AIRBORNE WARNING AND CONTROL SYSTEM (AWACS)
INTELLIGENT TUTORING SYSTEM (ITS)

David R. Strome
Systems Research Laboratories, Incorporated
2800 Indian Ripple Road
Dayton, OH 45440-3696

HUMAN RESOURCES DIRECTORATE
TECHNICAL TRAINING RESEARCH DIVISION
7909 Lindbergh Drive
Brooks Air Force Base, TX 78235-5352

May 1993
Final Technical Report for Period 1 April 1991 - 12 December 1991

Approved for public release; distribution is unlimited.
The Intelligent Training Branch of the Armstrong Laboratory supports and conducts research in intelligent tutoring systems (ITSs). The purpose of this project was to apply this technology in the construction of an ITS which applied to Air Force Specialty Code (AFSC) 17XX, Air Weapons Personnel. The specific domain included task conducted on the E-3B/C Airborne Warning and Control System (AWACS) aircraft. Results indicated the need for further development of the applicable software, but also indicated the plausibility for this type training in the selected domain.
Table of Contents

Background .................................................... 3
Developing Criteria ............................................... 3
Preparation ...................................................... 4
Benefits of an ITS ................................................ 5
Designing the ITS ................................................ 6
   ITS Paradigm ................................................. 6
   Knowledge Domain ............................................ 6
   Simulator ....................................................... 6
   Instructor Model ............................................. 7
   Student Model ................................................ 8
Necessary WD Knowledge/Skills .................................... 8
   Skill Levels ................................................... 9
Course Structure ................................................ 10
   Student Skills Assessment .................................. 10
   Lesson One .................................................... 11
Developing the Software ......................................... 12
AWACS ITS Program Documentation ................................ 12
   Silicon Graphics Software .................................. 13
      Implementation ............................................. 14
      Program Documentation .................................... 18
DEC VAX 780 .................................................... 22
      Implementation ............................................. 22
      Program Documentation .................................... 27
Recommended Development ....................................... 32

List of Appendices

Appendix A ..................................................... 3 SEP 91 Ltr from Capt. Fowler
Appendix B ..................................................... IQT-WD READING GUIDE
Appendix C ..................................................... Experience Questionnaire
Appendix D ..................................................... ITS Source Files
Appendix E ..................................................... ITS User’s Guide
Appendix F ..................................................... Decision Tree
AIRBORNE WARNING AND CONTROL SYSTEM (AWACS) 
INTELLIGENT TUTORING SYSTEM (ITS)

Background

The United States Air Force (USAF) at Tinker AFB, Oklahoma, trains USAF officers to fulfill the role of Weapons Directors (WDs) aboard the E-3B/C Airborne Warning and Control System (AWACS) aircraft. The AWACS platform supports Command, Control and Communications (C³) missions in the airborne environment. USAF Armstrong Laboratory, Crew Technology Division, Sustained Operations Branch (AL/CFTO), in cooperation with Systems Research Laboratories, Inc. (SRL), configured WD crewstations and affiliated systems in the Aircrew Evaluation Sustained Operations Performance (AESOP) facility to run defensive counter air (DCA) mission simulations. Teams of three WDs detect, identify, intercept, and destroy enemy aircraft attempting to attack friendly forces or penetrate friendly airspace. A Senior Director (SD) serves as the immediate supervisor in the chain of command. In a DCA mission, the SD oversees and coordinates WD efforts to execute the directives of the senior battle staff.

In cooperation with the Technical Training Research Division, Human Resources Directorate (AL/HRT), SRL began research and development of an Intelligent Tutoring System (ITS) in the Air Force Specialty Code (AFSC) 17XX C³ domain. The effort was to provide a proof of concept, with recommendations for future development.

Developing Criteria

One of the first tasks involved narrowing the instructional domain. The following criteria were established:

1. The domain was restricted to that of weapons controllers: WDs, SDs, Weapons Controllers (WCs), and Weapons Assignment Officers (WAOs).

2. Due to practical limitations on equipment and funds, interactive voice communications were not included.

3. The domain included areas that would fill an immediate need of the Air Force's 17XX operational training community.

4. The domain was limited in order to accomplish development in the time available.
5. The ITS was restricted to Initial Qualification Training (IQT). By focusing on initial training, changes in performance would be more dramatic and therefore easier to measure.

6. Finally, in order for the ITS to teach something more than a routine procedure, we wanted to incorporate some decision-making skills.

Using these criteria, training systems at Tyndall AFB and Tinker AFB were deemed appropriate for adapting to an ITS. At Tyndall AFB, all 17XXs receive their initial Air Force training in the weapons controller career field. After graduating from Tyndall’s introductory course, some 17XXs continue to Tinker for initial WD training in AWACS.

After initial contacts with training sites, we chose to focus on Tinker’s AWACS community. After a fact-finding trip to Tinker (11-13 August 1991), the main subject area for initial development was identified as Block I, AWACS WD Initial Qualification Training. This subject area is taught at Tinker in the mission simulations. It is an introduction to the switch actions a WD uses in performance of job tasks. Block I IQT met the first five criteria for narrowing down the domain for developing an ITS. In addition, the Director of Operations of the 552 Tactical Training Squadron indicated that an ITS for this instructional block would widen a traditional training bottleneck for AWACS WD training, thus fulfilling criterion # 3.

Criterion # 6, decision-making skills, was not met. However this criterion was of a lower priority for two reasons. First, only Block I IQT WD instructional materials were set and the AWACS WD training program was being rewritten to take into account lessons learned from Desert Shield/Storm. Second, there is an ongoing reorganization by the Air Force in the training of 17XXs.

Preparation

While at Tinker, SRL tried to ascertain how Instructor WDs (IWDs) taught and evaluated IQT WD students—particularly as related to Block I IQT WD training. Important findings included:

1. Time for task completion is ill-defined in early instructional blocks.

2. Number and types of errors are also ill-defined.

3. Window for task completion, while not critical at first, rapidly becomes more important than time for task completion. The size of the window remains ill-defined.
4. Fighter Weapons School is considered more important for good experience in initial categorization of student abilities and capabilities than any other prior experience.

5. Students are not accelerated through training, even if they show exceptional skill or aptitude.

6. Prior experience and training records play a part in evaluating students, but is an exception, not a rule.

7. Only some IWDs used a dual tracking task in training of Block I IQT WDs, but this technique was being institutionalized as part of the formal program. The dual tracking task consists of a single piece of symbology on a single track in an oval orbit. The student is required to keep the symbology on the track throughout the lesson and while performing the procedures. This dual tracking task is not objectively measured, but subjectively indicates to the IWD that the student WD learned the lesson well.

8. Most switch actions are taught and used for two simulations in a row and then not trained or used until a student progression test is administered.

9. Most student evaluation is subjective.

Benefits of an ITS

ITS can eliminate or ameliorate many obstacles to efficient and effective training, including those mentioned above. ITS can take full advantage of quantitative measures developed in cooperation with expert IWDs, allow individually paced escalation of complexity and volume of tasks, and implement consistent application of evaluation criteria. In addition:

- ITS allows better use of manpower by enabling an IWD to manage the training of many student WDs rather than two or three. At the same time, ITS effectively allows a one-to-one student-to-teacher ratio.

- Because of the self-paced aspect of ITS, training can proceed at a faster rate for more able or experienced students and allow less able students ample practice time and opportunities to hone their skills.

- As the need for rapid response escalates, a WD must increase the work pace. This increase can occur only where there is spare capacity. The optimum area for this increase is that of console operations. ITS can
objectively judge this spare capacity and thereby enhance WD training by developing high-performance knowledge and skills in console operations. ITS simulations can foster achievement of these skills through "consistent practice" (Regian, 1990).

- ITS allows a student WD to consistently build on recently acquired knowledge and skills rather than moving on to other tasks before these new skills are reinforced.

Designing the ITS

**ITS Paradigm**

The second task under this effort required the creation of a conceptual paradigm of an ITS. The five important elements that must be related to each other are:

- a Simulator,
- a Knowledge Domain,
- an Instructor Model,
- a Student Model, and
- an Intelligent Interface that links them together (Burns & Parlett, 1990).

These elements are graphically represented in Figure 1.
Figure 1. Intelligent Tutoring System Paradigm
Each of these five elements is interrelated to the others. A development in one element often requires a corresponding development in one or more of the other elements. In this way the different elements remain fitted together as a coherent whole. Take for example teaching the procedures for executing a Commit switch action. The procedural steps for executing this switch action must be included in the relevant Knowledge Domain. The Simulator must have this switch action and be able to execute it. The Instructor Model must indicate how, when, and where in the course of instruction to teach the Commit switch action. The how, when, and where of the instruction depends heavily on the Student Model. The Intelligent Interface must make the match of the Student Model to the student, pick the correct format of instructions for the lesson, execute the lesson on the simulator, and evaluate the student’s progress.

**Knowledge Domain**

For practical reasons, the Knowledge Domain element comes first. Specifically, this is identified as the 552 AWACW IQT WD Block I, Weapons Director Student Study Guide, Volume 1, of training course E3000BQODX. This guide introduces students to AWACS, the WD Multi-Purpose Console (MPC), and several switch actions, building a declarative knowledge base and a procedural knowledge base. The emphasis is on procedural knowledge base, that is, how to perform certain WD switch actions. A specific list of these switch actions is found in Appendix A, a letter from Capt. Fowler, representative of the 552 AWACW/DOQMWD office, which is responsible for the training course development at Tinker for WDs.

**Simulator**

The next element, the Simulator, is the nearest to completion. These are the C³ Generic Workstations (C³GWs) and the supporting AWACS Simulation software in the AESOP facility (Schiflett, Strome, Eddy, and Dalrymple, 1990). The C³GWs have most of the switch actions taught in Block I at Tinker AFB. The following list includes all the Block I switch actions that have not yet been implemented on the C³GW:

- Restricted Area
- RN/DES/NTN SD
- Add/Delete Airbase
- Area Define/Delete
- Corridor IFF SD
- Radar/IFF Tracking

These switch actions amount to 28% (6/21) of all the switch actions taught in Block I, IQT WD training. An estimated 280 man hours of programming development are needed to fully develop these switch actions. In all other respects, the Simulator element closely emulates the presentation and functions of an AWACS WD MPC.
Instructor Model

The Instructor Model element contains instructional goals, accounts for student attributes, structures the Knowledge Domain, and presents the structure. The goals center around transfer of knowledge from the Knowledge Domain to the student. Specifically, these goals are to teach the student:

1. The physical layout of the MPC, i.e., where the switch action buttons are located.
2. The procedural steps for executing MPC Console Checkout.
3. The procedural steps for executing the following switch actions:
   - Assign Console
   - Line
   - Coordinate
   - Restricted Area
   - RN/DES/NTN SD
   - Add/Delete Airbase
   - Locate SIF
   - Request SIF
   - Mode IV
   - Radar/IFF Tracking
   - Request/Assign IFF/SIF

   TD Index
   Circle
   Tactical Bearing & Range
   Bearing and Range
   Initialize Special Point
   Area Define/Delete
   Corridor IFF SD
   Initiate
   Reinitiate
   Assign/Defer
   ADS Panel Channel activations

These goals lay the foundation for evaluating the student’s performance. Acceptable execution of these procedures by a student WD indicates successful transfer of knowledge. However, defining acceptable execution involves several measures:

- time of complete task execution,
- time of execution of each step within the task,
- a time window for accepting task start and end, error toleration, and alternate step sequencing, when it exists.

The values that address these measures are not currently known with certainty. Therefore, our initial set of values for evaluation were determined by a Subject Matter Expert based on years of experience as a WD/SD/IWD. The set of values chosen for different students is based on student attributes.
Student Model

Accounting for student attributes requires recognizing appropriate student differences, setting appropriate goals based upon these student differences, and then selecting a mode of instruction that best matches both the student and the instructional goals. Recognizing appropriate student differences relies heavily upon the Student Model. A WD student is matched to his/her appropriate type in the Student Model. Once the student type is known, then the Instructor Model has the correct set of instructional goals to tie to that student type. Our prototype has only one set of instructional goals based on the lowest level of student type. Selecting a mode of instruction impacts how the Knowledge Domain is structured and how information is presented.

Necessary WD Knowledge/Skills

There are three distinct, but interrelated areas of knowledge/skill necessary to be a functioning and qualified AWACS WD. These are declarative knowledge, procedural knowledge, (Barr & Feigenbaum, 1981; Anderson, 1988) and operational skills.

Declarative knowledge refers to that specialized body of knowledge about the system, its purpose, components, events, and the relationships among them. A few examples are how radar works, the different types of intercept geometry, airpower doctrine, and brevity code words.

Procedural knowledge deals with task procedures needed to operate the equipment. Some examples include reading scope presentations, talking and listening using the communications equipment, and executing switch actions.

Operational skills encompass knowledge at both cognitive and meta levels (Burton, 1988). At the cognitive level, the WD needs to acquire some high level skills to cope with the demands of a complex task environment in order to properly apply the declarative and procedural knowledge in both normal and abnormal problem solving situations (Woods, 1988). Time management and cost-benefit analysis are two skills necessary for the proper coordination of multiple WD tasks in a complex and dynamic environment. Proper coordination is important because system events compete for the operator's attention. Other critical skills resulting from the dynamic, complex nature of the system include adaptiveness and disturbance management that are common to other complex systems. Two that are critical to the WD world include dichotic listening and three dimensional spatial orientation involving motion of objects while remaining stationary.

Meta skills concern knowledge about how to learn effectively. The body of knowledge the novice WD must learn can be overwhelming. Consequently, at the meta level, the WD
needs to know how to monitor the learning process and manage different activities to get the most out of training.

The declarative and procedural knowledge together form the domain knowledge the operator must have. Operational skill can be viewed as the operator's successful acquisition and application of the domain knowledge during training that transfers to the actual task environment.

Skill Levels

In each of the three areas, a WD is rated on a scale of 1-5, with 5 being the highest. Each numerical rating corresponds to a label ranging from 1-Naive, 2-Novice, 3-Journeyman, 4-Expert, and 5-Master. The following is a description of each label/rating:

1 NAIVE: Indicates a complete lack of knowledge and/or skill.

2 NOVICE: Has some knowledge and/or skill, but not enough to operate independently to complete assigned tasks in a timely manner. Does not recognize all patterns of stimuli. Does not know how to order behavioral actions in response to recognized patterns of stimuli.

3 JOURNEYMAN: Has enough knowledge (score 85% or better in written exams) and/or skill to operate independently to complete assigned tasks, but not always in a timely manner. Recognizes common stimuli variable patterns and applies domain rules of behavioral responses. Does not recognize situations in which the domain rules do not apply, and/or in which new variables or patterns exist. Perseveres in attempting to apply what is known. Not yet capable of solving difficult or complex problems or developing new rules.

4 EXPERT: Has superior knowledge and/or skill. Operates well independently to complete all assigned tasks in a timely manner. Recognizes common stimuli variable patterns and applies domain rules of behavioral responses. Recognizes existence of new variables and/or situations in which the domain rules no longer apply, and can usually develop a solution. Capable of solving most difficult or complex problems.
5 MASTER: Has superior knowledge and/or skill. Operates well independently to complete all assigned tasks in a timely manner. Recognizes common stimuli variable patterns and applies domain rules of behavioral responses. Recognizes existence of new variables and/or situations in which the domain rules no longer apply, and can develop new rules. Capable of solving difficult or complex problems.

The goal of the training in Block I AWACS WD training is to bring all students up to the NOVICE/level 2 for Procedural Knowledge, Declarative Knowledge, and Operational skill.

Course Structure

Since the Knowledge Domain consists largely of Procedural Knowledge, the required step sequences are already structured. Yet there is a need for a comprehensive course outline that addresses the goals of Procedural Knowledge to be taught in each lesson, how many lessons there will be, and how fast the pace of presentation will be. The course outline is currently taken from Appendix A and Appendix B, the IQT-WD READING GUIDE (6 JUN 90). Note that the maximum time allowed for the course is specified, but no minimum time is established.

Student Skills Assessment

For this proof of concept prototype, we developed a Student Skills Assessment module to categorize student WDs (from 1-Novice to 5-Master) according to experience and ability. This module consists of an Experience Questionnaire and a Switch Action Exercise that students complete on the console prior to the first lesson. A printed copy of this questionnaire is included as Appendix C. Although it is highly probable that the students taking this course are novice WDs just graduated from the introductory course at Tyndall AFB, exceptions occur with sufficient frequency that they must be addressed. For example, occasionally a student may have extensive experience either as a WD or as an aircraft controller and must be requalified as required by regulations. The multiple-choice questionnaire ascertains the same information usually gleaned from the student's records or informally gathered by the instructors during the lesson breaks. In addition, where the student indicates a proficiency above basic entry level, the Switch Action Exercise is presented to validate the evaluation. The normal Lesson One, Block I instruction is then presented.

The first lesson consists of three parts and is presented at only one pace of instruction. The Instructor Model does not evaluate the student during the lesson, only after the lesson is complete. The capability to evaluate the student during the lesson and the capability to change the lesson mode if necessary, need to be more fully developed.
Lesson One

The Simulator Model is the means of instructional presentation and involves five phases of instruction in this high performance Knowledge Domain (Fink, 1990):

1. Static overview knowledge,
2. General procedure-oriented knowledge,
3. Guided example exercises,
4. Unguided example exercises, and
5. Automated example exercises.

Static overview knowledge consists of a general description of the salient parts and features of the particular piece of equipment on which the task will be performed. This static overview is not included in the prototype. Instead, it is left to the written material found in the IQT WD Student Study, Vol. I.

General procedure-oriented knowledge consists of a description of the steps that must be performed in executing the procedure being taught. Parts of the equipment involved and the motivation or effect for each step are indicated. Each of the lessons should start here for the lowest level student. The first part of Lesson One describes the steps and indicates where the switches are. It doesn't give any motivations for each step or each step's effect on the overall status of the goal.

Guided example exercises provide the student with the opportunity to practice with specific examples while being prompted and coached in order to develop accuracy in the skill. The second part of Lesson One presents a procedure the student must perform on the Simulator. Instructions guiding the student are presented, but the student is not evaluated during the lesson.

Unguided example exercises provide the student the opportunity to practice the whole process with specific examples without interruption in order to develop speed with accuracy. This is the third and final part of the Lesson One. The student is given ten minutes to execute the Console Checkout while reading and following the Console Checkout checklist. After ten minutes, the lesson is terminated and the data on student performance is gathered and evaluated. In future ITS development, this process should be automated.
*Automated-example exercises* provide the student the opportunity to practice the entire process with specific examples while doing another task. These types of exercises allow the WD to develop the capability to perform tasks automatically. The first lesson is not designed to use this phase. However, Console Checkout is done at the beginning of each and every lesson. As the lessons proceed, the time for this procedure’s completion will narrow down from ten minutes to three minutes.

**Developing the Software**

The development of this prototype was accomplished with a minimum amount of coding change to the existing AWACS Simulation software. Rather than having the two pieces of software integrated, the ITS software executes as a separate process from the simulation, but uses the data collected in the Logger File as input for evaluation. The intent is to have both pieces of software execute in an alternating fashion so the results of one can be used in the next execution of the other. Hence, the ITS software brackets the simulation to provide the appearance of an imbedded simulation without actually accomplishing the imbedding process. On the front end, the ITS software provides the interaction necessary to instruct and evaluate the student before selecting an appropriate scenario for the AWACS Simulation. Then on the back end, the ITS software evaluates the student’s performance during the simulation before presenting the appropriate follow-on instruction area.

Admittedly, using two computers systems to provide the functionality for the AWACS ITS is not the ideal solution. However, it does provide a flexible test platform upon which to develop some quantitative metrics for estimating the skills of the student WD. While pedagogy suggests the development of high-performance skill is best accomplished via consistent practice, establishing the criteria to judge the WD student still requires significant amounts of definition and refinement.

**AWACS ITS Program Documentation**

The software for the AWACS ITS was developed on both the UNIX based Silicon Graphics (SG) 4D/50 and the DEC VAX 780 computers using C and the *Curses* screen handling package. With the exception of the simulation graphics (SG only), all user interaction can be accomplished using either the SG graphics or DEC VT200 series terminals. Source files are included as Appendix D.

The AWACS ITS operates as three separate parts.

1. The SG 4D/50 software provides the instructional capability for the ITS.
2. The Simulation software provides the hands-on aspect of the WD's console operation.

3. The VAX 780 software is the first iterative trial to evaluate a WD's actual console operation.

Since the Simulation software is documented elsewhere, it will not be reiterated and documentation of the remaining two functions follows. An ITS User's Guide is included as Appendix E.

**Silicon Graphics Software**

As developed, the software exists in 7 source files and have the following functional relationships:

- itsdef.h: Header file for the ITS software.
- ITS.c: The overall controlling function
- its_rutns.c: Common functions shared by the software
- instructor.c: Controls the presentation of the test material.
- instr_rutns.c: Common functions shared in the instructor function.
- student.c: Provides the student interface and student function.
- stdt_rutns.c: Common functions shared in the student function.

All source and executable files exist on the SG 4D/50 called Picard. In addition, there is an ancillary file, called ITS.mak, that can be used with the UNIX Make utility to control the compilation and linking of the modules that are required to form the executable.


---

Implementation

In order to provide a means of presenting a flexible curriculum, the instruct.blk file was created. This file exists in the Instructor directory and contains a catalog of available instruction blocks (max = 50) and the individual lessons (max = 50) that comprise each instruction block. Each catalog entry (logical record) is a string of 634 characters that consists of 52 fields as follows:

1. Brief description of the instruction block (max = 80 characters).
2. A two digit field containing a count of the lessons in this instruction block.
3-52. 50 10-character fields containing the names of the lesson files.

Additionally, each field is separated by a ". As developed, each catalog entry can thought to be an instructional block comprised of one or more lessons. Wherein, each lesson consist of a series of text and questions.

Since the intent was to create a subject matter independent ITS, the majority of the intelligence has been placed in the formation of the lesson file. Consequently, the lesson file has a "programmable" flavor to its format and usage. Currently, each lesson file can be created by specifying the following types of text_types via the use of the "vi editor".

a) Each text type is separated by a Ctrl-L character. In addition, this character also causes a form feed when the file is printed.

b) The first line of each text type has the following format:

\[\text{numeric}_\text{id} \text{ text}_\text{type}\]

where:

\[\text{numeric}_\text{id} = \text{a decimal value that uniquely identifies (within a given lesson context) the text}_\text{type}.\]

\[\text{text}_\text{type} = \text{one of the following:}\]

1. text
2. multiple choice
3. true/false
4. yes/no
5. noscore
6. score
7. instruct/lesson
For example, suppose you wanted to specify a multiple choice presentation and 4 other displays already exist in the lesson. Additionally, let the previous presentations be numbered 1, 10, 20, and 30, respectively, then this multiple choice presentation could be specified as:

```
12 multiple choice
```

c) The second line of each text type has the following format:

```
numeric_id1 numeric_id2 numeric_id3 numeric_id4 numeric_id5
```

where:

- `numeric_id` = the numeric_id of a text type that is associated with a given "knowledge level" where level 1 corresponds to the lowest and 5 to the highest.

Continuing the previous example, suppose we plan to advance to text type identified as 11 for knowledge levels 1 through 3, and 15 for all others. Our example would now appear as:

```
12 multiple choice
11 11 11 15 15
```

d) If the text type is that of text, then the third line contains a count of the number of lines in the presented text that follows it.

Varying our example slightly to account for the text specification, the example would appear as:

```
12 text
11 11 11 15 15
36
```

This is a test example of the text ...

where the ellipsis indicates the continuation of 36 lines of text as specified by the third line of our example.

e) For the remaining text types, the third line consists of multiple occurrences of the format:

```
z) y
```
For the score and noscore types, the $y_n$ values are maximums for intervals that are used in the determination of upgrades in knowledge level, i.e.,

- $0 < \text{score} \leq y_1 \rightarrow \text{knowledge level 1}$
- $y_1 < \text{score} \leq y_2 \rightarrow \text{knowledge level 2}$
- $y_2 < \text{score} \leq y_3 \rightarrow \text{knowledge level 3}$
- $y_3 < \text{score} \leq y_4 \rightarrow \text{knowledge level 4}$
- $y_4 < \text{score} \leq y_5 \rightarrow \text{knowledge level 5}$

For example, these options could appear as:

```
12 score
11 11 11 15 15
 1)10 2)20 3)30 4)40 5)50
```
or

```
12 noscore
11 11 11 15 15
 1)10 2)20 3)30 4)40 5)50
```

Whereupon, if the student's accumulated score were 23, then the next display would be the one that is annotated as 11.

For the instruct/lesson type, the $z_n$ is the index of an instruction block and the associated $y_n$ is also an index of the appropriate lesson within the instruction block.

An example of this type may appear as:

```
12 instruct/lesson
1 1 1 10 20
 0)0 0)0 0)0 1)0 2)0
```

Whereupon, if the student were estimated to have a knowledge level of 5, then the next instruction block to be presented has an index of 2 within which the lesson having the index of 0 will be used. Furthermore, the display annotated as 20 will be the beginning presentation.

f) For the text type of multiple choice, true/false, and yes/no, the fourth line specifies the number of lines in the text that follows it.

g) The text type of score and noscore resulted from attempted use of a lesson. The noscore type uses the accumulated score to branch to an appropriate
follow-on text/question. Whereas, score updates the knowledge level variable in the student's data base and resets the accumulated score to zero.

Each of the lesson files that are created using the above format must also reside in the Instructor directory along with the file labeled welcome. This file is another lesson file and contains the queries for the Experience Questionnaire.

In addition to the Instructor directory and its associated files, a Student directory must also exist. This contains a file that is associated with each student that uses the system. The purpose of each student file is to house the critical values from each student's run. However, these critical values have yet to be determined so each student file is for all practical purposes empty.

Each student's file is labeled as sdb_xxxx. Wherein, the xxxx field consist of the last four digits of the student's SSAN.

Program Documentation

itsdef.h

This is the header file for this portion of the ITS software. It contains the definitions of the variables and structures that are common to a large majority of the modules.

ITS.c

This file contains the basic controlling logic for the appropriate calling of the Instructor and Student functions. It also presets some of the global variables.

its_rutns.c

This module contains those functions that are, generally, common to all the other modules and functions in this portion the ITS.

Contains the following functions:

```c
void cntr_in(int y_coord, char *txt_str)
    This function will center the text in txt_str in the line specified by y_coord.
```
void get_path_str(char *directory, char *path_str)
    This function queries the user for the full directory path for directory and returns the full path string in path_str.

void file_str(char *path, char *filename, char *fullstr)
    This function returns the full file string in fullstr from path and filename.

void its_stop()
    This function causes the appropriate escape from Curses and terminates the execution of the program.

int chk_file(char *path, char *filename)
    This function checks the accessibility of the file specified from the conjunction of path and filename.

void lesson_blk_io(char io_type, int ndx)
    This function provides the read/write function for the instruct.blk file. (See section on the instruct.blk for full description of this file.) By specifying a R or W for the read or write function, respectively, this function will read/write the record that is indexed by ndx.

void work_msg()
    This function presents a blinking "Working..." message while some time consuming operation is being performed.

void getstr_echo(char *str)
    This function echoes the contents of the character string str as it is being entered.

void strg_blnk_pad(char *string, int str_len)
    This function pads a left justified string, string, with blanks until the specified length, str_len, is attained.

instructor.c

As currently implemented, this routine provides the maintenance functions (add, delete, and modify) for the entries in the instruct.blk file. Consequently, it is used to govern the flow of presentations for the ITS.

As seen on the main selection menu, provisions do exist for:

a. reviewing a prepared instruction block,
b. reviewing a student's record, and
c. adjusting a student's record, i.e., examining the scores that have the student has accumulated and adjusting these values as required.

Regrettably, none of these options were implemented because of the lack of quantifiable information concerning the criteria that could be used to appraise the skills of a WD student. It is hoped that with continued development these criteria will be further defined. Whereupon or concurrent with this development, the contents of the student record would be established as well as the methods and content of the material to be taught.

instr_rutns.c

These routines support the instructor.c.

void lesson_blk_rec_init(int ndx)
    This function initializes the record in instruct.blk that is specified by ndx.

int lesson_list(max_cnt)
    This function presents a list of the available instruction blocks, as specified by max_cnt, in instruct.blk and returns the index of the selected record.

void lesson_blk_chk()
    This function is a debug tool that will display a selected record from instruct.blk.

void instr_blk_vis(int ent_no)
    This function displays a record from instruct.blk that is specified by ent_no.

void make_pos(int ndx, int *y_pos, int *x_pos)
    This function will provide the screen coordinates, x_pos and y_pos, relative to the instr_blk_vis display that corresponds to index ndx of the desired lesson.

student.c

This routine uses a lesson file to present the material in a manner that resembles the traversal of a decision tree. The working version of this decision tree is included as Appendix F. The combined use of knowledge level and accumulated score are the attributes that key the order of presentation to the student.

It should be noted that there exists a software imposed limit of 100 entries (occurrences of text type) per lesson file. This limit can be easily changed as required. Also, the software does not preclude the concatenated presentation of multiple lesson files.
Consequently, the 100 entry limit can be circumvented by the subdivision of a large lesson file into smaller files.

stdt_rutns.c

Contains the following functions:

void student_ident(char *name_str, char *id_str)
   This routine queries the user for the student's name and SSAN and returns the values in name_str and id_str, respectively.

void get_ssan(char *a_strng)
   This function controls the user's specification of a SSAN and returns the value in a_strng.

void student_blk_io(char io_type, int recndx)
   This function provides the equivalent I/O capabilities as found in lesson_blk_io, but these apply to a student data base.

void read_prsnt_txt(int map, int *file_offset, int *disp_map, int *disp_ndx, char *disp_str)
   This routine reads a line of text, pointed to by file_offset, from a streamed file into a buffer that is pointed to by disp_str. If map equals a -1, then the entry that is indexed by disp_ndx in the seek address list, disp_map, is built. And in all cases, the next value of file_offset is computed.

int prsnt_txt(int mapndx)
   This routine presents a text that is pointed to by mapndx, which is an index into another list that contains some positional information about each block of text in a given lesson.

int score_txt(int mapndx)
   This routine uses the lesson map entry, indexed by mapndx, to score the preceding portion of a lesson block. It also asks the user to review any preceding answers before scoring the section.

int lesson_txt(int mapndx)
   This routine uses the lesson map entry, indexed by mapndx, to determine the next instruction/lesson block to be presented based upon the user's knowledge level.
void get_max_min(int map_ndx)
This routine uses the lesson map entry, indexed by map_ndx, and
determines the highest and lowest values that are in the answer set. The
sum of the max and min values are stored in the user's data base.

void reply_pos(int map_ndx)
This routine uses the lesson map entry, indexed by map_ndx, to position the
cursor in the appropriate screen position for a user's response to either a
multiple choice, true/false, or yes/no type of question.

DEC VAX 780

As stated earlier, the interviews with several IWDs were inconclusive in the attempt to
establish some quantifiable criteria for measuring the developing skills of a student WD.
But these interviews did indicate that a common appraisal basis did exist in a non-verbal
context. Consequently, the approach of this development was to provide a first
approximation and target for criticism using "timely completion of an action" as a starting
criterion in order to elicit critiques from the IWDs. In turn, these comments would be used
to improve the criteria, find new metrics, and improve the expert model. Hence, several
versions of the software are expected to evolve. So rather than saying that the software
"evaluates ...", the term "attempts to evaluate ..." is applied to connote that major revisions
to this software are expected with the possibility that our initial criteria could be
decomposed, tossed out, or embellished.

As developed, this software exists in 3 files and have the following functional relationships:

sdt_eval.h: This is the header file for the DEC VAX 780 portion of the software.
sdt_eval.c: This is the overall controlling function for this portion of the software.
           It attempts to evaluate the switch responses of the WD user.
eval_rtns.c: These are the common functions that are used in sdt_eval.c during
             the evaluation process.

Implementation

In order to evaluate the output from an AWACS scenario in the most flexible manner
possible, the software uses a script to evaluate a WD's console operation skills. This
script can consist of one or more user-supplied events that are described to the software
by the use of keyword values. All these values are taken from either a streamed file
called switches.dat or are part of the software.
The keyword values that are taken from switches.dat consist of the names, coded values, and code ids of the:

a. Feature and Category Select switches
b. Alarm/Display Control Panel switches
c. Function Select Panel switches

All fields are left-justified and separated with the | character to assist with visual discrimination. The format of each record is as follows.

Field 1 (27 characters): This is the name of the switch as they are identified on a WD's workstation console.

Field 2 (4 characters): A 4-digit field that corresponds to numerical value of the switch as it is encoded in the Simulation software (See file switches.h).

Field 3 (3 characters): When available, these characters correspond to an abbreviated value used by the switches.c software in its output to the Logger file.

For example the specification of Bearing & Range function switch appears as:

BEARING RANGE | 23 | B-R

Those keyword values not taken from the switches.dat file are coded into the software. These consist of the terms:

a. Checkout Console: Used to demarcate an event consisting of the Console Checkout procedure.

b. Done: Used to indicate the termination of a script.

c. Window: Used to specify a maximum period of time (no. of seconds) during which an event should occur.

d. Key: Used to specify the numerical value that is associated with an event on a logger file.

e. Text: Used to specify a text value that can be used for additional discrimination.
f. **Alternate**: Used to specify a function switch action. In addition, this option can be specified in groups such that each group represents another method of performing an equivalent action.

Each user-supplied event is listed in a file and separated from each other by an empty line. In addition, each event is tagged with a field [1-4 characters] that must start with a numeric and is terminated with a :. The follow-on field can be either:

a. A label [1-27 characters]
b. Keyword value Checkout Console, Done, or those that are found in the switches.dat file.

Thus, an event is denoted by a line that has the follow specification:

```
tag: {label, appropriate keyword}
```

where one of the fields enclosed between {...} is necessary, but interchangeable. Additionally, if a label is used, then the use of the key, text, and alternate options are also required.

For example, suppose we wanted to describe the Console Checkout event follow by some event called "Find and Tag a Tanker." These could appear as:

1: Checkout Console
   : Other options

2: Find & Tag Tanker
   : Other options

The next line that is associated with the specification of an event has the following format.

```
window = xxx.xx
```

where the field xxx.xx denotes the specification of a floating point value indicating the number of seconds within which the associated event must be completed.

In our example this specification would appear as:
1: Checkout Console
window = 180.0

: Other options

2: Find & Tag Tanker
window = 60.0

: Other options

The key option line has the following format.

key = xxxx

where the field xxxx is an integer value that corresponds to a keyword value that is found in the keywords.h file from the Simulation software.

For our example, if each event were tied to a message, then they would appear as:

1: Checkout Console
window = 180.0
key = 1014

: Other options

2: Find & Tag Tanker
window = 60.0
key = 1014

: Other options

The text option line has the following format.
text = '{text string (1-27 characters)}'

where the text string that is enclosed between single quotes consist of the same or all of the leading characters in the free text portion of a logger file record.

Again, in our example this specification would appear as:

1: Checkout Console
   window = 180.0
   key = 1014
   text = 'a text string'

2: Find & Tag Tanker
   window = 60.0
   key = 1014
   text = 'another text string'

Other options

The specified key and text values are used to find the next occurrence of these values in the logger file and constitutes the start of an event. Whereupon, the use of one or more groups of the alternate option takes place.

For the Checkout Console, the use of the next option is recognized. However, it is limited to the specification of the Assign Console switch action. This feature was added so that the Assign Console switch action could be tied to the Checkout Console event.

Each alternate option line has the following format.

   alternate {tag} = {switch action keyword}

where:

   tag = 1-8 character field that must start with a numeric.
   switch action keyword = the name of a switch that is found in the switches.dat file.

A group can consist of one or more repetitions of the alternate keyword. For example, suppose an event called "Find & Tag a Tanker" could be identified by the message "01 FIND & T", and could be accomplished using either the Request SIF SD, Locate SIF, or
Re-Init switch actions. In addition, suppose that this event had to be done in one minute from the time of the message. The event could appear in a script as:

```
1:  Find & Tag a Tanker
    window = 60.0
    key = 1014
    text = "'01 FIND & T"
    alternate 1 = Request SIF SD
    alternate 2 = Locate SIF
    alternate 3 = Re-Init
```

Now suppose that a fourth method of accomplishing the same event required two switch actions called Switch 1 and Switch 2, then the same event would appear as:

```
1:  Find & Tag a Tanker
    window = 60.0
    key = 1014
    text = "01 FIND & T"
    alternate 1 = Request SIF SD
    alternate 2 = Locate SIF
    alternate 3 = Re-Init
    alternate 4 = Switch 1
    alternate 4 = Switch 2
```

It should also be noted that the total number of alternate options that can be specified in any single event has a software limit of 25.

Another file of interest is the logger file. This file is a derivative of the original logger.out file that is produced during an execution of the Simulation software. This logger file is the output of the Pass 6 operation from the REDUCE software and represents all the captured data in a time order manner.

Program Documentation

sdt_eval.h

This is the header file for this portion of the ITS software. Like itsdef.h, it contains the definitions of the variables and structures that are shared by the programs in the application, i.e., sdt_eval.c and eval_rtns.c.

---

Of the structures that are included in this file, the \texttt{evnt skelt} specification and its use as \texttt{evnt desc} requires some explanation. As \texttt{evnt desc}, this structure contains the controlling information for the analysis of an event. The values are initialized and loaded from the event script file for each event at the onset of processing. The following is an explanation of this structure's usage.

\texttt{evnt seq}: This is a 4 character array that contains the tag value that is associated with the event.

\texttt{struct evnt var evnt label}: This structure contains the information extracted for the event demarcation line, the \texttt{Window}, \texttt{Key}, and \texttt{Text} specification options.

\texttt{no var}: The contains the number of variations that were specified via the \texttt{alternate} options. Suppose that the following event specification were used.

1: Find & Tag a Tanker
window = 60.0
key = 1014
text = "01 FIND & T"
alternate 1 = Request SIF SD
alternat e 2 = Locate SIF
alternat e 3 = Re-Init
alternat e 4 = Switch 1
alternat e 4 = Switch 2

The value of \texttt{no var} would be 4.

\texttt{step per var}: This is an array of 5 integer values. Each represents the number of function switch actions that comprise each specified alternate. In the above case, the assigned values would be 1, 1, 1, 2, and 0 for each of the entries in \texttt{step per var}.

\texttt{struct evnt var variant}: This is an array of structures that contains the function switch action information for each of the \texttt{alternate} option specifications. Again for the cited example, the first 5 structures would contain the appropriate values.

\texttt{sdtEval.c}

This program provides the controlling logic for the evaluation of the acquired console operation data.
This is a collection of routines that perform a partial evaluation of the data collected in the logger file.

```c
void cntr_ln(int y_coord, char *txt_str)
    This routine is identical to that found in its_rtns.c. It centers the text in txt_str in the line specified by y_coord.
```

```c
void quick_exit()
    This routine calls the endwin() function to clean up the Curses software before exiting.
```

```c
void get_fil_str(char *filename)
    This routine prompts the use for an appropriate filename and store the requested string in filename.
```

```c
void open_err(char *filename)
    This routine prints a message indicating that an error occurred while opening the file specified in filename.
```

```c
void read_swt_data(FILE *switch_fp, struct switch_ent *swt_ptr)
    This routine reads a record from the file pointed to by switch fp, switches.dat, loads the values into the entry of the table that is pointed to by swt_ptr.
```

```c
void read_log()
    This routine reads a line from the logger file and partially parse the data into the appropriately labeled variables found in the structure of cur_log_rec. It also ensures that all fields are null terminated strings.
```

In addition, this routine also counts excessive pushing of the message acknowledge button. (The term "excessive" means pushing the button when no message acknowledgement is required.) Three or more excessive depressions will trigger the start of the computation to determine the tapping variability and the occurrence of the next message will trigger the print of the variability statistic.

```c
void left_just(char *txt_str)
    This routine will left-justify the text in txt_str.
```
int swt_label_match(struct switch_ent *swt_tab_ptr,
int swt_cnt, int ent_no)
This routine will search the function switch table, with swt_cnt many entries,
that is pointed to by swt_tab_ptr for a matching value in the evnt_desc
structure. If ent_no is a -1, then the switch keyword specified in the event
demarcation line is used. Otherwise, the name specified in the alternate
line is used. Moreover, along with the indication that a match has occurred,
the associated switch number and abbreviated switch name are also
returned.

void blink_msg_on(int line_no, char *txt_str)
This routine will present the text in txt_str in a blinking format on the line
specified by line_no.

void blink_msg_off(int line_no)
This routine clears and turns off the blinking message on line number
line_no.

void read_event()
This routine reads the event script file and loads the evnt_desc structure that
describes the desired event to the software. It is the values in this evnt_desc
structure that are used to evaluate an event.

void opt_chk()
This routine is called by read_event to load the evnt_desc structure with the
values taken from the Window, Key, Text, and Alternate option specifications.

void swt_state()
This routine gets the state (on or off) of the category and feature select
switches as they exist prior to the use of the simulation by the WD.

void console_chk()
Currently, this routine attempts to identify the correct execution and
completion of steps that affect the category and feature select switches on
the Console Checkout checklist. Consequently, this routine's execution is
contingent upon the specification of the Checkout Console event in a script
file.

However, certain empirical assumptions were made as to the action that
constitutes the end of the all switches on and all switches off process.
Currently, the completion of the all switches off is indicated when the first
feature select switch goes to the on position and the all switches on is
indicated when the first function switch action is taken.
In addition, the inclusion of the alternate option specifying the Assign Console switch action is allowed with the event specification. Hence, taking this switch action is an optional part of the Console Checkout process.

```c
void switch_chk(struct evnt_var *swt_ent)
    This routine checks for the selection and complete status return of a function switch action, pointed to by swt_ent, within a specified time window. This check also exists for a follow-on switch action in which the antecedent switch action remains in effect.

In addition, some very minor checking of some associated switch options were included for only a few of the function switches. This was done as a check of the feasibility of including more expanded option checking.

void csl_chk(int ndx)
    This routine is one of the attempts to expand the option checking capability of the application. It tries to examine the Assign Console switch action for the WD station indicated by ndx.

void cmt_chk()
    This routine is an attempt to expand the checking associated with the Commit switch action. It tries to identify the pairing, intercept, and mission that were specified.

void win_chk()
    This routine is an attempt to expand the Init switch action. It tries to determine the object whose symbology was initiated.

void wmc_chk()
    This routine is an attempt to expand the Mode IV switch action. It tries to determine the object of the mode check.

void multiswitch()
    This routine provides the controlling logic to check the alternate option specifications that can be associated with an event.
Recommended Development

The following are some suggested software improvements that are directed toward the development of a functional AWACS ITS.

1. Integrate the AWACS Simulation and ITS software into a single package so that it will qualify as an expert system that is limited to switch action procedures. This limitation obviates the knotted problems that are often associated with the inclusion of strategy.

For the start of this effort, the current ITS operation will be continued, i.e., maintain execution in batch fashion, and improve the software so it can reliably detect the beginning and ending responses to events. In turn, each established detection scheme will be coded into the Simulation software so that the end result should be the limited expert system that is desired.

2. Using the just described, limited expert system as a kernel, this development would add the capability to detect and suggest solutions to switch action procedural problems while the Simulation is running.

Knowing that the Simulation software controls the scenario presentation to the student and that it can now determine the correctness of switch action procedures, it is now possible to determine the appropriateness of proffered switch action procedures in response to controlled scenario situations. This development would allow the software to passively monitor a student's actions and provide real-time diagnostics as procedural errors occur.

3. Incorporate a reactive Instructor and Student Model into the software that was developed during Step 2.

Since the software can now monitor a student's switch actions to known situations, this would begin the development of both the Instructor and Student Model software. Essentially, the software will be given the capability to evaluate a student and determine whether the student requires either advancement, remediation, or be kept at status quo. These software decisions will be based upon actions that were taken. Consideration of actions that were omitted will be avoided unless outcomes are blatantly obvious.
The following are suggested changes to the existing software.

1. Besides determining the start and stop of each switch action, each of the
   of the associated switch-checking modules could be made considerably
   more intelligent, e.g., devise methods of linking such things as time delays
   and erroneous inputs with help messages and switch menus. This would,
   in effect, provide an interim capability for both the student and instructor to
   review performance.

2. Provide a better prompt for Control key paging functions. This may entail
   just displaying paging options in appropriate places on the screen, i.e.,
   Ctrl-P, up-arrow, Ctrl-B at the top of the screen and Ctrl-N, down-arrow,
   Ctrl-F at the bottom of the screen.

3. Options to develop:
   a. Establish a criteria for those values that are to be scored and stored.
   b. Provide a mechanism for the student to review material and answers,
      but preclude the capability to change original values.
   c. Establish criteria for demotion, promotion, or instructor intervention
      based upon scores.

4. Build a utility to assist in the development of a lesson and/or instruction
   block so the instructor's attention can be focused on content and not
   procedure.


Appendix A

3 SEP 91 Ltr from Capt. Fowler
As promised, here’s a breakout of the switch actions taught during Block I of the new IQT WD course, which I believe will enter tryout in late Oct/early Nov. Sorry for the delay; seems something always comes up and obscures little projects like this.

Let me know if I can be of any more assistance. My new number is DSN 884-7232, but Roy Houchin and Tina Livingston are still at the old number (DSN 884-7785).

Day 1 - Course Orientation; Pubs Posting
Day 2 - Sim Safety; Console Checkout; Assign Console S/A
Day 3 - Assign Console S/A; TD Inde: S/A; Line S/A; Circle S/A; Coordinate S/A; Tactical Erg/Rng
Day 4 - Same as Day 3, but add ADS Panel familiarization
Day 5 - Restricted Area S/A; Bearing and Range S/A; RN/DES/MTN SD S/A; Initialize Special Point S/A; Add/Delete Airbase S/A
Day 6 - Same as Day 5, but add Area Define/Delete S/A, Locate SIF S/A; Corridor IFF SD S/A
Day 7 - Request SIF S/A; Initiate S/A; Mode IV S/A; Reinit S/A
Day 8 - Same as Day 7, but add Radar/IFF Tracking S/A; Assign/Def S/A; Request/Assign IFF/SIF S/A; and study of track blocks.
Day 9 - Building database and situational awareness through use of previously-taught switch actions; using assigned radios; performing flight follow missions
Day 10 - Flight follow; inflight separation; inflight emergencies; and completion of mission related paperwork
Day 11 - Same as Day 10
Day 12 - Practice for unit proficiency check
Day 13 - Block I proficiency check

Circles are switch actions we don’t have in our sim.
Appendix B
IQT-WD READING GUIDE

39
This guide lists the required daily reading for the IQT-WD course. The daily reading should be accomplished prior to each day training.

Section I lists the reading for Block I (Training Days 1-13). The left column shows the lesson title or switch action and the right column shows the required reading. The documents you'll be reading for Block I are broken down into two volumes (student study guides) and the 28AD HB 55-1, Vol II 20/25.1), also known as the "Positional". The abbreviation (SG) followed by a number tells you which chapter(s) to reference in the study guides. Read the information in the chapter(s) and positional to better prepare yourself for academics and simulator instruction. Many switch actions take up entire chapters. The abbreviation (P) followed by a number tells you on what page the information begins in the "Positional". Some (P) references list specific sections to read. The abbreviation (M) refers to the specific study module that relates to that subject. Again, accomplish all related reading before class.

### SECTION I

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Familiarization &amp; Mechanics</th>
<th>P</th>
<th>1-1 thru 1-12, 1-22 thru 1-28, 1-38 thru 1-40, 3-1 thru 3-14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SG</td>
<td>1 thru 7, 9 &amp; 10 Assign Cons</td>
</tr>
<tr>
<td></td>
<td>TD Index</td>
<td>P</td>
<td>3-30 SG 8</td>
</tr>
<tr>
<td></td>
<td>Hard Copy</td>
<td>P</td>
<td>3-77 SG 15 &amp; 16</td>
</tr>
<tr>
<td></td>
<td>TD Update</td>
<td>P</td>
<td>3-41 SG 15</td>
</tr>
<tr>
<td></td>
<td>Assign Console</td>
<td>P</td>
<td>3-83 SG 17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M</td>
<td>MS21: EMER Sim Emergency Proc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 2</th>
<th>Line</th>
<th>P</th>
<th>3-49 SG 12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Circle</td>
<td>P</td>
<td>3-56 SG 12</td>
</tr>
<tr>
<td></td>
<td>Coordinates</td>
<td>P</td>
<td>3-71 SG 13 &amp; 14</td>
</tr>
<tr>
<td></td>
<td>Arrow</td>
<td>P</td>
<td>3-60 SG 11</td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>P</td>
<td>3-64 SG 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 3</th>
<th>Winds Aloft TD (#44)</th>
<th>P</th>
<th>4-47 SG 17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Air Base Weather TD</td>
<td>P</td>
<td>3-92, 3-26 &amp; 27, SG 17</td>
</tr>
<tr>
<td></td>
<td>Initialize Special Point</td>
<td>P</td>
<td>3-132 SG 18</td>
</tr>
<tr>
<td></td>
<td>RN/DES/NTN SD</td>
<td>P</td>
<td>3-139 SG 19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 4</th>
<th>Identifying Tracks</th>
<th>P</th>
<th>3-15 thru 3-18, SG 22, 24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initiate</td>
<td>P</td>
<td>3-148 SG 23</td>
</tr>
<tr>
<td></td>
<td>reinitiate</td>
<td>P</td>
<td>3-151 SG 23</td>
</tr>
<tr>
<td></td>
<td>Mode IV</td>
<td>P</td>
<td>3-300 SG 34</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
<td>P</td>
<td>3-159 SG 23</td>
</tr>
<tr>
<td></td>
<td>Assign/Defer</td>
<td>P</td>
<td>3-239 SG 29 Request SIF P 3-121 SG 5 (Review), 20</td>
</tr>
<tr>
<td></td>
<td>Locate SIF</td>
<td>P</td>
<td>3-201 SG 29</td>
</tr>
</tbody>
</table>
Day 5  Track Blocks  P  1-28 thru 1-37  SG 25
Track TDs/Special Point TDS  P  3-164  SG 26

Day 6  Radar/IFF Tracking  P  3-207  SG 29
Request/Assign IFF/SIF  P  3-268  SG 29
Area Define/Delete  P  3-125  SG 21
Corridor IFF SD  P  3-129  SG 21

Day 7  RCT Initialization TD  P  3-184  SG 27
Add/Delete Air Base  P  3-204  SG 34
Commit  P  3-212  SG 28, 30
Cap  P  3-229  SG 28
RTB  P  3-236  SG 28
Alter Control  P  3-192  SG 24

Day 8  RCT Mission/Augmented MSN TD  P  3-184  SG 28
Manual Guidance  P  3-258  SG 29
Fuel/Configuration  P  3-94  SG 29
Armament Update/Override  P  3-97
Command Tracking  P  3  SG 29
RCT Mode Control  P  3-256  SG 34
Aircraft Down  P  3-190  SG 29

Day 9  Mission Modifier  P  3-217
Recovery Air Base  P  3-217  SG 28
Beam Rider  P  3-217
Bearing and Range  P  3-43  SG 31
Tactical Bearing and Range  P  3-4 & 3-89  SG 31

Day 10  Order of Battle Add/Delete  P  3-101  SG 33
Order of Battle SD  P  3-112  SG 32
Restricted Area  P  3-306  SG 33
Present Altitude  P  3-242  SG 34
Target Altitude  P  3-244  SG 34

Day 11  Intercept Line SD  P  3-143  SG 29
MA/Kill  P  3-179
UHF Tune  P  3-295  SG 34
WT Handover  P  3-262
Accept/Reject Handover  P  3-272
Wilco/Cantco  P  3-274

Day 12  Review All Block I Academics

Day 13  Written Eval and Critiques
**SECTION II**

Section II lists the reading for Block II (Training Days 14-27). The left column gives the lesson titles while the right column shows the required reading. The following abbreviations will be used: (55-3) refers to TAC Regulation 55-3 and associated supplements/attachments, (55-79) refers to Joint Regulation 55-79 and associated documents, (T.O.) refers to Technical Order 1-E-3A-43-1-2, (M) refers to the individual student modules, and (SDC) refers to the Readiness Enhancement Program modules. The SDC readings are optional but do contain valuable information. They are located in the Boeing Learning Resources Center.

<table>
<thead>
<tr>
<th>Day 14</th>
<th>Mission Flow/WD Responsibilities</th>
<th>WIL:CC03 Daily Training Missions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADS Panel</td>
<td>55-3 Ch 5-2c pgs 5-5 - 5-7</td>
</tr>
<tr>
<td></td>
<td>1vl Cutoff Geometry</td>
<td>T.O. Pgs 1-166-1-181,5-16&amp; 17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55-3 Ch 5-3e pg 5-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M WIL:CM01 Audio Distribution System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M ACC:CC0 Crew Coordination</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55-79 Chs 2, 5 &amp; 7 (TAC)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 15</th>
<th>Weapons Airspace</th>
<th>WIL:PE03/PE04 Airspace Types and Usage/FAA Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intercept R/T</td>
<td>M ICC: R/T Radio Telephone Transmissions</td>
</tr>
<tr>
<td></td>
<td>MCM 3-1 Brevity Code Words</td>
<td>MCM 3-1 Vol I Attach 1 pgs A1-1-A1-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 16</th>
<th>FAA/Ground Agency Procedures</th>
<th>Letters of Agreement (LOA)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Day 17</th>
<th>E-3 Restart Recovery Procedures</th>
<th>55-3 Sup 1 Ch 6-4D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intercept Mission Administration</td>
<td>55-3 Sup 1 Atch I-19 pgs A19-11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A19-21 (WD Log)</td>
</tr>
<tr>
<td></td>
<td>Contending With Variables/Stress</td>
<td>M WIL:CC05 Stress/Contending with Unknown Variables</td>
</tr>
<tr>
<td></td>
<td>Controlled and Uncontrolled</td>
<td>55-3 Ch 6-4C6 pgs 6-25C and 6-16</td>
</tr>
<tr>
<td></td>
<td>Aircraft Emergencies</td>
<td>(Sup 1)</td>
</tr>
<tr>
<td></td>
<td>M WIL:EE01 Controlled Aircraft Emergencies</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 18</th>
<th>55-79 Training Rules</th>
<th>55-79 Review Chs 2, 5 &amp; 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stern Geometry</td>
<td>55-79 Atch 10 pgs A10-1-A10-2</td>
</tr>
<tr>
<td></td>
<td>Special ROE/55-79 VID Procedures</td>
<td>MCM 3-1 Vol I Review Atch 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M Review WIL:CC03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 19</th>
<th>E-3 Weapons Guidance/Computer Logic</th>
<th>ACC:LOG Intercept Logic (Secret, in WPNS safe)</th>
</tr>
</thead>
</table>

43
Day 20  Transition Mission
ART Briefing

Day 21  Flyups/High Fast Flyer

SDC #19 E-3A Radar System General Description (Secret)
SDC #20 E-3A Radar Operations/
SDC #21 ECM/ECCM Concepts and Procedures (Secret)
SDC #22 ECM/ECCM Tactics and Techniques (Secret)

Day 22  Multiple Cutoff Positioning
M Review WIL:CCO3

Day 23  Stern Geometry Multiple/ Simultaneous Intercepts
M Review WIL:CCO3

Day 24  F-15 Liaison Briefing

Day 25  CDMT Briefing
CSO Briefing

Day 26  Review
Review All Academic Material

Day 27  Written Eval and Critiques

SECTION III

Section III lists the reading for Block III (Training Days 28-32). (SG refers to the Block III Study Guide.

Day 28  Tanker Rendezvous Overview
Receive Turn-ons
Tanker Turn-ons

Day 29  Rules of Thumb for Tankers

SG 3.4 Non-Standard Refueling Procedures
55-3 Sup 1 6-2Cd pg 6-6C
55-3 Sup 1 Atch C-3 pg A3-1C
Day 30 Master Question File (MQF)  
Local Operating Procedures (LOP)

Day 31 Review  
Review All Academic Material

Day 32 Written Eval and Critiques

SECTION IV

Section IV lists the reading for Block IV (Training Days 33-44). Abbreviations for the reading will be the same as Section II.

Day 33 NORAD/ADTAC Lane Defense and Threat Assessment
- 55-79 Ch 3, Review Ch 5, 7, Atch and Atch 10
- 55-3 Ch 5-1b pg 5-2
- 55-3 Ch 5-2a pgs 5-3 and 5-4
- MCM 3-1 Vol XIV Para 3-3 pgs 3-5 thru 3-8 (Secret)
- CINCNORAD Oplan 3000-84 pg C10-1 (Secret - in Battle staff Rm 118)
- Region/Sector Oplan 3000 read sector unique book and sup for one (1) of the regions (Secret - Rm 118)
- SDC #17 NORAD Identification Procedures (Secret)
- SDC #37 NORAD Organization
- SDC #39 Authentication Systems and Materials Handling

Day 34 NORAD Briefing
- SDC #38 NORAD/AWACS Interoperability
- SDC #40 Authentication Usage

Day 35 Authenticators
- Review MCM 3-1 Vol I Atch 1 & SDC #40

Day 36 Elements of Broadcast Control
Resource Battle Management
Strike Controller
- 55-79 Ch 2, 3, 5, 6 & 7 (TAC)
- SDC #11 Close Air Support
- SDC #12 Interdiction
- SDC #13 Interdiction/Battlefield Air Interdiction (Secret)
- SDC #32 E-3 Employment in the Tactical Arena (Secret)
Day 37 Tactical Battle Elements

SDC #23 Intelligence Support to
AWACS (Secret)
SDC #2 Elements of the TACS
SDC #3 Roles of TACS in Tactical
Air Operations
M PTP:MSN Point-to-Point
Missions

Day 38 CF Crew Concepts/Force Controller

SDC #1 Tactical Air Roles and
Missions

Day 39 Navy Briefing

Navy Interoperability

SDC #26 US Naval Organization
SDC #27 US Naval Operations
(Confidential)
Naval Anti-Air Warfare Interoper-
ability (Secret)

Breaking the ATO

SDC #5 Air Tasking Order

Day 40 Mission Planning Documents and Resources

55-3 Ch 3
55-3 28 AD Sup 1 Ch 6 pgs 6-1A/B
thru 6-16C
M WDA:INI Initialization
SDC #28 Mission Planning - Part
SDC #29 Mission Planning - Part

Mission Planning Considerations
Map Preparation

M WDA:RHO Relief Handover
M WIL:MP04 Performing AWACS
Monitor

Day 41 Emergency Equipment and Procedures Walk around(Flight line)

T.O. Section III pgs 3-1 thru 34
T.O. Section I pg 1-203 thru
252B
M WIL:MP03 Flight Procedures

Day 42 Pre-Eval Mission Planning Block V Aircraft Brief Prep
DOV Briefing

SDC #30 Aircrew Interfaces Part
SDC #31 Aircrew Interfaces Part

Day 43 Pre-Eval Mission Planning Block V Aircraft Brief Prep

Day 44 Final DOV Sim Check

46
SECTION V

Section V lists the reading for Block V (Training Days 45-47). Abbreviations for the reading will be the same as Section II. (SG) refers to Block V Study Guide.

Day 45  Introduction to ACT/DACT  SG  5.1  Introduction to ACT/DACT
       ACT R/T  SG  5.2  ACT Radio Transmissions
       ACT Formations and Tactics  SG  5.3  ACT Formations and Tactics
       F-15/16 Employment/Characteristics  SG  5.4  Aircraft Characteristics
                                          MCM 3-1  Volumes IV & V (Secret)
                                          SDC #4  Aircraft, Weapons and Tactics (Secret)

Student Aircraft Briefs  SDC #34  Allied Fighter Characteristics

Day 46  F-14/18 Employment/  Naval Anti-Air Warfare Interoperability (Secret)
       Student Aircraft Briefs Review

Day 47  Written Eval and Critiques

NOTE: This Reading Guide is subject to change.
Last update: 6 Jun 90  Capt Mike Challman

... PREVIOUS VERSIONS ARE OBSOLETE

Characteristics
Appendix C

Experience Questionnaire
EXPERIENCE QUESTIONNAIRE

1. Have you ever been Mission Ready (MR) qualified as a 17XX?
   A. Yes
   B. No

2. Were you MR as a ______?
   A. U.S. or NATO E-3 WD/IWD/SD/ISD
   B. U.S. or NATO E-3 ASO
   C. Other

3. You are here because _______.
   A. You've been DNIF for more than 180 days.
   B. You failed an EVAL and are being retrained.
   C. You've only been NATO qualified.

4. Do you have any other 17XX experience?
   A. Yes
   B. No

5. Your other 17XX experience includes _______
   A. MCE
   B. 407L CRC/CRP WC/WAO
   C. 407L CRC/CRP ASO
   D. NORAD/ICELAND/ALASKA ROCC/SOCC WC/WAO
   E. NORAD/ICELAND/ALASKA ROCC/SOCC ASO
   F. Manual Radar (FACP/TSP-43E)
   G. Other

6. Does your other 17XX experience include a stint as an exchange officer with the
   US Navy in the E-2C?
   A. Yes
   B. No

7. Did you graduate from Tyndall's Basic WC Course (E30BP-1741A-004)?
   A. Yes
   B. No
8. Were you ____?
   A. a Distinguished Graduate
   B. in the top 25% of your class
   C. average
   D. barely passed

9. Did you take the automated course?
   A. Yes
   B. No

10. Were you prior enlisted?
    A. US Air Force
    B. US Navy
    C. US Army
    D. US Marine
    E. No

11. Are you a former 16XX?
    A. Yes
    B. No

12. While you were prior enlisted in the Air Force, did you have the AFSC _____?
    A. 11XXX
    B. 276XX
    C. 30XXX (Air Traffic Controller at a RAPCON)
    D. Other

13. While you were prior enlisted in the Army, did you have a previous job as ______.
    A. Helicopter pilot
    B. Fire control officer for Patriot or I-HAWK battery
    C. Radar Air Traffic Controller
    D. Joint Stars operator
    E. Other

14. While you were prior enlisted in the Navy, did you have a previous job in ______.
    A. Radar Air Traffic Control
    B. Air Intercept Control
    C. Other
15. While you were prior enlisted in the Marines, did you have a previous job in ________?
   A. Radar Air Traffic Control
   B. Air Intercept Control
   C. Other

16. Are you an exchange/liaison officer from ________?
   A. the U.S. Army
   B. the U.S. Navy
   C. the U.S. Marines
   D. another nation

17. In the US Army, your previous jobs included ________.
   A. Pilot
   B. Joint Stars
   C. Air Defense Artillery
   D. Radar Air Traffic Control
   E. ASOC (Air Support Operations Center)
   F. Other

18. In the US Navy, your previous jobs included ________.
   A. Pilot
   B. Radar Intercept Officer
   C. E-2C Naval Flight Officer
   D. Radar Air Traffic Control
   E. Air Intercept Control
   F. Other

19. In the US Marines, your previous jobs included ________.
   A. Pilot
   B. Air Defense Artillery
   C. Radar Air Traffic Control
   D. F-4 Weapons System Operator (WSO)
   E. MACS 5/6/7 Radar Control/Air Intercept Control
   F. Other

20. You are a ________ officer.
    A. United Kingdom
    B. Canadian
    C. French
    D. German
    E. Other
21. You have experience in/as ________.
   A. Pilot/Navigator
   B. Weapons System Officer (WSO) in F-4/Tornado IDS
   C. Schackelton AEW.2
   D. NATO E-3 or UK E-3D AEW.Mk.1 ASO
   E. NATO E-3 or UK E-3D AEW.Mk.1 WC/WAO
   F. Other Radar Control Air Defense Work
   G. Other (work not involving Radar)

22. You have experience in/as ________.
   A. U.S. E-3 ASO
   B. NATO E-3 WC/WAO
   C. NATO E-3 ASO
   D. NORAD ROCC/SOCC
   E. Ground Radar Weapons Controller
   F. Other

23. You have experience in/as ________.
   A. Pilot/Navigator
   B. NATO E-3 WC/WAO
   C. NATO E-3 ASO
   D. NADGE WC
   E. Other

24. You have experience in/as ________.
   A. Pilot/Navigator
   B. French E-3F WC/WAO
   C. French E-3F ASO
   D. Other

25. Are you a Fighter Weapons Instructor Course graduate?
   A. Yes
   B. No
Appendix D

ITS Source Files
NAME: SWITCHES.DAT

/*------------------------------------------------------------------------------------*
 For the following, the numerous switches are presented in numerical order, but the use of pagination is done to infer a logical order. This logical ordering is as follows:

 a. 0-79: Function select switches

 b. 80-87: Feature & Vector switches Required Switches
 88-105: Category Select switches for WD Console
 142-159: Category Select switches Checkout

 c. 106-141: Optional Category Select switches

 d. 160-184: Display Panel switches

------------------------------------------------------------------------------------*/

<table>
<thead>
<tr>
<th>Switches</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>UHF TUNE</td>
<td>0</td>
</tr>
<tr>
<td>PDA</td>
<td>1</td>
</tr>
<tr>
<td>ORD BAT ADD/DEL</td>
<td>2</td>
</tr>
<tr>
<td>FLIGHT PLAN</td>
<td>3</td>
</tr>
<tr>
<td>FLT PLN ADD/DEL</td>
<td>4</td>
</tr>
<tr>
<td>FLT PLN ASC/DIS</td>
<td>5</td>
</tr>
<tr>
<td>TRACK TD</td>
<td>6</td>
</tr>
<tr>
<td>TD INDEX</td>
<td>7</td>
</tr>
<tr>
<td>INIT SPCL PT</td>
<td>8</td>
</tr>
<tr>
<td>RN/DES/NTN SD</td>
<td>9</td>
</tr>
<tr>
<td>ORDER BAT SD</td>
<td>10</td>
</tr>
<tr>
<td>RADAR IFF TRK</td>
<td>11</td>
</tr>
<tr>
<td>AIRBASE WX TD</td>
<td>12</td>
</tr>
<tr>
<td>ADD DEL AIRBASE</td>
<td>13</td>
</tr>
<tr>
<td>HARD COPY</td>
<td>14</td>
</tr>
<tr>
<td>TD UPDATE</td>
<td>15</td>
</tr>
<tr>
<td>RCT INIT TD</td>
<td>16</td>
</tr>
<tr>
<td>WEAPONS SUMY TD</td>
<td>17</td>
</tr>
<tr>
<td>RESTRICTED AREA</td>
<td>18</td>
</tr>
<tr>
<td>OPNL COND TD</td>
<td>19</td>
</tr>
<tr>
<td>UNUSED 1</td>
<td>20</td>
</tr>
<tr>
<td>UNUSED 2</td>
<td>21</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>22</td>
</tr>
<tr>
<td>BEARING RANGE</td>
<td>23</td>
</tr>
<tr>
<td>COMMAND</td>
<td>24</td>
</tr>
<tr>
<td>AADCP OPTION</td>
<td>25</td>
</tr>
<tr>
<td>ADA ENG STATUS</td>
<td>26</td>
</tr>
<tr>
<td>DATA REQUEST</td>
<td>27</td>
</tr>
<tr>
<td>UNUSED 3</td>
<td>28</td>
</tr>
<tr>
<td>UNUSED 4</td>
<td>29</td>
</tr>
<tr>
<td>LINE</td>
<td>30</td>
</tr>
<tr>
<td>COORDS</td>
<td>31</td>
</tr>
<tr>
<td>WILCO/CANTCO</td>
<td>32</td>
</tr>
<tr>
<td>HANDOVER FREQ</td>
<td>33</td>
</tr>
<tr>
<td>CONTROL UNIT TD</td>
<td>34</td>
</tr>
</tbody>
</table>

57
FORCE TELL
SAF PAS CORIDOR
AADCP SD
ARROW
MESSAGE
WT HANDOVER
ACC/REJ HANDOVER
CORIDOR IFF SD
AREA MONITOR
AREA DEF DEL
A/C DOWN
DIALOG TEST
ASSIGN CONSOLE
MODE IV
LOCATE SIF
FUEL/CONFIG
ARM UP/ORIDE
MA KILL
INT L/R LINE SD
TRK ALT ORIDE
REQUEST SIF SD
RCT MSN TD
COMMAND TRACKING
PRESENT ALT
TARGET ALT
RCT MOD CONTROL
REQ/ASG IFF/SIF
DROP
ALL
MANUAL GDNC
CRUISE ORIDE
COMBAT ORIDE
PROFILE ORIDE
RCVY AIRBASE
ALTER CONTROL
INIT
OFF
COMMlT
MISSION
CAP
RTB
BEAM RIDER
ASSIGN/DEFER
RE-INIT
HOOK
<p>| FEATURE SEL B | 80 |
| FEATURE SEL A | 81 |
| FEATURE SEL D | 82 |
| FEATURE SEL C | 83 |
| FEATURE SEL F | 84 |
| FEATURE SEL E | 85 |
| VECTOR SW DOWN | 86 |
| VECTOR SW UP | 87 |
| IDBO          | 106 |
| IDBO MOM     | 107 |
| REQUESTED/FORCED SDS | 108 |
| REQUEST/FORCED SDS MOM | 109 |
| NET PART PRIMARY E-3A | 110 |
| NET PART PRIMARY E-3A MOM | 111 |
| SPECIAL POINTS | 112 |
| SPECIAL POINTS MOM | 113 |
| FRIENDLY | 114 |
| FRIENDLY MOM | 115 |
| HSTL/UNK/FAKER TRK | 116 |
| HSTL/UNK/FAKER TRK MOM | 117 |
| CROSSTOLD TRACKS-AIR | 118 |
| CROSSTOLD TRACKS-AIR MOM | 119 |
| SPECIAL MISSION | 120 |
| SPECIAL MISSION MOM | 121 |
| INTERCEPTOR | 122 |
| INTERCEPTOR MOM | 123 |
| CROSSTOLD TRACKS-SURF | 124 |
| CROSSTOLD TRACKS-SURF MOM | 125 |
| ASSIGNED TRACKS | 126 |
| ASSIGNED TRACKS MOM | 127 |
| UNASSIGNED TRACKS | 128 |
| UNASSIGNED TRACKS MOM | 129 |
| TADIL-A/LINK 11 DATA | 130 |
| TADIL-A/LINK 11 DATA MOM | 131 |
| STROBE HISTORY C | 132 |
| STROBE HISTORY C MOM | 133 |
| STROBE PRESENT C | 134 |
| STROBE PRESENT C MOM | 135 |
| JTIDS/ERCS DATA | 136 |
| JTIDS/ERCS DATA MOM | 137 |
| STROBE HISTORY U | 138 |
| STROBE HISTORY U MOM | 139 |
| STROBE PRESENT U | 140 |
| STROBE PRESENT U MOM | 141 |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA INTEN TRK</td>
<td>160</td>
</tr>
<tr>
<td>DATA INTEN SNSR</td>
<td>161</td>
</tr>
<tr>
<td>DSPL MODE SID</td>
<td>162</td>
</tr>
<tr>
<td>DSPL MODE FTAB</td>
<td>163</td>
</tr>
<tr>
<td>TEST MODE STANDALONE</td>
<td>164</td>
</tr>
<tr>
<td>TEST MODE WRAP AROUND</td>
<td>165</td>
</tr>
<tr>
<td>TACT B/R RESET</td>
<td>166 BRR</td>
</tr>
<tr>
<td>NOT USED3</td>
<td>167 CLR</td>
</tr>
<tr>
<td>NOT USED4</td>
<td>168 TST</td>
</tr>
<tr>
<td>ALARM/ALERT CLEAR</td>
<td>169 AAC</td>
</tr>
<tr>
<td>MSG ACK</td>
<td>170 ACK</td>
</tr>
<tr>
<td>ALERT</td>
<td>171 ALT</td>
</tr>
<tr>
<td>ALARM</td>
<td>172 ALM</td>
</tr>
<tr>
<td>SCALE EXPANSION 1</td>
<td>173</td>
</tr>
<tr>
<td>SCALE EXPANSION 2</td>
<td>174</td>
</tr>
<tr>
<td>SCALE EXPANSION 4</td>
<td>175</td>
</tr>
<tr>
<td>SCALE EXPANSION 8</td>
<td>176</td>
</tr>
<tr>
<td>SCALE EXPANSION 16</td>
<td>177</td>
</tr>
<tr>
<td>SCALE EXPANSION 32</td>
<td>178</td>
</tr>
<tr>
<td>RETURN TO CENTER</td>
<td>179 RTC</td>
</tr>
<tr>
<td>CANCEL OFFSET</td>
<td>180 CAN</td>
</tr>
<tr>
<td>OFFSET</td>
<td>181 PAN</td>
</tr>
<tr>
<td>SID CURSOR BLINK/STDY</td>
<td>182 BSD</td>
</tr>
<tr>
<td>TAB CURSOR BLINK/STDY</td>
<td>183 BTD</td>
</tr>
<tr>
<td>TACTICAL B/R</td>
<td>184</td>
</tr>
</tbody>
</table>
NAME: SDTEVAL.H

#define MAX_LOG_BUF 200

FILE *pass6_fp;
FILE *event_fp;

int x_mid, y_mid; /* Center coordinates for screen */

int wd_cnt; /* No. of WDs that were tested */

For the following arrays, the values are ordered corresponding to the value sequence in "wd_console_no". In addition, "wd_cnt" contains the number of valid entries in each array.

char wd_console_no[3]; /* Console no. of each WD tested (ASCII) */
char wd_id_no[3]; /* WD id no. (ASCII) for associate console */
int msg_ack_cnt[3]; /* Count of extraneous sequential ACKs */
float msg_prev_tim[3]; /* Time of previous message action */
double msg_elap_tim[3]; /* Sum of elapse time */
double msg_elap_tim2[3]; /* Sum of (elapse time)**2 */

int evl_skl_lvl; /* Skill level for evaluation */

long int prev_pass6_pos;

struct switch_ent
{ /* Function switch entry */
  char func_sw_name[28];
  char func_sw_num[4];
  char func_sw_alt_nm[4];
  char func_sw_state[3];
}; /* Function switch entry */

int func_sw_cnt; /* Number of function switches */
struct switch_ent func_sw[80];
#define FUNC_SW_LEN (sizeof(func_sw))

int req_sw_cnt; /* Number of required feature & category switches */
struct switch_ent req_sw[44];
#define REQ_SW_LEN (sizeof(req_sw))

int opt_sw_cnt; /* Number of optional switches */
struct switch_ent opt_sw[36];
#define OPTSWLEN (sizeof(opt_sw))

int panel_sw_cnt; /* Number of Alarm/Display panel switches */
struct switch_ent panel_sw[25];
#define PANEL_SW_LEN (sizeof(panel_sw))

struct log_rec
{ /* Partially parsed log record */
  char time_str[13];
  char mod_str[10];

 64
char key_str[5];
char trns_str[MAX_LOG_BUF-28];
};/* Partially parsed log record */

struct evnt_mrk
{ /* Event marker */
    long int evnt_file_offset;
    struct log_rec mrk_evnt;
};/* Event marker */

struct log_rec cur_log_rec;

struct evnt_mrk strrt_rec;
long int sim_strrt_time;
long int prev_evnt_strrt_rec;

struct evnt_var
{ /* Event variation */
    char name_fld[28];
    char nbr_fld[5];
    char alt_nm_fld[4];
    char add_match_txt[28];
    float tim_win;
    float strt_tim;
    float end_tim;
};/* Event variation */

struct evnt_var prev.swt;
long prev_rec_pos;

struct evnt_skelt
{ /* Event description */
    char evnt_seq[4];
    struct evnt_var evnt_label;
    int no_var;
    int step_per_var[5];
    struct evnt_var variant[25];
};/* Event description */

struct evnt_skelt evnt_desc;
This program is an attempt to evaluate a student WD for proficiency in the subject matter from Instruction Block 1.

main()
{
    extern int x_mid;
    extern int y_mid;
    extern FILE *pass6_fp;
    extern int wd_cnt;
    extern char wd_console_no[];
    extern int eval_skil_lvl;
    extern int func_sw_cnt;
    extern int req_sw_cnt;
    extern int opt_sw_cnt;
    extern int panel_sw_cnt;
    extern struct switch_ent func_sw[];
    extern struct switch_ent req_sw[];
    extern struct switch_ent opt_sw[];
    extern struct switch_ent panel_sw[];
    extern FILE *event_fp;
    extern struct log_rec cur_log_rec;
    extern struct evnt_mrkr str_rec;
    extern long int sim_strt_time;
    extern struct evnt_skelt evnt_desc;
    extern char wd_id_no[];
    extern int msg_ack_cnt[];
    extern long int prev_pass6_pos;
    extern void cntr ln();
    extern void quick_exit();
    extern void read_swt_data();
    extern void get_fil_str();
    extern void open_err();
    extern void blink_msg_on();
    extern void blink_msg_off();
    extern void console_chk();
    extern void swt_state();
    extern void multi_switch();

    struct switch_ent *swt_ptr;
    char filename[30];
    char txt_str[81];
    char reply;
    int i;
int valid, noevents;
FILE *switch_fp;
int tmp_int;
char *tmp_chr_ptr;

/* Initialize curses screen management */
initscr();
x_mid = COLS/2 - 1;
y_mid = LINES/2 - 1;

/* Initialize program variables */
func_sw_cnt = 80; /* Number of function switches */
req_sw_cnt = 44; /* Number of required feature & category switches */
opt_sw_cnt = 36; /* Number of optional feature switches */
panel_sw_cnt = 25; /* Number of alarm/display panel switches */
for (i=0; i<3; i++)
    msg_ack_cr.t[i] = 0;
prev_pass6_pos = 0;

/* Get name of data file */
mvaddstr(y_mid-1, 10, "Enter name of data file");
get_file(filename);

/* Verify that data file was generated from REDUCE pass6 */
sprintf(txt_str, "%s file "$, filename);
cntr_ln(y_mid+1, txt_str);
cntr_ln(y_mid+2, "a \"REDUCE pass6\" output [Y/N]?");
reply = tolower(getch());
if (reply != 'y')
{ /* File requires processing */
clear();
cntr_ln(y_mid, "Input data file must be preprocessed by REDUCE");
cntr_ln(y_mid+1, "and the resultant of at least \"pass6\" processing");
getch();
quick_exit();
} /* File requires processing */

/* Open specified file */
pass6_fp = fopen(filename, "r");
if (pass6_fp == NULL)
    open_err(filename);

/* Get number of WDs that were tested */
valid = FALSE;
while (!valid)
{ /* Get number */
clear();
cntr_ln(y_mid, "Enter number of WDs that were tested: ");
reply = getch();
sscanf(&reply, "%d", &wd_cnt);
if (wd_cnt<=0 || wd_cnt > 3)
{ /* Invalid count */
cntr_ln(y_mid+1, "An invalid number of WDs was entered");
getch();
}
else
valid = TRUE;
} /* Get number */

/* Get WD's console number */
for (i=0; i<wd_cnt; i++)
{ /* Get console numbers */
clear();
if (i == 0)
  sprintf(txtstr, "Enter console no. of %ldst WD: ", i+1);
else if (i == 1)
  sprintf(txtstr, "Enter console no. of %ldnd WD: ", i+1);
else
  sprintf(txtstr, "Enter console no. of %ldrd WD: ", i+1);
cntrl(y_mid, txt_str);
wd_console_no[i] = getch();
} /* Get console numbers */

/* Get WD id number */
for (i=0; i<wd_cnt; i++)
{ /* Get WD id */
clear();
  sprintf(txt_str, "Enter WD id no for console %c: ", wd_console_no[i]);
cntrl(y_mid, txt_str);
wd_id_no[i] = getch();
} /* Get WD id */

/* Get skill level */
clear();
mvinsstr(y_mid-4, x_mid-17, "Select skill level for evaluation:");
mvinsstr(y_mid-2, x_mid-6, "1) Naive");
mvinsstr(y_mid-1, x_mid-6, "2) Novice");
mvinsstr(y_mid, x_mid-6, "3) Journeyman");
mvinsstr(y_mid+1, x_mid-6, "4) Expert");
mvinsstr(y_mid+2, x_mid-6, "5) Master");
mvinsstr(y_mid+4, x_mid-5, "Selection: ");
valid = FALSE;
while (!valid)
{
  reply = tolower(getch());
  if (reply == 'q')
    quick_exit();
  sscanf(&reply, "%d", &evl_skl_lvl);
  if (evl_skl_lvl < 1 || evl_skl_lvl >= 4)
  { /* Not available */
    cntrl(y_mid+5, "Evaluation criteria not available");
    getch();
    move(y_mid+4, x_mid+6);
    clrbot();
    refresh();
  } /* Not available */
  else
    valid = TRUE;
/* Load Feature and Category switch tables */
switch_fp = fopen("switches.dat", "r");
if (switch_fp == NULL)
    open_err("switches.dat");

/* Loading Function switches */
for (swt_ptr=func_sw; swt_ptr<func_sw+func_sw_cnt; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);
fgets(txt_str, 80, switch_fp);

/* Loading Feature Select switches (Console Checkout) */
for (swt_ptr=req_sw; swt_ptr<req_sw+8; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);
fgets(txt_str, 80, switch_fp);

/* Loading top 9 Category Select switches (Console Checkout) */
for (swt_ptr=req_sw+8; swt_ptr<req_sw+26; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);
fgets(txt_str, 80, switch_fp);

/* Loading middle 18 Category Select switches */
for (swt_ptr=opt_sw; swt_ptr<opt_sw+opt_sw_cnt; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);
fgets(txt_str, 80, switch_fp);

/* Loading bottom 9 Category Select switches (Console Checkout) */
for (swt_ptr=req_sw+26; swt_ptr<req_sw+req_sw_cnt; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);
fgets(txt_str, 80, switch_fp);

/* Loading Panel switches */
for (swt_ptr=panel_sw; swt_ptr<panel_sw+panel_sw_cnt; swt_ptr++)
    read.swt_data(switch_fp, swt_ptr);

fclose(switch_fp);

/* Get event script file */
clear();
mvaddstr(y_mid-1, 10, "Enter name of event script file");
get.fill_str(filename);
event_fp = fopen(filename, "r");
if (event_fp == NULL)
    open.err(filename);

clear();
blink_msg_on(y_mid, "Getting initial switch settings");
swt_state();
blink_msg_off(y_mid);

clear();
blink_msg_on(y_mid, "Searching for Sim start");
valid = FALSE;
while (!valid)
{
    /* Find start of simulation */
    strt_rec.evnt_file_offset = ftell(pass6_fp);
    read_log();
    if ((strncmp(cur_log_rec.key_str,"848") != 0) ||
        (strncmp(cur_log_rec.trns_str,"SCN 2",5) != 0))
        continue;
    tmp_chr_ptr = strchr(&cur_log_rec.trns_str[5], 'P');
    if (tmp_chr_ptr == NULL)
        continue;
    tmp_int = -1;
    for (i=tmp_chr_ptr-cur_log_rec.trns_str+l;
        i<strlen(cur_log_rec.trns_str); i++)
    {
        /* Search for blanks */
        if (cur_log_rec.trns_str[i] != ' ')
        {
            /* Possible start */
            if (cur_log_rec.trns_str[i] == 'l')
            {
                /* Start found */
                strcpy(strt_rec.mrk_evnt.time_str, cur_log_rec.time_str);
                strcpy(strt_rec.mrk_evnt.key_str, cur_log_rec.key_str);
                strcpy(strt_rec.mrk_evnt.trns_str, cur_log_rec.trns_str);
                tmp_int = 0;
                break;
            }
            /* Start found */
            break;
        }
        /* Possible start */
    } /* Search for blanks */
    if (tmp_int == 0)
        valid = TRUE;
} /* Find start of simulation */
sscanf(strt_rec.mrk_evnt.time_str, "%d", &sim_strt_time);
blink_msg_off(y_mid);

no_events = FALSE;
while (!no_events)
{
    /* Process Event file */
    read_event();
    if (strncmp(evnt_desc.evnt_label.name_fld, "CONSOLE CHECKOUT",16) == 0)
        console_chk();
    else if (strncmp(evnt_desc.evnt_label.name_fld, "DONE", 4) == 0)
        no_events = TRUE;
    else if (strlen(evnt_desc.evnt_label.alt_nm_fld) != 0)
        switch_chk(&evnt_desc.evnt_label);
    else if (evnt_desc.no_var != 0)
        multi_switch();
    else
    { /* Invalid event */
        clear();
        move(y_mid,10);
        printf("Event: %s lacks sufficient description", evnt_desc.evnt_label.name_fld);
        getch();
    } /* Invalid event */
} /* Process Event file */
/* Terminate curses screen management */
endwin();
NAME: EVAL_RTN.C

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <curses.h>
#include <math.h>

#include "sdteval.h"
void cntr_ln(int y_coord, char *txt_str)
{ /* Center text on specified line */
extern int x_mid;
extern int y_mid;
int i, x_offset;

x_offset = 0;
for (i=0; i<=strlen(txt_str)-1; i++)
{ /* Search for printable characters */
    if (isprint(*(txt_str+i)))
        x_offset++;
} /* Search for printable characters */
x_offset /= 2;
mvinsstr(y_coord, x_mid-x_offset, txt_str);
return;

} /* Center text on specified line */
void quick_exit()
{
    /* Stop Curses and exit */
    endwin();
    exit(0);
} /* Stop Curses and exit */
void get_fil_str(char *filename)
{ /* Get a file string */
extern void cntrl_n();
extern void quick_exit();

*filename = '\0';
while (strlen(filename) == 0)
{ /* Filename */
    mvaddstr(y_mid, 10, "Name: ");
    getstr(filename);
    if (strlen(filename) == 0)
    { /* No name entered */
        cntrl_n(y_mid+1, "ERROR: No name was entered");
        getch();
        move(y_mid, 0);
        clrtobot();
    } /* No name entered */
} /* Filename */
if (strlen(filename) == 1 && tolower(*filename) == 'q')
    quick_exit();
return;
} /* Get a file string */
void open_err(char *filename)
{ /* Error opening a file */
 extern int y_mid;

 extern void cntr_ln();
 extern void quick_exit();

 clear();
 cntr_ln (y_mid-1, "Error occurred while opening");
 cntr_ln(y_mid, filename);
 getch();
 quick_exit();
 return;
} /* Error opening a file */
void read.swt.data(FILE *switch.fp, struct switch.ent *swt.ptr)
{ /* Read a record from "switch.dat" */
  char txt.str[80];
  char *blnk;
  char fill[2];
  int ndx;

  fgets(txt.str, 80, switch.fp);
  strncpy(swt.ptr->func_sw.name, txt.str, 27);
  swt.ptr->func_sw.name[27] = '0';
  while (swt.ptr->func_sw.name[strlen(swt.ptr->func_sw.name)-1] == ' ')
    swt.ptr->func_sw.name[strlen(swt.ptr->func_sw.name)-1] = '0';
  strncpy(swt.ptr->func_sw.num, &txt.str[28], 3);
  swt.ptr->func_sw.num[3] = '0';
  while (swt.ptr->func_sw.num[strlen(swt.ptr->func_sw.num)-1] == ' ')
    swt.ptr->func_sw.num[strlen(swt.ptr->func_sw.num)-1] = '0';
  if (txt.str[32] == '1')
  { /* Alternate name */
    strncpy(swt.ptr->func_sw.alt_nm, &txt.str[33], 3);
    if (swt.ptr->func_sw.alt_nm[2] == '1')
      !isprint(swt.ptr->func_sw.alt_nm[2])
    swt.ptr->func_sw.alt_nm[2] = '0';
    else
      swt.ptr->func_sw.alt_nm[3] = '0';
  } /* Alternate name */
  else
    swt.ptr->func_sw.alt_nm[0] = '0';

  return;
} /* Read a record from "switch.dat" */
void read_log()
{ /* Read a logger entry */
extern FILE *pass6_fp;
extern struct log_rec cur_log_rec;
extern int msg_ack_cnt[];
extern float msg_prev_tim[];
extern double msg_elap_tim[];
extern double msg_elap_tim2[];
extern long int prev_pass6_pos;

extern void left_just();

int i;
unsigned int tmp_long_int;
float tmp_flt;
double tmp_dbl;
char log_str[MAX_LOG_BUF];
char *sep;
char *beg_scan;
long int rec_pos;
char *rd_stat;

rec_pos = ftell(pass6_fp);
rd_stat = fgets(log_str, MAX_LOG_BUF, pass6_fp);
if ((rd_stat == NULL)
{ /* End of file or error */
    cur_log_rec.time_str[0] = '\0';
    return;
} /* End of file or error */

/* Partially parse logger record */
beg_scan = log_str;
sep = strchr(beg_scan, '│');
strncpy(cur_log_rec.time_str, beg_scan, sep-beg_scan);
cur_log_rec.time_str[12] = '\0';
beg_scan = sep + 1;
sep = strchr(beg_scan, '│');
strncpy(cur_log_rec.mod_str, beg_scan, sep-beg_scan);
cur_log_rec.mod_str[9] = '\0';
while (cur_log_rec.mod_str[strlen(cur_log_rec.mod_str)-1] == ' ')
    cur_log_rec.mod_str[strlen(cur_log_rec.mod_str)-1] = '\0';
beg_scan = sep + 1;
sep = strchr(beg_scan, '│');
strncpy(cur_log_rec.key_str, beg_scan, sep-beg_scan);
cur_log_rec.key_str[4] = '\0';
left_just(cur_log_rec.key_str);
beg_scan = sep + 1;
strcpy(cur_log_rec.trns_str, beg_scan);
cur_log_rec.trns_str[strlen(cur_log_rec.trns_str)-1] = '\0';

if (rec_pos < prev_pass6_pos)
    return;
prev_pass6_pos = rec_pos;
/* Evaluate Message ACK tapping */
if (strcmp(cur_log_rec.key_str, "1014") == 0)
{ /* Message Presented */
  for (i=0; i<3; i++)
  { /* Find WD console */
    if (cur_log_rec.mod_str[8] != wd_console_no[i])
      continue;
    if (abs(msg_ack_cnt[i]) > 2)
      {/* Compute variability */
        tmpdbl = (abs(msg_ack_cnt[i])*msg_elap_tim2[i]) -
        msg_elap_tim[i];
        tmpdbl /= abs(msg_ack_cnt[i])*(abs(msg_ack_cnt[i])-1);
        tmpdbl = sqrt(tmpdbl);
        clear();
        move(y_mid, 10);
        printf("ACK tapping:Avg: %f Var: %f", tmpflt, tmpdbl);
      }
  }
  msg_ack_cnt[i] = 0;
  msg_ack_cnt[i]++;
  msg_elap_tim[i] = 0.0;
  msg_elap_tim2[i] = 0.0;
} /* Find WD console */
} /* Message Presented */
if (strcmp(cur_log_rec.key_str, "170") == 0)
{ /* Message Acknowledged */
  for (i=0; i<3; i++)
  { /* Find WD console */
    if (cur_log_rec.mod_str[7] != wd_console_no[i])
      continue;
    msg_ack_cnt[i]--;
    tmp_long_int = atol(cur_log_rec.time_str);
    if (msg_ack_cnt == 0)
      msg_prev_tim[i] = tmp_long_int/30000.0;
    else
      { /* Multiple ACK */
        tmpflt = tmp_long_int/30000.0;
        msg_elap_tim[i] += (tmpflt - msg_prev_tim[i]);
        msg_elap_tim2[i] += pow((tmpflt - msg_prev_tim[i]),
        2.0);
        msg_prev_tim[i] = tmpflt;
      }
  }
  msg_ack_cnt[i] = 0;
} /* Find WD console */
} /* Message Acknowledged */
return;
} /* Read a logger entry */
void left_just(char *txt_str)
{ /* Left justify string */
  int blnk_sp;
  int i;

  blnk_sp = TRUE;
  while (blnk_sp)
  { /* Left justify */
    if (isspace(txt_str[0]))
    { /* Space found */
      for (i=0; i<strlen(txt_str); i++)
        txt_str[i] = txt_str[i+1];
    } /* Space found */
    else
      blnk_sp = FALSE;
  } /* Left justify */
  return;
} /* Left justify string */
int swt_label_match(struct switch_ent *swt_tab_ptr, int swt_cnt, int ent_no)
{ /* Search for matching switch name */
    extern struct evnt_skelt evnt_desc;

    struct switch_ent *swt_ptr;
    int match;

    match = FALSE;
    for (swt_ptr=swt_tab_ptr; swt_ptr<swt_tab_ptr+swt_cnt; swt_ptr++)
    { /* Search for matching Function switch */
        if (ent_no == -1)
            if (strcmp(swt_ptr->func_sw_name, evnt_desc.evnt_label.name_fld) != 0)
                continue;
        else
            { /* Alternate switch event */
                if (strcmp(swt_ptr->func_sw_name, evnt_desc.variant[ent_no].name_fld) != 0)
                    continue;
            }
        else
            { /* Alternate switch event */
                match = TRUE;
                if (ent_no == -1)
                    { /* Main event */
                        strcpy(evnt_desc.evnt_label.nbr_fld, swt_ptr->func_sw_num);
                        strcpy(evnt_desc.evnt_label.alt_nm_fld, swt_ptr->func_sw_alt_nm);
                    }
                else
                    { /* Alternate switch */
                        strcpy(evnt_desc.variant[ent_no].nbr_fld, swt_ptr->func_sw_num);
                        strcpy(evnt_desc.variant[ent_no].alt_nm_fld, swt_ptr->func_sw_alt_nm);
                    }
            }
    }
    return match;
} /* Search for matching switch name */
void blink_msg_on(int line_no, char *txt_str)
{ /* Blink message - ON */
extern void cntr_ln();

setattr(_BLINK);
 cntr_ln(line_no, txt_str);
 clrattr(_BLINK);
 refresh();
 return;
} /* Blink message _ ON */
void blink_msg_off(int line_no)
{ /* Blink message - OFF */
move(line_no, 0);
clrtoeol();
refresh();
return;
} /* Blink message - OFF */
void read_event()
{     /* Read event file */
    extern FILE *event_fp;

    extern void left_just();
    extern int swt_label_match();
    extern void opt_chk();

    char evnt_rec[81];
    int i;
    char *colon_ptr;
    int ndx;
    int match;
    struct switch_ent *swt_ptr;

    fgets(evnt_rec, 80, event_fp);
    for (i=0; i<strlen(evnt_rec); i++)
        evnt_rec[i] = toupper(evnt_rec[i]);    /* Change to UPPER case */

    ndx = strlen(evnt_rec) - 1;
    if (iscntrl(evnt_rec[ndx]))
        evnt_rec[ndx] = '\0';    /* Eliminate appended control character */
    left_just(evnt_rec);
    if (isdigit(evnt_rec[0]))
        /* Event found */
        /* Initialize event description */
        evnt_desc.evnt_seq[0] = '\0';
        evnt_desc.evnt_label.name_fld[0] = '\0';
        evnt_desc.evnt_label.nbr_fld[0] = '\0';
        evnt_desc.evnt_label.alt_nm_fld[0] = '\0';
        evnt_desc.evnt_label.tim_win = 0.0;
        evnt_desc.evnt_label.strt_tim = 0.0;
        evnt_desc.evnt_label.end_tim = 0.0;
        evnt_desc.no_var = 0;
        for (i=0; i<5; i++)
            evnt_desc.step_per_var[i] = 0;

    colon_ptr = strchr(evnt_rec, ':');
    ndx = colon_ptr - evnt_rec;
    strncpy(evnt_desc.evnt_seq, evnt_rec, ndx);
    evnt_desc.evnt_seq[ndx] = '\0';
    strcpy(evnt_desc.evnt_label.name_fld, &evnt_rec[ndx+1]);
    left_just(evnt_desc.evnt_label.name_fld);
    match = FALSE;
    match = swt_label_match(func_sw, func_sw_cnt, -1);
    if (!match)
        match = swt_label_match(req_sw, req_sw_cnt, -1);
    if (!match)
        match = swt_label_match(opt_sw, opt_sw_cnt, -1);
    if (!match)
        match = swt_label_match(panel_sw, panel_sw_cnt, -1);
if (match)    /* Check if single switch */
    opt_chk();

if (!match)
{ /* Action Label (Multiple switch sequence) */
    if (strncmp(evnt_desc.evnt_label.name_fld, "DONE", 4) == 0)
        return;
    opt_chk();
} /* Action Label (Multiple switch sequence) */
} /* Event found */
return;
} /* Read event file */
void opt_chk()
{ /* Parse for event options */
    extern FILE *event_fp;
    extern struct evnt_skelt evnt_desc;

    extern void left_just();
    extern int swt_label_match();
    extern void cntr_lno();

    char evnt_rec[81];
    int input_options;
    char *chr_ptr;
    int ndx, ndx1;
    int i;
    char cur_alt_label[9];
    char alt_label[9];
    int match;
    char err_str[81];
    char tmp_str[80];
    int tmp_int;

    input_options = TRUE;
    alt_label[0] = '\0';
    while (input_options)
    { /* Process event record */
        fgets(evnt_rec, 80, event_fp);
        if (strlen(evnt_rec) == 0) { input_options = FALSE; continue; }
        /* No more options */
        left_just(evnt_rec);
        for (i=0; i<strlen(evnt_rec); i++)
            evnt_rec[i] = toupper(evnt_rec[i]);
        evnt_rec[strlen(evnt_rec)-1] = '\0';
        while (evnt_rec[strlen(evnt_rec)-1] == ' ')
            evnt_rec[strlen(evnt_rec)-1] = '\0';

        if (strncmp(evnt_rec, "ALTERNATE", 9) == 0)
        { /* Alternate switch sequence */
            /* Search for label */
            /* Label found */
            if (isdigit(evnt_rec[i]))
                /* Label found */
            /* Search for label */
            if (strcmp(cur_alt_label, alt_label) != 0)
                /* Another variation */
strcpy(cur_alt_label, alt_label);
evnt_desc.no_var++;
} /* Another variation */
chr_ptr = strchr(evt_rec, '=');
ndx = chr_ptr - evt_rec + 1;
ndx1 = 0;
for (i=0; i<evnt_desc.no_var; i++)
    ndx1 += evnt_desc.step_per_var[i];
evnt_desc.step_per_var[evnt_desc.no_var-1]++;
strcpy(evnt_desc.variant[ndx1].name_fld, &evt_rec[ndx]);
left_just(evnt_desc.variant[ndx1].name_fld);
match = FALSE;
match = swt_label_match(func_sw, func_sw_cnt, ndx1);
if (!match)
    match = swt_label_match(req_sw, req_sw_cnt, ndx1);
if (!match)
    match = swt_label_match(opt_sw, opt_sw_cnt, ndx1);
if (!match)
    match = swt_label_match(panel_sw, panel_sw_cnt, ndx1);
if (!match)
    sprintf(err_str, "\"%s\" is either misspelled or does not exist",
            evnt_desc.variant[ndx1].name_fld);
clear();
cntr_in(y_mid, err_str);
getch();
return;
} /* Invalid switch */
continue;
} /* Alternate switch sequence */

if (strncmp(evt_rec, "WINDOW", 6) == 0)
{ /* Window option specified */
    chr_ptr = strchr(evt_rec, '=');
    ndx = chr_ptr - evt_rec + 1;
    for (i=ndx; i<strlen(evt_rec); i++)
    { /* Search for first numeric */
        if (isdigit(evt_rec[i]))
            /* Digit found */
            ndx = i;
            break;
    } /* Digit found */
} /* Search for first numeric */
if (evnt_desc.no_var == 0)
    sscanf(&evt_rec[ndx], "%f",
            &evnt_desc.evnt_label.tim_win);
else
    sscanf(&evt_rec[ndx], "%f",
            &evnt_desc.variant[evnt_desc.no_var-1].tim_win);
    continue;
} /* Window option specified */

if (strncmp(evt_rec, "KEY", 3) == 0)
{ /* Keyword - numeric form */
    chr_ptr = strchr(evnt_rec, '=');
    ndx = chr_ptr - evnt_rec + 1;
    strcpy(tmp_str, &evnt_rec[ndx]);
    left_just(tmp_str);
    if (evnt_desc.no_var == 0)
        strcpy(evnt_desc.evnt_label.nbr_fld, tmp_str);
    else
        strcpy(evnt_desc.variant[evnt_desc.no_var-1].nbr_fld, tmp_str);
    continue;
} /* Keyword - numeric form */

if (strncmp(evnt_rec, "TEXT", 4) == 0)
{ /* Matching Text */
    chr_ptr = strchr(evnt_rec, '\"');
    ndx = chr_ptr - evnt_rec + 1;
    chr_ptr = strchr(&evnt_rec[ndx], '\"');
    ndx1 = chr_ptr - &evnt_rec[ndx];
    if (evnt_desc.no_var == 0)
        strncpy(evnt_desc.evnt_label.add_mtvch_txt, &evnt_rec[ndx], ndx1);
    else
        strncpy(evnt_desc.variant[evnt_desc.no_var-1].add_mtvch_txt, &evnt_rec[ndx], ndx1);
    continue;
} /* Matching Text */
} /* Process event record */
return;
} /* Parse for event options */
void swt_state()
{
    /* Get state of switches */
    extern FILE *pass6_fp;
    extern int wd_cnt;
    extern char wd_console_no[];
    extern struct logrec cur_log_rec;

    extern void read_log();

    int i, j;
    char mod_name[10];
    long unsigned int swt_bit_map[2];
    long unsigned int bit_pic;
    int found;

    strcpy(mod_name, "Display_x");
    for (i=0; i<wd_cnt; i++)
    {
        /* Get switch state for each WD console */
        mod_name[8] = wd_console_no[i];
        fseek(pass6_fp, 0, SEEK_SET);
        found = FALSE;
        while (!found)
        {
            /* Search for switch state */
            read_log();
            if (strlen(cur_log_rec.time_str) == 0)
            {
                /* End of file or error */
                clear();
                cntr ln(y_mid, "EOF/error reading Logger: swt_state");
                getch();
                return;
            }
            /* Search for switch state */
            sscanf(&cur_log_rec.time_str[2], "%8x %8x",
                   &swt_bit_map[0], &swt_bit_map[1]);

            bit_pic = 1;
            for (j=0; j<9; j++)
            {
                /* Top 9 Category switches */
                if (((bit_pic<<j) & swt_bit_map[1]) != 0)
                {
                    req_sw[(j+4)*2].func_sw_state[i] = 1;
                    else
                        req_sw[(j+4)*2].func_sw_state[i] = 0;
                }
                /* Top 9 Category switches */
            }

            for (j=9; j<27; j++)
            {
                /* Optional Category switches */
                if (((bit_pic<<j) & swt_bit_map[1]) != 0)
                {
                    opt_sw[(j-9)*2].func_sw_state[i] = 1;
                    else
                        opt_sw[(j-9)*2].func_sw_state[i] = 0;
                }
            }
        }
    }
}
opt_sw[(j-9)*2].func_sw_state[i] = 0;
} /* Optional Category switches */

/* Bottom 9 Category switches */
for (j=27; j<32; j++)
{
    if (((bit_pic<<j) & swt_bit_map[1]) != 0)
        req_sw[(j-14)*2].func_sw_state[i] = 1;
    else
        req_sw[(j-14)*2].func_sw_state[i] = 0;
}
for (j=0; j<4; j++)
{
    if (((bit_pic<<j) & swt_bit_map[0]) != 0)
        req_sw[(j+18)*2].func_sw_state[i] = 1;
    else
        req_sw[(j+18)*2].func_sw_state[i] = 0;
}
for (j=4; j<12; j++)
{
    if (((bit_pic<<j) & swt_bit_map[0]) != 0)
        req_sw[j-4].func_sw_state[i] = 1;
    else
        req_sw[j-4].func_sw_state[i] = 0;
}
} /* Get switch state for each WD console */
fseek(pass6_fp, 0, SEEK_SET);
return;
} /* Get state of switches */
void console_chk()
{
/* Evaluate CONSOLE CHECKOUT */
extern int wd_cnt;
extern char wd_console_no[];
extern int req_sw_cnt;
extern struct switch_ent req_sw[];
extern int opt_sw_cnt;
extern struct switch_ent opt_sw[];
extern struct evnt_skelt evnt_desc;
extern struct log_rec cur_log_rec;

extern void read_log();
extern void cntr_lno();
extern void switch_chk();
extern void cs1_chk();
extern void quick_exit();

struct evnt_mrk beg_con;
int valid, chk_on_off;
int i,j;
int tmp_int;
char mod_name[10];
int key_code, on_off_state;
float elap_tim;
unsigned int tmp_long_int;
float tmp_flt;
char tmp_str[80];

/* Find starting event */
valid = FALSE;
while (!valid)
{
/* Search for starting event */
    beg_con.evnt_file_offset = ftell(pass6_fp);
    read_log();
    if (strlen(cur_log_rec.time_str) == 0)
    {
/* End of file or error */
        clear();
        cntr_lno(y_mid, "EOF/error in Logger: console_chk (1)" );
        getch();
        quick_exit();
    }
/* End of file or error */
    if (strcmp(cur_log_rec.key_str, evnt_desc.evnt_label.nbrFld) != 0) continue;
    if (strcmp(cur_log_rec.trns_str, evnt_desc.evnt_label.add_match_txt, strlen(evnt_desc.evnt_label.add_match_txt)) != 0)
    continue;
    valid = TRUE;
    strcpy(beg_con.mrk_evnt.time_str, cur_log_rec.time_str);
    tmp_long_int = atol(cur_log_rec.time_str);
    evnt_desc.evnt_label.strt_tim = tmp_long_int/30000.0;
    strcpy(beg_con.mrk_evnt.key_str, cur_log_rec.key_str);
}
strcpy(beg_con.mrk_evnt.trns_str, cur_log_rec.trns_str);
} /* Search for starting event */

for (i=0; i<wd_cnt; i++)
{ /* Cycle all WDs */
  fseek(pass6_fp, beg_con.evnt_file_offset, SEEK_SET);
  tmp_int = 0;
  for (j=0; j<req_sw_cnt; j++)
    tmp_int += req_sw[j].func_sw_state[i];
  for (j=0; j<opt_sw_cnt; j++)
    tmp_int += opt_sw[j].func_sw_state[i];
  strcpy(mod_name, "Switch_x");
  mod_name[7] = wd_console_no[i];
  if (tmp_int != 0)
  { /* All switches must be turned off */
    chk_on_off = FALSE;
    while (!chk_on_off)
    { /* Check for switch turn off */
      read_log();
      if (strlen(cur_log_rec.time_str) == 0)
      { /* End of file or error */
        clear();
        cntr_ln(y_mid, "EOF/error in Logger: console_chk (2)");
        getch();
        fseek(pass6_fp, beg_con.evnt_file_offset, SEEK_SET);
        return;
      } /* End of file or error */
      if (strcmp(mod_name, cur_log_rec.mod_str) != 0 ||
          strcmp(cur_log_rec.key_str, "101") != 0)
        continue;
      sscanf(cur_log_rec.trns_str, "%d %d", &key_code,
             &on_off_state);
      if (key_code < 80 || key_code > 159)
        continue;
      if (key_code > 87 && (fmod((double)key_code, 2.0) !=
          0.0))
        continue; /* Momentary switch */
      if (on_off_state == 1)
      { /* Check if all switches are off */
        if (key_code == 87)
        { /* Vector */
          req_sw[6].func_sw_state[i] = 0;
          req_sw[7].func_sw_state[i] = 1;
          continue;
        } /* Vector */
        if (key_code < 80 || key_code >7)
          continue; /* Ignore all other switches */
        tmp_int = 0;
        for (j=0; j<req_sw_cnt; j++)
          tmp_int += req_sw[j].func_sw_state[i];
        for (j=0; j<opt_sw_cnt; j++)
          tmp_int += opt_sw[j].func_sw_state[i];
      }
  }
} /* End of file or error */
if (tmp_int != 0)
    { /* Error - All switches not off */
        clear();
        cntr_ln(y_mid+1, "Error: Not all category and
        feature switches");
        cntr_ln(y_mid+2, "have been turned off");
        for (j=0; j<req_sw_cnt; j++)
            { /* Search for switch */
                if (req_sw[j].func_sw_state[i] == 1)
                    { /* Found a switch */
                        move(y_mid+3,0);
                        clrtoeol();
                        sprintf(tmp_str, "%s is ON",
                                req_sw[j].func_sw_name);
                        cntr_ln(y_mid+3, tmp_str);
                        getch();
                    } /* Found a switch */
            } /* Search for switch */
        for (j=0; j<opt_sw_cnt; j++)
            { /* Search for switch */
                if (opt_sw[j].func_sw_state[i] == 1)
                    { /* Found a switch */
                        move(y_mid+3,0);
                        clrtoeol();
                        sprintf(tmp_str, "%s is ON",
                                opt_sw[j].func_sw_name);
                        cntr_ln(y_mid+3, tmp_str);
                        getch();
                    } /* Found a switch */
            } /* Search for switch */
        move (y_mid,0);
        clrtoeol();
        refresh();
    } /* Error - All switches not off */
    chk_on_off = TRUE;
    continue;
} /* Check if all switches are off */

if (key_code >= 80 && key_code <= 87)
    { /* Feature & Vector - off */
        if (key_code == 80 || key_code == 81)
            { /* A/B - off */
                req_sw[0].func_sw_state[i] = 0;
                req_sw[1].func_sw_state[i] = 0;
            } /* A/B - off */
        else if (key_code == 82 || key_code == 83)
            { /* C/D - off */
                req_sw[2].func_sw_state[i] = 0;
                req_sw[3].func_sw_state[i] = 0;
            } /* C/D - off */
        else if (key_code == 84 || key_code == 85)
            { /* E/F - off */
                req_sw[4].func_sw_state[i] = 0;
                req_sw[5].func_sw_state[i] = 0;
            } /* E/F - off */
    } /* Feature & Vector - off */

93
else
{
    /* Vector */
    req_sw[6].func_sw_state[i] = 0;
    req_sw[7].func_sw_state[i] = 0;
}
/* Feature & Vector - off */

else if (keycode >= 106 && keycode <= 141)
{ /* Turning off optional category switch */
    for (j=0; j<opt_sw_cnt; j++)
    { /* Find switch and reset state */
        if (strncmp(optsw[j].funcswnum, cur_log_rec.trns_str, 3) != 0)
            continue;
        optsw[j].func_sw_state[i] = 0;
        break;
    }
    /* Find switch and reset state */
}
/* Turning off optional category switch */

else
{ /* Turning off other require switch */
    for (j=0; j<req_sw_cnt; j++)
    { /* Find switch and reset state */
        tmp_int = 2;
        if (keycode > 99)
            tmp_int = 3;
        if (strncmp(reqsw[j].funcswnum, cur_log_rec.trns_str, tmp_int) != 0)
            continue;
        reqsw[j].func_sw_state[i] = 0;
        break;
    }
    /* Find switch and reset state */
}
/* Turning off other require switch */
/* Check for switch turn off */
/* All switches must be turned off */

chk_on_off = FALSE;
fseek(pass6_fp, beg_con.evnt_file_offset, SEEK_SET);
while(!chk_on_off)
{ /* Required switches must be on */
    read_log();
    if (strlen(cur_log_rec.time_str) == 0)
    { /* End of file or error */
        clear();
        cntr_ln(y_mid, "EOF/error in Logger: console_chk (3)");
        getch();
        fseek(pass6_fp, beg_con.evnt_file_offset, SEEK_SET);
        return;
    }
    /* End of file or error */
    if (strcmp(mod_name, cur_log_rec.mod_str) != 0 ||
        strcmp(cur_log_rec.key_str, "101") != 0)
        continue;
}
sscanf(cur_log_rec.trns_str, "%d %d", &key_code, &on_off_state);
    if (on_off_state == 0)
        continue;
    if (key_code > 159)
        continue;
    if ((key_code > 87 && key_code < 160) &&
        (fmod((double)key_code, 2.0) != 0.0))
        continue; /* Momentary switch */

    if (key_code < 80)
        { /* Assume checkout complete */
            tmp_int = 0;
            for (j=1; j<8; j+=2)
                tmp_int += req_sw[j].func_sw_state[i];
            if (tmp_int < 4)
                continue;
            for (j=8; j<req_sw_cnt; j+=2)
                tmp_int += req_sw[j].func_sw_state[i];
            if (tmp_int < 22)
                { /* Error - Required switches not set */
                    clear();
                    cntr_ln(y_mid+1, "Error: Required switches not set");
                    for (j=1; j<3; j+=2)
                        { /* Feature & Vector */
                            if (req_sw[j].func_sw_state[i] == 0)
                                { /* Not set */
                                    move(y_mid+2,0);
                                    clrtoeol();
                                    sprintf(tmp_str, "%s is OFF",
                                    req_sw[j].func_sw_name);
                                    cntr_ln(y_mid+2, tmp_str);
                                    getch();
                                } /* Not set */
                        } /* Feature & Vector */
                    for (j=8; j<req_sw_cnt; j+=2)
                        { /* Category switches */
                            if (req_sw[j].func_sw_state[i] == 0)
                                { /* Not set */
                                    move(y_mid+2,0);
                                    clrtoeol();
                                    sprintf(tmp_str, "%s is OFF",
                                    req_sw[j].func_sw_name);
                                    cntr_ln(y_mid+2, tmp_str);
                                    getch();
                                } /* Not set */
                        } /* Category switches */
                    move(y_mid,0);
                    clrtoeol();
                    refresh();
                } /* Error - Required switches not set */
    tmp_long_int = atol(cur_log_rec.time_str);
    evnt_desc.evnt_label.end_time = tmp_long_int/30000.0;
    fseek(pass6_fp, beg_con.evnt_file_offset, SEEK_SET);
chk_on_off = TRUE;
continue;
) /* Assume checkout complete */

if (key_code == 86)
{ /* Vector */
    req_sw[6].func_sw_state[i] = 1;
    req_sw[7].func_sw_state[i] = 0;
} /* Vector */
else if (key_code >= 106 && key_code <= 141)
{ /* Turning on optional switch */
    for (j=0; j<opt_sw_cnt; j++)
    { /* Find and reset state */
        if (strncmp(opt_sw[j].func_sw_num,
                    cur_log_rec.trns_str, 3) != 0)
            continue;
        opt_sw[j].func_sw_state[i] = 1;
        break;
    } /* Find and reset state */
} /* Turning on optional switch */
else
{ /* Turning on required switch */
    for (j=0; j<req_sw_cnt; j++)
    { /* Find and reset state */
        tmp_int = 2;
        if (key_code > 99)
            tmp_int = 3;
        if (strncmp(req_sw[j].func_sw_num,
                    cur_log_rec.trns_str,
                    tmp_int) != 0)
            continue;
        req_sw[j].func_sw_state[i] = 1;
        break;
    } /* Find and reset state */
    tmp_int = 0;
    for (j=1; j<8; j+=2)
    { tmp_int += req_sw[j].func_sw_state[i];
        if (tmp_int < 4)
            continue;
        for (j=8; j<req_sw_cnt; j+=2)
        { tmp_int += req_sw[j].func_sw_state[i];
            if (tmp_int < 22)
                continue;
            tmp_long_int = atol(cur_log_rec.time_str);
            evnt_desc.evnt_label.end_tim = tmp_long_int/30000.0;
            chk_on_off = TRUE;
            continue;
        } /* Turning on required switch */
    } /* Turning on required switch */
} /* Required switches must be on */

if (evnt_desc.no var != 0)
{ /* Assume console assignment */
    evnt_desc.variant[0].tim_win =
    evnt_desc.evnt_label.tim_win;

}
event.evnt_desc.variant[0].strt_tim =
event.evnt_desc.evnt_label.strt_tim;
    switch_chk(&event.evnt_desc.variant[0]);
    if (event.evnt_desc.variant[0].end_tim != 0.0)
        event.evnt_desc.evnt_label.end_tim =
event.evnt_desc.variant[0].end_tim;
    } /* Assume console assignment */
} /* Cycle all WDs */
if (event.evnt_desc.evnt_label.strt_tim != 0.0
    && event.evnt_desc.evnt_label.end_tim != 0.0)
    { /* Console checkout complete */
        clear();
        elap_tim = event.evnt_desc.evnt_label.end_tim -
event.evnt_desc.evnt_label.strt_tim;
        move(y_mid, 10);
        printf("Console Checkout Completed (%f sec)", elap_tim);
        tmp_flt = elap_tim - event.evnt_desc.evnt_label.tim_win;
        move(y_mid+1, 15);
        if (tmp_flt <= 0.0)
            { /* Within window */
                printf("%f sec. within %f sec. window", tmp_flt,
                        event.evnt_desc.evnt_label.tim_win);
            } /* Within window */
        else
            { /* Outside window */
                printf("%f sec. outside of %f sec. window", tmp_flt,
                        event.evnt_desc.evnt_label.tim_win);
            } /* Outside window */
        getch();
    } /* Console checkout complete */
return;
} /* Evaluate CONSOLE CHECKOUT */
void switch_chk(struct evnt_var *swt_ent)
{ /* Evaluate switch action */
  extern struct evnt_skelt evnt_desc;
  extern struct log_rec cur_log_rec;
  extern int wd_cnt;
  extern char wd_console_no[];
  extern FILE *pass6_fp;
  extern struct evnt_var prev.swt;
  extern long int prev_rec_pos;

  extern void read_log();
  extern void csl_chk();
  extern void cnt_chk();
  extern void win_chk();
  extern void wmc_chk();

  int i;
  int valid;
  long int rec_pos;
  char mod_name[10];
  int key_code, on_off_state;
  int tmp_int;
  unsigned int tmp_long_int;
  float tmp_flt;
  float elap_tim;
  int swt_err_cnt;

  rec_pos = ftell(pass6_fp);
  if (prev_rec_pos < 0 || prev_rec_pos > rec_pos)
    prev_rec_pos = rec_pos;
  swt_ent->end_tim = 0.0;
  for (i=0; i<wd_cnt; i++)
    { /* Process switch action */
      fseek(pass6_fp, rec_pos, SEEK_SET);
      swt_err_cnt = 0;
      strcpy(mod_name, "Switch -x");
      mod_name[7] = wd_console_no[i];
      valid = FALSE;
      while (!valid)
        { /* Search for starting switch selection */
          read_log();
          if (strlen(cur_log_rec.time_str) == 0)
            { /* End of file or error */
              clear();
              cntr_ln(y_mid, "EOF/error in Logger: switch_chk");
              getch();
              fseek(pass6_fp, rec_pos, SEEK_SET);
              return;
            } /* End of file or error */
          if (swt_ent->strt_tim != 0.0)
            { /* Time constraint search */
              tmp_long_int = atol(cur_log_rec.time_str);
              tmp_flt = tmp_long_int/30000.0;
            } /* Time constraint search */
        } /* Search for starting switch selection */
    } /* Process switch action */
  if (prev_rec_pos < 0 || prev_rec_pos > rec_pos)
    prev_rec_pos = rec_pos;
  swt_ent->end_tim = 0.0;
  for (i=0; i<wd_cnt; i++)
    { /* Process switch action */
      fseek(pass6_fp, rec_pos, SEEK_SET);
      swt_err_cnt = 0;
      strcpy(mod_name, "Switch -x");
      mod_name[7] = wd_console_no[i];
      valid = FALSE;
      while (!valid)
        { /* Search for starting switch selection */
          read_log();
          if (strlen(cur_log_rec.time_str) == 0)
            { /* End of file or error */
              clear();
              cntr_ln(y_mid, "EOF/error in Logger: switch_chk");
              getch();
              fseek(pass6_fp, rec_pos, SEEK_SET);
              return;
            } /* End of file or error */
          if (swt_ent->strt_tim != 0.0)
            { /* Time constraint search */
              tmp_long_int = atol(cur_log_rec.time_str);
              tmp_flt = tmp_long_int/30000.0;
            } /* Time constraint search */
        } /* Search for starting switch selection */
    } /* Process switch action */
} /* switch_chk */
if ((tmp_fit - swt_ent->strt_tim) > swt_ent->tim_win)
{ /* Check if same follow-on switch action */
  if (strcmp(prev_swt.name_fid, swt_ent->name_fid) == 0)
  { /* Same follow-on switch action */
    fseek(pass6_fp, prev_rec_pos, SEEK_SET);
    valid = TRUE;
    continue;
  } /* Same follow-on switch action */
  return;
} /* Check if same follow-on switch action */

/* Time constraint search */
if (strcmp(cur_log_rec.mod_str, mod_name) != 0)
  strcmp(cur_log_rec.key_str, "101") != 0)
  continue;
if ((strncmp(swt_ent->nbr_fid, cur_log_rec.trans_str, strlen(swt_ent->nbr_fid)) != 0)
  continue;
if (swt_ent->strt_tim == 0.0)
{ /* Get start time */
  tmp_long_int = atol(cur_log_rec.time_str);
  swt_ent->strt_tim = tmp_long_int/30000.0;
} /* Get start time */
valid = TRUE;

/* Search for starting switch selection */

swt_ent->end_tim = 0.0;
strcpy(mod_name, "Display_x");
mod_name[8] = wd_console_no[i];
valid = FALSE;
while (!valid)
{ /* Search for switch action completion */
  read_log();
  if (strlen(cur_log_rec.time_str) == 0)
  { /* End of file or error */
    clear();
    cntr ln(y_md, "EOF/error in Logger: switch_chk");
    getch();
    fseek(pass6_fp, rec_pos, SEEK_SET);
    return;
  } /* End of file or error */
  tmp_long_int = atol(cur_log_rec.time_str);
  tmpflt = tmp_long_int/30000.0;
  if ((tmpflt - swt_ent->strt_tim) > swt_ent->tim_win)
  { /* Switch action not completed in time */
    clear();
    move(y_md, 10);
    printf("%s not completed in %f sec. (%d errors)", 
      swt_ent->name_fid,
      swt_ent->tim_win, swt_err_cnt);
    getch();
    valid = TRUE;
    continue;

99
if (strcmp(curlogrec.modstr, modname) != 0 ||
    strcmp(curlogrec.keystr, "848") != 0)
    continue;
if (strncmp(curlogrec.trnsstr, swtent->alt nm fld, strlen(swtent->altnmfld)) != 0)
    continue;
if (curlogrec.trnsstr[strlen(swtent->altnmfld)+1] != '2')
    { /* Input error */
        swt_err_cnt++;
        continue;
    } /* Input error */
tmplongint = atol(curlogrec.time_str);
swt_ent->end_time = tmplongint/30000.0;
elap_tim = swt_ent->end_time - swt_ent->start_time;
clear();
move(ymid,10);
printw("%s COMPLETED (%f sec) (%d errors)",
swt_ent->name fld, elap_tim,
    swt_err_cnt);
tmpflt = elap_tim - swt_ent->time_win;
move(ymid+1,15);
if (tmpflt <= 0.0)
    { /* Within window */
        swtent->time_win;
    }
else
    { /* Outside window */
        swtent->time_win;
    }
getch();

/* Save last processed switch action */
prev_rec_pos = ftell(pass6_fp);
strcpy(prevswt.namefld, swtent->namefld);
strcpy(prevswt.nbrfld, swtent->nbrfld);
strcpy(prevswt.alt nfld, swtent->altnmfld);
strcpy(prevswt.add mtchtxt, swtent->add mtchtxt);
prevswt.time_win = swtent->time_win;
prevswt.start time = swtent->start_time;
prevswt.end time = swtent->end_time;
if (strcmp(swtent->altnmfld, "CSL") == 0)
    csl chk(i);
if (strcmp(swtent->altnmfld, "CMT") == 0)
    cmt chk();
if (strcmp(swtent->altnmfld, "WIN") == 0)
    win chk();
if (strcmp(swtent->altnmfld, "WMC") == 0)
    wmc chk();
    valid= TRUE;
} /* Search for switch action completion */
{ /* Process switch action */
    return;
} /* Evaluate switch action */
void csl_chk(int ndx)
{ /* Evaluate Assign Console Switch */
extern struct log_rec cur_log_rec;
extern int wd_cnt;
extern char wd_id_no[];

if (strlen(cur_log_rec.trns_str) == 6)
{ /* Assign console TD */
    move(y_mid+2, 0);
    clrtoeol();
    mvinsstr(y_mid+2, 15, "Assign Console TD requested");
    getch();
    return;
}

/* Assign console TD */
if (strncmp(&cur_log_rec.trns_str[6], "WD", 2) != 0)
{ /* Error - Station type */
    move(y_mid+2, 0);
    clrtoeol();
    mvinsstr(y_mid+2, 15, "Error: (CSL) Station type specification");
    getch();
}

/* Error - Station type */
if (cur_log_rec.trns_str[9] != wd_id_no[ndx])
{ /* Error - Position number */
    move(y_mid+2, 0);
    clrtoeol();
    mvinsstr(y_mid+2, 15, "Error: (CSL) Position number specification");
    getch();
}

/* Error - Position number */
return;
} /* Evaluate Assign Console Switch */
void cmt_chk()
{ /* Evaluate Commit Switch */
extern struct log_rec cur_log_rec;

extern void left_just();

int i, j;
char *brkt;
int ndx;
char cmt_str[80];
char obj_str[5];
char tgt_str[5];
int tgt_fnd;
char *bink;
char cmt_type, cmt_intercept;

strcpy(cmt_str, cur_log_rec.trns_str);
for (i=0; i<strlen(cmt_str); i++)
{ /* Eliminate "hook" character */
   if (cmt_str[i] != '\x9f')
      continue;
   cmt_str[i] = '\';
   brkt = strchr(&cmt_str[i], '}');
   ndx = brkt - cmt_str;
   for (j=i; j<ndx-4; j++)
      cmt_str[j] = '\';
   cmt_str[ndx] = '\';
   left_just(&cmt_str[i]);
} /* Eliminate "hook" character */

strncpy(obj_str, &cmt_str[6], 4);
obj_str[4] = '\0';

for (i=6; i<10; i++)
   cmt_str[i] = '\';
left_just(&cmt_str[6]);

ndx = 6;
tgt_fnd = FALSE;
tgt_str[0] = '\0';
while (!tgt_fnd)
{ /* Search for target */
   if (strlen(&cmt_str[ndx]) < 4)
   { /* No string */
      tgt_fnd = TRUE;
      continue;
   } /* No string */
   blnk = strchr(&cmt_str[ndx], ' ');
   if (blnk == NULL)
   { /* End of "cmt_str" */
      if (strlen(&cmt_str[ndx]) == 4)
      { /* Assume target */
         strncpy(tgt_str, &cmt_str[ndx], 4);
      }
   }
      } /* Search for target */
} /* Evaluate Commit Switch */
tgt_str[4] = '\0';

/* Assume target */
tgt_fnd = TRUE;
continue;

} /* End of "cmt_str" */
if (blnk & cmt_str[ndx]) < 3)
{ /* Not target */
    ndx += (blnk - &cmt_str[ndx]) + 1;
    continue;
} /* Not target */
strncpy(tgt_str, &cmt_str[ndx], 4);
tgt_str[4] = '\0';
tgt_fnd = TRUE;

} /* Search for target */

if (strlen(tgt_str) != 0)
{ /* Shorten string */
    for (i=ndx; i<ndx+4; i++)
        cmt_str[i] = ' ';
    left_just(&cmt_str[ndx]);
} /* Shorten string */

cmt_type = 'D';
cmt_intercept = 'C';
for (i=6; i<strlen(cmt_str); i++)
{ /* Get Commit type & intercept */
    if (isspace(cmt_str[i]))
        continue;
    if (cmt_str[i] == 'I' || cmt_str[i] == 'D')
        cmt_type = cmt_str[i];
    else
        cmt_intercept = cmt_str[i];
} /* Get Commit type & intercept */
move(y_mid+2, 15);
clrtoeol();
printw("%s committed on %s using", obj_str, tgt_str);
move(y_mid+3, 15);
clrtoeol();
printw("Intercept geometry: %c Mission: %c", cmt_intercept, cmt_type);
getch();

return;
} /* Evaluate Commit Switch */
void win_chk()
{ /* Evaluate Init Switch Action */
 extern struct log_rec cur_log_rec;

 extern void read_log();

 int crrlt;
 char cor_obj[5];

crrlt = FALSE;
while (!crrlt)
{ /* Find correlation */
   read_log();
   if (strcmp(cur_log_rec.mod_str, "target_1") != 0 ||
      strcmp(cur_log_rec.key_str, "845") != 0)
      continue;
   strncpy(cor_obj, &cur_log_rec.trns_str[strlen(cur_log_rec.trns_str)-4]);
crrlt = TRUE;
} /* Find correlation */

move(y_mid+2, 15);
printw("%s was initiated", cor_obj);
getch();

return;
} /* Evaluate Init Switch Action */
void wmc_chk()
{ /* Evaluate Mode IV Switch Action */
extern struct log_rec cur_log_rec;

char *brkt;
int ndx;
char mode_obj[5];

if (strchr(cur_log_rec.trns_str, '\x9f') != NULL)
{ /* Find object */
    brkt = strchr(cur_log_rec.trns_str, '}');
    ndx = brkt - cur_log_rec.trns_str - 4;
    strncpy(mode_obj, &cur_log_rec.trns_str[ndx], 4);
    mode_obj[4] = '\0';
} /* Find object */
else
    return;

move(y_mid+2,15);
printw("Mode IV check on %s", mode_obj);
getch();
return;
} /* Evaluate Mode IV Switch Action */
void multi_switch()
{
    /* Evaluate switch sequence variations */
    extern struct evnt_skelt evnt_desc;
    extern int wd_cnt;
    extern char wd_console_no[];
    extern char wd_id_no[];
    extern FILE *pass6_fp;
    extern struct log_rec cur_log_rec;
    extern long int prev_evnt_strt_rec;

    int i, j, k;
    long int rec_pos;
    char mod_name[10];
    int valid;
    int strt_ent;
    unsigned int tmp_long_int;
    float elap_tim;

    rec_pos = ftell(pass6_fp);
    if (prev_evnt_strt_rec < 0 || prev_evnt_strt_rec > rec_pos)
        prev_evnt_strt_rec = rec_pos;
    else
        rec_pos = prev_evnt_strt_rec;
    for (i=0; i<wd_cnt; i++)
    {
        /* Process switch sequences */
        fseek(pass6_fp, rec_pos, SEEK_SET);
        strcpy(mod_name, "Display x");
        mod_name[8] = wd_console_no[i];
        valid = FALSE;
        while (!valid)
        {
            /* Find start */
            fseek(pass6_fp, rec_pos, SEEK_SET);
            char temp[] = mod_name;
            if (strcmp(cur_log_rec.mod_str, mod_name) != 0 ||
                strcmp(cur_log_rec.key_str, evnt_desc.evnt_label.nbrFld) != 0 ||
                strncmp(cur_log_rec.trans_str, evnt_desc.evnt_label.addMtchText, strlen(evnt_desc.evnt_label.addMtchText)) != 0)
                continue;
            tmp_long_int = atol(cur_log_rec.time_str);
            evnt_desc.evnt_label.strtTim = tmp_long_int/30000.0;
            rec_pos = ftell(pass6_fp);
            prev_evnt_strt_rec = rec_pos;
            valid = TRUE;
        }
    /* Find start */

    strt_ent = 0;
    for (j=0; j<evnt_desc.no_var; j++)
    {
        /* Process a switch sequence */
        fseek(pass6_fp, rec_pos, SEEK_SET);
        for (k=strt_ent; k<strt_ent+evnt_desc.step_per_var[j]; k++)
        {
            /* Variation j */
            evnt_desc.variant[k].strtTim =

    /* Other code follows */
}
evnt_desc.evnt_label.strt_tim;
evnt_desc.variant[k].tim_win =
evnt_desc.evnt_label.tim_win;
switch_chk(&evnt_desc.variant[k]);
if (evnt_desc.variant[k].end_tim == 0.0)
{ /* Switch start not found */
    break;
} /* Switch start not found */
evnt_desc.variant[k].end_tim =
evnt_desc.evnt_label.end_tim =
} /* Variation j */
if (evnt_desc.evnt_label.end_tim == 0.0)
{ /* Start not found */
    strt_ent += evnt_desc.step_per_var[j];
    continue;
} /* Start not found */
elap_tim = evnt_desc.evnt_label.end_tim -
evnt_desc.evnt_label.strt_tim;
clear();
move(y_mid,10);
printw("%s Completed (%f sec)",
      evnt_desc.evnt_label.name_fld, elap_tim);
getch();
break;
} /* Process a switch sequence */
if (evnt_desc.evnt_label.end_tim == 0.0)
{ /* Event not found */
clear();
move(y_mid,10);
printw("%s Not Found", evnt_desc.evnt_label.name_fld);
getch();
} /* Event not found */
} /* Process switch sequences */
return;
} /* Evaluate switch sequence variations */
NAME: ITS.C

#include <curses.h>
#include <string.h>
#include "itsdef.h"

main()
{
  /* Intelligent Tutor */
  extern int x_mid;
  extern int y_mid;
  extern char instr_path[];
  extern instruct_blk_open;
  extern char stdt_path[];
  extern int instr_review;
  extern int user_type;
  extern int first_ques;

  int pick;
  int x_coord, y_coord;
  int rtn_state;
  char buf[81];

  /******************************************************************************/
  /* Initialize window for Intelligent Tutor interface */
  VT100: no action is required to establish a window.
  IRIS: "wsh -n 'ITS Menu' -p2,2 -s113,52" is used to establish a full screen window.
  /******************************************************************************/
  initscr();

  /******************************************************************************/
  /* Gain control of keyboard */
  /******************************************************************************/
  cbreak();     /* Pass all keystroke to program */
  noecho();     /* Suppress character echoing */
  nonl();       /* Suppress carriage return recognition */
  keypad(stdscr,TRUE);  /* Access user's keypad */

  /******************************************************************************/
  /* Instructor/Student function selection */
  /******************************************************************************/
  x_mid = COLS/2 - 1;
  y_mid = LINES/2 - 1;
  instr_path[0] = '\0';
  instruct_blk_open = FALSE;
  stdt_path[0] = '\0';
  instr_review = -1;
  first_ques = -1;

  cntr_ln(y_mid - 2, "Use cursor to select:");

109
attrset(A_STANDOUT);
mvaddstr(y_mid, x_mid - 10, "Instructor");
attrset(0);
mvaddstr(y_mid, x_mid + 3, "Student");
cntr_ln(y_mid + 2, "then press 'Enter'");
move(y_mid, x_mid - 10);

******************************************************************************
Move and highlight appropriate field according to cursor use, then
retrive desired selection.
******************************************************************************

for (;;)
 { /* Wait for Instructor/Student selection */
   x_coord = 0;
   y_coord = 0;
   pick = getch();
   switch(pick)
   { /* Examine keyboard input */
     case KEY_LEFT:
       attrset(A_STANDOUT);
       mvaddstr(y_mid, x_mid - 10, "Instructor");
       attrset(0);
       mvaddstr(y_mid, x_mid + 3, "Student");
       move(y_mid, x_mid - 10);
       break;

     case KEY_RIGHT:
       attrset(0);
       mvaddstr(y_mid, x_mid - 10, "Instructor");
       attrset(A_STANDOUT);
       mvaddstr(y_mid, x_mid + 3, "Student");
       attrset(0);
       move(y_mid, x_mid + 3);
       break;

     case '\r':
     case 0x157:
       getyx(stdscr,y_coord,x_coord);
       break;

     default:
       break;
   } /* Examine keyboard input */
   if (x_coord != 0)
     break;
 } /* Wait for Instructor/Student selection */

if (x_coord == x_mid - 10)
 { /* Instructor */
   getstr(buf);
   if (strcmp("password",buf) != 0)
     { /* Feeble attempt at security */
       endwin();
       break;
     }
   else
     /* Do something */
 }
exit(0);
} /* Feeble attempt at security */
user_type = Instructor;
rtm_state = 1;
while (rtm_state != 0)
{ /* Continue to call */
    instructor(&rtm_state);
    if (instr_review >= 0)
    { /* Review an instruction block */
        student(&rtm_state);
        instr_review = -1;
    } /* Review an instruction block */
    } /* Continue to call */
} /* Instructor */
else
{ /* Student */
    user_type = Student;
    student(&rtm_state);
    if (rtm_state == 0)
        its_stop();
    clear();
    refresh();
} /* Student */
endwin();
exit(0);
} /* Intelligent Tutor */
NAME: ITSDEF.H

/*******************************************************************************
 Common variables
*******************************************************************************/
#define Instructor 157
#define Student 208
#define instr_path_def "/usr/people/neal/instructor"
#define stdt_path_def "/usr/people/neal/student"

int user_type; /* Indicator of user type (Instructor/Student) */
int x_mid; /* x coordinate of screen center */
int y_mid; /* y coordinate of screen center */

/*******************************************************************************
 Variables for Instructor
*******************************************************************************
char instrpath[80]; /* Directory string to Instructor files */
int instructblk_open; /* Indicator that file open and in memory */
int instr_blk_cnt; /* Count of exiting instruction blocks */
int instr_review; /* If >= 0 then curricula review occuring */
    /* & value is entry index into instruct.blk */
FILE *lesson_fp; /* Declare file pointer for lesson block */

/*******************************************************************************
 Structure for a block of instruction
 Each occurrence of "curricula" represents a block of instruction
 title = brief description
 no_qa_sets = number of Q&A sets in this instructional block
 qa_sets = each occurrence is a file containing one or more
 related Q&As.
*******************************************************************************
struct filnms
{ /* 10 character fields */
    char names[10]; /* Name of a lesson file */
}; /* 10 character fields */

struct curricula
{ /* Lesson blk record */
    char title[80]; /* Brief description of instruction block */
    int no_qa_sets; /* Number of lessons in instructional block */
    struct filnms qa_sets[50]; /* Names of lessons in instruction block */
}; /* Lesson blk record */

struct curricula instr_blks[50];

/*******************************************************************************
 Variables for Student
*******************************************************************************
char stdt_path[80]; /* Directory string to Student files */
FILE *student_fp; /* Declare file pointer for student profile */
FILE *present.fp; /* Declare file pointer for lesson presentation */
int stdt_acc_score; /* Student's accumulated score */
int first ques; /* Index of 1st question in a lesson */

="/***********************************************************************************/
 Structure for a student's data base
***********************************************************************************/
struct level
{
    int max_val; /* Maximum value accumulation */
    int min_val; /* Minimum value accumulation */
    int act_val; /* Actual value accumulation */
    int spare_val; /* Spare */
};

struct stdt_db
{
    /* Student's DB */
    char ssan[9]; /* SSAN */
    char name[26]; /* Name */
    int instr_blkndx; /* Index into instr_blks */
    int lessonndx; /* Index of lesson In instr_blks entry */
    int textblkndx; /* Numeric id of text to be presented */
    int knowphase_ndx; /* Knowledge level/phase index */
    struct level know_phase[5]; /* Student's DB */
};

struct stdt_db pupil;

struct ans_val_entry
{ /* Entry for multiple choice */
    char ans_desig; /* Designation of answer, e.g. a,b,c...,or 1,2,3,... */
    int ans_val; /* Value of the answer */
}; /* Entry for multiple choice */

struct lesson_file_entry
{ /* Entry in lesson file map */
    int num_id; /* Numeric identifier of text */
    int text_type; /* Type of text */
    long int file_pos; /* Offset from start of file */
    int prev_num_id; /* Numeric id of previous text */
    int know_level[5]; /* The follow-on text for each knowledge level */
        /* and correspond to the num_id of the text. */
    int mc_ans_cnt; /* Number of entries in mc_ans */
    struct ans_val_entry mc_ans[8];
    char stdt_ans;
}; /* Entry in lesson file map */

struct lesson_file_entry lesson_map[200];
NAME: ITS_RUTN.CWP

#include <curses.h>  /* <stdio.h>, <termio.h>, <unctrl.h> */
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include "itsdef.h"
void cntr_ln(int y_coord, char *txt_str)
{ /* cntr_ln */
extern int x_mid;
extern int y_mid;

int i, x_offset;

    x_offset = 0;
    for (i=0; i<=strlen(txt_str)-1; i++)
    { /* Search for printable characters */
        if (isprint(*(txt_str+i)))
            x_offset++;
    } /* Search for printable characters */
    x_offset /= 2;
    mvaddstr(y_coord, x_mid - x_offset, txt_str);
    return;
}
} /* cntr_ln */
void get_path_str(char * directory, char * path_str)
( /* Get path to specified directory */
extern int x_mid;
extern int y_mid;

int reply;
char buf[80];

if (strlen(path_str) != 0)
( /* Directory path already specified */
  return;
) /* Directory path already specified */

strcpy(buf, "Enter path to "");
strcat(buf, directory);
strcat(buf, " file directory");
clear();
ctnl(y_mid-1, buf);
move(y_mid, x mid-(strlen(buf)/2), "->");
move(y_mid, x mid-(strlen(buf)/2)+3);

while (TRUE)
( /* Get directory path */
echo();
clrtoeol();
getstr(path_str);
noecho();
if (strlen(path_str) == 0)
( /* Use default path */
  move(y mid, x mid-(strlen(buf)/2)+3);
  if (strcmp(directory, "Instructor") == 0)
    ( /* Use instructor path default */
      addstr(instr_path_def);
      strcpy(path_str, instr_path_def);
    ) /* Use instructor path default */
  else
  ( /* Use student path default */
    addstr(stdt_path_def);
    strcpy(path_str, stdt_path_def);
  ) /* Use student path default */
  refresh();
) /* Use default path */
ctnl(y mid+2, "Is path correct? [Y/N/Q]: ");
reply = tolower(getch());
if (reply == 'y')
  break;
if (reply == 'q')
( /* Terminate process */
  *path_str = '\0'; /* Set NULL string */
  break;
) /* Terminate process */
move(y mid+2, 0);
clrtoeol();
refresh();
move(y_mid, x_mid-(strlen(buf)/2)+3);
) /* Get directory path */
clear();
refresh();
return;

) /* Get path to specified directory */
void file_str(char *path, char *filename, char *fullstr)
{
    /* Form full filestring */
    int trim;
    char *blank_ptr;

    strcpy(fullstr, path);
    strcat(fullstr, "/");
    strcat(fullstr, filename);
    trim = TRUE;
    while (trim)
    {
        /* Trim of trailing blanks */
        blank_ptr = strchr(fullstr, ' ');
        if (blank_ptr == NULL)
        {
            /* Blanks trimmed off */
            trim = FALSE;
            continue;
        } /* Blanks trimmed off */
        fullstr[blank_ptr-fullstr] = NULL;
    } /* Trim of trailing blanks */
    return;
} /* Form full filestring */
void its_stop()
{
    /* Halt ITS unconditionally */
    endwin();
    exit(0);
}
/* Halt ITS unconditionally */
int chk_file(char *path, char *filename)
{
    /* Check accessibility */
    extern int x_mid;
    extern int y_mid;
    extern int user_type;

    int reply;
    int status;
    char fullstr[80], txtstr[80];

    file_str(path, filename, fullstr);
    status = access(fullstr, F_OK);
    if (status == 0) return(0);
    clear();
    if (errno == ENOTDIR)
        printf("Path specification error in \"\%s\"", path);
    else if (errno == ENOENT)
    {
        /* File does not exist */
        if (user_type == Student)
            return(1);
        clear();
        sprintf(txtstr, "File \"\%s\" does not exist", fullstr);
        cntr_ln(y_mid-1, txtstr);
        cntr_ln(y_mid, "Continue [Y/N]: ");
        reply = tolower(getch());
        if (reply == 'y')
        { /* Indicate condition */
            clear();
            refresh();
            return(1);
        } /* Indicate condition */
    } /* File does not exist */
    else
        printf("File access denied w/ errno = \%d", errno);
    refresh();
    its_stop();
}

/* Check accessibility */
void lesson_blk_io(char io_type, int ndx)
{ /* Provide I/O for lesson block file */
extern FILE *lesson_fp;

char shrt_str[81];
char rec_str[635];
int offset;
int i;

if (tolower(io_type) == 'w')
{ /* Write a record of lesson block */
    sprintf(rec_str, "%s;%02d", instr_blks[ndx].title,
            instr_blks[ndx].no_qa_sets);
    for (i=0; i<50; i++)
    { /* Format qa file names */
        strcat(rec_str, "\n");
        strcat(rec_str, instr_blks[ndx].qa_sets[i].names, 10);
    }
    sprintf(rec_str, "%\n");
    fseek(lesson_fp, 634*ndx, SEEK_SET);
    fputs(rec_str, lesson_fp);
}
/* Write a record of lesson block */

else if (tolower(io_type) == 'r')
{ /* Read a record of lesson block */
    fseek(lesson_fp, 634*ndx, SEEK_SET);
    fgets(rec_str, 634, lesson_fp);

    strncpy(instr_blks[ndx].title, rec_str, 80);
    offset = 81;
    sscanf(&rec_str[offset], "%2d", &instr_blks[ndx].no_qa_sets);
    offset += 3;
    for (i=0; i<50; i++)
    { /* Unblock qa file names */
        strncpy(instr_blks[ndx].qa_sets[i].names, rec_str+offset, 10);
        offset += 11;
    }
    /* Unblock qa file names */
}
/* Read a record of lesson block */

else
{ /* Error in I/O specification */
    clear();
    addstr("lesson_blk_io error: I/O type specification");
    refresh();
    its_stop();
}
/* Error in I/O specification */
return;
}
/* Provide I/O for lesson block file */
void work_msg()
{
    /* Present blinking 'working...' message */
    clear();
    attrset(A_BLINK);
    cntr_ln(y_mid, "Working...");
    attrset(0);
    refresh();
    return;
} /* Present blinking 'working...' message */
void getstr_echo(char *str) {
    /* Enable character echoing with getstr function */
    echo();
    getstr(str);
    noecho();
} /* Enable character echoing with getstr function */
void strg_blkn_pad(char *strng, int str_len)
{ /* Pad a left justified string with blanks */
    int i;

    if (strlen(strng) > str_len)
    {
        cntr_ln(22, "String too long");
        cntr_ln(23, "Press any key to continue");
        getch();
        return;
    }

    if (strlen(strng) < str_len)
    { /* Pad with blanks */
        for (i=strlen(strng); i<str_len; i++)
            strng[i] = ' ';
    } /* Pad with blanks */
    return;
}

/* Pad a left justified string with blanks */
NAME: ITS_RUTN.C

#include <curses.h>    /* <stdio.h>, <termio.h>, <unctrl.h> */
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include "itsdef.h"
void cntr_ln(int y_coord, char *txt_str)
{ /* cntr_ln */
extern int x_mid;
extern int y_mid;

int i, x_offset;

    x_offset = 0;
    for (i=0; i<=strlen(txt_str)-1; i++)
    { /* Search for printable characters */
        if (isprint(*(txt_str+i)))
            x_offset++;
    } /* Search for printable characters */
    x_offset /= 2;
    mvaddstr(y_coord, x_mid - x_offset, txt_str);
    return;

} /* cntr_ln */
void get_path_str(char * directory, char * path_str)
{ /* Get path to specified directory */
  extern int x_mid;
  extern int y_mid;

  int reply;
  char buf[80];

  if (strlen(path_str) != 0)
  { /* Directory path already specified */
    return;
  } /* Directory path already specified */

  strcpy(buf, "Enter path to ");
  strcat(buf, directory);
  strcat(buf, " file directory");
  clear();
  cntr_ln(y_mid-1, buf);
  mvaddstr(y_mid, x_mid-(strlen(buf)/2), "->");
  move(y_mid, x_mid-(strlen(buf)/2)+3);

  while (TRUE)
  { /* Get directory path */
    echo();
    clrtoeol();
    getstr(path_str);
    noecho();
    if (strlen(path_str) == 0)
    { /* Use default path */
      move(y_mid, x_mid-(strlen(buf)/2)+3);
      if (strcmp(directory, "Instructor") == 0)
      { /* Use instructor path default */
        addstr(instr_path_def);
        strcpy(path_str, instr_path_def);
      } /* Use instructor path default */
      else
      { /* Use student path default */
        addstr(stdt_path_def);
        strcpy(path_str, stdt_path_def);
      } /* Use student path default */
      refresh();
    } /* Use default path */
    cntr_ln(y_mid+2, "Is path correct? [Y/N/Q]: ");
    reply = tolower(getch());
    if (reply == 'y')
      break;
    if (reply == 'q')
    { /* Terminate process */
      *path_str = '\0'; /* Set NULL string */
      break;
    } /* Terminate process */
    move(y_mid+2, 0);
    clrtoeol();
  } /* Get directory path */
} /* Get path to specified directory */
refresh();
move(y_mid, x_mid-(strlen(buf)/2)+3);
) /* Get directory path */

clear();
refresh();
return;

} /* Get path to specified directory */
void file_str(char *path, char *filename, char *fullstr)
{ /* Form full filestring */
  int trim;
  char *blank_ptr;

  strcpy(fullstr, path);
  strcat(fullstr, "/");
  strcat(fullstr, filename);
  trim = TRUE;
  while (trim)
    { /* Trim of trailing blanks */
      blank_ptr = strrchr(fullstr, ' ');
      if (blank_ptr == NULL)
        { /* Blanks trimmed off */
          trim = FALSE;
          continue;
        } /* Blanks trimmed off */
      fullstr[blank_ptr-fullstr] = NULL;
    } /* Trim of trailing blanks */
  return;
} /* Form full filestring */
void its_stop()
{ /* Halt ITS unconditionally */
    endwin();
    exit(0);
} /* Halt ITS unconditionally */
int chk_file(char *path, char *filename)
{
    extern int x_mid;
    extern int y_mid;
    extern int user_type;

    int reply;
    int status;
    char fullstr[80], txtstr[80];

    file_str(path, filename, fullstr);
    status = access(fullstr, F_OK);
    if (status == 0) return(0);
    clear();
    if (errno == ENOTDIR)
        printf("Path specification error in \"%s\", path);
    else if (errno == ENOENT)
    { /* File does not exist */
        if (user_type == Student)
            return(1);
        clear();
        sprintf(txtstr, "File \"%s\" does not exist", fullstr);
        cntr_ln(y_mid-1, txtstr);
        cntr_ln(y_mid, "Continue [Y/N]: ");
        reply = tolower(getch());
        if (reply == 'y')
        { /* Indicate condition */
            clear();
            refresh();
            return(1);
        } /* Indicate condition */
    } /* File does not exits */
    else
        printf("File access denied w/ errno = %d", errno);
    refresh();
    its_stop();
}
/* Check accessibility */

void lesson_blk_io(char io_type, int ndx)
{ /* Provide I/O for lesson block file */
  extern FILE *lesson_fp;

  char shrt_str[81];
  char rec_str[635];
  int offset;
  int i;

  if (tolower(io_type) == 'w')
  { /* Write a record of lesson block */
    sprintf(rec_str, "%s%02d", instr_blks[ndx].title,
            instr_blks[ndx].no_qa_sets);
    for (i=0; i<50; i++)
    { /* Format qa file names */
      strcat(rec_str, "\"n");
      strcat(rec_str, instr_blks[ndx].qa_sets[i].names, 10);
    } /* Format qa file names */
    strcat(rec_str, instr_blks[ndx].qa_sets[i].names, 10);
    fseek(lesson_fp, 635*ndx, SEEK_SET);
    fputs(rec_str, lesson_fp);
  } /* Write a record of lesson block */

  else if (tolower(io_type) == 'r')
  { /* Read a record of lesson block */
    fread(rec_str, 635, lesson_fp);
    strncpy(instr_blks[ndx].title, rec_str, 80);
    offset = 81;
    sscanf(&rec_str[offset], "%2d", &instr_blks[ndx].no_qa_sets);
    offset += 3;
    for (i=0; i<50; i++)
    { /* Unblock qa file names */
      strncpy(instr_blks[ndx].qa_sets[i].names, &rec_str[offset], 10);
      offset += 11;
    } /* Unblock qa file names */
  } /* Read a record of lesson block */

  else
  { /* Error in I/O specification */
    clear();
    addstr("lesson_blk_io error: I/O type specification");
    refresh();
    its_stop();
  } /* Error in I/O specification */
  return;

} /* Provide I/O for lesson block file */
void work_msg()
{
    /* Present blinking 'working...' message */
    clear();
    attrset(A_BLINK);
    cntr_ln(y_mid, "Working...");
    attrset(0);
    refresh();
    return;
} /* Present blinking 'working...' message */
void getstr_echo(char *str)
{ /* Enable character echoing with getstr function */
  echo();
  getstr(str);
  noecho();
} /* Enable character echoing with getstr function */
void strg_blnk_pad(char *strng, int str_len)
{ /* Pad a left justified string with blanks */
    int i;

    if (strlen(strng) > str_len)
    {
        cntr_ln(22, "String too long");
        cntr_ln(23, "Press any key to continue");
        getch();
        return;
    }

    if (strlen(strng) < str_len)
    { /* Pad with blanks */
        for (i=strlen(strng); i<str_len; i++)
            strng[i] = ' ';
    } /* Pad with blanks */
    return;

} /* Pad a left justified string with blanks */
NAME: INSTRUCT.C

#include <curses.h>
#include <string.h>
#include "itsdef.h"

void instructor(int *rtn_state)
{ /* Instructor */
    extern int x_mid;
    extern int y_mid;
    extern struct curricula instr_blks[];
    extern FILE *lesson_fp;
    extern char instr_path[];
    extern instr_blk open;
    extern instr_blk_cnt;
    extern instr_review;

    int pick, valid, select, reply, reply1;
    int file_state;
    int i,j,k;
    int x_pos, y_pos;

    char filestr[80];
    char any_str[81];

    /****************************************************************************
     Present Instructor's Menu
     ****************************************************************************/

clear();
cntr_ln(y_mid - 4, "Enter index of desired option:\n");
mvaddstr(y_mid - 2, x_mid - 13, "1. Add an instruction block");
mvaddstr(y_mid - 1, x_mid - 13, "2. Delete an instruction block");
mvaddstr(y_mid, x_mid - 13, "3. Modify an instruction block");
mvaddstr(y_mid + 1, x_mid - 13, "4. Review an instruction block");
mvaddstr(y_mid + 2, x_mid - 13, "5. Review a student's performance");
mvaddstr(y_mid + 3, x_mid - 13, "6. Adjust a student's instruction");
cntr_ln(y_mid + 5, "Selection:");
move(y_mid + 5, x_mid + 6);

    /****************************************************************************
     Get selection
     ****************************************************************************/

    valid = FALSE;
    select = 0;
    while (!valid)
    { /* Examine user's selection */
        pick = getch();

        if (pick == '\r' || pick == 0x157)
        { /* Possible termination of input or process */
            if (select != 0)
{ /* Terminate selection process */
    valid = TRUE;
} /* Terminate selection process */
else
{ /* Check if process terminating */
    cntr_ln(y_mid+7, "Terminate ITS [Y/N]:");
    move(y_mid+7, x_mid+12);
    pick = tolower(getch());
    if (pick == 'y')
    { /* Terminate */
        clear();
        refresh();
        *rtn_state = 0;
        return;
    } /* Terminate */
    else
    { /* Continue selection process */
        move(y_mid+5, x_mid+6);
        clrtobot();
        select = 0;
    } /* Continue selection process */
} /* Check if process terminating */

/* Possible termination of input or process */

else if (!isalpha(pick) || (pick <= 0x30 || pick > 0x36))
{ /* Invalid response */
    mvaddch(y_mid+5, x_mid+7, pick);
    cntr_ln(y_mid+7, "Invalid response");
    refresh();
    select = 0;
} /* Invalid response */

else
{ /* Valid response */
    valid = TRUE;
    select = pick;
    mvaddch(y_mid+5, x_mid+7, pick);
    *rtn_state = 1;
} /* Valid response */

/* Examine user's selection */

if ((select <= 0x34) && (!instruct_blk_open))
{ /* Get desired path */
    get_path_str("Instructor", instr_path);
    if (strlen(instr_path) == 0)
    { /* No path specified */
        *rtn_state = 0; /* Terminate Instructor */
        return;
    } /* No path specified */

    file_state = chk_file(instr_path, "instruct.blk");
    file_str(instr_path, "instruct.blk", filestr);
    if (file_state == 0)
        lesson_fp = fopen(filestr, "r+"); /* Open existing file */
else
    lesson_fp = fopen(filestr, "w+"); /* Open new file */

if (lesson_fp == NULL)
{ /* Error */
    clear();
    printf("Error opening ", filestr);
    refresh();
    its_stop();
} /* Error */

if (file_state != 0)
{ /* Initialize new file */
    work_msg();
    for (i=0; i<=49; i++)
    { /* Initialize instruction directory and file */
        lesson_blk_init(i);
        lesson_blk_io('W', i);
    } /* Initialize instruction directory and file */
    } /* Initialize new file */

} /* Get desired path */

if (!instruct_blk_open)
{ /* File already in memory */
    work_msg();
    instr_blk_cnt = 0;
    for (i=0; i<50; i++)
    { /* Find available lesson blocks */
        lesson_blk_io('R', i);
        if (instr_blks[i].no_qa_sets != 0)
            instr_blk_cnt++;
    } /* Find available lesson blocks */
    instruct_blk_open = TRUE;
} /* File already in memory */

/* Check if list of instruction blocks is desired */
clear();
if (instr_blk_cnt == 0)
{ /* No list available */
    cntr Ln(y_mid+1, "No instruction blocks exist");
    cntr Ln(y_mid+2, "Press any key to continue");
    pick = getch();
    if (select != '1') return;
} /* No list available */

/* Process chosen option */
if (select == '1')
{ /* Adding an instruction block */
    pick = lesson_list(instr_blk_cnt);
    if (instr_blk_cnt == 0)
        pick = 0;
    else
        pick = instr_blk_cnt;

138
/* Get a title */
clear();
mvaddstr(y_mid-1, 0, "Enter title (max = 80 chars):");
move(y_mid+1, 0);
getstr_echo(any_str);

if (strlen(any_str) == 0)
{ /* Return to main menu */
    *rtn_state = 1;
    return;
} /* Return to main menu */

instr_blk_cnt++;
strg_blnk_pad(any_str, 80);
strncpy(instr_blks[pick].title, any_str, 80);

/* Get list of associated lessons */
clear();
cntr_in(y_mid-1, "Any lessons in this instruction block? [Y/N]:");
reply = tolower(getch());

if (reply == 'n')
{ /* Create a dummy entry */
    strncpy(instr_blks[pick].qa_sets[instr_blks[pick].no_qa_sets].names,
            "dummy", 10);
    strg_blnk_pad(
            instr_blks[pick].qa_sets[instr_blks[pick].no_qa_sets].names,
            10);
    instr_blks[pick].no_qa_sets++;
    lesson_blk_io('W', pick);
    return;
} /* Create a dummy entry */

instr_blk_vis(pick);
valid = TRUE;
select = 0;
while (valid)
{ /* Get lesson filenames */
    make_pos(select, &y_pos, &x_pos);
    mvaddstr(23, 0, "Enter lesson name (max = 10 char):");
    move(23, 34);
    getstr_echo(any_str);
    if (strlen(any_str) == 0) continue;
    if (strlen(any_str) == 1 && (any_str[0] == 'q' || any_str[0] == 'Q'))
    { /* Exiting */
        lesson_blk_io('W', pick);
        valid = FALSE;
        break;
    } /* Exiting */
    if (strlen(any_str) > 10)
    { /* String too long */
move(23,50);
addstr("Too long");
getch();
move(23,34);
clrtoeol();
refresh();
continue;
} /* String too long */
mvaddstr(y_pos, x_pos, any_str);
refresh();
str_g_blink_pad(any_str, 10);
strncpy(instr_blks[pick].qa_sets[select].names, any_str, 10);
instr_blks[pick].no_qa_sets++;
select++;
move(23,34);
clrtoeol();
refresh();
} /* Get lesson filenames */

) /* Adding an instruction block */

else if (select == '2')
{
/* Deleting an instruction block */
pick = lesson_list(instr_blk_cnt);
instr_blk_vis(pick);
mvaddstr(23,0,"Do you want to delete lessons? [Y/N]:");
reply = tolower(getch());
if (reply == 'y')
{/*
Determine complete or selective deletion */
move(23,0);
clrtoeol();
adstr("Delete (a)ll or (s)ome of these lessons?");
reply = tolower(getch());

if (reply == 'a')
{/*
Delete all lessons */
for (i=0; i<instr_blks[pick].no_qa_sets; i++)
{ /* Get lesson file */
strncpy(any_str, instr_path);
strcat(any_str, "/");
strncat(any_str, instr_blks[pick].qa_sets[i].names, 10);
any_str[strlen(instrpath)+ll] = '\0';
remove(any_str);
instr_blks[pick].qa_sets[i].names[0] = '\0';
str_g_blink_pad(instr_blks[pick].qa_sets[i].names, 10);
} /* Get lesson file */
} /* Delete all lessons */

if (reply == 's' || reply == 'S')
{/*
Delete selected lessons */
valid = TRUE;
while (valid)
{ /* Delete selection */
move(23,0);

} /* Deleting an instruction block */
clrtoeol();
mvaddstr(23, 0, "Enter index of lesson to be deleted: ");
getstr_echo(any_str);
move(23, 37);
clrtoeol();
refresh();
if (any_str[0] == 'q' || any_str[0] == 'Q')
{ /* Exiting */
    valid = FALSE;
    continue;
}
scanf(any_str, "%d", &reply);
strcpy(filestr, instr_path);
strcat(filestr, "/");
strncat(filestr, instr_blks[pick].qa_sets[reply-1].names,
10);
filestr[strlen(instr_path)+11] = '\0';
remove(filestr);
for (i=reply-1; i<=instr_blks[pick].no_qa_sets-2; i++)
{ /* Compress lesson list */
    strncpy(instr_blks[pick].qa_sets[i].names,
            instr_blks[pick].qa_sets[i+1].names, 10);
} /* Compress lesson list */
instr_blks[pick].qa_sets[instr_blks[pick].no_qa_sets-1].names[0] = '\0';
strg_blk_pad(instr_blks[pick].
            qa_sets[instr_blks[pick].no_qa_sets-1].names, 10);
instr_blk_cnt--;
instr_blk_vis(pick);
} /* Delete selection */
} /* Determine complete or selective deletion */
for (i=pick; i<instr_blk_cnt-1; i++)
{ /* Delete selected instruction block entry */
    strncpy(instr_blks[i].title, instr_blks[i+1].title, 80);
instr_blks[i].no_qa_sets = instr_blks[i+1].no_qa_sets;
    for (j=0; j<50; j++)
        strncpy(instr_blks[i].qa_sets[j].names,
                instr_blks[i+1].qa_sets[j].names, 10);
} /* Delete selected instruction block entry */
lesson_blk_rec_init(instr_blk_cnt-1);
for (i=0; i<instr_blk_cnt; i++)
    lesson_blk_io('W', i);
instr_blk_cnt--;
} /* Deleting an instruction block */

else if (select == '3')
{ /* Modifying an instruction block */
    pick = lesson_list(instr_blk_cnt);
    valid = TRUE;
    while (valid)
    { /* Modify selected fields */
        instr_blk_vis(pick);
        }
mvaddstr(23, 0, "Modifying (t)itle or (l)esson:");
reply = tolower(getcho());

if (reply == 'q')
{ /* Modification completed */
    valid = FALSE;
} /* Modification completed */

else if (reply == 't' || reply == 'T')
{ /* Modifying title */
    mvaddstr(22, 0, "Enter title:");
clrtobot();
move(23, 0);
getstr_echo(any_str);
str_blnk_pad(any_str, 80);
strncpy(instr_blks[pick].title, any_str, 80);
lesson_blk_io('W', pick);
} /* Modifying title */

else if (reply == 'l' || reply == 'L')
{ /* Modifying a lesson name */
    move(22, 0);
clrtobot();
mvaddstr(22, 0, "Enter index of lesson to be changed: ");
getstr_echo(any_str);
sscanf(any_str, "%d", &reply);
reply--;
mvaddstr(23, 0, "Change: (d)elete, (m)odify, (i)nsert:");
reply1 = tolower(getcho());

    if (reply1 == 'd')
    { /* Delete lesson from list */
        for (i=reply; i<instr_blks[pick].no_qa_sets-1; i++)
            strncpy(instr_blks[pick].qa_sets[i].names,
                instr_blks[pick].qa_sets[i+1].names, 10);
instr_blks[pick].qa_sets[instr_blks[pick].no_qa_sets-1].
                names[0] = '0';
str_blnk_pad(instr_blks[pick].
                qa_sets[instr_blks[pick].no_qa_sets-1].names, 10);
instr_blks[pick].no_qa_sets--;
lesson_blk_io('W', pick);
} /* Delete lesson from list */

else if (reply1 == 'm' || reply1 == 'M')
{ /* Modify lesson name */
    move(23, 0);
clrtoeol();
addstr("Enter lesson name: ");
getstr_echo(any_str);
str_blnk_pad(any_str, 10);
strncpy(instr_blks[pick].qa_sets[reply].names, any_str, 10);
lesson_blk_io('W', pick);
} /* Modify lesson name */
else if (reply1 == 'i' || reply1 == 'I')
{ /* Insert a lesson name */
    for (i=instr_blks[pick].no_qa_sets-1; i>=reply; i--)
        strncpy(instr_blks[pick].qa_sets[i+1].names,
                instr_blks[pick].qa_sets[i].names, 10);
    move(23,0);
    clrtoeol();
    addstr("Enter lesson name: ");
    getstr_echo(anystr);
    strcncpy(instr_blks[pick].qa_sets[reply].names, anystr, 10);
    instr_blks[pick].noqasets++;
    lesson_blk_io('W', pick);
} /* Insert a lesson name */

else
{ /* No change */
    continue;
} /* No change */

} /* Modifying a lesson name */

else
{ /* No change */
    continue;
} /* No change */

} /* Modify selected fields */

} /* Modifying an instruction block */

else if (select == '4')
{ /* Review an instuction block */
    instr_review = lesson_list(instr_blk_cnt);
    *rtn_state  = 1;
    return;
} /* Review an instuction block */

else
{ /* Option not yet available */
    cntr_ln(y_mid, "Option is not available");
    refresh();
    *rtn_state  = 1;
} /* Option not yet available */

return;

} /* Instructor */
NAME: INSTR_RU.C

#include <curses.h>
#include <string.h>
#include "itsdef.h"
void lesson_blk_rec_init(int ndx)
{ /* Initialize a record in lesson block file */
    int i,j;
    for (i=0; i<80; i++) /* Blank fill title area */
        instr_blks[ndx].title[i] = ' ';
    instr_blks[ndx].no_qa_sets = 0; /* Zero number of Q&A sets */
    for (i=0; i<50; i++)
    { /* Blank fill each Q&A set name */
        for (j=0; j<10; j++)
            instr_blks[ndx].qa_sets[i].names[j] = ' ';
    }
    return;
} /* Initialize a record in lesson block file */
int lesson_list(max_cnt)
{ /* Display lesson block titles */

/* Routine assumes that the area consisting of lines 2-22 is clear and will
return the values of:
  (value) = entry number of selection
-1 = display terminated without selection */

extern int x_mid;
extern int instr_blk_cnt;

int loop_cnt1;
int beg_ent, end_ent;
int dsp_ln, cur_in;
int cur_ent;
int i;
int reply;

char str_buf[81];

clear();
mvprintw(0, x_mid-21, "The following %d instruction blocks exist:",
  instr_blk_cnt);
mvprintw(23, 0,
  "Use cursor to position, 'Enter' to select, or 'Q' to quit");

loop_cnt1 = TRUE;
beg_ent = 0;
if (max_cnt < 9)
  end_ent = max_cnt-1;
else
  end_ent = 9;
cur_in = 2;

while (loop_cnt1)
{ /* Prepare and display list */
  dsp_ln = 2;
  for (i=beg_ent; i<=end_ent; i++)
    { /* Present list */
      strncpy(str_buf, instr_blks[i].title, 80);
      str_buf[80] = '\0';
      if (cur_in == dsp_ln) attrset(A_REVERSE);
      mvaddstr(dsp_ln, 0, str_buf);
      if (cur_in != dsp_ln) attrset(0);
      dsp_ln++;
    } /* Present list */
  move(23,60);

  reply = getch();
cur_ent = beg_ent + cur_in - 2;
  switch (reply)
  { /* Examine user's input */
case KEY_UP:
    if (cur_ent == 0) break;
    cur_ln--;     
    cur_ent--;     
    if (cur_ent < beg_ent)
    { /* Scroll display down */
        cur_ln++; 
        beg_ent--; 
        end_ent--; 
    } /* Scroll display down */
    break;

case KEY_DOWN:
    if (cur_ent == (max_cnt-1)) break;
    cur_ln++;
    cur_ent++; 
    if (cur_ent > end_ent)
    { /* Scroll display up */
        cur_ln--; 
        beg_ent++; 
        end_ent++; 
    } /* Scroll display up */
    break;

case '\r':
    case 0x157:
        loop_cnt1 = FALSE;
        break;

    case 'q':
    case 'Q':
        loop_cnt1 = FALSE;
        cur_ent = -1;
        break;

    default:
        break;
    } /* Examine user's input */
} /* Prepare and display list */

    return cur_ent;

} /* Display lesson block titles */
void lesson_blk_chk()
{ /* Display a selected record from lesson.blk */
  int view;
  int ent_no;
  int i,j;
  int reply;

  char txt_str[81];

  struct
  {
    char names[11];
  } lessons[5];

  view = TRUE;
  while (view)
  { /* View an instruction block record */
    clear();
    addstr("Enter index of entry to be viewed: ");
    echo();
    getstr(txt_str);
    noecho();
    sscanf(txt_str, "%d", &ent_no);
    if (ent_no == -2)
    { /* Exit */
      its_stop();
    } /* Exit */
    if (ent_no < 0)
    { /* End viewing of records */
      view = FALSE;
      break;
    } /* End viewing of records */
  }

  clear();
  strncpy(txt_str, instr_blks[ent_no].title, 80);
  txt_str[80] = '\0';
  printf("Title: \n%s\n", txt_str);
  printf("Q&A set count = %2d\n", instr_blks[ent_no].no_qa_sets);

  for (i=0; i<10; i++)
  { /* Display lesson files */
    for (j=0; j<5; j++)
    {
      strncpy(lessons[j].names,
              instr_blks[ent_no].qa_sets[i+(j*10)].names,10);
      lessons[j].names[10] = '\0';
    }
    printf("%2d)|$%2d)|%s %2d)%s %2d)%s %2d)%s %2d)%s
", i,lessons[0].names,
            i+10,lessons[1].names, i+20,lessons[2].names,
            i+30,lessons[3].names, i+40,lessons[4].names);
  } /* Display lesson files */
  mvprintw(23,0,"Press any key to continue");
  reply = getch();

  return;
}
} /* View an instruction block record */
return;

} /* Display a selected record from lesson.blk */
void instr_blk_vis(int ent_no)
{ /* Display entry from 'instruct.blk' */
 extern struct curricula instr_blks[];

 int i, j;
 char txt_str[81];
 struct /* Lesson filename strings */
 { 
   char names[11];
 } lesson[5];

 clear();
 strncpy(txt_str, instr_blks[ent_no].title, 80);
 txt_str[80] = '\0';
 printf("Title:\n%s\n", txt_str);

 for (i=0; i<10; i++)
 { /* Display lesson files */
   for (j=0; j<5; j++)
   { /* Make filename strings */
     strncpy(lesson[j].names,
           instr_blks[ent_no].qa_sets[i+(j*10)].names, 10);
     lesson[j].names[10] = '\0';
   } /* Make filename strings */
   printf("%2d)%s %2d)%s %2d)%s %2d)%s %2d)%s \n", i+1,lesson[0].names,
          i+11,lesson[1].names, i+21,lesson[2].names, i+31,lesson[3].names,
          i+41,lesson[4].names);
 } /* Display lesson files */
 refresh();
 return;
}

 /* Display entry from 'instruct.blk' */
void make_pos(int ndx, int *y_pos, int *x_pos)
{ /* Convert ndx to screen coordinates */
  int quo, rem;

  quo = ndx/10;
  rem = ndx - (quo*10);
  *y_pos = (rem*2) + 3;
  *x_pos = (quo*14) + 3;
  return;
}
} /* Convert ndx to screen coordinates */
NAME: STUDENT.C

/**************************************************************************************
For the following, the numerous switches are presented in numerical order, but the use of pagination is done to infer a logical order. This logical ordering is as follows:

a.  0-79: Function select switches

b.  80-87: Feature & Vector switches
   88-105: Category Select switches
   142-159: Category Select switches
           for WD Console
           Checkout

c.  106-141: Optional Category Select switches

d.  160-184: Display Panel switches
***************************************************************************************/

#include <curses.h>
#include <string.h>
#include "itsdef.h"

void student(int *stdt_status)
{ /* Student */
  extern void get_max_min();
  extern int user_type;
  extern char instr_path[];
  extern char stdt_path[];
  extern FILE *lesson_fp;
  extern instr_blk_cnt;
  extern int y_mid;
  extern int x_mid;
  extern struct stdt_db pupil;
  extern FILE *present_fp;
  extern struct lesson_file_entry lesson_map[];
  extern int stdt_acc_score;

  int i;
  int file_status;
  int valid, next_lesson;
  int reply;
  int map_cnt;
  int mapndx;
  int scan_it;
  int prev_text;

  long int file_offset;

  char name_str[27], ssan_str[10];
  char any_str[81];
  char any_strl[81];
  char *srch_char;
Error opening file

for (i=0; i<50; i++)
{
  /* Load 'instruct.blk' into memory */
  lesson_blk_io('R', i);
  if (instr_blks[i].no_qa_sets != 0)
    instr_blk_cnt++;
}

if (instr_blk_open = TRUE;

get_path_str("Student", stdt_path);
if (strlen(stdt_path) == 0)
  return;

if (user_type == Student)
{
  valid = FALSE;
  while (!valid)
  {
    /* Validate */
    student_ident(name_str, ssan_str);
    str_blink_pad(name_str, 26);
    strcpy(any_str, "sdb ");
    strcat(any_str, &ssan_str[5], 4);
    file_status = chk_file(stdt_path, any_str);
    file_str(stdt_path, any_str, any_str1);"}

  /* Possible New Student */
  clear();
  cntr_ln(y.midd-1, "Are you a new student? [Y/N]:");
  reply = tolower(getch());
  if (reply == 'y')
  {
    /* Open new student file */
    valid = TRUE;
    student_fp = fopen(any_str1, "w+");
    /* Initialize student db file */
    strcpy(pupil.name, name_str);
    strcpy(pupil.ssan, ssan_str);
    pupil.instr_blk_ndx = -1;
    pupil.lesson_ndx = -1;
    pupil.text_blk_ndx = 1;
    pupil.know_phase_ndx = 1;
    student_blk_io('W', 0);
    for (i=0; i<5; i++)
    {
      /* Initialize knowledge phase data */
      pupil.know_phase[i].max_val = 0;
      pupil.know_phase[i].min_val = 0;
      pupil.know_phase[i].act_val = 0;
      pupil.know_phase[i].spare_val = 0;
      student_blk_io('W', i+1);
    }
    /* Initialize knowledge phase data */
  }
} /* Open new student file */

153
else
    { /* Error */
        clear();
        cntr_ln(y_mid,"Please validate name and SSAN again");
        cntr_ln(y_mid+1, "Enter any key to continue");
        getch();
    } /* Error */
} /* Possible New Student */
else
    { /* Existing Student */
        student_fp = fopen(any_strl, "r+");
        student_blk_io('R', 0); /* Read header */
        for (i=1; i<=5; i++)
            student_blk_io('R', i); /* Read knowledge phase data */
        if (strncmp(pupil.name, namestr, 26) != 0 ||
            strncmp(pupil.ssan, ssan_str, 9) != 0)
            { /* Name or SSAN mismatch */
                clear();
                cntr_ln(y_mid, "Name or SSAN mismatch with original registration");
                cntr_ln(y_mid+1, "Please validate name and SSAN again");
                cntr_ln(y_mid+2, "Press any key to continue");
                getch();
            } /* Name or SSAN mismatch */
        else
            valid = TRUE;
    } /* Existing Student */
} /* Validate */
} /* Check student's registration */

if (user_type == Instructor)
    { /* Reviewing an instruction block */
        pupil.instr_blk_ndx = instr_review;
        pupil.lessonndx = 0;
    } /* Reviewing an instruction block */
valid = TRUE;
present_fp = NULL;
stdt_acc_score = 0;
file_offset = 0;
prev_text = pupil.text_blk_ndx;
while (valid)
    { /* Presenting a lesson */
        if (present_fp == NULL)
            { /* Open lesson file */
                if (pupil.instr_blk_ndx == -1)
                    strcpy(any_str, "welcome"); /* New Student */
                else
                    { /* Get next lesson */
                        strncpy(any_str,
                            instr_blks[pupil.instr_blk_ndx].qa_sets[pupil.lessonndx].names,
                                10);
                        any_str[10] = '\0';
                    } /* Get next lesson */
            } /* Open lesson file */
        else
            { /* Presenting a lesson */
                if (present_fp == NULL)
                    { /* Open lesson file */
                        if (pupil.instr_blk_ndx == -1)
                            strcpy(any_str, "welcome"); /* New Student */
                        else
                            { /* Get next lesson */
                                strncpy(any_str,
                                    instr_blks[pupil.instr_blk_ndx].qa_sets[pupil.lessonndx].names,
                                        10);
                                any_str[10] = '\0';
                            } /* Get next lesson */
                    } /* Open lesson file */
            } /* Presenting a lesson */
    } /* Presenting a lesson */
/* Get next lesson */
file_status = chk_file(instr_path, any_str);
if (file_status == 1)
    {/* File doesn't exist */
        valid = FALSE;
        continue;
    } /* File doesn't exist */
file_str(instr_path, any_str, any_str1);
present_fp = fopen(any_str1, "r");

/* Build map of lesson file */
file_offset = 0;
map_cnt = 0;
next_lesson = TRUE;
fseek(present_fp, file_offset, SEEK_SET);
while (fgets(any_str, 81, present_fp) != NULL)
    {/* Build map */
        if (next_lesson)
        { /* Build map entry */
            lesson_map[map_cnt].file_pos = file_offset;
            file_offset += strlen(any_str);
            sscanf(any_str, "%d", &lesson_map[map_cnt].num_id);
            srch_char = strchr(any_str, 'l');
            if (strcmp(&any_str[srch_char-1], "text", 4) == 0)
                lesson_map[map_cnt].text_type = 1;
            else if (strcmp(&any_str[srch_char-1], "multiple choice", 15) == 0)
                lesson_map[map_cnt].text_type = 2;
            else if (strcmp(&any_str[srch_char-1], "true/false", 10) == 0)
                lesson_map[map_cnt].text_type = 3;
            else if (strcmp(&any_str[srch_char-1], "yes/no", 6) == 0)
                lesson_map[map_cnt].text_type = 4;
            else if (strcmp(&any_str[srch_char-1], "score", 5) == 0)
                lesson_map[map_cnt].text_type = 5;
            else if (strcmp(&any_str[srch_char-1], "instruct/lesson", 15) == 0)
                lesson_map[map_cnt].text_type = 6;
            else if (strcmp(&any_str[srch_char-1], "noscore", 7) == 0)
                lesson_map[map_cnt].text_type = 7;
            else
            { /* Problem with file format */
                clear();
                any_str[strlen(any_str)-1] = '\0';
                mvprintw(y_mid, 0, "%s is not a valid text type",
                    &any_str[srch_char-1]);
                refresh();
                return;
            } /* Problem with file format */
        } /* Build map */
    } /* Build map */
/* Get next identifiers of follow-on texts */
fsseek(present_fp, file_offset, SEEK_SET);
fgets(any_str, 81, present_fp);
file_offset += strlen(any_str);
src_char = any_str;
for (i=0; i<5; i++)
{ /* Get each follow-on */
    sscanf(&any_str[src_char-any_str], "%d",
            &lesson_map[map_cnt].know_level[i]);
    if (i < 4)
        /* Search for blank and advance */
        src_char = strchr(&any_str[src_char-any_str], ' ');
        src_char++;
    } /* Search for blank and advance */
} /* Get each follow-on */

/* Indicate no previous display */
lesson_map[map_cnt].prev_num_id = -1;

/* Blank out student response */
lesson_map[map_cnt].stdt_ans = ' ';

/* Get answers and values for: */
/* 2) multiple choice */
/* 3) true/false */
/* 4) yes/no */
/* 5) scoring function */
/* 6) instrct/lesson branch */
/* 7) No scoring function */

if (lesson_map[map_cnt].text_type >= 2 &&
    lesson_map[map_cnt].text_type <= 7)
{ /* Other text types */
    fsseek(present_fp, file_offset, SEEK_SET);
    fgets(any_str, 81, present_fp);
    file_offset += strlen(any_str);
    src_char = any_str;
    map_ndx = 0;
    scan_it = TRUE;
    while (scan_it)
    { /* Get answers and associated values */
        src_char = strchr(&any_str[src_char-any_str], ')');
        if (src_char != NULL)
            /* Store possible answer and its value */
            lesson_map[map_cnt].mc_ans[map_ndx].ans_desig =
            any_str[src_char-any_str-1]; /* Answer */
            sscanf(&any_str[src_char-any_str+1], "%d",
                   &lesson_map[map_cnt].mc_ans[map_ndx].ans_val);
            src_char++;
            map_ndx++;
    } /* Store possible answer and its value */
    else
        scan_it = FALSE;
} /* Get answers and associated values */
```c
lesson_map[map_cnt].mc_ans_cnt = map_ndx;
} /* Other text types */

next_lesson = FALSE;
map_cnt++;
} /* Build map entry */

else
{ /* Skip this line */
    file_offset += strlen(any_str);
    if (any_str[0] == '\f')
        next_lesson = TRUE;
} /* Skip this line */

fseek(present_fp, file_offset, SEEK_SET);
} /* Build map */
} /* Open lesson file */

if (pupil.text_blk_ndx == -1)
{ /* End of current lesson */
    clear();
    cntr_ln(y_mid, "End of lesson");
    getch();
    fclose(present_fp);
    student_blk_io('W', 0);
    for (i=0; i<5; i++)
        student_blk_io('W', i+1);
    valid = FALSE;
    *stdt_status = 1;
    break;
} /* End of current lesson */

file_offset = -1;
for (i=0; i<map_cnt; i++)
{ /* Search for selected text */
    if (pupil.text_blk_ndx == lesson_map[i].num_id)
    { /* Match found */
        file_offset = lesson_map[i].file_pos;
        if (lesson_map[i].prev_num_id == -1)
            lesson_map[i].prev_num_id = prev_text;
        map_ndx = i;
        break;
    } /* Match found */
} /* Search for selected text */

if (file_offset < 0)
{ /* Text does not exist */
    clear();
    mvprintw(y_mid, x_mid-23, "Text %3d does not exist for block %2d, lesson %2d",
             pupil.text_blk_ndx, pupil.instr_blk_ndx, pupil.lesson_ndx);
    refresh();
    its_stop();
} /* Text does not exist */
```
fseek(present_fp, lesson_map[mapndx].file_pos, SEEK_SET);
switch (lesson_map[mapndx].text_type)
{ /* Display text types */
case 1:
case 2:
case 3:
case 4:
    reply = prsnt_txt(mapndx);
    break;

case 5:
case 7:
    reply = score_txt(mapndx);
    break;

case 6:
    reply = lesson_txt(mapndx);
    break;

default:
    clear();
    cntr_ln(y_mid, "This is impossible and should not have happened");
    refresh();
    its_stop();
} /* Display text types */

if (reply == 1)
    valid = FALSE; /* Terminate lesson presentation */
else if (reply == 2)
{ /* Get next text block */
    if (lesson_map[mapndx].text_type <= 4)
        prev_text = pupil.text_blkndx;
    if (lesson_map[mapndx].text_type != 7)
        pupil.text_blkndx = lesson_map[mapndx].know_level[pupil.know_phase_ndx-1];
} /* Get next text block */
else if (reply == 3)
{ /* Get previous text */
    if (pupil.text_blkndx != lesson_map[mapndx].prev_num_id)
        pupil.text_blkndx = lesson_map[mapndx].prev_num_id;
    for (i=0; i<map_cnt; i++)
    { /* Get index */
        if (pupil.text_blkndx == lesson_map[i].num_id)
        { /* Match */
            map_ndx = i;
            break;
        } /* Match */
    } /* Get index */
    if (lesson_map[mapndx].text_type != 1)
        get_max_min(mapndx, '-');
} /* Get previous text */
else if (reply == 4)
{ /* Get next instruction block/lesson */
else
  {  /* Oops ! */
      clear();
      cntr_ln(y_mid, "This shouldn't have happened either");
      refresh();
      its_stop();
  }  /* Oops ! */

}  /* Presenting a lesson */

  return;
}  /* Student */
NAME: STDTRUT.C

#include <curses.h>
#include <string.h>
#include <ctype.h>

#include "itsdef.h"
void student_ident(char *name_str, char *id_str)
{
    /* Get student's name and identifier */
    extern int x_mid;
    extern int y_mid;
    extern void get_ssan();

    int valid, ack;

    clear();
    cntr_ln(y_mid-2,"Enter name and SSAN:
    mvaddstr(y_mid, x_mid-11, "Name:");

    valid = FALSE;
    while (!valid)
    {
        /* Get name */
        movu(y_mid, x_mid-5);
        getstr_echo(name_str);
        cntr_ln(y_mid+2, "Is name spelled correctly? [Y/N]");
        ack = tolower(getch());
        if (ack == 'y')
            valid = TRUE; /* Valid Name */
        move(y_mid+2, 0);
        clrtoeol();
    } /* Get name */

    mvaddstr(y_mid+1, x_mid-11, "SSAN:");
    valid = FALSE;
    while (!valid)
    {
        /* Get Id */
        mvaddstr(y_mid+1, x_mid-5, "xxx-xx-xxxx");
        move(y_mid+1, x_mid-5);
        refresh();
        get_ssan(id_str);
        cntr_ln(y_mid+3, "Is SSAN correct? [Y/N]");
        ack = tolower(getch());
        if (ack == 'y')
            valid = TRUE;
        move(y_mid+3, 0);
        clrtoeol();
    } /* Get Id */

    return;
}
/* Get student's name and identifier */
void get_ssan(char *a_strng)
{
    /* Get a SSAN string */

    int pos;
    int not_digit;
    int x_coord, y_coord;

    getyx(stdscr, y_coord, x_coord);
    not_digit = FALSE;
    pos = 0;
    while (pos < 9)
    {
        /* Get digits */
        *(a_strng) = getch();
        if (not_digit)
        {
            /* Clear error msg */
            move(y_coord+2, 0);
            clrtoeol();
            move(y_coord, x_coord);
            refresh();
            not_digit = FALSE;
        }
        /* Clear error msg */
        if (!isdigit(*(a_strng)))
        {
            /* Must be a digit */
            cntr_ln(y_coord+2, "Must be a digit");
            move(y_coord, x_coord);
            refresh();
            not_digit = TRUE;
        }
        /* Must be a digit */
        else
        {
            /* Next digit */
            addch(*a_strng);
            a_strng++;
            pos++;
            if (pos == 3 || pos == 5)
            {
                x_coord++;
                x_coord++;
                move(y_coord, x_coord);
            }
            /* Next digit */
        }
        /* Get digits */
    }
    noecho();
    return;
}
/* Get a SSAN string */
void student_blk_io(char io_type, int rec_ndx)
{ /* Provide I/O for student db file */
extern FILE *student_fp;
extern struct stdt_db pupil;

char stdt_rec[81];

if (tolower(io_type) == 'w')
{ /* Write a record to student db */
  if (rec_ndx == 0)
  { /* Header record */
    fseek(student_fp, 0, SEEK_SET);
    sprintf(stdt_rec, "%9.9s-%26.26s-%02d-%02d-%02d-%02d\n",
            pupil.ssan, pupil.name, pupil.instr_blk_ndx, pupil.lesson_ndx,
            pupil.text_blk_ndx, pupil.know_phase_ndx);
    fputs(stdt_rec, student_fp);
  } /* Header record */
  else
  { /* Knowledge phase record */
    fseek(student_fp, ((rec_ndx-1)*20)+49, SEEK_SET);
    sprintf(stdt_rec, "%04d%04d%04d%04d\n",
            pupil.know_phase[rec_ndx-1].max_val,
            pupil.know_phase[rec_ndx-1].min_val,
            pupil.know_phase[rec_ndx-1].act_val,
            pupil.know_phase[rec_ndx-1].spare_val);
    fputs(stdt_rec, student_fp);
  } /* Knowledge phase record */
} /* Write a record to student db */

else if (tolower(io_type) == 'r')
{ /* Read a record from student db */
  if (rec_ndx == 0)
  { /* Header record */
    fseek(student_fp, 0, SEEK_SET);
    fgets(stdt_rec, 49, student_fp);
    strncpy(pupil.ssan, stdt_rec, 9);
    strncpy(pupil.name, &stdt_rec[10], 26);
    sscanf(&stdt_rec[37], "%2d", &pupil.instr_blk_ndx);
    sscanf(&stdt_rec[40], "%2d", &pupil.lesson_ndx);
    sscanf(&stdt_rec[43], "%2d", &pupil.text_blk_ndx);
    sscanf(&stdt_rec[46], "%2d", &pupil.know_phase_ndx);
  } /* Header record */
  else
  { /* Knowledge phase record */
    fseek(student_fp, ((rec_ndx-1)*20)+49, SEEK_SET);
    fgets(stdt_rec, 20, student_fp);
    sscanf(stdt_rec, "%4d", &pupil.know_phase[rec_ndx-1].max_val);
    sscanf(stdt_rec[4], "%4d", &pupil.know_phase[rec_ndx-1].min_val);
    sscanf(&stdt_rec[4], "%4d", &pupil.know_phase[rec_ndx-1].act_val);
    sscanf(&stdt_rec[12], "%4d",
           &pupil.know_phase[rec_ndx-1].spare_val);
  } /* Knowledge phase record */
} /* Read a record from student db */
else
{ /* Error in I/O specification */
   clear();
   addstr("student_blk_io error: I/O type specification");
   refresh();
   its_stop();
} /* Error in I/O specification */
return;

} /* Provide I/O for student db file */
void read_prsnt_txt(int map, int *file_offset, int *disp_map, int *disp_ndx, char *disp_str)
{
    /* Read and map a line of text */
    extern FILE *present_fp;

    fseek(present_fp, *file_offset, SEEK_SET);
    fgets(disp_str, 81, present_fp);
    if (map == -1)
    { /* Map the text block */
        disp_map[*disp_ndx] = *file_offset;
        (*disp_ndx)++;
    } /* Map the text block */
    *file_offset += strlen(disp_str);
    return;

} /* Read and map a line of text */
int prsnt_txt(int map_ndx)
{
    /* Present text to user */
    extern struct lesson_file_entry lesson_map[];
    extern FILE *present_fp;
    extern int stdt_acc_score;
    extern void get_max_min();
    extern void reply_pos();
    extern int firstques;

    int dispndx;
    int line_cnt;
    int i;
    int beg_ln, end_ln;
    int display_it;
    int reply;
    int paint_it;
    int match_found;
    int key_state;

    long int file_offset;
    long int disp_map[100];

    char disp_str[81];

    /* Skip the first and second line */
    dispndx = 0;
    file_offset = lesson_map[map_ndx].file_pos;
    read_prsnt_txt(0, &file_offset, disp_map, &dispndx, disp_str);
    read_prsnt_txt(0, &file_offset, disp_map, &dispndx, disp_str);

    /* Skip next line if multiple choice, true/false, or scoring */
    if (lesson_map[map_ndx].text_type != 1)
        read_prsnt_txt(0, &file_offset, disp_map, &dispndx, disp_str);

    /* Get line count */
    read_prsnt_txt(0, &file_offset, disp_map, &dispndx, disp_str);
    sscanf(disp_str, "%d", &line_cnt);

    /* Create line display map */
    for (i=0; i<line_cnt; i++)
        read_prsnt_txt(-1, &file_offset, disp_map, &dispndx, disp_str);

    /* Present first page */
    beg_ln = 0;
    if (line_cnt <= LINES-2)
        end_ln = line_cnt - 1;
    else
        end_ln = LINES - 2;

    if (lesson_map[map_ndx].text_type > 1 && firstques == -1)
        firstques = map_ndx;

    if (lesson_map[map_ndx].text_type > 1 && lesson_map[map_ndx].stdt_ans !=
' */
{
  /* Backout existing value */
  for (i=0; i<lesson_map[map_ndx].mc_ans_cnt; i++)
  {
    /* Search for match */
    if (lesson_map[map_ndx].stdt_ans ==
        lesson_map[map_ndx].mc_ans[i].ans_desig)
    {
      /* Matching answer */
      if ((stdt_acc_score - lesson_map[map_ndx].mc_ans[i].ans_val) >= 0)
      {
        /* Adjust score */
        stdt_acc_score -=
          lesson_map[map_ndx].mc_ans[i].ans_val;
        if (stdt_acc_score == 0 && first ques != map_ndx)
          stdt_acc_score +=
            lesson_map[map_ndx].mc_ans[i].ans_val;
        break;
      }
      /* Adjust score */
      /* Matching answer */
    }
    /* Search for match */
  }
  /* Backout existing value */

  if (lesson_map[map_ndx].text_type != 1)
    get_max_min(map_ndx, '+');

  paint_it = TRUE;
  display_it = TRUE;
  while (display_it)
  {
    /* Display a text block */
    if (paint_it)
    {
      /* Paint screen */
      clear();
      file_offset = disp_map[beg_ln];
      for (i=beg_ln; i<=end_ln; i++)
      {
        /* Display text */
        read_prsnt_txt(i, &file_offset, disp_map, &disp_ndx, disp_str);
        mvprintw(i-beg_ln, 0, "%'s", disp_str);
      }
      /* Display text */
      if (end_ln < line_cnt-1)
      {
        /* Display "more" message */
        attrset(A_BLINK);
        cntr_ln(LINES-1, "More");
        attrset(0);
      }
      /* Display "more" message */
      else
        /* Clear "more" message and prompt for answer, if necessary */
        move(LINES-1, 0);
        clrtoeol();
        if (lesson_map[map_ndx].text_type != 1)
        {
          /* Possible response */
          if (lesson_map[map_ndx].text_type == 2)
            mvaddstr(LINES-1, 0, "Selection: "); /* Multiple Choice */
          else if (lesson_map[map_ndx].text_type == 3)
            mvaddstr(LINES-1, 0, "(t)rue/(f)alse: "); /* True/False */
        }
      }
    }
  }
}
else
    mvaddstr(LINES-1, 0, "(y)es/(n)o: "); /* Yes/No */
    if (lesson_map[map_ndx].stdt_ans != ' ' &&
        lesson_map[map_ndx].stdt_ans != 'x')
    { /* Display previous answer */
        addch(lesson_map[map_ndx].stdt_ans);
        reply_pos(map_ndx);
    } /* Display previous answer */
    /* Clear "more" message and prompt for answer, if necessary */
paint_it = FALSE;
/* Paint screen */
}
reply = tolower(getch());
switch (reply)
/* Get user's reply */
case KEY_UP:
    if (end_ln < line_cnt-1)
    { /* Move up one line */
        beg_ln++;
        if (beg_ln+LINES-2 <= line_cnt)
            end_ln = beg_ln + LINES - 2;
        else
            end_ln = line_cnt - 1;
        paint_it = TRUE;
    } /* Move up one line */
basic

    case KEY_DOWN:
    case 0x43:
        if (beg_ln > 0)
        { /* Move down one line */
            beg_ln--;
            if (beg_ln+LINES < line_cnt)
                end_ln = beg_ln+ LINES - 2;
            else
                end_ln = line_cnt -1;
            paint_it = TRUE;
        } /* Move down one line */
basic

    case 'q': /* Quit */
        key_state = 1;
        display_it = FALSE;
        break;

    case 0x02: /* Ctrl-B: Page backward */
        if (beg_ln > 0)
        { /* Paging backwards */
            if (beg_ln-LINES+2 < 0)
                beg_ln = 0;
            else
                beg_ln = beg_ln - LINES + 2;
if (beg_ln+LINES-2 > line_cnt)
    end_ln = line_cnt - 1;
else
    end_ln = beg_ln + LINES - 2;
paint_it = TRUE;
} /* Paging backwards */
break;

case 0x06: /* Ctrl-F: Page forward */
    if (end_ln < line_cnt - 1)
    { /* Paging forward */
        if (end_ln+LINES-2 > line_cnt)
            end_ln = line_cnt - 1;
        else
            end_ln = end_ln + LINES - 2;
        if (end_ln-LINES+2 < 0)
            beg_ln = 0;
        else
            beg_ln = end_ln - LINES + 2;
paint_it = TRUE;
    } /* Paging forward */
break;

case 0x0e: /* Ctrl-N: Next text */
case 'r': /* Carriage return from VT100 */
case 0x157: /* Carriage return from SG */
    if (lesson_map[map_ndx].text_type == 1)
        reply == 'r' || reply == 0x157)
    (end_ln < line_cnt-1)
break;
    if (lesson_map[map_ndx].text_type > 1)
    { /* Score result */
        for (i=0; i<lesson_map[map_ndx].mc_ans_cnt; i++)
        { /* Search for match */
            if (lesson_map[map_ndx].stdt_ans ==
                lesson_map[map_ndx].mc_ans[i].ans_desig)
            { /* Accumulate value */
                stdt_acc_score +=
                    lesson_map[map_ndx].mc_ans[i].ans_val;
                break;
            } /* Accumulate value */
        } /* Search for match */
    } /* Score result */

key_state = 2;
display_it = FALSE;
break;

case 0x10: /* Ctrl-P: Previous text */
    if (lesson_map[map_ndx].text_type != 1)
        get_max_min(map_ndx, '-');
    key_state = 3;
display_it = FALSE;
break;
default:
    if ((lesson_map[map_ndx].text_type == 1) || (end_ln < line_cnt-1))
        break;
    reply_pos(map_ndx);
    clrtoeol();
    addch(reply);
    reply_pos(map_ndx);
    match_found = FALSE;
    for (i=0; i<lesson_map[map_ndx].mc_ans_cnt; i++)
    { /* Determine if valid response */
        if (lesson_map[map_ndx].mc_ans[i].ans_desig == reply)
        { /* Match found */
            match_found = TRUE;
            break;
        } /* Match found */
    } /* Determine if valid response */
    if (! match_found)
    { /* In valid response */
        printf("%c - invalid response", reply);
        reply_pos(map_ndx);
    } /* In valid response */
    else
    { /* Valid response */
        lesson_map[map_ndx].stdt_ans = reply;
    } /* Valid response */
    break;
} /* Get user's reply */
} /* Display a text block */

return key_state;
} /* Present text to user */
int score_txt(int mapndx)
{ /* Assess student's score */
    extern struct lesson_file_entry lesson_map[];
    extern int stdt_acc_score;
    extern struct stdt_db pupil;
    extern int first_ques;

    int i;
    int reply;

    if (lesson_map[mapndx].text_type == 5)
    { /* Verify satisfaction with previous answers */
        clear();
        cntr_ln(y_mid, "Are you satisfied with your previous answers? [Y/N]: ");
        reply = tolower(getch());
        if (reply != 'y')
            { /* Reviewing answers */
                return 3;
            } /* Reviewing answers */
    } /* Verify satisfaction with previous answers */

    if (lesson_map[mapndx].text_type == 5)
    {
        pupil.know_phase[pupil.know_phase_ndx-1].act_val += stdt_acc_score;
        first_ques = -1;
    }
    for(i=0; i<lesson_map[mapndx].mc_ans_cnt; i++)
    { /* Search of interval */
        if (stdt_acc_score <= lesson_map[mapndx].mc_ans[i].ans_val)
            { /* Interval established */
                if (pupil.know_phase_ndx != i+1 && lesson_map[mapndx].text_type == 5)
                    { /* Changing knowledge level */
                        pupil.know_phase[i].max_val = 0;
                        pupil.know_phase[i].min_val = 0;
                        pupil.know_phase[i].act_val = 0;
                        pupil.know_phase[i].spare_val = 0;
                    } /* Changing knowledge level */
                if (lesson_map[mapndx].text_type == 5)
                    pupil.know_phase_ndx = i + 1;
            else
                pupil.text_blkndx = lesson_map[mapndx].know_level[i];
    } /* Interval established */
} /* Search of interval */

if (lesson_map[mapndx].text_type != 7)
    stdt_acc_score = 0;

    return 2;
}

/* Assess student's score */
int lesson_txt(int map_ndx)
{ /* Change lesson or instruction block */
extern FILE *present_fp;
extern struct stdt_db pupil;
extern struct lesson_file_entry lesson_map[];
extern int stdt_acc_score;

int i;
int reply;

fclose(present_fp);
present_fp = NULL;
sscanf(&lesson_map[map_ndx].mc_ans[pupil.know_phase_ndx-1].ans_desig,
"%d", &pupil.instr_blk_ndx);
pupil.lesson_ndx =
lesson_map[map_ndx].mc_ans[pupil.know_phase_ndx-1].ans_val;
pupil.text_blk_ndx =
lesson_map[map_ndx].know_level[pupil.know_phase_ndx-1];
student_blk_io('W', 0);
for (i=0; i<5; i++)
    student_blk_io('W', i+1);
clear();
cntr_ln(y_mid, "End of an instruction block");
cntr_ln(y_mid+1, "Continue [Y/N]: ");
reply = tolower(getch());
if (reply == 'y')
    return 4;
else
    return 1;
}

} /* Change lesson or instruction block */
void get_max_min(int map_ndx, char op)
{ /* Get maximum and minimum answer values */
  extern struct stdt_db pupil;
  extern struct lesson_file_entry lesson_map[];

  int i;
  int temp_max, temp_min;

  temp_max = 0;
  temp_min = 9999;
  for (i=0; i<lesson_map[map_ndx].mc_ans_cnt; i++)
  { /* Find maximum and minimum values */
    if (lesson_map[map_ndx].mc_ans[i].ans_val > temp_max)
      temp_max = lesson_map[map_ndx].mc_ans[i].ans_val;
    if (lesson_map[map_ndx].mc_ans[i].ans_val < temp_min)
      temp_min = lesson_map[map_ndx].mc_ans[i].ans_val;
  } /* Find maximum and minimum values */
  if (op == '+')
  { /* Add max & min values */
    pupil.know_phase[pupil.know_phase_ndx-1].max_val += temp_max;
    pupil.know_phase[pupil.know_phase_ndx-1].min_val += temp_min;
  } /* Add max & min values */
  if (op == '-')
  { /* Subtract max & min values */
    pupil.know_phase[pupil.know_phase_ndx-1].max_val -= temp_max;
    pupil.know_phase[pupil.know_phase_ndx-1].min_val -= temp_min;
  } /* Subtract max & min values */
  if (op == ' ') /* Display values */
    mvprintw(22,0,"high = %4d score = %4d low = %4d indx = %4c",
      pupil.know_phase[pupil.know_phase_ndx-1].max_val,
      stdt_acc_score,
      pupil.know_phase[pupil.know_phase_ndx-1].min_val,
      map_ndx);
  refresh();
  getch();
  move(22,0);
  clrtoeol();
  refresh();
} /* Display values */
return;
} /* Get maximum and minimum answer values */
void reply_pos(int mapndx)
{ /* Position cursor for response */
    extern struct lesson_file_entry lesson_map[];

    if (lesson_map[mapndx].text_type == 2)
        move(LINES-1, 11);
    else if (lesson_map[mapndx].text_type == 3)
        move(LINES-1, 16);
    else
        move(LINES-1, 12);

    return;
} /* Position cursor for response */
Appendix E

ITS User’s Guide
Intelligent Tutoring System

User's Guide

You may run the ITS program using a SGI terminal even though its software is spread across a SGI/UNIX and DEC/VMS based system. Functionally, the software is used as follows:

- The presentation of the questionnaire function is accomplished using only the SGI/UNIX system.
- The presentation of the Simulation is accomplished using a combination of both the SGI/UNIX and DEC/VMS systems.
- The presentation of the evaluation software is accomplished using only the DEC/VMS system.

Each function is started separately. While having to start each function is not ideal, the modularity did provide a better software development environment with no impact to the existing Simulation software.

Questionnaire Function

To begin the Questionnaire Function, you must have the "ITS Window" on the screen. This window is sized to an 80 character by 24 line (80x24) display for the presentation of questionnaire material. Enter the following command to initiate this window:

```
wsh -n 'ITS Window' -p175,300 -s80,24
```

Enter ITS at the prompt within the window.

ITS prompts you as follows:

1. Use cursor to select:
   Instructor  Student
   then press 'Enter'

The desired selection is highlighted on the screen.

If you select the Instructor option, continue with Step 2.
If you select the Student option, continue with Step 13.
2. At this point, nothing on the screen changes. However, ITS expects you to enter the proper password before continuing. Entry of any other value causes the program to terminate without any indication. Once you enter the proper password, continue with Step 3.

3. ITS displays the following prompt:

   Enter index of desired option:

   1. Add an instruction block
   2. Delete an instruction block
   3. Modify an instruction block
   4. Review an instruction block
   5. Review a student's performance
   6. Adjust a student's instruction

   Selection:

   Enter the desired option by specifying 1, 2 etc.

   Note: Only options 1-3 are implemented. Selections 4-6 are not implemented at this time.

   Enter one of the options 1-3, continue with Step 4.

   Note: If you enter any value other than 1-6, ITS displays Invalid response. Press the Enter key to acknowledge and ITS displays the following message:

       Terminate ITS [Y/N]:

   If you enter any value other than Y, ITS interprets it as a N response and returns to the start of this Step. If you enter Y, the program terminates.

4. ITS prompts you for the specification of the path to the Instructor directory. However, if you have already accomplished this Step (on a previous pass through the program), ITS continues with Step 5. If not, the following display appears:

   Enter path to "Instructor" file directory
   ->
You may either specify the Instructor file directory or press the Enter key to get the default value as stipulated by instr_path_def in itsdef.h. In both cases ITS prompts you to verify the specification:

Is path correct? [Y/N/Q]:

If you enter Q, the program terminates.
If you enter N, this Step is repeated.
If you enter Y, continue with Step 5.

5. ITS displays titles of existing Instruction Blocks\(^1\) along with the following message:

Use cursor to position, 'Enter' to select, or 'Q' to quit

If you selected option 1 in Step 3, Add an Instruction Block, ITS displays a reminder of existing titles. For options 2 or 3, you may choose the Instruction Block that is to be modified or deleted, respectively.

If you selected option 1 in Step 3, continue with Step 6.
If you selected option 2, Delete an instruction block, ITS continues with Step 9.
If you selected option 3, Modify an instruction block, ITS continues with Step 12.

6. ITS asks you to enter a title to the new instruction block:

Enter title (max = 80 chars):

Enter the title and continue with Step 7.

---

\(^1\) Refer to the section Implementation under Silicon Graphics Software of this document.
7. ITS asks you if there are any lessons\(^2\) associated with this Instruction Block:

Any lessons in this instruction block [Y/N]:

You may create the instruction block entry by entering a N and ITS continues with Step 3, or if lessons are to be added, enter Y and continue with Step 8.

8. ITS displays the title of the instruction block and a numbered list of empty lesson entries, followed by the prompt:

Enter lesson name (max = 10 char):

Enter the names of the lessons that are to comprise this instruction block. When you have finished, terminate this Step by entering a q to lesson name and ITS continues with Step 3.

9. ITS displays the title of the instruction block and a numbered list of the lessons in the block along with the following prompt:

Do you want to delete lessons? [Y/N]:

If you want to delete only the Instruction Block, enter N and ITS continues with Step 3.

If you also want to delete lesson files, enter Y and continue with Step 10.

10. ITS displays the following prompt:

Delete (a)ll or s(ome) of these lessons:

If you want to delete all the lesson files listed, enter a and ITS continues with Step 3.

If you want to selectively delete some of the lesson files, enter s and continue with Step 11.

\(^2\) Refer to the section Implementation under Silicon Graphics Software for additional details.
11. ITS displays the following prompt:

   Enter index of lesson to be deleted:

   Enter the number that appears with the lesson name.

   Note: The numbered list is reordered after each deletion so indices can change for each specified deletion.

   When you are finished deleting, enter q and ITS continues with Step 3.

12. ITS displays the title of the instruction block and a numbered list of the lessons in the block along with the following prompt:

   Modifying (t)itle or (l)esson:

   If you want to modify the title of the instruction block, enter a t and the prompt Enter title will overwrite the previous prompt and allow you to specify a new title. ITS continues with a repetition of this Step.

   If you want to modify a lesson, enter l and ITS displays the following prompt:

   Enter index of lesson to be changed:

   Enter the number associated with the lesson to be changed and ITS displays the following prompt:

   Change: (d)elete, (m)odify, (i)nsert:

   If you want to delete the selected lesson, enter d ITS continues with the repetition of this Step.

   If you want to insert or modify a lesson, enter m or i. ITS displays the prompt Enter lesson name. Modify or insert the lesson name and ITS continues with a repetition of this Step.

   To terminate this Step, enter a q and ITS continues with Step 3.

13. ITS asks you to specify the path for the Instructor and Student directories. If one or both use the default values, the values for the instructor and student paths are taken from the instr_path_def or the stdt_path_def values in itsdef.h, respectively.
For the **instructor** path, ITS displays:

Enter path to "Instructor" file directory

For the **student** path, ITS displays:

Enter path to "Student" file directory

Both are followed by the prompt:

Is path correct? [Y/N/Q]:

If you enter **Q**, the program stops.

If you enter **N**, this Step is repeated.

If you enter **Y**, then:

a. If this response is to the **instructor** path prompt, ITS continues with a request for the **student** path.

b. If this request is to the **student** path prompt, ITS continues with Step 14.

14. ITS prompts you (the student) for identification:

Enter name and SSAN:

Name:

ITS asks you to verify identification:

Is name spelled correctly? [Y/N]

If you answer **N**, this request is repeated.

If you answer **Y**, ITS requests your Social Security Number (SSAN):

SSAN: xxx-xx-xxxx

Verify your entry by answering:

Is SSAN correct? [Y/N]
If you enter N, SSAN portion of the this Step is repeated.

If you enter Y, continue with Step 15.

15. If the student is new, ITS displays the following verification step:

Are you a new student? [Y/N]

If you enter N, ITS repeats Step 14. Otherwise, if the student is either new, as indicated by a Y reply, or has not completed the Experience Questionnaire, as indicated in the student’s database, ITS presents the questionnaire.

At the conclusion of the questionnaire, ITS begins the Simulation. If the student is above the base entry level, then the Simulation is used to present a scenario to try and validate the determined level. Otherwise, the Simulation consists of the Console Checkout lesson.

Simulation Function

Documentation for the Simulation Function can be found in:


Evaluation Function

To execute the Evaluation Function, you must be in a window with access to the VAX. It is also assumed that the student has completed a session with the Simulation. Prior to running the software to evaluate the student’s performance, the data captured by the Simulation must be preprocessed so it is time ordered. This is accomplished by executing the following command on the logger file:

```
reduce "mindisk 0 f 1 l 6 sim 90 dir [logger file directory string] status"
go
quit
```

Upon completion of the REDUCE run, begin the evaluation by entering:

```
run sdt_eval
```
Processing continues with the following steps.

1. ITS asks you to specify the location of the output of the previously mentioned REDUCE process.

   Enter name of the data file
   Name:

   Enter the catalog/file string of the Pass 6 logger file from REDUCE. Upon completion, ITS asks you to verify that the file specified is a REDUCE pass 6 output.

2. ITS asks you to specify the number of WDs tested:

   Enter the number of WDs that were tested:

3. ITS asks you to associate the WDs with a specific console number by answering the following query for each WD tested:

   Enter console no. of xxx WD:

4. ITS asks you to specify the WD id number for each console that contained a tested WD.

   Enter WD id no. for console x:

   where x is substituted with the appropriate console number.

5. For this Step, ITS displays the following:

   Select skill level for evaluation:

   1) Naive
   2) Novice
   3) Journeyman
   4) Expert
   5) Master

   Selection:

   Enter an appropriate value. At this time, this value is not used. The intent is to provide multiple criteria when evaluating the scenario from a Simulation run.
6. ITS asks for the location of the event script file³:

   Enter name of event script file
   Name:

   Upon answering the name of the event script file, processing continues until the event script is exhausted. As the absence or presence of each is event is detected, a brief message about the condition is displayed. You must acknowledge each message with the Enter key before the program continues. Currently, display of the captured data is limited to the screen. However, recording these data to a file could be easily accomplished.

³ Refer to the section Implementation under DEC VAX 780 for an explanation of the event script file.
Appendix F

Decision Tree