 MINUTES.

I hope that you enjoy it!

**SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:**

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>1</td>
<td>If, If-Else, Nesting, and Switch</td>
</tr>
<tr>
<td>@</td>
<td>2</td>
<td>Loops (While, For, and Do-While)</td>
</tr>
<tr>
<td>@</td>
<td>3</td>
<td>Break and Continue Statements</td>
</tr>
<tr>
<td>@</td>
<td>4</td>
<td>Goto statement and Labels</td>
</tr>
<tr>
<td>@</td>
<td>5</td>
<td>Test Over Lesson 3</td>
</tr>
</tbody>
</table>

**NOTE:** A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.

Control statements are used in programming languages to provide a
means of altering the "normal" flow of the program.

Without the use of control statements, program execution would proceed
in a sequential fashion starting with the first executable statement
and ending with the last executable statement. In most cases this is
not the desired way in which the programmer wants the program to exe-
cute. Thus, the need and capability for imposing control over program
execution using program control statements.

The control statements that we will be looking at in this topic are
the "if", "if-else", and "switch". We will also cover "nesting" of
"if" statements.

Let's get started.
The "if" statement is used to control the execution of a statement or statements by testing an expression. The expression is checked to see if it is "true" (non zero) or "false" (zero). If the statement is indeed "true", then the statement (or statements) following the "if" is executed. If the expression is "false", then the next sequential statement is executed.

The structure of the basic "if" statement is as follows:

```
if (test_expression)
    statement_to_be_executed;
next_sequential_statement;
```

Let's take a look at an actual example of the basic "if" statement.

For this example let tax_val, high_tax, and tax_rate be of type "int".

```
if(tax_val >= 10)
    high_tax++;
```

In this example, the expression "tax_val >= 10" is tested. If the value of "tax_val" is greater than or equal to 10, then the statement "high_tax++" is executed, otherwise program execution continues with the statement "tax_rate = tax_val / 100;"

By now you should be asking yourself: "How does the compiler know what statement is associated with the 'if' statement?" The answer of course is quite simple. Let's clear up the question and expand the "if".

The example we just looked at could just as easily have been written as:

```
if(tax_val >= 10)
    high_tax++;
    tax_rate = tax_val / 100;
```

This is confusing to the programmer, but not to the compiler. When the "if" statement is encountered, the next sequential statement is the only one that is associated with it. Therefore, only "high_tax+++" is subject to conditional execution. The statement "tax_rate = tax_val / 100;" is not subject to conditional execution.
This brings up the question of how do we provide for the execution of several statements after the "if" statement? Let's take a look.

If it is desired to have a group of statement's execution controlled by an "if" statement, then you must use braces "{}" to form a "block" of one or more statements to be conditionally executed. For example:

```c
if(test_expression)
{
  first_statement;
  second_statement;
  ...
  last_statement;
}
```

Which of the following is used to "block" statements into a group to be conditionally executed?

A: 
B: 
C: 
D: 
E: I'm sorry, "E" was not one of your choices.

Correct.

An option to the "if" statement is the use of an "else".

I will use the same example as before to illustrate the structure and use of the "if-else" control structure.

```c
if(tax_val >= 10)
  high_tax++;  
else
  low_tax++; 
```
tax_rate = tax_val / 100;

Here, the expression "tax_val >= 10" is tested. If the value of the "tax_val" is greater than or equal to 10, then "high_tax++" is executed, otherwise "low_tax++" is executed before execution continues with the statement "tax_rate = tax_val / 100;"

Of course you may use a "block" of statements after either or both the "if" or "else" parts of the control structure. For example:

```c
if(test_expression)
{
    statement_1;
    statement_2;
}
else
{
    alt_statement_1;
    alt_statement_2;
    statement_3;
}
next_sequential_statement;
```

It is often the case that you may need to test more than one expression within an "if-else" structure. This may be done by using what is called a multi-way decision structure. I will show you one way to do this using the "if-else" structure now, and later we will see another way using the "switch" structure. Using "if-else" structure:

```c
if(tax_val >= 10)
   high_tax++;
else if(tax_val <= 5)
   low_tax++;
else
   medium_tax++;

// Using this structure, one of the variables: "high_tax", "low_tax", or "medium_tax" will get incremented depending on the value of "tax_val".
// If you didn't want to keep track of "medium_tax", you could leave off "medium_tax++";
// the last else and its statement.
```

When using the "if-else" control structure, you are limited to only one executable statement for each part of the structure. (True or False)

Right. "Blocks" of statements can be defined by the use of braces "{}".
Wrong. "Blocks" of statements can be defined by the use of braces "{}".
### Nesting ###

Another capability of the "if-else" structure is being able to "nest" other "if" or "if-else" structures within the original "if-else".

For example:

```c
if(test_exp_1)
    if(test_exp_2)
        statement_one;
    else
        alt_statement_one;
else
    statement_two;
next_seq_statement;
```

In this structure, if "test_exp_1" is true then "test_exp_2" is checked and if found true, then "statement_one" is executed if however, "test_exp_2" is false, then the "alt_statement_one" is executed. If the "test_exp_1" was found to be false, then only "statement_two" would be executed.

Note: True is any "non zero" value and false is a "zero" value.
Also, "blocks" of statements can be used in the structure.

Caution must be exercised when nesting "if-else" structures. Remember, the "else" part of the "if-else" structure is optional. Thus, it is fairly easy to have an "else" apply to the wrong "if" statement.

Let's take a look at an example to show how this can happen.

For our example, let's say we want to check an expression and if it is true, then we want to check a second expression and if it is true, then we want to execute a statement, but if the second expression is false, we don't want any statement executed. If however, our first expression is false, then we want to execute a different statement. How would we code such a thing? Well, let's give it a try.

```c
if(test_exp_1)
    if(test_exp_2)
        statement_one;
    else
        statement_two;
else
    statement_two;
next_seq_statement;
```

Even though I indented the code to look like the "else" goes with the first "if", it really goes with the last "if" that doesn't have an "else". Thus, the above code doesn't solve the problem as I stated it.

In order to solve the stated problem, we must use braces to force
program execution.

Compare the code I gave before (left) to the correct code (right).

```
if(test_exp_1)
  if(test_exp_2)
    statement_one;
  else
    statement_two;
else
  statement_two;
```

As you can see, the execution of the code is greatly affected by the placement of the braces in the "if-else" control structure.

Given:
```
if(x > 0)
  if(x > 10)
    x_large = 1;
  else
    x_small = 1;
else
  if(x == 0)
    x_zero = 1;
```

Which of the following would be true if x = -1?

A x_large would be set to 1
B x_small would be set to 1
C x_zero would be set to 1
D none of the above
E A, B, and C would be true

Very good.

```
$** Switch Statement $$
```

We saw earlier that one way to do multi-way decisions was with the use of several "if-else" statements linked together.

A common use of such a structure is when you test a variable and depending on its value (as compared to a constant) a statement or group of statements is executed. For example:

```
if(test_var == 10)
  statement_to_be_executed;
else if (test_var == 15)
  alt1_statement_to_be_executed;
else if (test_var == 20)
  alt2_statement_to_be_executed;
```
In the example, we saw how to use the "if-else" structure to accomplish the testing of one variable and execution of different statements depending on the value of the variable.

Well, in C we have another way to accomplish the same thing. We can use the "switch" statement. In the "switch" statement each constant value we wish to test the variable against is labeled with the keyword "case". The last statement (following the last "else" in our example) is labeled with the keyword "default".

Let's take a look at our example again, but this time we will use the "switch" statement structure.

Essentially, the "switch" is just a special case of the "if-else" structure, and its use is really just "programmer preference". (True or False)

Right. You can do the same thing using the "if-else" structure.

Wrong. You can do the same thing using the "if-else" structure.

In this topic we have looked at the "if" statement, the "if-else" structure, nesting of the "if-else" structure, and the "switch" statement.
We have seen many examples of what these statements and structures look like, and how they are used.

In the next topic area I will describe and show examples of loop statements and structures.

See you there!

*** This concludes this topic area. ***

Loops are used in programming languages to provide a way of repeatedly executing a statement or group of statements within the program.

The way in which a loop is written can vary. The most common reason for this variability is again "programmer preference". Most, if not all, loops can be written using only one of the structures that we will be covering in this topic area.

The loop control statements and structures that we will be looking at in this topic are the "While", "For", and "Do-While".

Let's get started.

The "while" loop is a two part control structure. The first part is the loop control expression, and the second part is the executable body.

The loop control expression is a expression that is tested at the beginning of the loop and after execution of the body. The loop control expression is "true" whenever it is "non zero" and "false" when it is "zero". Execution of the body will continue until the control expression is "false". If the expression is "false" the first time, then program control will drop to the next sequential program statement.

The structure of the "while" loop looks like this:

```
while (test_expression) {  
    statement_to_be_executed;  
}  
```

Of course, braces can be used to define a "block" of statements.

Here is an example using the "while" loop control statement:

```c
sum = 0;
```
loop_var = 0;
while (loop_var == 0) {
    if (sum < 10)
        sum += 2;
    else
        loop_var++;
}

When the "while" is encountered, the test expression is checked and found to be "true", so the loop body is then executed. Execution of the loop body will continue until the loop control expression is no longer true. That will occur, in this example, after 6 iterations.

If the loop control expression is "false" the first time it is checked, then the loop body will be executed only once before program control drops to the next sequential program statement after the "while" loop. (True or False)

That's right. The loop body will be skipped entirely.

Wrong. The loop body will be skipped entirely.

The "for" loop is a three part control structure. The first part is the loop control initialize expression, the second part is the loop control test expression, and the third part is the loop control increment expression.

The loop control initialize expression is a expression that is evaluated once and can serve to initialize variables used within the loop body. The loop control test expression is tested at the beginning of the loop and after execution of the body. Again, the loop control expression is "true" whenever it is "non zero" and "false" when it is "zero". Execution of the body will continue until the control expression is "false". If the expression is "false" the first time, then program control will drop to the next sequential program statement. The loop control increment expression is evaluated after execution of the loop body.

The structure of the "for" loop looks like this:

for (initialize_exp; test_exp; increment_exp)
    statement_to_be_executed;

Again, the braces can be used to define a "block" of statements.
Such as:
for (loop_var = 0; loop_var < 50; loop_var++) {
    first_statement;
    next_statement;
    last_statement;
}

Here is an example using the "for" loop control statement:
for (i = 0; i < 20; i++)
    if((i % 2) == 0)
        printf("i value is even");
    else
        printf("i value is odd");

When the "for" is encountered, the loop control initialize expression
is executed setting i equal to zero. Next the loop control test ex-
pression is checked and found to be "true", so the loop body is then
executed. After execution of the loop body the loop control increm-
ent expression is executed setting i equal to i plus one. The loop control
test expression is then checked again. The execution of the loop body
will continue until the loop control expression is no longer "true".

Which of the following is "not" a part of the "for" loop control structure?
A) initialize expression
B) test expression
C) terminate expression
D) increment expression

Correct. Keep up the good work.

As I mentioned before, loop structures serve basically the same purpose
and can usually be accomplished by using one such structure. We have
looked at both "while" and "for" loops so far. Let's compare the struc-
ture of these two loop types.

The "while" structure:  
init_exp;  
while (test_exp) {  
    statement_to_be_executed;
}  

The "for" structure:  
for (init_exp; test_exp; incr_exp) {
    statement_to_be_executed;
}  

C - 61
Which of these two structures you use is up to you, but there are times when one may be more appropriate than the other.

The final loop structure available in C is the "do-while".

The "do-while" loop is a two part control structure just like the "do" loop. The basic difference between the "do" loop and the "do-while" loop is that the first part of the "do-while" is the executable body, and the second part is the loop control expression. This is just the opposite of the "do" loop control structure.

The loop body will be executed once, before the loop control expression is tested at the end of the loop. If the loop control expression is "true" then the loop body will be executed again. Execution of the loop body will continue until the loop control expression is "false". The biggest difference, I'm sure you have noticed, is that the loop body will be executed at least one time before program control drops to the next sequential program statement.

The major difference between the "while" loop and the "do-while" loop is that the "do-while" will always be executed at least once whereas the "while" loop may be skipped altogether if the loop control expression is "false".
Wrong. That is a true statement.

In this topic we have looked at the "while", "for", and "do-while" loops.

We have seen many examples of what these statements and structures look like, and how they are used.

In the next topic area I will describe and show examples of the "break" and "continue" statements.

Hope I see you there!

*** This concludes this topic area. ***

### Break Statement Example #1: "While" loop ###

The "break" statement is used to terminate a "while", "for", or the "do-while" loop before the loop control expression becomes "false". It is also used in the "switch" control statement to prevent further statement execution after a "case" has been found that satisfies the switch.

When a "break" statement is encountered, it is executed and the loop or case in which it is located is terminated immediately. Program control then passes to the next sequential statement following the loop or switch.

I will show you how this looks in each of the loop structures as well as the "switch" structure.

But first I want to be sure you want to see these examples.

Do you want to see examples of how the "break" statement is used?

(Yes or No)

Y

OK. Here we go.

OK. Let's take a look at the "Continue" statement.

### Break Statement Example #2: "For" loop ###

The following is an example of how the "break" statement can be used in the "while" loop.
Without getting into details of how the "scanf" statement works or where you would use this section of code, this example shows how the "break" statement can be used to terminate the "while" loop before the loop control statement becomes "false". If the variable named "in_int" ever becomes a negative number, the "break" will be executed and program execution will continue with the next sequential statement after the loop.

### Break Statement Example #2: "For" loop

The following is an example of how the "break" statement can be used in the "for" loop.

```c
for (i=0; i<=10; i++) {
    in_char = getchar();
    if(in_char == '.')
        break;
    last_name[i] = in_char;
}
```

Here, the loop will be terminated if "in_char" becomes a period (.) and program execution will, once again, continue with the next sequential statement after the loop. Note: We will cover "arrays" in lesson 4 and "input & output" in lesson 6.

### Break Statement Example #3: "Do-While" loop

The following is an example of how the "break" statement can be used in the "do-while" loop.

```c
do {
    count++;  
    if(count > 10)
        break;
    avg_num = (total / tot_num);
    scanf("%d", &in_int);
    tot_num++;
    total += in_int;
} while (avg_num < 70);
```

In this example, there are really two loop control expressions. The loop would be terminated if the value of "count" becomes greater than 10, or if variable "avg_num" ever exceeds the value of 69. Since the "do-while" is executed before the loop control expression is tested, the "break" statement could be used to control the loop's execution the first time through.

### Break Statement Example #4: "Switch" statement

```c
```
The following is an example of how the "break" statement is used in the "switch" statement.

```java
switch(temp) {
  case 70:
    nice++;
    break;
  case 80:
    hot++;
    break;
  case 90:
    cool++;
    break;
  default:
    break;
}
```

The "break" is used in the "switch" statement in order to prevent the unnecessary evaluation of expressions that will turn out to be "false". The "break" statement will terminate the "switch" after the "case" is found that is "true" and the statement(s) is executed. It is important to note that the execution of the switch is sequential, therefore if in our example the value of "temp" is 70, there is no need to check any "cases" after the execution of the statement "nice++;".

Which of the following will the "break" statement "not" work with?

A) "while" loop
B) "for" loop
C) "switch" statement
D) "if-else" statement
E) "do-while" loop

Right you are. The "if-else" works the same as the "switch" without the use of the "break" statement.

The "continue" statement is used within a loop structure in order to force the loop's next iteration. The "continue" is used with the "while", "for", and "do-while" loops, but NOT with the "switch" statement.

When you use the "continue" in the "while" and "do-while" loops, it forces the immediate evaluation of the "loop control expression".

When you use the "continue" in the "for" loop, it executes the "loop control increment expression" and then the "loop control expression" is evaluated.

Let's take a look at an example.
The following is an example of the use of the "continue" statement in a "for" loop.

```c
for (i=0; i<max_i; i++) {
    if(name_area[i] != ' ')
        continue;
    num_found++;
}
```

In this example the "continue" statement causes the loop to be executed until a "space" is encountered or the "loop control test expression" becomes "false". Once a space is found, "num_found" is incremented, the "loop control increment expression" is executed, and the "loop control test expression" is evaluated. Execution will continue in this fashion until the "loop control test expression" becomes "false".

The use of the "continue" is only effective in the loop control structures of "while", "for", and "do-while". (True or False)

Right. It can be used in a "switch", but only if the switch is inside of a loop structure, in which case it would cause the next iteration of the loop structure.

Wrong. It can be used in a "switch", but only if the switch is inside of a loop structure, in which case it would cause the next iteration of the loop structure.

In this topic we have looked at the "break" and "continue" statements. I have presented you the opportunity to see many examples of how the "break" statement is used in the three different loop structures as well as the "switch" statement. You also saw an example of how the "continue" statement can be used within a "loop" structure.

In the next topic area I will describe and show examples of the "goto" and "label" statements.

Hope I see you there!

*** This concludes this topic area. ***

The use of "goto" statements has come under attack within the software
engineering community of experts. Although most languages provide for
the use of "goto", it is highly discouraged. Most instances of the
statement can be eliminated by careful software development. This is
especially true in a language such as C.

Even though use of the statement is discouraged, it is a part of the
language and therefore I will give a brief description of how it is
used.

In order to use the "goto" statement, you must have some way of identi-
fying where to "goto" to. In other languages such as BASIC or Fortran,
this is done by using statement numbers. C doesn't use statement num-
bers but instead uses "labels".

A label is declared in a function by using the following form:

```c
label_name:
```

When naming a "label", follow the same rules that you use when naming
a variable.

The "goto" statement is used to transfer program control to some point
within a function other than the next sequential statement. The point
MUST be a labeled point in the same function.

The most common use of the "goto" statement is to terminate execution
of a deep nested loop structure. As we learned in the last topic area,
we can use the "break" statement to terminate a loop but it will only
terminate the innermost loop (the one it is physically in).

A "goto" statement has the following form: goto label_name;

Note: "goto" is one word. The use of: go to label_name; will cause
a compile error.

The following two sections of code provide an example of how the "goto"
statement is used in conjunction with a label and how to write the same
section without using a "goto" statement.

```c
Code with the "goto" & "label" ; Code without using the "goto"

---
in_out() {
  }
```

C - 67
char c;
begin:
    c = getchar();
    if(c!='\n') {
        printf("%c",c);
        goto begin;
    }
    return;
}

You can only use the "goto" statement to transfer program control to a label within the function where the "goto" is located. (True or False)

Right.

Wrong. You can not transfer control to any other part of the program using the "goto" statement.

*** Lesson Three Summary ***

Well, we have come to the end of another lesson. If you have seen the four subject topics in this lesson, you should now be ready to take the final test. If you feel that you don’t understand something well enough to pass the test, please retake the topic that is giving you problems.

Topic 1 described the "if", "if-else", "nesting", and "switch".

Topic 2 described the "while", "for", and "do-while" loops.

Topic 3 described the "break" and "continue" statements.

Topic 4 described the "label" and "goto" statements.

Good Luck on the test.

Well, enough said. Let’s get on with it. Good luck!
if
if-else
switch
while
Right. (1,100)

Wrong. (1,100)

E was not one of your choices.

B:905

Braces "{}" are used to form a "block" of one or more statements to be
conditionally executed. (True or False)

Right. (1,120)

Wrong. (1,120)

Wrong. (1,120)

Wrong. (1,120)

Wrong. (1,120)

Essentially, the "switch" is just a special case of the "if-else"
structure, and its use is really just "programmer preference".

(True or False)

Right. (2,320)

Wrong. (2,320)

Wrong. (2,320)

If the "loop control expression" in the "while" loop is "false" the first
time it is checked, which of the following statements would be true?

A The loop body would be executed one time only.

B The loop would be executed until the control expression becomes "true".
53 53C+ The loop body would be skipped altogether.
53 53D The loop would become an infinite loop.
54 Right. (2,305)
54 B:930
55ABD Wrong. (2,305)
55 B:930
55E "E" was not one of your choices.
55 B:925
51Frame 930 QM
526. Which of the following is "not" a part of the "for" loop control
52structure?
53A initialize expression
53B test expression
53C increment expression
53D+ terminate expression
54 Right. (2,320)
54 B:935
55ABC Wrong. (2,320)
55 B:935
55E "E" was not one of your choices.
55 B:930
51Frame 935 QF
527. The major difference between the "while" loop and the "do-while" loop is
52that the "do-while" will always be executed at least once whereas the "while"
52loop may be skipped altogether if the loop control expression is "false".
52(True or False)
53Y
54 Right. (2,345)
54 B:940
55 Wrong. (2,345)
55 B:940
51Frame 940 QF
528. The "break" statement can only be used to terminate a "while" or "for"
52loop before the "loop control expression" becomes false. (True or False)
53N
54 Right. (3,500)
54 B:945
55 Wrong. (3,500)
55 B:945
51Frame 945 QM
529. Which of the following structures is the "continue" statement not
52effectively used with?
53A for loop
53B do-while loop
53C switch
when loop

Right. (3,535)
B:950

Wrong. (3,535)
B:950

"E" was not one of your choices.
B:945

Frame 950 GM

When using the "goto" statement in your C program, which of the
following must be adhered to?
A+ The target "label statement" must be in the same function.
B
The target "statement number" must be in the same function.
C
The "goto" statement must not be in a loop structure.
D
The "goto" statement must be before the "flagged" statement.

Right. (4,710)
B:955

Wrong. (4,710)
B:955

"E" was not one of your choices.
B:950

Frame 955 T

This marks the end of lesson number three. I hope that it was of some
benefit to you. I am looking forward to seeing you in lesson number
four. I hope that you didn’t have too much trouble with the material
presented in this lesson. If you did, please voice your comments to
your training monitor who will in turn contact the CAI Plans Branch
at Keesler AFB, MS.

Well, let’s take a look at how you did with the test ...
THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON ARRAYS, POINTERS, AND ADDRESS ARITHMETIC USED IN C PROGRAMMING.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Approx. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introducing Arrays - This topic introduces the declaration, initialization, and use of arrays.</td>
<td>15 min.</td>
</tr>
<tr>
<td>2</td>
<td>Introducing Pointers - This topic introduces the declaration and use of pointers.</td>
<td>15 min.</td>
</tr>
<tr>
<td>3</td>
<td>Working with Pointers I - This topic is the first of two that covers how to work with pointers. Emphasis is on how pointers are passed to functions.</td>
<td>10 min.</td>
</tr>
<tr>
<td>4</td>
<td>Working with Pointers II - This topic is the second of two that covers how to work with pointers. Emphasis is on how pointers are used in conjunction with arrays and the use of address arithmetic.</td>
<td>10 min.</td>
</tr>
</tbody>
</table>
# Topic 5: Lesson 4 Test - This is the lesson test over items that have been presented in the previous four lesson topics.
(Approx. time = 5 min.)

# TOTAL LESSON TIME IS APPROXIMATELY 55 MINUTES.

# I hope that you enjoy it!

*******************************************************************************

SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:

*******************************************************************************

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>1</td>
<td>Introducing Arrays</td>
</tr>
<tr>
<td>@</td>
<td>2</td>
<td>Introducing Pointers</td>
</tr>
<tr>
<td>@</td>
<td>3</td>
<td>Working with Pointers I</td>
</tr>
<tr>
<td>@</td>
<td>4</td>
<td>Working with Pointers II</td>
</tr>
<tr>
<td>@</td>
<td>5</td>
<td>Test Over Lesson 4</td>
</tr>
</tbody>
</table>

*******************************************************************************

NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.

*******************************************************************************

1) Frame 100 T  Introducing Arrays
   *** Introduction ***
   
   An "array" is a group of contiguously stored related variables.
   
   In this topic area we will take a look at the basic use of arrays and some advanced concepts involving arrays.
   
   To be more specific, we will be looking at: one dimensional arrays, multidimensional arrays, and array initialization.
   
   Let's get started.

2) Frame 105

3) Frame 105 T

4) *** One Dimensional Arrays ***

5) The language C does not have a "string" variable type, therefore it uses an array of characters to accomplish the same thing. If you
think of a string of characters such as a sentence. How would you
store it in your program? Well, the answer of course is to use an
array of characters.

The structure of the basic one dimensional character array declaration
statement is:

char var_name[n]; where "n" is the number of characters in the array.

Now comes the tricky part. The individual characters in the array
are called the "elements" of the array. Accessing these elements is
a very common procedure in programming. Let's look at an example that
uses an array and see how this is done.

For our example, let's say we want to store the word Payment.
The first thing we must do is decide on the size of the array that will
hold this word. This can be done by counting the number of characters
in the word. So, let's see... I count 8.

At first glance it looks like I made a mistake in counting the charac-
ters in Payment. This is not the case. In C the first element of
an array is stored in array position 0 (zero), and the last (string)
array position element is always a null character (\0). So, using
the following statement to declare our word as a character string
constant...

char ex_word[8] = "Payment";

the array will be filled as follows:

ex_word[0] = P
ex_word[1] = a
ex_word[2] = y
ex_word[3] = m
ex_word[4] = e
ex_word[5] = n
ex_word[6] = t
ex_word[7] = \0

The "null character" stored at the end of a string array is put there
automatically by the C compiler. All you have to worry about is to
leave room for it in your array. What if you don't want to worry
about such things? Well, there is a way to get around counting the
number of characters in a string constant and then adding one for the
null character. Let's take a look.
Using the statement: char ex_word[] = "Payment"; will accomplish the same thing as the example we just looked at. Namely, an array consisting of eight elements will be declared and filled by the compiler.

The way in which the individual elements in an array are accessed is by referencing the element using an index. In our example an index of 4 would look like this: ex_word[4] and yield the character e.

Our discussion thus far has only dealt with the C character type. The use of arrays is by no means restricted to this C variable type. Here are a couple examples of arrays of other variable types:

int ex_ints[35]; This is an array of integers (36 of them).
float ex_floats[67]; An array of floating point reals (68 of them).

Which of the following is the correct number for "n"?
A 10
B 9
C 8
D 7

Right. Seven characters plus the "null character", therefore 8.

Wrong. There are seven characters plus the "null character", example[0] thru example[7], therefore the correct answer is 8 ("C").

As we have seen, a one dimensional array is declared using a statement such as char ex_word[8]. The dimension of this array is seen as a list of characters running from ex_word[0] to ex_word[7].

A two dimensional array can be thought of as a table consisting of rows and columns. The way in which a two dimensional array is declared is as follows:

int ex_int[n][m]; where "n" is the number of rows and "m" is the number of columns.

Let's look at an example.
If we want to store the test scores for a class of 5 students who have each taken 4 tests, we could do it like this:

```c
int scores[5][4] = {
    {75, 80, 70, 95},
    {85, 85, 90, 95},
    {60, 90, 80, 90},
    {70, 80, 90, 90},
    {75, 85, 95, 85}
};
```

This form is very representative of how the table would look. How these numbers are stored is as follows: scores[0][0] = 75, scores[0][1] = 80, scores[0][2] = 70, and scores[0][3] = 95. You then increment the first index and continue: scores[1][0] = 85 ... scores[4][3] = 85.

This can be done several ways, but I would define a symbolic constant for each student. Such as:

```c
#define Jones 0
#define Smith 1
#define Brown 2
#define Green 3
#define White 4
```

Now if you want to find out what Brown got on his third test you could use the statement: Brown_3 = scores[Brown,2]; This will retrieve the score stored in array position scores[2][2], which was 80.

A good way I've found to get used to arrays is to experiment with them.

As you might have deduced by now, you can define arrays of more than two dimensions. All that needs to be done is add more brackets [[]] after the array name.

For example:
```
int four_D_array[5][10][5][20];
```

Don't ask me to give you a visual picture of such a thing, but I can tell you that there are 5000 integer storage locations allocated by such a declaration (5 x 10 x 5 x 20 = 5000).

Given the array declaration:

```c
int array[2][5] = {
    {75, 80, 70, 95, 65},
};
```
Which of the following is the value stored in position `array[1][2]`?

- A 80
- B 60
- C 90
- D 95
- E 70

Very good.

15 ABDE No. Answer "C" is the correct one.

You may NOT initialize arrays that are "automatic". This means any arrays that are contained within a function. In order to initialize an array within a function it must be declared as "static". The way this is done is by use of the keyword "static".

For example:

```c
sample() {
    char array[] = "Example";
}
```

This initialization is wrong. This is the correct way.

```c
static char array[] = "Example";
```
When initializing arrays other than character arrays, the initializing is accomplished with values enclosed in braces. For example:

A one dimensional global integer array can be initialized using:

```c
int array[5] = {24, 67, 82, 90, 41};
```

Or, if all values of the array are being specified, the dimension can be left out, as in:

```c
int array[] = {24, 67, 82, 90, 41};
```

Again, if the array is local to a function and needs to be initialized, use the keyword "static".

**Multidimensional arrays are initialized by rows, as in one of our previous examples:**

```c
int scores[5][4] = {
    (75, 80, 70, 95),
    (85, 85, 90, 95),
    (60, 90, 80, 90),
    (70, 80, 90, 90),
    (75, 85, 95, 85)
};
```

If any of the values are missing, then the array value will be stored as 0 (zero). Note: If values are missing, then dimensions must be specified. Of course "static" must be used for local function arrays that you want to initialize.

The integer array initialization: `int array[] = {2, 4, 6, 8};` is valid for a one dimensional integer array having 5 elements. (True or False)

Right. If you intend for the array to have 5 elements then either 5 values must be given in the list or a dimension of 5 must be explicitly stated.

Wrong. If you intend for the array to have 5 elements then either 5 values must be given in the list or a dimension of 5 must be explicitly stated.

In this topic we have looked at "one dimensional" and "multidimensional" arrays. We have also seen how to initialize these arrays.

We have seen examples of what these arrays and initialization statements
look like, and how they are used.

In the next topic area I will describe pointers and give a few examples of their use.

See you there!

*** This concludes this topic area. ***

---

Introducing Pointers

A "pointer" is a variable that contains the address of where some other variable resides in memory.

In this topic area I will describe how pointers are declared and used within a C program.

Since pointers can be very confusing to someone who has not seen them before, I will restrict my discussion to elementary concepts and leave their more advanced uses for your research.

Let's get started.

---

Pointers

In the declaration: int var_one = 500; a storage location is set aside in memory for an integer variable and the value of 500 is stored in that memory location. That memory location also has a memory address.

In C, you can determine the memory address by the use of the unary operator &.

The way that you would assign a pointer variable to the memory location where "var_one" is located is: point_v1 = &var_one; this assigns the address of the variable "var_one" to the variable "point_v1".

Note: Pointer names follow the same rules as other variable types and must be declared as the same type of the variable being pointed to (as we'll see later).

---

Pointers Continued

That's fine. Now we know how to find out the memory address of a variable, but what good is it?

It would be nice if we could now find out the value stored at the address pointed to by our pointer. C just happens to have a special
operator that allows us to do just that.

In C, you can determine the value stored at an address pointed to by
pointer by the use of the unary operator *.

The way that you would use this operator to find the value stored at
a pointed to address is: var1_val = *point_v1; this statement assigns
the value stored at the memory location pointed to by "point_v1" to
the variable "var1_val". Which, in our example, would be 500.

To help clear up what we have done so far, let’s look at our example
again and compare it to statements we are familiar with.

The sequence of statements: var_one = 500;
                        point_v1 = &var_one;
                        var1_val = *point_v1;

Is the same as the sequence of statements: var_one = 500;
                        var1_val = var_one;

In both of the above cases, the variable "var1_val" is assigned the
value of 500. Although the use of the first set of statements seems
to be an unnecessary complication of a straightforward assignment,
keep in mind that this is just an example to demonstrate how a pointer
is used but does not show the true power of pointer usage.

The two unary operators used when working with pointers are the _____ and
the ______.

A # and &
B + & and *
C $ and &
D $ and #
E # and *

Wrong. Answer "B" is the correct response.

In order for pointers to be used in a C program, you must declare a
pointer variable before you can use it. The type of the pointer
variable must be the same as the variable that it is to point to.
In our example, the statement: `point_v1 = &var_one;` must be preceded
by the declaration: `int *point_v1;` which states that the value to be
pointed to by "point_v1" is of type "int".

Pointers to other types of variables are declared in the same way.
For example:

```
char *char_point; declares the pointer variable "char_point" which
is to point to a variable of type "char".
```

The declaration: `float *var_point;` declares the pointer variable
"var_point" to be of type "float".

Very good. It declares the pointer variable "var_point" which "points"
to a variable of type "float".

No. It declares the pointer variable "var_point" which will "point"
to a variable of type "float".

```
*** Pointer Facts ***
```

Pointers can be used in expressions. For example:

```
answer = *point + 35; adds 35 to the value pointed to by "point" and
stores the result in variable "answer".
```

```
*p_1 = *p_2 * 5; multiplies the value pointed to by "p_2" by 5 and
stores the result in the variable pointed to by "p_1".
```

```
p_one = p_two; will make "p_one" point to the same variable that
"p_two" points to if both "p_one" and "p_two" are
declared to point to the same variable type.
```

(i.e. int *p_one, *p_two;)

```
*** Topic Review ***
```

In this topic we have looked at pointer declaration and a few elemen-
tary examples of how they are used.

The rest of this lesson will discuss some other uses of pointers in
C programming.

In the next topic area (3) I will describe and show examples of how
to pass pointers as function arguments. In topic area four I will
discuss the use of pointers in conjunction with arrays and explain
how to do address arithmetic.

Hope I see you there!

C - 81
In the last topic area we saw that a "pointer" is actually a variable that contains the address of where some other variable resides in memory.

In this topic area I will describe how pointers are passed to functions, a rationale for doing it, and a few examples.

Let's get started!

We have seen two methods of passing arguments to a function, although I have not explicitly named these methods. Now is as good a time as any to do so. They are: "Call by value" and "Call by reference". The main difference in the two is that the actual value stored in a variable can only be changed by using the "Call by reference" method of argument passing. Let's look at a couple of examples to help make this clear.

Let's say we have a C program that has two functions "main" and "add". The "main" function calls the "add" function and passes it two variables "x" and "y". The "add" function takes the two arguments and adds 50 to the first (x) and 75 to the second (y). The "main" function then prints out the two variables "x" and "y".

Let's see what these two functions might look like.

```
add(x,y)

int x, y;
{
    x += 50;
    y += 75;
    return;
}

main()
{
    int x, y;
    x = 10;
    y = 30;
    add(x,y);
    printf("\n%d %d", x, y);
}
```
For this example let's say we have two functions "main" and "init". The "main" function declares an array called "line" to be a sequence of 80 characters. The "main" function calls the "init" function and passes it the array to be initialized to blanks.

```c
init(b_line)     // This is a clear example of the "Call by reference" method. Although I called the array different names in the two functions, the "init" function will actually change the array "line" declared in function "main".

for (i=0;i<80;i++)     // This is because the function "main" actually passes the address of where the array "line"
  b_line[i] = ' ';     // begins in memory to the function "init".

main() {
  char line[80];     // This "Call by reference" only works in the case of arrays. Before we look at pointer
  init(line);         // passing, let me ask you a quick question.
}
```

The "Call by value" method of argument passing only passes a copy of a variable, whereas the "Call by reference" method passes the address of the argument. (True or False)

Right. You have been paying close attention.

Wrong. I hope you aren't falling asleep on me.

We've seen in another lesson that a called function can only return one value to the calling function. Thus, only one value of the calling function is truly changed. This of course precludes the use of global variables by the functions in question.

If it is necessary for the called function to change more than one variable of the calling function, then the preferred method is to use addresses or pointers as passed arguments.

There are three ways in which to accomplish the task introduced above.

1. Pass the address of the variable.
2. Pass a pointer to the variable.
3. Pass an array name.

If we have a function that is to be called and its "function" is to change two variables (as in our first example), we can set up the function to receive pointers as its arguments as follows:
In this example I have identified the variables "px" and "py" to be pointers to variables of type "int". When the function is executed, the values stored in the variables, pointed to by these pointers, will change by "50" and "75" respectively.

Let's look at how we would pass the "addresses" of the variables to this function from our "main" function.

One way we have identified as being a way to pass a pointer to a function is by passing the "address". The following illustrates this method.

main() {
    int x,y,*px,*py;
    x = 10;
    y = 30;
    add(&x,&y);
    printf("\n%d %d",x,y);
}

Now let's look at another way to pass pointers from the calling function to the called function.

An alternate way of passing pointer information is to pass the pointer itself. The following illustrates this method.

main() {
    int x,y,*px,*py;
    x = 10;
    y = 30;
    px = &x;
    py = &y;
    add(px,py);
    printf("\n%d %d",x,y);
}

The third method of passing pointer information (pass an array name) was already discussed.

Which of the following is "not" one of the ways in which to pass information that will allow the value of a variable to be changed by a called function?

A. Pass a pointer to the variable.
33A Pass an array name.
33B Pass the variable name.
33C Pass the address of the variable.
34 Very good.
35ABD No. That is one of the ways "to" do it. The correct response is "C".
35E "E" was not a given choice. Please try again.
36
31 *** Topic Review ***
32 In this topic area we have looked at the "Call by reference" and "Call by value" methods of argument passing as well as how to pass pointers as function arguments.
32 We have seen several examples to help illustrate all of these methods.
32 In the next topic area I will describe the use of pointers in conjunction with arrays and explain how to use address arithmetic.
32 Hope to see you there!

*** This concludes this topic area. ***
33
31 *** Introduction ***
32 In this topic area I will describe how pointers are used in conjunction with arrays and how to use address arithmetic.
32 We have seen already that when you declare an array with a statement like: char line[] = "This is an example"; the compiler sets up 19 contiguous storage locations in memory. These locations have names line[0] thru line[18].
32 We also have seen how to refer to each individual storage location using an "index" value. If "i" is an integer then line[i] refers to the "i"th element in array "line". You can manipulate "i" in order to give you quick and easy access to any of the elements of the array.
32 Let's now see how we can use pointers to give us access and manipulative power over arrays.
33
42 *** Array Access Thru Pointers ***
42 When an array is declared (char line[10];) the array can be passed
between fuctions by just giving the array name. For example:

init(line); This calls the function "init" and passes the array "line".

What actually happens is the C compiler passes the address of the "0"th element of the array. So in essence, a pointer to the beginning of the array is passed ("line" being the pointer).

A:710

The same thing can be accomplished by explicitly defining a pointer in the following manner:

char *p_line; This identifies "p_line" as a pointer to a variable of type "char".

p_line = &line[0]; This assigns the address of the "0"th element of array "line" to the pointer variable "p_line".

init(p_line); This calls the function "init" and passes the address of the starting location of array "line".

Once the above declarations have been made, the two expressions: "line" and "p_line" are interchangeable.

B:715

If you have the declaration: char line[10]; which of the following statements will assign the address of the "0"th element to a pointer variable that has been declared using the statement: char *p_line; ?

A: *p_line = line[0];
B: p_line = line[0];
C: *p_line = &line[0];
D: p_line = &line[0];
E: "E" was not a given choice. Please try again.

Right.

B:720

ABC Wrong. Answer "D" is the correct response.

B:720

E "E" was not a given choice. Please try again.

B:715

The next logical step in our discussion is to look at how we can access the individual elements of an array using our declared pointer.

We already know that "line[0]" will give us access to the "0"th element of the array "line", but now that "p_line" has the address of the "0"th element of the array, we can also use the expression "*p_line" to accom-
Now that we have pointer access to the array, we can manipulate the pointer to point to any of the array elements by use of address arithmetic.

The most common use of address arithmetic is through the use of the increment, decrement, addition, and subtraction operators.

The operation must involve a pointer and an integer with the exception of the subtraction operator (subtraction/comparison of two pointers is allowed).

The use of "relational" operators is legal as long as the pointers point to members of the same array. The use of the "operational assignment" operators "+=" and "-=" are also legal.

Let's look at an example of how to use some of these operators.

When we first started this topic area I used the declaration statement: char line[] = "This is an example"; to declare and initialize the array "line".

Using the declarations: char *p_line; and p_line = &line[0]; we established a pointer to the "0"th element of array "line".

We also saw that the expressions "line[0]" and "*p_line" are equivalent.

Both would return a value of T if used in a statement such as:

```
char_val = line[0]; OR char_val = *p_line;
```

We can move forward and backward in the array by using our pointer and the legal operators mentioned before.

If we want to move one element forward in the array we can use the increment operator (++), the addition operator (+), or the operational assignment operator (+=).

For example: p_line++ will make the pointer point to the next sequential element in the array. Likewise, p_line = p_line + 1; and p_line += 1; will have the same effect.
In general, it can now be said that if "p_line" is a pointer and "i" is an integer, then \texttt{p_line += i} will increment "p_line" by "i" thus making "p_line" point to an element "i" elements beyond its present location. Decrementing is done in a similar fashion.

Given that "pa_val", "pb_val", and pc_val are pointers. Which of the following statements is "not" a "legal" address arithmetic operation?

\begin{enumerate}
\item \texttt{pc_val = pb_val + pa_val;}
\item \texttt{pc_val = pb_val - pa_val;}
\item \texttt{pa_val += (pb_val += pc_val);}
\item \texttt{pa_val -= (pb_val - pc_val);}
\end{enumerate}

Very good. Addition of two pointers is not allowed.

Wrong. That is a valid statement involving address arithmetic.

"E" was not a given choice. Please try again.

Well, we have come to the end of lesson four. If you have seen the four subject topics in this lesson, you should now be ready to take the final test. If you feel that you don't understand something well enough to pass the test, please retake the topic that is giving you problems.

Topic 1 gave an introduction to one and multidimensional arrays.

Topic 2 gave an introduction to pointers and their use.

Topic 3 gave a description of how pointers are passed to functions.

Topic 4 gave a description of pointer use in conjunction with arrays.

Good Luck on the test.
521. In the array declaration: char word[10] = "Sample"; which of the following is the correct value for "\n"?
A 9
B 8
C + 7
D 6
E Right. (1,110)

523. Given the array declaration: int array[2][4][6]; how many integer storage locations are allocated?
A 12
B 24
C 36
D 48
E Right. (1,145)

525. The integer array initialization: int array[5] = {4,8,12}; is valid for a one dimensional integer array having 5 elements. (True or False)
A True
B False

526. Which of the following is the unary operator that is used to determine the memory address of a variable?
A @
B #
C %
D &
535 **Right.** (2,305)
535 **Wrong.** (2,305)
535 Frame 925 QP
536 The declaration: char *char_point; declares the pointer variable "char_point" which points to a variable of type "char". (True or False)
537 **Y**
537 **N**
537 Frame 925 QP
538 The "Call by reference" method of argument passing only passes a copy of a variable, whereas the "Call by value" method passes the address of the argument. (True or False)
539 **N**
539 **Y**
539 Frame 930 QP
540 Given: main () {
540 	 int x,y,*px,*py;
540 	 x = y = 0;
540 	 px = &x;
540 	 py = &y;
540 	 change(px,py); }
541 Which of the following is the method of pointer passing used?
542 **A** Pass a pointer to the variable.
542 **B** Pass an array name.
542 **C** Pass the address of the variable.
542 **D** Pass the variable name.
543 **Right.** (3,540)
543 **Wrong.** (3,540)
543 Frame 935 QM
544 Given the declaration: int array[10]; which of the following statements will assign the address of the third element to a pointer variable that has been declared using the statement: int *p_array; ?
545 **A** *p_array = &array[2];
545 **B** p_array = &array[2];
545 **C** *p_array = array[2];
545 **D** p_array = array[2];
Right. (4,710)
54 B:945
55ACD Wrong. (4,710)
55 B:945
55E "E" was not one of your choices.
55 B:940
51Frame 945 OP
52The statement: init(p_var); calls the function "init" and passes the
52address of the variable pointed to by the pointer "p_var", provided the
52pointer was declared using a statement like "int *p_var;". (True of False)
53Y
54 Right. (4,710)
54 B:950
55 Wrong. (4,710)
55 B:950
51Frame 950 GM
5210. Which of the following operators is "not" a legal operator in address
52arithmetic?
53A +
53
53B -
53
53C +=
53
53D --
53
53E+ /
54 Right. (4,725)
54 B:955
55ABCD Wrong. (4,725)
55 B:955
51Frame 955 T
52 *** End of Lesson Material ***
52
52 This marks the end of lesson number four. I hope that it was of some
52 benefit to you. I am looking forward to seeing you in lesson number
52 five. I hope that you didn't have too much trouble with the material
52 presented in this lesson. If you did, please voice your comments to
52 your training monitor who will in turn contact the CAI Plans Branch
52 at Keesler AFB, MS.
52
52 Well, let's take a look at how you did with the test ...
THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON STRUCTURES THAT ARE USED IN C PROGRAMMING.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Introducing Structures - This topic introduces the idea of structures and two methods of declaring them.
(Approx. time = 10 min.)

Topic 2: Structures and Arrays - This topic describes the use of structures within structures and arrays of structures.
(Approx. time = 5 min.)

Topic 3: Structures and Pointers - This topic describes how to use pointers in conjunction with structures. (Approx. time = 5 min.)

Lesson Breakdown Continued:

Topic 4: Structures and Functions - This topic describes how structures are passed between functions. (Approx. time = 5 min.)
# Topic 5: Lesson 5 Test - This is the lesson test over items that have
been presented in the previous four lesson topics.
(Approx. time = 5 min.)

# TOTAL LESSON TIME IS APPROXIMATELY 30 MINUTES.

I hope that you enjoy it!

******************************************************************************
** SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:
******************************************************************************

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>1</td>
<td>Introducing Structures</td>
</tr>
<tr>
<td>@</td>
<td>2</td>
<td>Structures and Arrays</td>
</tr>
<tr>
<td>@</td>
<td>3</td>
<td>Structures and Pointers</td>
</tr>
<tr>
<td>@</td>
<td>4</td>
<td>Structures and Functions</td>
</tr>
<tr>
<td>@</td>
<td>5</td>
<td>Test Over Lesson 5</td>
</tr>
</tbody>
</table>

NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.

******************************************************************************
** 11Frame 100 T  Introducing Structures
12    *** Introduction ***
12
12    A "structure" is typically a group of related variables, of possibly
12      different types, under a single structure name.
12
12    In this topic area we will take a look at the concept of a "structure"
12      and two methods of declaring them.
12
12    We will also be discussing how to access the individual members of a
12      declared structure. We will see several examples of elementary
12      structures in order to get you introduced to their declaration and
12      use.
12
12    Let's get started.
13B:105
11Frame 105 T
12    *** Structures ***
12
C - 93
Whenever you have a group of related items it is nice to be able to group them in such a way as to give quick and easy access. In C, the way this is done is through the use of "structures".

For example, if you have information about a student at a university, this information might include items such as: Name, Address, Major, GPA, and Advisor. Instead of keeping all this information stored separately we can form a structure with five parts containing the needed information.

Let's take a look at one way to declare our structure.

```
struct C
{
    char name[NAMESIZE];
    char address[ADDRESS_SIZE];
    char major[MAJOR_SIZE];
    float gpa;
    char advisor[ADVISOR_SIZE];
}
```

Each of the character arrays must have predeclared constant values for their sizes, hence the use of capital letter names. You could have broken "name" or "address" into several variables or even other structures as we'll see later.

The "structure name" need not be a single variable name. You can give several different names to the same structure type by listing the names separated by commas.

For example:

```
struct C
{
    int wing_span;
    int num_tires;
    double tonage;
    double fuel_cap;
}
```

This example shows how you can define a standard information structure that can be used for several different types of aircraft.

The use of structures allows for the grouping of related variables into a form which will be easy and quick to access. (True or False)
13 That's right.
14 B:125
15 Wrong. It is easy and quick, as you will shortly see.
15 B:125

11 Frame 125 T
* Declaring Structures Continued *

Our second way of declaring a structure uses a sort of "template" for the composition of the structure variables.

This way of declaring a structure uses the keyword "struct" followed by a structure tag followed by an open brace "{" followed by the declaration of the item variables followed by the close brace "}" followed by a semicolon. For our "student" example this would look something like:

```
struct stu_rec {
    char name[NAME_SIZE];
    char address[ADDRESS_SIZE];
    char major[MAJOR_SIZE];
    float gpa;
    char advisor[ADVISOR_SIZE];
};
```

As you can see, the structure name has been dropped and I have added the structure tag name of "sturec". Defining structures in this way will allow you to define a variable of this type within your program whenever you need it.

The major difference between the two methods of declaring structures is that the first method will allocate memory space for the structure variable when the program is run through the C compiler, and the second method doesn't.

The second method only defines a structure type which you can use in later variable declarations. For example, if you have several students that you wish to identify within your program, you can use the following declaration to allocate memory space for them:

```
struct stu_rec student_1, student_2, student_3;
```

This declares the variables "student_1", "student_2", and "student_3" to be structures of type "stu_rec".

11 Frame 130 T
* Declaring Structures Continued *

11 Frame 130 T

11 Frame 135 T
### Structure Variable Access ###

Now that we have seen how to declare structures, it is now time to see how to access the individual members of the structure.

Access to these individual structure members is gained through the use of the structure member operator . (period).
For example: student.gpa would be how you reference the "gpa" float variable within the "student" structure that I declared using the first method of structure declaration. Whereas, student_1.gpa is how to reference the "gpa" float variable within the "student_1" structure (of type "stu_rec") that was declared using the second method.

Let's look at another example to be sure you understand this concept.

```c
struct employees { int num_male; int num_female; int num_over_40; int num_under_40; } dep_1;
```

This declaration sets up a "template" for a structure of type "employees" as well as declares "dep_1" to be a variable of that type. This is a legal declaration that combines both methods of structure declaration.

The way in which you would reference the individual members of the declared structure "dep_1" is as follows:

```c
dep_1.num_male
```

Each of these individual variable members of the structure can be used as you would any variable of their individual type ("int").

```c
dep_1.num_female
```

Let's now take a quick look at how you can initialize a structure.

```c
struct planes {
    int tot_num;
    int tot_maint;
    int tot_avail;
} planes = (50,5.45);
```

```c
struct planes F_16 = (50,5,45);
```

A structure may be initialized by listing the member values after the structure name declaration. The following two examples show how this is done.

```c
struct { int tot_num;
    int tot_maint;
    int tot_avail;
} planes = (50,5,45);
```

```c
struct planes F_16 = (50,5,45);
```

Which of the following is a way to increase the "num_brick" variable to 137?

A) houses.num_brick += 10;
B) quarters.houses.num_brick += 10;
C) quarters.num_brick += 10;

```c
L = 96
```
In this topic we have looked at the concept of a structure and we examined two methods of declaring them.

We have seen how to access the individual members of a declared structure, and we also saw how you can initialize a structure when it is declared. We have seen examples of what these structures look like and how they can be used.

In the next topic area I will describe "structures within structures" and "arrays of structures".

See you there!

*** This concludes this topic area. ***
struct employee {
  char f_name[FSIZE];
  char m_init;
  char l_name[LSIZE];
};

struct home {
  char name[FSIZE];
  char street[SSIZE];
  char city[CSIZE];
  long zip;
  float wage;
};

struct wage_earner {
  *struct employee name;
  *struct home address;
  char l_name[LSIZE];
  long zip;
  float wage;
};

Given the above, which of the following is "not" a valid variable access expression?

A. wage_earner.wage
B. wage_earner.address.zip
C. wage_earner.name.m_init
D. Fright.
E. "E" was not a given choice, please try again.

Answer "C" is the correct response.
First we need to declare a structure:

```c
struct address {
    char street[S_SIZE];
    char city[C_SIZE];
    long zip;
};
```

We can now declare an array of this type of structure:

```c
struct address student[100];
```

arrays of structures continued

The statement: "struct address student[100];" will allocate memory space for 100 structures of type "address". Each of these structures can now be accessed by using an array index and the structure member operator (.).

For example:

```c
student[49].zip will access the variable "zip" of the 50th structure (of type "address") in the array "student".
```

```c
student[9].city = "New York"; will assign the character string "New York" to the character array "city" of the 10th structure (of type "address") in the array "student".
```

 arrays of structures continued

given the declaration: `struct name {
    char f_name[F_SIZE];
    char m_init;
    char l_name[L_SIZE];
} roster[50];`

"template" structure of type "name" is declared and an array of 50 of these structures called "roster" is declared as well. (True or False)

very good.

Wrong. This is one way we have seen to combine the two methods of structure declaration.

*** Topic Review ***

In this topic we have looked at how structures are declared and used within other structures and we saw how to declare and use an array of structures.
Although we didn't look at very many or very involved examples of the uses of these two capabilities, I think that it is enough to introduce you to their use and will spark your ingenuity for programming applications. The rest of this lesson will discuss some other ways of working with structures in C programming.

In the next topic area (3) I will describe and show examples of how to use pointers to structures. In topic area four I will discuss how to pass structure data between functions.

*** This concludes this topic area. ***
float gross;
float fitw;
float s_tax;
float fica;
)
pay;

We can use the pointer declaration: struct income *p_pay; to declare a pointer "p_pay" that points to a structure of type "income".

Using the statement: p_pay = &pay; we assign the starting address of variable "pay" (of structure type "income") to variable "p_pay".

Given the structure declaration: struct address {
    char street[S_SIZE];
    char city[C_SIZE];
    long zip;
}
) home;

Which of the following will assign the starting address of the structure "home" to the pointer "p_home"?

A p_home = home;
B p_home = address;
C p_home = &home;
P_Fight.
D p_home = &address;

Wrong. Response "C" is the correct one.

"E" was not a given choice. Please try again.

Now that we have defined a pointer to the structure "pay", we need to learn how to use this pointer to access the members of the structure.

The way in which this is done in C is through the use of the a special operator which is composed of a minus and greater than sign "->".

For example, if we wish to access the variable "gross" of the structure "pay" in our example, we could use the expression: p_pay->gross

This, of course, would be used in a statement such as:

p_pay->gross = gross_pay; which would store the value of "gross_pay" in the memory location represented by "gross" within structure "pay".

C - 101
The special operator -> is provided as a shorthand way of accomplishing the same thing that the unary operator * does.

The statement we just saw, p_pay->gross = gross_pay; , could have been just as easily written as: (*p_pay).gross = gross_pay; and would have the same result.

The problem with using the unary operator * (asterisk) is that it has a lower precedence than the structure member operator . (period). Hence, you must use parentheses to ensure proper execution.

With this in mind it is easy to see that using the provided special operator -> is easier and clearer.

Given the structure declaration:

```c
struct address {
    char street[S_SIZE];
    char city[C_SIZE];
    long zip;
} home;
```

The variable "zip" can be accessed by using the expression: p_home->&zip

Provided "p_home" has been declared a pointer to type "address".

(True or False)

Very good. The correct expression is: p_home->zip.

No. The correct expression is: p_home->zip.

*Pointers to Structures Continued*

As a quick review.

If you have a structure declaration like:

```c
struct income {
    float gross;
    float fitw;
    float s_tax;
    float fica;
} p_pay;
```

And a pointer variable declaration like:

```c
struct income *p_pay;
```

You can access the individual structure members with expressions:

```
p_pay->gross
p_pay->fitw
p_pay->s_tax
p_pay->fica
```

In this topic area we have looked at how pointers to structures are declared and how to access the individual members of a structure using a declared pointer.
We have seen a couple examples to help illustrate this technique.

In the next topic area I will describe how to pass structure data between functions.

Hope to see you there!

### This concludes this topic area. ###

---

**Structures and Functions**

**Introduction**

In this topic area I will describe how structure data is passed between functions.

We have seen already how to pass variables as well as pointers between functions. Passing structure data is done in much the same way. We will look at a few examples to help illustrate this concept.

Let's get started!

**Passing Structure Data**

Using the structure we defined in the last topic area:

```
struct income {
    float gross;
    float fitw;
    float s_tax;
    float fica;
} pay;
```

One way to pass the data contained in the structure to a called function is to pass the structure members individually. For example:

```
compute(pay.gross,pay.fitw,pay.s_tax,pay.fica);
```

Calls function "compute" and passes the four members of structure "pay".

**Passing Structure Data Continued**

The called function would look something like the following in order to receive and use the passed variables:

```
float compute(gross,fitw,s_tax,fica)
    float gross,fitw,s_tax,fica;
```
42 {  
43     take_home_pay = gross - (fitw + s_tax + fica);  
44     return(take_home_pay);  
45 }

A second way to pass the structure data to the called function is to pass the entire structure. For example:
42 compute(pay); will pass the address of the beginning of structure "pay" to function "compute".

The called function would look something like the following in order to receive and use the passed structure address.

42 float compute(p_data)  
43     struct income p_data;  
44     {  
45         t_h_p = p_data.gross - (p_data.fitw + p_data.s_tax + p_data.fica);  
46         return(t_h_p);  
47     }

A third way to pass the structure data to the called function is to pass a pointer to the structure. For example, if you have a structure defined as:

42 struct income {  
43     float gross;  
44     float fitw;  
45     float s_tax;  
46     float fica;  
47 } pay;

Define a pointer variable with the statement: struct income *p_pay;

Assign the address to the pointer variable: p_pay = &pay;

Then call the function: compute(p_pay);
Which of the following is not one of the three ways of passing structure data to a called function?

A) Pass structure members individually.
B) Pass the structure template name.
C) Pass the entire structure.
D) Pass a pointer to the structure.

Your right.

ACD Wrong. Answer "B" is not a valid way to pass structure data.

B: 735

E) "E" was not a give choice. Please try again.

B: 730

**Lesson Five Summary**

Well, we have come to the end of lesson five. If you have seen the four subject topics in this lesson, you should now be ready to take the final test. If you feel that you don't understand something well enough to pass the test, please retake the topic that is giving you problems.

Topic 1 gave an introduction to structures and their use.
Topic 2 gave a description of structures within structures and arrays of structures.
Topic 3 gave a description of how pointers to structures are used.
Topic 4 described how structure data is passed between functions.

Welcome to the final test of lesson five. This test consists of seven questions over material presented in the previous four topic areas.
In order to successfully complete this lesson, you must achieve a minimum score of 71.4% (five out of seven questions correct).
If you miss a question, the correct answer will not be shown. It is up to you to research the correct answer.

Well, enough said. Let's get on with it. Good luck!
51Frame 905 OM
521. Which of the following can be used to declare a structure?
52A struct structure_tag { variable declarations );
52B struct: ( variable declarations ); structure name;
52C struct { variable declarations } structure name;
52D None of the above.
52E* Both "A" and "C" above.
53 Right. (1,110 & 125)
54 B:910
55 AB CD Wrong. (1,110 & 125)
56 B:910
51Frame 910 OM
522. Given the structure declaration: struct houses {
522 int num_wood;
522 int num_brick;
522 int num_stucco;
522 } resident;
522
52Which of the following is a way to access the variable "num_brick"?
52A houses.num_brick
52B houses.resident.num_brick
52C resident.num_brick
52D resident.houses.num_brick
52 Right. (1,135-140)
54 B:915
55 ABD Wrong. (1,135-140)
56 B:915
55 E "E" was not one of your choices.
57 B:910
51Frame 915 GF
522. In the C programming language there is no provision for the use of
522 structures within structures because it would require to much memory
522 overhead. (True or False)
523 Right. (2,305)
54 B:920
55 Wrong. (2,305)
56 B:920
51Frame 920 OM
524. Given the structure declaration: struct address {
524 char street[S_SIZE];
524 char city[C_SIZE];
524 long zip;
524 };
524
52Which of the following is a way to declare an array of 50 such structures?
53A array_of_address struct address[50];
53B+ struct address array_of_address[50];
53C struct array_of_address address[50];
53D address[50] struct array_of_address;
54 Right. (2,320-325)
54 B:925
55ACD Wrong. (2,320-325)
55 B:925
51Frame 925 QM
52. Given the structure declaration: struct address {
53        char street[S_SIZE];
54        char city[C_SIZE];
55        long zip;
56    } home;
57
58And the pointer declaration: struct address *p_home;
59
60The statement: p_home = &home; will assign the starting address of the
61structure "home" (of type "address") to the pointer "p_home".
(52. True or False)
52Y
63 Right. (3,510)
64 B:930
65 Wrong. (3,510)
66 B:930
51Frame 920 QM
57. Given the declaration: struct name {
58        char f_name[F_SIZE];
59        char m_init;
60        char l_name[L_SIZE];
61    } roster[50];
62
63Which of the following expressions can be used to access the variable
64"m_init" (Assume pointer "p_roster" has been properly declared.)?
65A+ p_roster->m_init
66B p_roster->m_init
67C p_roster->roster.m_init
68D p_roster->name->roster.m_init
69 Right. (3,515)
70 B:935
55BCD Wrong. (3,515)
55 B:935
55E "E" was not one of your choices.
55 B:930
51Frame 925 QM
57. Which of the following is not one of the three ways of passing structure
58data to a called function?
59A Pass structure members individually.
60B Pass the entire structure.
61C Pass the structure template name.
62D Pass a pointer to the structure.
63 Right. (4,706,715,720)
64 B:940
55ABC Wrong. (4,706,715,720)
55 B:940
55E "E" was not one of your choices.
55 B:945
This marks the end of lesson number five. I hope that it was of some benefit to you. I am looking forward to seeing you in lesson number six. I hope that you didn't have too much trouble with the material presented in this lesson. If you did, please voice your comments to your training monitor who will in turn contact the CAI Plans Branch at Keesler AFB, MS.

Well, let's take a look at how you did with the test ...
THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INTRODUCTORY INFORMATION ON INPUT AND OUTPUT CAPABILITIES OF THE C PROGRAMMING LANGUAGE.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Getchar and Putchar - This topic gives a description of the use of the standard I/O functions "getchar" and "putchar". (Approx. time = 5 min.)

Topic 2: Getline - This topic gives a description of the use of the standard input function "getline" and presents an example "getline" function. (Approx. time = 5 min.)

Topic 3: Scanf - This topic gives a description and examples of the standard input function "scanf". (Approx. time = 15 min.)

Lesson Breakdown Continued:

Topic 4: Printf - This topic gives a description and examples of the standard output function "printf". (Approx. time = 10 min.)
# Topic 5: Lesson 6 Test - This is the lesson test over items that have been presented in the previous four lesson topics.
(Approx. time = 5 min.)

TOTAL LESSON TIME IS APPROXIMATELY 40 MINUTES.

I hope that you enjoy it!

******************************************************************************

SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:

<table>
<thead>
<tr>
<th>STATUS</th>
<th>TOPIC #</th>
<th>TOPIC TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Getchar and Putchar</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Getline</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Scanf</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Printf</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Test Over Lesson 6</td>
</tr>
</tbody>
</table>

******************************************************************************

NOTE: A "STATUS" OF "*" INDICATES TOPIC SUCCESSFULLY COMPLETED.

******************************************************************************

11Frame 100 T Getchar and Putchar
12 *** Introduction ***
12
12 Input/Output (I/O) is "not" a part of the C programming language.
12 Statements such as Print, Write, or Read are "not" available for use.
12
12 The way in which you compensate for C's lack of I/O capability is to
12 make use of library functions supplied by the C compiler's manufacturer.
12
12 The types of functions that are provided with a specific C compiler
12 vary from manufacturer to manufacturer, so it is suggested that you
12 review your C compiler's documentation in order to determine what
12 functions you can make use of.
12
12 In this topic area we will take a look at some basic I/O functions
12 that most manufacturers provide.
128 105
11Frame 105 T
12 * Introduction Continued *
In order for you to have access to the standard I/O functions provided with your C compiler you may need to include a header file that contains the definitions and declarations needed by the I/O functions.

The file name that you include depends on the compiler you are using. Typically the include statement will look something like the following:

```
#include <stdio.h> OR #include <bdscio.h>
```

Please check your compiler's documentation for the proper header file to be included, if any.

In this lesson topic we will be discussing how to use the standard I/O functions "getchar" and "putchar". We will see examples of how these two functions are called and what they do. Let's get started.

*** Getchar ***

The function "getchar" is used to read one character at a time from the standard input device. The standard input device is by default the user's terminal keyboard.

Note: The standard input device can be changed on most systems, but how this is done will not be discussed in this course.

The format of the call to the function "getchar" is as follows:

```
c = getchar(); Where "c" is any variable of type "int".
```

What was that? Variable "c" is of type "int"! Well, that just doesn't sound right. Let's look at this a little closer.

The requirement that the variable that receives the character returned by the function "getchar" be of type "int" stems from the fact that "getchar" is a function that returns an integer value.

The only time you would run into problems in making the variable "c" a "char" type is if you were trying to detect an end of file condition. The reason for this is that EOF is typically equal to -1, which is of course an integer.

Thus, when the EOF is encountered it must be read into a variable of type "int".

For example, the following program will "not" work.
main()
    char c;
    while ((c = getchar()) != EOF)
        < some statement to deal with variable "c" >;
}

The proper way to write the program is:

main()
    int c;
    while ((c = getchar()) != EOF)
        < some statement to deal with variable "c" >;
}

As another example, the following program will work since no check is made against "EOF".

main()
    char c;
    while ((c = getchar()) != 'n')
        < some statement to deal with variable "c" >;
}

Here the terminating condition is when "c" is equal to the "newline" C escape sequence. As you can see, the requirement for the receiving variable of the function "getchar" to be of type "int" is not without exception. Just be aware of the fact that "getchar" returns an "int" type and this may cause you a problem if the receiving variable is not of the same type.

The "getchar" function is used to read one character at a time from standard input to the executing C program. (True or False)

Right.
Wrong. Wake up!

The function "putchar" is used to write one character at a time to the standard output device. The standard output device is by default the users terminal screen.

Note: The standard output device can be changed on most systems, but how this is done will not be discussed in this course.
The formats of the call to the function "putchar" is as follows:

- `putchar(c);` Where `c` is any character variable.
- `putchar(\c);` Where \c is any character constant.
- `putchar(\c);` Where \c is any C escape sequence.

For example:

```c
main() {
    putchar('I');
    putchar(\');
    putchar('I');
    putchar('I');
    putchar('k');
    putchar(\e');
    putchar(\c');
    putchar(\');
}
```

This program will write the sentence: I like C. to the standard output device (terminal screen).

As another example:

```c
char string[] = "I like C."
for (i = 0; string[i] != \0; i++)
    putchar(string[i]);
```

This program will also write the sentence: I like C. to the standard output device (terminal screen). The loop terminating expression will become "true" when the end-of-string marker (\0) is encountered.

Which of the following is "not" a correct way to use the "putchar" function?

- A. `putchar(c);` Where `c` is any character variable.
- B. `putchar(*c);` Where `*c` is a pointer to any character array.
- C. `putchar(\c);` Where \c is any character constant.
- D. `putchar(\c);` Where \c is any C escape sequence.

Very good.

ACD No. Answer "b" is the correct one.
This example shows how you can combine both the "getchar" and "putchar" functions to read & write a line of text from/to the standard I/O device.

```c
main() {
    char c;
    while ((c = getchar()) != '\n')
        putchar(c);
}
```

This program will terminate when the user hits the "Return" key at the end of his/her typed line.

In this topic we have looked at the standard I/O functions "getchar" and "putchar".

We have seen a few examples of how to access and use these functions and discussed a couple of things to be aware of in their use.

In the next topic area I will describe the I/O function "getline" and give a few examples of its use.

See you there!

This concludes this topic area.

In this topic area I will describe the I/O function "getline".

This function is used to read in one line of input from the standard input device (users terminal keyboard). In addition to reading a line of input, the "getline" function also keeps track of how many characters were read in.

We saw in the last topic area how to accomplish the reading of a line of input using the "getchar" function, but as you can well imagine, if you need to do this task in several points in your program it would pay to have a separate function defined which you could call.
Most C compilers have this function as part of its I/O library, but just in case your compiler manufacturer didn’t include it, I will present a version of "getline" that you can use in your programs.

The format of the call to the function "getline" is as follows:

```c
n = getline(input_line, 80);
```

Where "n" is any variable of type "int", "input_line" is a character array, and "80" is the maximum length of the array. When the above statement is executed the "getline" function will read a line of input from the users terminal keyboard. The above call will read in at most 78 characters. If the user were to type 78 characters and then hit the "Return" key, the actual contents of the "input line" array would be as follows:

```c
input_line[0] thru input_line[77] = characters (78 characters)
input_line[78] = \n (end of line character)
input_line[79] = \0 (end of string marker)
```

As I stated before, the "getline" function will keep track of the number of characters it reads in. What I didn’t mention is that it will return this number to the calling function if so desired.

In our example statement: `n = getline(input_line, 80);`

The variable "n" (of type "int") is where the number of characters read in is stored. This number will include the 78 characters of user input and the end of line character, but not the end of string marker. For our example this would give us a total count of 79.

How you use this number, if at all, depends on your programs application.

Which of the following is "not" true.

A. "n" must be a variable of type "int".
B. "getline" will return two values "n" and "input_line".
C. "input_line" must be a character array.
D. "80" is the maximum input line size.

Right. "getline" will return an integer value to "n", but the array "input_line" is passed as a pointer to array position input_line[0].

ACD Wrong. Answer "B" is the correct response. "getline" will return an
integer value to "n", but the array "input_line" is passed as a pointer to
array position input_line[0].

"E" was not a given choice, please try again.

Let's take a look at a sample program that uses the function "getline".

```c
main() {
    char input_line[80];
    getline(input_line,80);
    i = 0;
    while (input_line[i] != '\0') {
        putchar(input_line[i]);
        i ++;
    }
}
```

This program will read in one line of input from the user's terminal
keyboard and print the stored line (one character at a time) on the
user's terminal screen.

Now that we have seen how to use the "getline" function that is usually
provided with your C compiler by the manufacturer, let's take a look at
how you can define your own version of the "getline" function.

The following will perform the same as the "getline" function we have
just looked at and can be included in your programs if the "getline"
function is not available.

```c
getline(i,nln,max)
char in_ln[];
int max;
{
    int i,c;
    for (i = 0; i < (max-1) && (c = getchar()) != EOF && c != '\n'; i++)
        in_ln[i] = c;
    if (c == '\n')
        in_ln[i++] = c;
    in_ln[i] = '\0';
    return(i);
}
```
The maximum number of characters that will be read by the function "getline" is 35. (True or False)

Very good. 34 characters can be read. One character is used to store the end of string marker.

Wrong. 34 characters can be read. One character is used to store the end of string marker.

In this topic we have looked at the I/O function "getline" which may or may not be included with your C compiler's standard I/O library. We have seen a few examples of how to access and use this function and we saw a version of the function that you can include in your program if it is not available with your compiler.

In the next topic area I will describe the I/O function "scanf" and give a few examples of its use.

See you there!

This concludes this topic area.
The format of the "scanf" function call is composed of two parts: a format control string and the pointer arguments.

A skeleton of the function call looks like this:

```c
scanf("format control string", &arg_1, &arg_2, ..., &arg_n);
```

The format control string will be described in detail shortly. The arguments following the string must be pointers to the memory locations where the read in arguments are to be stored.

It is a fairly common mistake to try and read values into a variable by just specifying the variable name. This can not be done since "scanf" is a function and as such can only return one value. Thus, you must somehow pass it the address of where the variable is stored.

Let's now look at each part of the "format control string" of the scanf function call.

The "format control string" is made up of individual conversion specifications. Each of these conversion specifications "must" begin with a "percent sign" (%).
The next (optional) character is an "argument suppression character". This character is an asterisk (*) and indicates that the next input field is to be skipped. Thus, no assignment is made into the corresponding input argument.

The next (optional) part of the "string" is an "integer field width specifier" which is used to specify the maximum field width of the input.

The next (optional) part of the "string" is the "length modification character". This character can be one of two letters: 1 or h. These two letters can only be used in conjunction with certain "conversion characters" as will be described forthwith.

The last part of the "string" is the "conversion character or character class". The "conversion character" can be one of 13 different characters.

I will now give a brief description of each of these characters.

- d = decimal integer (argument should point to "int" variable type.)
- o = octal integer (argument should point to "int" variable type.)
- x = hexadecimal integer (argument should point to "int" variable type.)
- D = decimal integer (argument should point to "long" variable type.)
- O = octal integer (argument should point to "long" variable type.)
- X = hexadecimal integer (argument should point to "long" variable type.)
- e or f = floating point number (argument should point to "float" variable type.)
- E or F = floating point number (argument should point to "double" variable type.)
- c = character (argument should point to "character" variable type.)
- s = string (argument should point to "character array" variable type.)

As a refresher: Integer input: d, o, x, D, O, or X
Floating point input: e, f, E, or F

Character input: c

String input: s

Percent sign input: %

Let’s look at a couple of examples involving the "scanf" function call.

scanf("%d%f", &int_var, &float_var);

The above call will read from standard input (users terminal keyboard) two numbers of the types "integer" and "floating point real".

The users typed input numbers would be of the form: 23 45.78

The "scanf" function will read into the first argument ("int_var") until a "white space" character or a character that is incompatible with the specified "format control string" is encountered.

Note: A "white space" character is defined as a "blank", "tab" (\t), or "newline" (\n).

As another example: scanf("%s%c%d%%", s_array, &int_var);

This "scanf" call will read a "string", "integer", and "percent sign".

The user’s input would look something like this: Tax = 5%

The function "scanf" will read the word "Tax" into the array "s_array", then skip the character "=". then read the integer "5", and finally read the "percent sign". No space is needed after the "5" in the user’s input since the "percent sign" is not compatible with the "%d" format control string. The "percent sign" is not stored anywhere.

Given the function call: scanf("%d%f%s%c", &w, &x, &y, &z);

Which of the following variables will contain a number with a decimal point?

A. w
B. x
C. y
D. z
Right.

B: 555
One more point on the "format control string" that I promised to talk about, namely the "length modification character".

As I mentioned, this optional character can be either the letter 1 or the letter h. The "length modification character" can only be used with certain "conversion characters".

You may use the letter 1 with the conversion characters d, o, or x to indicate that the value being read in is to be stored in a "long" rather than "int" variable type. i.e., scanf("%ld",&lint);

You may use the letter h with the conversion characters d, o, or x to indicate that the value being read in is to be stored in a "short" rather than "int" variable type. i.e., scanf("%hd",&s_int)

As I mentioned during the description of the format of the "scanf" function call, the "format control string" begins with a percent sign and ends with either a "conversion character" or "character class".

We have seen what the "conversion character" is, but we still need to cover the "character class".

A "character class" is identified by a set of brackets [ ] following the percent sign. The "character class" is used in conjunction with a character array argument.

Let's look at two examples to demonstrate the use of "character class".

Example #1: scanf("%[abcdefghijkml]",valid_letters);

In this example an input string is read until a letter is encountered that "is not" in the "character class" specified. The character array "valid_letters" must be big enough to hold the read in input string.

Example #2: scanf("%[^abcdefghijkml]",valid_letters);

An alternate form of the "character class" uses a circumflex (^). When this form is used, the valid input becomes any character not specified in the "character class". Therefore, for example #2 above,
the input string will be read until a letter is encountered that "is"
in the "character class" specified.

In this topic we have looked at the I/O function "scanf" which is
usually included with your C compiler's standard I/O library.

We have seen a few examples of how to access and use this function
and discussed many of the special features of the function.

In the next topic area I will describe the I/O function "printf" and
give a few examples of its use.

See you there!

*** This concludes this topic area. ***

In this topic area I will describe the I/O function "printf".
This function is used to convert and print specified arguments to
the standard output device (user's terminal screen). In essence the
function is used to do formatted output.

We saw in the last topic area how to accomplish formatted input by
using the "scanf" function. We will now cover how to accomplish the
task of producing output from your C program in any form you like.

All C compilers should have the function "printf" as part of its I/O
library. Please check your compiler's documentation to be sure of
this function's availability.

The format of the "printf" function call is composed of two parts:
a format control string and the arguments.

A skeleton of the function call looks like this:

printf("format control string", arg_1, arg_2, ..., arg_n);

The format control string will be described in detail shortly.
The arguments following the string have two important restrictions:

1. Their "type" must agree with the corresponding conversion
control character within the "format control string".

2. The number of arguments must agree with the number of conversion control specifications in the "format control string".

The format control string will usually contain the conversion specifications to be applied to the output sequences being printed to the output device.

However, you may also use the "printf" function to print character sequences "character for character".

For example: printf("C is GREAT"); will print: C is GREAT

The format control string usually begins with a percent sign (%) and ends with a conversion character, but can begin with C character escape sequences.

For example: printf("\n\t%d",arg1); will execute a "new line" and a "tab", then print an integer.

The "printf" function call: printf("\nI Love C"); will execute a "new line" and then print the character sequence: I Love C (True or false)

Right. Good work!

Sorry, that is a true statement.
begin with a "percent sign" (%).

The next (optional) character is a "minus sign". The minus sign, if present, indicates that the corresponding argument is to be printed left justified in its field. If no minus sign is present then the argument is printed right justified.

The next (optional) part of the "string" is an "integer field width specifier" which is used to specify the minimum field width in which the converted argument is to be printed.

The next (optional) part of the "string" is a "period". The period is used to separate the "integer field width specifier" from the next field of the "format control string".

The next (optional) part of the "string" is an "integer precision specifier". This is used to specify the maximum number of digits to be printed to the right of the decimal point (in the case of "double and float" argument types) or the maximum number of characters (in the case of a "character string" argument).

The next (optional) part of the "string" is the "length modification character". This character is the letter "l". This letter can only be used in conjunction with the "conversion characters": d, u, o, x, f, e, a, i, c.

The last part of the "string" is the "conversion character". The "conversion character" can be one of 9 different characters:
- d = signed decimal notation
- u = unsigned decimal notation
- o = unsigned octal notation
- x = unsigned hexadecimal notation
- f = float or double decimal notation (precision default = 6)
- e = float or double scientific notation (precision default = 6)
- a = float or double using the shorter of e or f above
- s = string
- c = character

Let's look at a couple of examples involving the "printf" function call.

```c
printf("%d %f", int_var, float_var);
```
The above call will print to standard output (users terminal screen) two numbers of the types "integer" and "floating point real".

The users printed output numbers would be of the form: 23 45.78

The "printf" function will print the first argument ("int_var") and then print the second argument ("float_var").

Note: A "white space" or blank character is printed between the arguments since one space appears between the conversion specifications in the "format control string".

As another example: printf("\n%6.2f",float_var);

This "printf" call will execute a "new line" and then print a "floating point real" right justified in a field of 6 print positions with 2 digits after the decimal point.

The users output would look something like this: 2561.89

The function "printf" will print the value in "float_var" using the specified format unless more print positions are needed, in which case, more print positions will be used.

Given the function call: printf("%4d %-4.2f %s %c",w,x,y,z);

Which of the following variables corresponds to the printed output: HI

A w
B x
C+ y
D z

Right. HI is a string.

Well, we have come to the end of lesson six. If you have seen the four subject topics in this lesson, you should now be ready to take the final test. If you feel that you don't understand something well enough to pass the test, please retake the topic that is giving you problems.

Topic 1 gave a description of the I/O functions "getchar" and "putchar".

Topic 2 gave a description of the I/O function "getline".

*** Lesson Six Summary ***
Topic 3 gave a description of the I/O function "scanf".

Topic 4 gave a description of the I/O function "printf".

Welcome to the final test of lesson six. This test consists of seven questions over material presented in the previous four topic areas.

In order to successfully complete this lesson, you must achieve a minimum score of 71.4% (five out of seven questions correct).

If you miss a question, the correct answer will not be shown. It is up to you to research the correct answer.

Well, enough said. Let's get on with it. Good luck!

1. The "getchar" function is used to read one character at a time from standard input to the executing C program. (True or False)

   A: True.  (1,110)
   B: False. (1,110)

2. Which of the following is "not" a correct use the "putchar" function?

   A: putchar(c); Where c is any character variable.
   B: putchar(*c); Where *c is a pointer to any character array.
   C: putchar('c'); Where 'c' is any character constant.
   D: putchar('\c'); Where \c is any C escape sequence.

   A: Correct.  (1,135)
   B: Wrong. (1,135)
   C: "E" was not one of your choices.
   D: "E" was not one of your choices.

3. Given the function call statement: n = getline(input_line,80);

   Which of the following is "not" true.

   A: "n" must be a variable of type "int".
   B: "getline" will return two values "n" and "input_line".
   C: "input_line" must be a character array.
   D: "80" is the maximum input line size.

   A: Correct.  (2,305)
   B: Wrong. (2,305)
   C: "E" was not one of your choices.
   D: "E" was not one of your choices.
The maximum number of characters that will be read by the function "getline" is 35. (True or False)

Frame 925 OP

The format of the "scanf" function call is composed of two parts: "format control string" and the "pointer arguments". (True or false)

Which of the following variables will contain a number with a decimal point?

A: w
B: x
C: y
D: z

Given the function call: scanf("%d%f%s%c", &w, &x, &y, &z);

Which of the following variables will contain a number with a decimal point?

A: w
B: x
C: y
D: z

The "printf" function call: printf("\nI Love C"); will execute a "new line" and then print the character sequence: I Love C (True or false)

This marks the end of lesson number six and hence the end of the course. I hope that the lesson as well as the course was of some benefit to you.

I hope that you didn't have too much trouble with the material presented in this or any of the lessons in this course. If you did, please voice your comments to your training monitor who will in turn contact the CAI Plans Branch at Keesler AFB, MS.

Well, let's take a look at how you did with the test ...
File "EXIT"

# THE COURSE YOU ARE NOW LEAVING WAS WRITTEN BY CAPT FRANK DEMARCO
# IN PARTIAL FULFILLMENT OF HIS MASTERS DEGREE IN INFORMATION SYSTEMS.

#

# GGGGGGGG 000000 000000 DDDDDD BBBBBB YY YY EEEEEEEE
# GGGGGGGG 00000000 00000000 DDDDDD BBBBBB YY YY EEEEEEEE
# GG 00 00 00 00 DD DD BB BB YY YY EE
# GG GGG 00 00 00 00 DD DD BBBBBB YYYY EEEEEEE
# GG GG 00 00 00 00 DD DD BB BB YY YY EE
# GGGGGGGG 00000000 00000000 DDDDDD BBBBBB YY EEEEEEEE
# GGGGGGGG 000000 000000 DDDDDD BBBBBB YY EEEEEEEE

# FFFFFFFF 000000 RRRRRR NN NN 000000 WW WW !!
# FFFFFFFF 00000000 RRRRRRRR NNNN NN 00000000 WW WW WW !!
# FF 00 00 RR RR NN NN NN 00 00 WW WW WW !!
# FF 00 00 RR RR NN NN NN 00 00 WW WW WW !!
# FF 00000000 RR RR NN NNNN 00000000 WWWWWWWW
# FF 000000 RR RR NN NN 000000 WWWWWW 00

C - 128
VITA

Captain Frank W. DeMarco was born on 8 June 1954 in Wheeling, West Virginia. He graduated from St. Johns High School in Bellaire, Ohio, in 1972 and entered the Air Force at the age of eighteen. He was honorably discharged from the Air Force in 1976 and joined the Ohio Air National Guard. In 1978 he joined the Air Force Reserve Officer Training Corps at Ohio University in Athens, Ohio. He received the degree of Bachelor of Science in Education (Mathematics) in June of 1980. Upon graduation, he received his commission in the USAF. Entering active duty in July 1980 he was assigned to the 3300 Technical Training Wing (TCHTW) at Keesler AFB, Mississippi. His duties while at Keesler included working as a World Wide Military Command and Control System (WWMCCS) mobile training team member and as a course writer for the Computer Assisted Instruction (CAI) Plans Branch of the 3300 TCHTW. In May of 1984 he entered the School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Permanent address: 212 South 8th Street
Martins Ferry, Ohio
43935
REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/MA/85D-2

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6c. ADDRESS (City, State and ZIP Code)
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

6b. OFFICE SYMBOL (If applicable)
AFIT/ENC

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Computer Assisted Instruction Plans Branch

8c. ADDRESS (City, State and ZIP Code)
CAI Plans Branch
3300 TCHM/TTXZ
Keesler AFB, MS 39534

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
TTGXZ

11. TITLE (Include Security Classification)
See Box 19

13b. TIME COVERED FROM TO

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Computer Applications
Teaching Methods
Computer Aided Instruction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
Title: COMPUTER ASSISTED INSTRUCTION FOR THE "C" PROGRAMMING LANGUAGE ON THE ZENITH Z-100 MICROCOMPUTER SYSTEM

Thesis Chairman: Dr. Henry B. Potoczny
Professor of Mathematics

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Dr. Henery B. Potoczny

22b. TELEPHONE NUMBER
513-255-3098
The field known as "computer assisted instruction" or CAI as it is commonly called, has gained considerable interest and support since the advent of the microcomputer. More and more people, including those in supervisory positions are beginning to see the advantages, both cost and time, in having training available in the workplace. This study developed a training package for use on the Zenith Z-100 microcomputer. The package consists of six lessons and three programs. The six lessons cover various topics dealing with the "C" programming language. The objective of these lessons is to present an introduction to the "C" programming language. The three programs are written in the Pascal programming language and are used for the following functions:

1. Provide a means of displaying the lesson material.
2. Provide a means of checking student progress.
3. Provide a means of displaying course statistics.