NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A HYPERTEXT-BASED INTELLIGENT LEARNING ENVIRONMENT FOR THE X WINDOW SYSTEM

by

James Michael Delani Jr.

September 1994

Thesis Advisor: Yuh-jeng Lee

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

4. TITLE AND SUBTITLE
A HYPERTEXT-BASED INTELLIGENT LEARNING ENVIRONMENT FOR THE X WINDOW SYSTEM (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Delani, James Michael Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*

Intelligent tutoring systems have failed to achieve practical benefits commensurate with their potential for time savings, increased efficiency, and improved effectiveness. The problem consists of three inter-related factors. First, the complexity and computational requirements of these systems often causes user interface design to be relegated to an ancillary role in the development cycle. Second, creating an effective and usable interface is an inherently difficult task that often requires as much time and effort as the logical components. Finally, the lack of an intuitive and navigable interface interferes with the student's primary task of learning new material, precluding the success of these systems.

Our approach is to provide an alternative framework for tutorial developers which assumes the burden of interface design. This framework consists of a hypertext-based instructional format that authors can easily convert their lessons into, facilitating rapid prototype development. By creating an Intelligent Learning Environment (ILE) designed to dynamically integrate hypertext-based tutors and reference materials, we can also provide users with a consistent, familiar, and highly navigable interface which will simplify the learning process.

Using this approach we have successfully implemented an integrated environment for the X Window System. We have created a course of instruction for the Ada programming language, including the Ada 9X Language Reference Manual in hypertext format. The ILE may be dynamically extended through the modification of a text-based configuration file. This enables additional hypertext-based tutorials, reference manuals, and help systems to be incorporated into the learning environment without requiring recompilation. This also allows developers to dynamically extend the functionality of individual tutor systems through the incorporation of student evaluation modules, software testing tools, or syntax directed editors.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Intelligent Tutoring Systems, Intelligent Computer Aided Instruction, Computer Based Training, X Window Applications	128
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

A HYPERTEXT-BASED INTELLIGENT LEARNING ENVIRONMENT FOR THE X WINDOW SYSTEM

James Michael Delani Jr.
Captain, United States Marine Corps
B.S.E.E., Boston University, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: James Michael Delani Jr.

Approved By: Yuh-jeng Lee, Thesis Advisor

Mantak Shing, Second Reader

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Intelligent tutoring systems have failed to achieve practical benefits commensurate with their potential for time savings, increased efficiency, and improved effectiveness. The problem consists of three inter-related factors. First, the complexity and computational requirements of these systems often causes user interface design to be relegated to an ancillary role in the development cycle. Second, creating an effective and usable interface is an inherently difficult task that often requires as much time and effort as the logical components. Finally, the lack of an intuitive and navigable interface interferes with the student's primary task of learning new material, precluding the success of these systems.

Our approach is to provide an alternative framework for tutorial developers which assumes the burden of interface design. This framework consists of a hypertext-based instructional format that authors can easily convert their lessons into, facilitating rapid prototype development. By creating an Intelligent Learning Environment (ILE) designed to dynamically integrate hypertext-based tutors and reference materials, we can also provide users with a consistent, familiar, and highly navigable interface which will simplify the learning process.

Using this approach we have successfully implemented an integrated environment for the X Window System. We have created a course of instruction for the Ada programming language, including the Ada 9X Language Reference Manual in hypertext format. The ILE may be dynamically extended through the modification of a text-based configuration file. This enables additional hypertext-based tutorials, reference manuals, and help systems to be incorporated into the learning environment without requiring recompilation. This also allows developers to dynamically extend the functionality of individual tutor systems through the incorporation of student evaluation modules, software testing tools, or syntax directed editors.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The primary advantage of individualized instruction is that it provides the student with an environment specifically tailored to his or her learning needs and goals. The promise of achieving the effectiveness of personalized instruction without incurring the high cost of personal tutors has been one of the driving factors in the research and development efforts in the field of computer-aided instruction (CAI). Early efforts were less than successful due to the prohibitive cost and limited capability of existing hardware and software. Despite revolutionary advances in both the hardware and software industry, CAI applications have still failed to deliver on their promise for time savings, higher efficiency, and improved effectiveness.

Kearsley [Kear87] attributes the field's inability to achieve practical benefits to three critical factors: the computational requirements of these systems; the lack of an adequate research basis to properly understand learning strategies; and the lack of researchers and scientists involved in CAI. Although these factors have certainly had substantial limiting influences, the field has still experienced tremendous advances in the quality and sophistication of these systems. An even greater factor in the success of these systems that is often overlooked is the quality of the user interface.

One of the major problems with traditional programmed instruction is that it confines the student to a prescribed and immutable sequence of screen presentations, which expedites the inevitable loss of interest. Recent efforts have shifted the focus of development on creating student models designed to completely represent current levels of understanding, comprehension, and ability to apply problem solving skills. The problem with these systems is that the tools needed to develop these elaborate models have interfered with the student's primary task of exploring and learning new information.

The problem is further exacerbated by the conflicting goals of the principle communities involved with the development of these systems. The primary goals of the

research community center around the advancement of their respective fields. Often the knowledge and insight gained in the development process is of equal or greater importance than the finished product. There is little reason to divert their efforts to include development of an effective and usable interface when this is not part of their objectives. On the other hand, developers who are members of the business community have goals that are related to their financial success. They are understandably more concerned with creating a polished and usable product, but are much less concerned with advancing the research basis of any of the fields involved.

Our approach is to provide an intelligent learning environment as an alternative framework for the development of these systems. By providing a means for tutorial developers to intelligently organize and present their instructional material, we can greatly facilitate their efforts. This enables them to continue to focus on the computational aspects of their systems, since we will be providing the interface mechanisms for them. The concept of a learning environment offers substantial advantages to users as well. They are provided with a consistent, familiar, and highly navigable user interface that will facilitate the learning process. This allows them to focus on their primary task of exploring and learning new information. Our objective is to facilitate the learning process, not over-complicate it.

We believe that this objective can be accomplished through the simplification of two critical areas of existing system models. The first and most dramatic change involves abandoning efforts to have the system explicitly model a human tutor in order to replace the human element altogether. Tutorials in our system will act as intelligent assistants, designed to supplement human teachers or instructional material. We are more interested in what the interaction between man and machine can accomplish than in what machines can accomplish by themselves. The second change involves strictly enforcing the separation of the presentation of instructional material from the student evaluation process. This enables our system to facilitate the interaction with its users, providing them with complete access and control of the information. Users will still be expected to view the

instructional material in the order the authors intended, but there will be no control mechanisms in place to enforce this.

B. GOALS AND OBJECTIVES

The Intelligent Learning Environment (ILE) has been developed to fill a perceived void that exists in the area of programming language tutorials. The principle problem area is the usability and effectiveness of the tutor system itself. One of the fundamental goals of the ILE is to develop an intelligent and useful user interface that can provide a program development environment suitable for both novice and advanced programmers. To accomplish this, the environment should be able to integrate the following hypertext based materials: instructional tutorials, on-line reference manuals, programming language help files, and system help files. In addition, specific mechanisms for extending the capabilities of the environment through incorporation of additional software tools must be provided to tutorial developers.

C. SCOPE

As the underlying goal of this endeavor has been to provide a learning environment that is useful to potential students, the main focus of effort has been in providing a usable interface which integrates diverse tutoring subsystems and on-line reference materials, not in the implementation of a particular language tutorial. Although we have used this tool to implement a tutorial in Ada, we could just as easily have provided a tutorial in any programming language. Our main concern has been to create an integrated environment that is functional, extendible, and most importantly, usable. We have provided mechanisms for dynamically extending the environment through the incorporation of additional tutoring systems or reference material, and have provided mechanisms for dynamically extending the power and functionality of individual tutoring systems through the incorporation of additional student evaluation modules, software testing tools, or syntax directed editors.

D. ORGANIZATION

Chapter II of this thesis provides an overview of intelligent tutoring systems as well as an introduction to the evaluation criteria that currently exists for these systems. Chapter III discusses the conceptual design of the intelligent learning environment that we have developed. In Chapter IV we describe the implementation in greater detail, discussing specific design considerations and trade-offs. Chapter V contains a summary of the development of an Ada tutorial which has been integrated as a subsystem of the learning environment. This exemplifies the ease with which tutorial authors can develop rapid prototypes. In Chapter VI we provide a detailed example to demonstrate how the system works from the users's perspective. We also illustrate how easily the system can be extended by either tutorial developers, system administrators, or users themselves. Finally, in Chapter VII we present our conclusions and ideas for future work.

II. INTELLIGENT SYSTEMS

A. INTRODUCTION

Traditional computer applications can be characterized as time-saving tools that have automated many of the manual tasks associated with office work, desktop publishing, and scientific research and development. They are typically implemented using well defined algorithms, and do not attempt to perform creative tasks such as problem solving, diagnosis, or planning. Intelligent systems, in contrast, are designed to emulate the human thinking process, and have a strong basis in artificial intelligence. Typical applications for intelligent systems include natural language processing, speech recognition, robotics, computer vision, diagnosis, and tutors. [BiLe91]

The use of intelligent systems for the development of tutoring applications is the focus of this thesis. Although these systems have seen tremendous advances in the past decade, the enormous potential for computer-assisted instruction (CAI) has been recognized throughout the history of modern computers. Some of the earliest users of CAI were members of the computer industry, who, in the late 1950's, used CAI programs to train their personnel [Skin86]. Unfortunately, the cost of the hardware and the complexity of the programming languages of that era effectively precluded the use of CAI in educational settings. Despite this shortcoming, the possibilities attracted researchers from such diverse fields as cognitive psychology, education and training, and of course, computer science.

These prohibitive factors have become virtually inconsequential with the technological revolutions that have taken place in the computer hardware and software industries. Extensive research and development has also transformed the structure of CAI as well. Traditional applications in computer based instruction have primarily consisted of sequences of screen presentations that students were required to view in some predetermined order. Small sections of the instructional material, often called frames, are successively displayed on the screen according to a predefined set of responses from a

student. This method was prevalent in the 1970s, and is often referred to as frame-oriented CAI [Weng87].

Developments in the fields of cognitive studies and artificial intelligence have generated significant changes to this model. The traditional methods of programmed instruction evolved into intelligent computer-assisted instruction (ICAI), which have become more commonly referred to as intelligent tutoring systems (ITS). These systems are programs that incorporate three major disciplines: computer science, cognitive psychology, and education and training. The field that encompasses these three disciplines is often referred to as cognitive science [Kear87]. Within computer science, the fields having the most influence are artificial intelligence, software engineering, and human factors engineering. Recent trends in ITS development have changed the focus to creating elaborate student models that reflect users' current level of understanding and ability to apply problem solving skills. Despite the tranformations that the field of CAI has experienced, it still remains an "attempt to produce computational models of interactive knowledge communication". [Weng87]

B. CURRENT TUTORING PARADIGMS

Most of the current examples of intelligent tutoring systems can be classified according to one of five teaching strategies or paradigms [Kear87].

1. Mixed Initiative Tutors

Mixed Initiative Tutors are the oldest style of intelligent tutoring systems. This method incorporates the Socratic method of teaching by engaging the student in conversation and attempting to lead the student to discovery. The tutor first attempts to discover the student's specific weaknesses or misconceptions, and then presents instructions designed to help the student discover these errors. Examples of this approach include the SCHOLAR and SOPHIE programs, both of which are discussed in extensive detail in [Weng87].

2. Diagnostic Tutors

Diagnostic tutors try to identify the misconceptions a student may have in solving a problem by using a catalog or knowledge base of common problems. These systems typically focus on problems that stem from conceptual or semantic errors, which extends the analysis well beyond the issue of syntactic correctness. Although these tutors are suitable for most problem-solving applications, they are easier to implement for problems with closed solutions [Kear87]. Examples of the diagnostic paradigm include BUGGY, DEBUGGY, and PROUST.

3. Microworlds

Microworlds involve developing a computational tool that allows a student to explore a problem domain. A relatively well known example of this paradigm is the LOGO programming environment developed at the Massachusetts Institute of Technology (MIT) by Seymour Papert as a tool for teaching mathematics and science to children. LOGO combines computational theory and artificial intelligence with Piaget's theory of learning. In this theory, people learn by naturally and spontaneously interacting with their environment [Pape80].

This actually represents a distinct shift in traditional computer based tutoring systems. Normally the computer is utilized to teach, or in the case of programmed instruction, to "program" the student. In LOGO, the student's interaction with the tutor consists of programming the computer. This provides students with a sense of mastery over an otherwise intimidating machine, and simultaneously familiarizes them with conceptual ideas from science, mathematics, and the cognitive problem solving process [Pape80]. The lack of rigidity, discipline, and absolute control enforced by the tutor is typical of programs that use the microworld concept. The student is allowed to explore and discover the various relationships and scientific truths contained in the knowledge domain. Inherent in this paradigm, then, is the need for some type of supervision and assistance from other than the tutor. The astute reader will notice that this paradigm has received quite a bit more analysis

than the other models presented. This is because Piaget's theory that people learn by naturally and spontaneously interacting with their environment is entirely consistent with our thesis. We feel that it is much more important to provide users with a learning environment than it is to provide them with a rigidly structured tutorial.

4. Articulate Expert Systems

Articulate Expert Systems can explain their decisions, which helps provide reasoning or analysis skills to the student. These systems typically consist of expert systems, which can be utilized to provide practice in problem solving and decision making skills. By providing the reasoning process used to arrive at conclusions in addition to the conclusions themselves, these systems help develop the cognitive process. Well known examples of these systems include GUIDON and MYCIN [Clan87].

5. Coaches

Coaches observe the student's performance in some problem solving activity and provide advice and guidance that will help the student perform better. The complexity of these systems stems from the requirement that the tutor knows when in the course of the problem solving activity it should interrupt the student and offer advice. Once it has interrupted the student, it must also be able to determine the degree of assistance that is required to continue with the problem solving effort [Kear87].

C. EVALUATING INTELLIGENT TUTORING SYSTEMS

There is a distinct difference in the evaluation criteria that has been established for CAI as opposed to the standards typically applied to ITS. The success of CAI programs is directly related to the level of instructional effectiveness and efficiency. To ensure the quality of these programs there is extensive analysis and review during the development process, as well as a validation process following completion [Kear87].

With ITS, however, the success is primarily determined by its ability to perform those aspects of the system which provide the intelligence to the system. This often

involves evaluation of special processes involved in the instruction, including the system's inference and reasoning mechanisms, diagnostic capabilities, cognitive process modeling, and remediation. Since the researchers involved with ITS come from such diverse fields as cognitive psychology, education and training, and computer science, the evaluation criteria used in any particular application is heavily influenced by the primary research goals of the developers. While this diversity has contributed significantly to the advances made in the field of cognitive science, it has also undermined standard software development practices. "Apparently, no systematic evaluation procedures (formative or summative) have been used in ICAI to assure the quality of the program during the development process or to validate its success after development". [Kear87]

While a formal evaluation criteria is not universally used, prominent developers of ITS have constructed a set of standards to apply to their own systems. One such set proposes that an ITS can be evaluated based on how well it accomplishes four main activities: modeling of knowledge and reasoning, communication, cognitive processing, and tutoring [Wool92]. From this set and the previous discussion we can conclude that the primary concern of intelligent tutoring system developers is in ensuring that their systems possess sufficient "intelligence" to facilitate the task of knowledge communication.

D. A CRITICAL VIEW

As the field of CAI gradually evolved into ITS, the change in priorities and development strategies has produced a number of impressive programs. The problem, however, is that very few of these systems are actually used. As we discussed in our introduction, there are a number of valid reasons that have been provided to explain the lack of practical benefits that this field has produced, all of which conveniently overlook one critical factor. These systems aren't being used because they are too hard to use! Users who are trying to learn a new subject domain do not need the additional burden of learning a clumsy and inefficient interface. By relegating the effectiveness of these systems to a secondary role in the development and evaluation process, ITS developers have effectively

guaranteed that their systems are incapable of accomplishing their stated purpose of tutoring students.

An assumption that requires considerable rethinking is that these tutoring systems shall be the only means for students to gather information and learn the subject matter, and that these machines must be able to perform all of the tasks of a human tutor. One of the design criteria proposed earlier stated that ITS should be able to dynamically determine a student's learning strategy so that it could employ the most appropriate teaching style from its diverse repertoire [Wool92]. This criteria expects computers to demonstrate human skills that many professional educators have been unable to master. This also implicitly assumes that a sufficient knowledge base or understanding of how people actually learn already exists, which is simply not true. "Almost every cognitive scientist would quickly acknowledge that our current theories of learning are woefully inadequate to explain or predict how people learn." [Kear87] This lack of an adequate research basis further substantiates the requirement that these programs be evaluated on their level of instructional effectiveness and efficiency.

Hamming [Hamm93] has stated unequivocally in several of his classes here at the Naval Postgraduate School (NPS) that the focus of computer research and development should be on the interaction between man and machine, and what the two of them can accomplish together that is important, not what they can do in isolation. Developers of intelligent systems seem to have the mistaken belief that in order to demonstrate the intelligence of a system it must be capable of acting completely autonomously. Intelligent tutoring systems do not have to replace teachers altogether in order to increase efficiency, effectiveness, and productivity. They can instead function as intelligent assistants, augmenting a particular course of instruction or extending a teacher's influence [BiLe91].

Another key consideration that ITS developers overlook is the fact that most of their users will have a tangible desire to learn. Students who use these systems do so because they are interested in learning a particular knowledge domain, not because they are problem students who have been assigned a personal tutor for remediation. ITS developers have

tried to incorporate a multitude of teaching strategies and diagnostic capabilities so that they can accurately match the appropriate teaching and learning strategies. This assumes too much of the burden of the learning process.

> The burden of teaching is on the person who wants to teach. The burden of learning is on the person who wants to learn; not on the person who wants to teach. [Hill77]

These students already possess the motivation to learn. What is needed of these tutorial systems is to facilitate these students' efforts, to reduce the burden of learning for them by organizing and intelligently preparing the knowledge, and by providing an efficient and usable mechanism to access this information.

One of the most critical evaluation criteria of an intelligent tutoring system is its interface with the student. How it presents and communicates the subject matter has profound influence on its success as a learning tool. Gause and Weinberg [GaWe89] claim that a majority of product development efforts fail simply because they produce a product that people don't like or enjoy using. Several examples of tutoring systems whose failure has been solely attributed to lack of a usable interface can be found in [ABCL90]. We shall conclude our discussion of evaluation criteria with two quotes which succinctly capture the essence of our thesis.

> One might have thought that the discussion to this point would complete the description of our tutoring systems. We have stated how a tutor models a student and how it uses the model to achieve pedagogical goals. However, the discussion is abstract and leaves completely unspecified what the student actually experiences, which is the computer interface. We have learned that design of the interface can make or break the effectiveness of the tutor. [ABCL90]

The significance of this critical aspect has been emphasized in every area of computer science. It is evidenced not only in end applications themselves, but in programming methodologies as well. If a tool, application, or methodology is unnecessarily difficult to learn or use, it simply will not be used, regardless of the potential for time savings, improved efficiency, or productivity.

> Intelligent system developers sometimes short change the user interface of their programs. Once they succeed in getting the program to perform logically, they sometimes consider their job done. But much software, possibly otherwise worthwhile, goes unused because the user interface is not effective. To interest users in an intelligent system, the interface must be so well designed as to be inviting to use and also to inspire confidence in the system. [BiLe91]

What this statement fails to convey is that design of the interface cannot be done at the end of the development - it must be developed concurrently with the computational aspect. Waiting until the final stages of the software development life cycle to begin work on the user interface establishes this attribute as the lowest priority quality attribute, effectively guaranteeing substandard results.

III. DESIGN CONSIDERATIONS

The idea for the Intelligent Learning Environment grew from the author's frustration with available tutoring systems. In addition to the unnecessarily difficult task of learning an entirely new interface for each new tutorial, the interfaces provided were often counter-intuitive, clumsy, and an unwelcome burden in the learning process. While many of the systems' developers have created programs which satisfy most of the evaluation criteria discussed in the previous section, especially with respect to the computational aspect, they seem to have totally disregarded the interface element.

It is important to realize that the creation of an effective and usable interface for a particular application will not solve the problem. Users would still be required to learn a new (hopefully superior) interface, and nothing would be done to facilitate the development of a consistent and familiar interface among diverse tutorial applications. What is needed is a system that not only provides a superior interface, but also provides a mechanism for other developers to easily incorporate this interface into their systems as well. It was with this goal in mind that the concept of a learning environment was developed.

A. INTEGRATED SYSTEM DESIGN

The purpose of developing a tutoring system is to communicate specific knowledge to a user or group of users. The development of a tutoring environment involves taking the level of abstraction used in development of tutorial applications one step further. Instead of developing a single application, creating an environment involves developing a system of applications. Inherent in this design is providing a mechanism for integrating a diverse set of subsystems. In order to maintain a consistent interface throughout this set, an alternative framework must be provided that tutorial developers may easily convert their prepared instructional material into. This process is greatly simplified by separating the presentation of instructional material from the student evaluation process, thereby creating a generic interface mechanism. This provides users with a familiar environment in which they can explore and learn different programming languages. They can exploit this

familiarity and concentrate on the task at hand - learning a new subject matter, without having to learn a new system as well.

B. SYSTEM SPECIFICATIONS

The intelligent learning environment has been developed to provide an effective and consistent means of communicating diverse knowledge domains to its users. In addition to providing the framework for intelligent tutoring system development, it also provides the means for incorporating hypertext-based on-line reference manuals and help systems. The primary purpose of these latter features is to provide users with a tool which can be used in subsequent programming endeavors. The usefulness of this environment can be enjoyed even by advanced users who have no interest in the tutorial applications.

As discussed in the previous chapter, the Microworld paradigm is generally recognized as the design which is most flexible and gives the most control to the student. A student is allowed the freedom to learn about the topics that are of immediate interest, thereby increasing the student's motivation to learn. Although the lack of discipline enforced by this paradigm often requires the need for some type of supervision and assistance from other than the tutor, this methodology is particularly suitable for tutor systems that attempt to cover a broad range of subjects in a single system, or for a group of smaller tutor systems all under the control of a single integration system [Camp90]. These qualities are precisely what is necessary for the development of our integrated environment.

The fundamental design considerations of the intelligent learning environment can be prioritized as follows:

- Usability
- Portability
- Extensibility

There are two important elements to notice about this priority list. First of all, the computational and logical aspects of the system are not even accounted for. This is because they are secondary considerations in the design of the learning environment. They are still vitally important in the development of the tutorial subsystems, especially in the student

evaluation components. The second point to be aware of is that these characteristics are often conflicting, meaning that improving the quality in one criteria requires diminishing the quality of another. In particular, creation of an elaborate and usable interface makes maintaining portability extremely challenging, primarily due to the close tie between screen manipulation actions and machine architecture [Delo91]. As the underlying goal of our development effort is to provide an interface which empowers our users, usability will always be the overriding concern. Our first and foremost objective is to facilitate the learning process.

C. USER INTERFACE

1. Graphical User Interface

The term graphical user interface (GUI) basically describes an interface that utilizes windows, menus, buttons, icons, and other graphical objects which allows users to interact with the application through manipulation of a mouse or other pointing device. One important consideration is that although the system may contain a multitude of modules and subsystems which offer tremendous computational capabilities, these aspects are normally transparent to the user of the application. To the end user the interface represents the entire system; if the interface is lousy, then so is the system.

Usability is defined as a combination of the following characteristics [HiHa93]:

- Ease of learning
- High speed of user task performance
- Low user error rate
- Subjective user satisfaction
- User retention over time

In other words, in addition to the qualitative aspects relating to the appearance and control mechanisms provided by an application, usability is directly related to the efficiency and effectiveness of the interface, as well as to the user's reaction to the interface. It is important to note that this focus on user-centered design often increases the level of difficulty in developing the application tremendously. This is one of the reasons

that this attribute must be addressed early in the software development cycle. Since the purpose of our system is to provide a learning environment to its users, it was decided that providing an intuitive and easily navigable graphical user interface was a minimum requirement of our system.

2. Selecting an Interface Development Platform

While research has shown that windowing interfaces are easier to learn and use than character-based interfaces, there is a plethora of successful and popular windowing systems to choose from. Most notable for personal computer development are the Apple Macintosh, Microsoft Windows, and IBM's OS/2, while MIT's X Window System, Sun Microsystem's OpenWindows, and Digital Equipment Corporation's (DEC) DECWindows are available for workstations. The X Window System, by virtue of its operating system independence and network transparency, offers a great deal more flexibility and potential than any of the other systems mentioned. [JoRe92]

The X Window System is an industry-standard software system that allows programmers to develop portable GUIs. It was developed at MIT's Laboratory for Computer Science for Project Athena to fulfill that project's need for a distributed, hardware-independent user interface platform [YoPe92]. It was this promise of hardware-independence which first attracted us to this windowing platform. The X Window System is currently available for UNIX systems, DEC's VAX/VMS systems, and many personal computers. While portability and cross-platform development capabilities represent severe limitations of most of the windowing systems listed above, this is not the case for the X Window system. The hardware and operating system independence of X make it extremely portable and ideally suited for cross-platform development. "To a large extent, X Window is the right windowing system at the right time." [JoRe92]

The software architecture of the X Window system is actually comprised of several layers, as illustrated in Figure 1. At the lowest level above the network protocol is the standard C library routines, called Xlib. These routines are the same on every machine

running X, which provide its operating system independence. The next layer in the hierarchy is the Xt Intrinsics library, which is built upon Xlib and offers object-oriented capabilities that support user interface development. This functionality is provided by widgets, which are basically interface component classes.

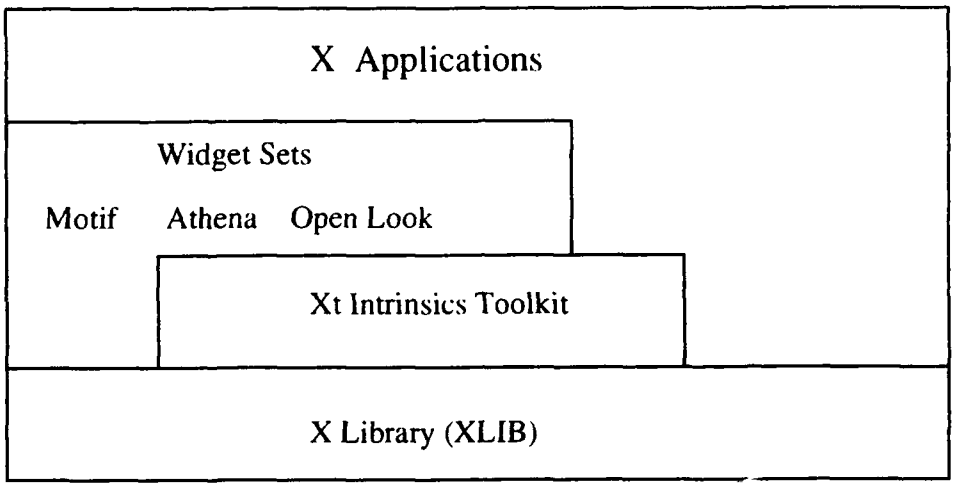

Figure 1. X Window System Hierarchy

The widget sets that comprise the layer above the Intrinsics level offer more advanced user interface components, each of which is normally comprised of several of the Xt widget classes. X application developers normally use these widget sets in their programming efforts, since they offer integrated components at an appropriate level of abstraction. If specific functionality is required but is not offered at a particular level, developers can access the layer which does provide this capability.

An important difference between X and many other window systems is that X does not define or restrict developers to any particular user interface style. In the words of its designers, X provides "mechanism without policy." The Xlib and Xt layers are standard because they can support any kind of user interface [NyOr92]. Thus, in addition to the portability inherent in this windowing environment, X also offers tremendous flexibility to application developers as well.

3. Providing Hypertext Capabilities

Many of the guidelines for creation of usable interfaces and intelligent tutoring systems are contradictory. Part of the problem exists because many of these rules are heuristic in nature, especially in the relatively new field of graphical interface design. One guideline that appears to be based predominantly on common sense and experience is the admonition to use real-world analogies as much as possible [HiHa93]. In order to follow this advice we must recognize what analogy we are trying to represent in the first place.

Trying to create the analogy of a human tutor has thus far consternated many tutorial developers, and is the essence of many of the problems with ITS's today. This is one of the primary considerations in our decision to develop a learning environment for the user. The possible interactions with a human tutor are potentially unlimited, and would be infeasible to implement with current technology and available research. A much simpler analogy, that of an instructional manual or book, has still eluded adequate emulation, despite it representing the most natural and familiar interface to users. This analogy appears to satisfy the criteria that the best interfaces are those that are natural for the task that they are designed to perform [Blum92]. While many tutoring systems claim to provide this functionality, in reality they fall far short of creating even primitive capabilities.

Rather than try to incorporate a myriad of tutoring styles into a single application in order to accommodate a diverse range of users, we shall instead try to model the paradigm of an instructional manual or tutorial that a user is interested in using. Although an author of any text book usually intends for the material to be read in the order presented, it would be preposterous to somehow restrict or force a reader to follow this sequence. A great deal of thought and design considerations went into the format and organization of the material to be presented. The order in which each topic is addressed is designed to provide the most logical flow of information to its reader, or the expected path to take. There are, however, absolutely no mechanisms for enforcing that every user follow this path. The antithesis of this, in fact, does exist. Readers are typically provided a table of contents which outlines major topics and subtopics, a list of tables and figures, and an extensive index. It would

seem that every effort is being made to encourage users not to read this manual sequentially, although in reality we know that its designers are merely trying (quite successfully) to increase the usefulness of this reference material. It is intuitively obvious that any reference material that provided less capability would be inadequate, restrictive, and would not be willingly used.

Similarly, the material presented in most tutorials is designed to be viewed in the same sequence as it was prepared, and mechanisms have always been available to do so. If additional mechanisms are not also provided to jump around, skim certain sections, or just get a feel for the depth of the material, then the user is effectively forced to follow a prescribed sequence. By providing the user with the control mechanism afforded through incorporation of hypertext capabilities it is possible to more closely model the instructional model paradigm.

Providing hypertext capabilities in the tutorial module requires relinquishing explicit control on the part of developers. The prepared and intended sequence of instruction has not changed, only the enforcement mechanisms. The degree of control that tutorial authors sacrifice is transferred to the users, thereby empowering them with complete access and control. If a user is exposed to information that contains related topics or ideas that are not completely familiar, it is possible to immediately "jump" to those areas and gain the necessary understanding. If the user is already comfortable with these concepts, the normal course of instruction is not unnecessarily interrupted by digression.

While the benefits of incorporating hypertext into the instructional material of a tutoring system have been established, we still need a mechanism which could provide these capabilities and not degrade our interface. Fortunately, the developers at the National Center for Supercomputer Applications (NCSA) have created a hypertext widget that can be used with the X Window System, known as the hypertext markup language (HTML) widget. This widget is a special purpose object that provides additional functionality not found in the widget sets that occupy the upper level of the X Window System hierarchy (see Figure 1). This is the same widget that provides the hypertext capabilities for Mosaic

and most world wide web (WWW) browsers. This widget allows us to provide users with the hypertext capabilities from within the graphical environment we have chosen for our development platform.

D. PORTABILITY

As mentioned in the discussion on design considerations, creating a usable interface often precludes maintaining portability. This is not the case, however, with the development platform which we have chosen. The X Window System is an industry-standard software system that allows programmers to develop portable graphical user interfaces. Furthermore, the hypertext widget developed at NCSA is written for the X system, which allows us to incorporate this additional capability without any sacrifice in portability.

The code for the HTML widget is written so that is may be used with either the Motif widget set or the Athena widget set. These widget sets represent the top layer of the X System hierarchy, as shown in Figure 1. Although Motif is emerging as the standard among X platforms (and provides a much more polished interface), it is proprietary software that is not included with the standard X distribution from MIT. The Motif widget set must be purchased separately from the Open System Foundation vendors themselves, and restrictions apply to applications developed using their libraries. Since the HTML widget can be used with the Athena widget set, we chose to exploit the availability of this set over the ease of programming and enhanced features offered by Motif (the Athena widget set is included with both Release 4 and 5 distributions from MIT).

The only major consideration remaining is the choice of the implementation language. Of the possible choices, the C programming language represents several distinct and significant advantages over other languages. C is widely available and a powerful programming language. In addition to being recognized as one of the best languages to use in implementing an application for several different machines, C offers excellent interfacing capabilities to external routines [Kear87].

Several other key design considerations also suggest that C be our implementation language. While X is not restricted to a single language or operating system, the most logical choice of programming languages is C. All of the low-level X library routines, called Xlib, are written in C. The Xt Intrinsics library, built upon Xlib (see Figure 1), which offers an object-oriented layer that supports the development of user-interface components, is also written in C. From the perspective of X application developers, however, the most important consideration may very well be the fact that numerous reference books which provide detailed examples and instruction are based on the C programming language.

An additional consideration is that one of the fundamental components of intelligent tutoring systems is their student modeling and evaluation component. Often this consists of an expert system shell or program which must be integrated into the tutorial. One popular expert system language that is often used for such applications is CLIPS, which is an acronym for the C Language Integrated Production System. CLIPS was designed at NASA/Johnson Space Center with the specific purposes of providing high portability, low cost, and easy integration with external systems. CLIPS was written using the C programming language to facilitate these objectives. [GiRi94]

Another feature which would greatly improve the effectiveness of any programming language tutorial is the incorporation of a syntax directed editor, which is a language-based program editor that can apply knowledge of a language's context-free syntax to ensure that programs are always syntactically correct. They can also be used to enforce structured programming techniques or a particular programming style. The Synthesizer Generator, [Reps89], is a system that is designed to facilitate the creation of these language based editors and other programming tools. The synthesizer generator is written in C and runs under the UNIX operating system.

Finally, the UNIX operating system (also written in C), provides several powerful programming utilities for application developers. Of particular interest are the two utilities, lex and yacc, which are tools for writing programs that transform structured input. They are capable of rapidly producing powerful lexical analyzers, parsers, and other input

handling programs. These tools automatically generate C program code, which can easily be integrated into separate applications or be used to create separately executable programs [LeMB92]. These tools are often used to perform the lexical analysis for code evaluators.

While all of these tools are capable of interfacing with other programming languages, they have all beer specifically designed to exploit the portability and excellent interfacing capabilities of C. It would therefore seem quite illogical to attempt to implement an integrated system in any other programming language.

E. EXTENSIBILITY

Our final design consideration is that of extensibility. It is not sufficient to create a system that meets all of the known requirements of a particular set of users. This would guarantee that the system is obsolete before the completion of testing. The system must be flexible enough to accommodate modifications without requiring extensive rework and redesign.

In order to provide this capability we have provided several mechanisms for expanding the system. Incorporating some of these mechanisms was actually necessitated by our decision to separate the instructional modules from the evaluation modules in order to provide non-linear access to the tutorial lessons. By relinquishing explicit control over the presentation of material we empowered our users, but forfeited access to standard modeling information. Traditional tutoring systems maintain information on which screens have been viewed by their users, and use this information in the construction of their student models. It is our contention that information on which screens have been viewed is irrelevant and misleading, and should not be used in the creation of a student model or in controlling the presentation of instructional material.

There are, however, other means of accurately measuring student understanding and mastery. One common method is to administer quizzes to the students, as is done in many programming classes. Another is to analyze small sections of code for syntactic and semantic correctness. We have provided a simple and straight forward mechanism which

allows tutorial developers to implement either or both of these capabilities. This mechanism also allows independent development and modification of these separate tools, which facilitates production of prototype systems which can be tested and evaluated. While these features are obviously desirable in many applications, they may not be necessary or cost-effective in others. Tutorial developers are free to develop applications without any of these capabilities, and merely provide the instructional material as an on-line reference to augment course instruction.

In addition to the mechanisms we have incorporated for system administrators and tutorial authors to extend the environment, we have also provided a means for end users to extend the environment as well. In addition to the standard configuration file that is used to dynamically create the menu entries for the system, the environment is also designed to accept a configuration file as its only command line argument. The format of this file is identical to that of the system configuration file, although it will only affect the current invocation of the program. Thus users may develop and incorporate their own on-line hypertext help and quick reference manuals for use within the environment.

IV. IMPLEMENTATION DETAILS

A. SYSTEM OVERVIEW

The intelligent learning environment has been designed to dynamically integrate diverse tutoring subsystems, on-line reference manuals, and hypertext-help systems into a cohesive learning system. The environment is intended to be used on a multi-user system, such as UNIX, and to provide system administrators a great deal of flexibility in setting up a directory and file-system structure necessary to support this tool. The integration of the various subsystems is accomplished through creation of a system control center from which users can launch individual applications. These applications are all built upon the same basic framework, which provides a consistent look and feel throughout the environment. Tutorial developers are freed from the burden of creating an effective interface mechanism, and can focus on the preparation of instructional material instead. Although the format for the hypertext markup language is fairly straightforward and easy to learn, numerous tools are available via the internet which can transform several different file formats into the HTML format.

B. INITIAL SETUP REQUIREMENTS

In order to provide this flexibility to system administrators, two configuration files are necessary for the proper execution of the system. The first file is necessary to establish the proper path and environment variable settings which allow multiple users to access the system. A UNIX environment variable, "ILEHOME", establishes the full path to the directory where the executable program and the second configuration file are located. While it is not strictly necessary that the program and configuration files be located in the same directory, it does simplify administration. The user's path must also be modified so that it looks for the executable program in the proper directory. A sample setup file is shown in Figure 2. The additional information contained in this example is to accommodate special external applications, as will be discussed in following sections.

```
# Sample setup file for Intelligent Learning Environment
# First section contains environment settings for system operation
setenv ILEHOME  /usr/local/iledir
set path = (/usr/local/iledir  $path)
# Following sections are for setting up special external applications that
# go with individual tutorials
# Setup for Ada tutorial
setenv XADAHELP  /usr/local/tutor/ada
set path = (/usr/local/utils  $path)
# Setup for Clips tutorial
set path = (/usr/work1/bin  $path)
```

Figure 2. ILE Setup File

Note that on UNIX systems the setup file can be centrally located and that users' ".cshrc" file can be modified to include a statement that reads this file when starting a session. This allows an administrator to completely change the hierarchical structure of the entire system without affecting every potential user. The single, centrally located file can be modified to reflect these changes, maintaining complete access for all users.

The second file, "ilemenu.dat", is the system configuration file. The format of this file is fairly rigid in order to simplify the parsing requirements. It is through this file that the system can be dynamically extended and modified. The format of each line of this file consists of the following semi-colon separated fields:

- Name of Application
- Type of Application
- Full path to directory where files are located
- Name of first file to load (Home Page)
- Name of auxiliary quiz or testing application
- Name of auxiliary code evaluator or syntax directed editor front end

The first entry is the name of the application and its appearance in this file is exactly how it will appear in one of the submenus of the system control center. Which specific

submenu it appears in is determined by the second field, whose valid entries consist of "TUTOR", "MANUAL", or "HELP". The next two fields designate the directory in which all of the application files are located as well as the first page which is to be loaded when the application is selected. The remaining two fields correspond to the names of separately executable programs. These fields are only applicable to applications of type "TUTOR", and provide the mechanism by which tutorial developers can incorporate student evaluation tools into the system. An example configuration file is shown in Figure 3. The actual system configuration file contains additional comments explaining the meaning and specific requirements of each field. This file must be named "ilemenu.dat" and be stored in the directory identified by the "ILEHOME" environment variable.

```
# Sample configuration file for the Intelligent Tutoring System
<Begin Data>
Ada Tutorial;TUTOR;/usr/local/tutors/itsAda;itsada00.htm;NONE;xAdaEval;
C Tutorial;TUTOR;/usr/local/tutors/itsCeedir;itscee00.htm;NONE;NONE;
C++ Tutorial;TUTOR;/usr/local/tutors/itsCxx;itscxx00.htm;NONE;NONE;
Ada LRM;MANUAL;/usr/local/manual/Ada9x/;rm9x_00.htm;NONE;NONE:
Using UNIX:HELP;/usr/local/pub/introCS/;jxxl_00.htm;NONE:NONE;
Clips Help;HELP;/usr/local/pub/clips/;clips00.htm;NONE;NONE;
```

Figure 3. Sample Configuration File

Note that because these special application programs identified in fields five and six are separately executable, the environment will most likely need to be modified in order to provide multiple users access. System administrators can easily modify the system setup file to include the information necessary for the proper execution of these external applications as well (see Figure 2).

C. CONTROL CENTER

In order to minimize the potential of overwhelming new users, a system menu is provided which serves as the control center for the environment (see Figure 4). Each of the

major categories of subsystems is represented on this menu, which should provide an intuitive interface to even novice users. Users can select the specific category they wish to explore, and are provided a pop-up menu which lists the available applications.

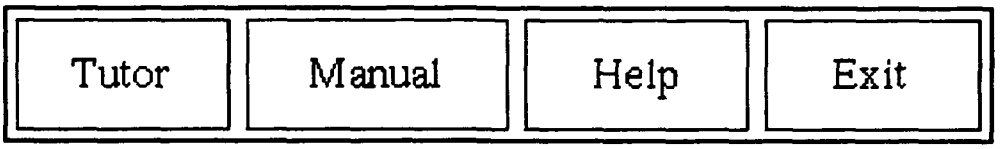

Figure 4. System Menu

In order to provide extensibility to our system, the topics that comprise each of the submenus are determined at run-time by parsing the configuration file, "ilemenu.dat", as discussed in the previous section. This file determines the basic characteristics of the system that all users will experience. Advanced users can further extend the selection of subtopics available through a command line configuration file. This file must be in the same format as the system configuration file, which can conveniently be used as a template. The configuration file used as a command line parameter only affects the current execution of the program, and provides power users with a means of extending the usefulness of the environment even further. For a detailed explanation of this process, see "EXTENDING THE APPLICATION BY THE USER" on page 53.

D. SUBSYSTEM OPERATION

Although the system menu provides selections for three distinct subsystem areas, operation of these subsystems is actually defined by two modes: a tutoring mode and a reference mode. The reference mode may be viewed simply as an on-line hypertext document browsing tool. The non-linear access provided through hypertext capabilities is particularly well suited to reference manuals and help systems. The tutoring subsystem provides basically the same capabilities as the reference section, and offers additional features not available in the reference mode. It is possible to have both a tutor and reference window open concurrently, effectively operating the system in both modes. Although this capability is allowed by the environment, users will normally find that most monitors are

best suited for single mode operation. Below we first discuss the capabilities common to both subsystems to provide the conceptual framework necessary to understand operation of the integrated environment.

1. Reference Mode

Choosing an option from either the manual or help submenus causes a new window to be created, displaying the home page of the specific application (as identified in the configuration file). This file will be an actual hypertext document, with the hypertext links indicated by being underlined. These links can be selected by using the mouse or other pointing device to place the cursor over one of these items and pressing the selection button. This causes the hypertext link to be executed, which loads the new topic into the window. In order to encourage exploration without causing users to get irretrievably lost we have provided two mechanisms for backtracking through the sequence of hypertext jumps. The "Back" button is a single-step backtrack, reloading the previous document and positioning the hypertext link that originally caused the jump at the top of the viewing window. If multiple links have been explored and a user has lost a sense of exactly where in the document he or she should be, the "Home" button clears the record of previous jumps and reloads the home page of the document.

Once the window has been created users may wish to replace the current material with another manual or help section. Rather than require them to close the window and restart the reference section, which may involve repositioning and resizing the window, we allow users to simply select the new material to be loaded. In order to minimize the overhead associated with maintaining the history list of hypertext jumps, once a user selects a new subtopic to load all of the backtracking information is cleared from memory, effectively restarting this section. This includes replacing the previous home page as well.

Note that this feature has been included in order to extend the usefulness of the application, and is not available from within the tutorial section. The manual and help subsystems have been designed to provide on-line hypertext access to reference material

that may be used externally to any of the tutoring systems, especially in subsequent programming endeavors.

2. Tutor Mode

We stated that the tutoring subsystem provides fundamentally the same operational capabilities as the reference mode, but with additional features. These additional capabilities, however, made it imperative that we preclude users from restarting a tutorial application without properly closing the currently open one first. This is to enable applications to properly save any student modeling information obtained and to close all open files. This should not be viewed as a design trade-off, since it would be unreasonable to open multiple tutorials or jump from one tutorial to another, a capability quite reasonable to expect of reference material.

The additional capabilities provided by the tutorial subsystem include a quiz application and a front end to a myriad of software engineering and testing tools, including code evaluators and syntax directed editors. These applications must be identified in the configuration file in order to be accessible from within a tutorial application. These features have been designed so that they are initiated as hypertext jumps from within the tutorial. Rather than load another hypertext document as is done with normal hypertext jumps, these special links cause the external applications to be loaded, passing the filename associated with the link as a command line parameter. This allows implementation of context-sensitive quiz evaluations and programming assignments.

It is important to note that tutorial developers are not required to incorporate these additional features in order to implement a prototype tutorial. We purposely separated the process of presenting the instructional material from the student evaluation process in order to provide this flexibility. Tutorial authors can easily transform existing course work or prepared instructional material into the HTML format needed to provide hypertext capabilities and focus on the quality and content of the instructional material without being distracted by concurrent development of quizzes, code evaluators, or other tools designed

to evaluate students and create elaborate student models. Once satisfied with the instructional material, tutorial authors can then focus their attention on these other tools. More importantly, this allows development teams to work independently on these projects as well. Once these external tools are properly tested and validated, they may be incorporated into the specific tutoring subsystem through modification of the ascii based configuration file.

E. ADDITIONAL FUNCTIONALITY

To facilitate the task of tutorial developers and to provide them with an additional feature that they can exploit, a default quiz application has been included. Authors may use the functionality of this tool in developing prototypes, or as a model in developing their own evaluators. We considered it important to include this capability since it provides an interrogative capability which can help develop the student's cognitive process. In addition to illustrating key points that may be typical sources of confusion, users are provided with explanations as to why certain responses may be correct or incorrect.

In order to use the default quiz, the corresponding entry in the configuration file must be "NONE". Any other entry will be interpreted as a desire to override the default. As indicated in the previous section, these settings only set up the capability within the learning environment. Actual calls to these applications are made through hypertext links with non-standard filename extensions. The tutor subsystem examines these hypertext requests, and determines whether the jump is a standard hypertext jump or a special flag. If the jump is special, it then must determine if a corresponding special application has been specified. If one has been specified, the application is started via a system call, using the filename as a command line argument. If an external application has not been specified, then the default quiz function will be called, and this time the filename is passed as the parameter.

The format of the quiz file is almost identical to that of a standard HTML file. The differences between the two types of jumps are illustrated in Figure 5.

The format code for a standard jump is:

<A HREF=#Anchor>Standard Jump</A>

which appears to the user as:

Standard Jump

The format for a quiz answer is:

<A HREF=##3_A>(A)</A> Response A<P>

which appears to the user as:

(A) Response A

Figure 5. Quiz Formatting Codes

The fundamental difference is that selections which designate user responses to the quiz questions must be distinguishable from standard hypertext jumps so that the application can tell whether the user is attempting to answer a question or jump to an alternative question. Note that to the user these links appear identical, and it is within the context with which they appear that distinguishes their purpose. The "##3_A" signals the quiz processor that this jump represents the user's choice for question 3 is A.

When the default quiz application is started it takes control over the system and opens a new hypertext window, loading the contents of the filename into the hypertext widget. In addition to the prepared quiz file, this function requires a corresponding solution file whose filename is identical to that of the quiz file, except that the "quiz" extension is replaced with a "soln" extension. Note that both of these files must be present in the tutorial application's directory in order for the quiz to work properly. Failure to locate and read either of these files will generate an error message and return control back to the tutorial.

The format of the solution file is shown below in Figure 6, and has been designed to simplify parsing while still providing explanatory comments for each possible response to quiz questions.

```
QUEST_1 = B
ANS_A
The explanation for answer A goes here, explaining why incorrect.
END_ANS
ANS_B
The justification for this being the correct answer goes here.
END_ANS
ANS_C
A message explaining why C is incorrect goes here.
END_ANS
END_QUEST
```

Figure 6. Quiz Solution Format

These messages appear in a popup dialog box in response to the user selecting a particular solution. Note that if the responses are too long to fit into the dialog box that scrolling capabilities will automatically be incorporated. This provides authors with the means of providing a complete and meaningful explanation.

The solution file can contain up to ten questions, and each question may have up to four possible alternatives. It is important for the solutions to match precisely with the questions, not only to ensure the accuracy of the responses to the students, but also to preclude any unpredictable responses from the system. The results of attempting to retrieve the solution to a non-existent question are undefined.

F. EXTENDING THE LEARNING ENVIRONMENT

We have already seen how easy it is to extend the environment from the perspective of system administrators and tutorial developers. A brief summary of this process is included here to ensure complete understanding of this concept. The key to extending the environment for all users is the file, "ilemenu.dat". This file contains the menu entries for all of the major subtopics of the system. While the format of this file is fairly rigid, it is easy to install additional applications by simply following the examples provided.

In addition to the capability of adding new tutorials, manuals, and help materials, the capabilities of tutorials already incorporated into the environment can be modified as

well. The last two fields of the configuration file facilitate this process. They correspond to external applications that can be invoked through special hypertext links from within the tutorial documents. This provides context-sensitive quiz and program evaluation capabilities to tutorial developers. As an added feature, authors can use the default quiz application which should expedite the prototype development process.

Finally, users may extend the functionality of the environment for their own personal use by invoking the system with a configuration file as a command line parameter. This allows users to develop their own personal help files, quick reference files, and other program development tools which they may use in future efforts. An example of a typical use of this capability would be in the development of an example code or data structures help reference. Through the clipboard capabilities afforded by the X Window System, users could then cut and paste these examples into their own code, thus ensuring syntactic correctness and improving efficiency and productivity.

G. CROSS PLATFORM CAPABILITY

The intelligent learning environment has been designed to work on any system capable of operating the X Window System. The ILE has been compiled and tested on UNIX workstations as well as on a personal computer. The majority of the development work was actually completed on an Intel 80486 based personal computer running the Linux operating system. This operating system is a freely distributed UNIX clone that is available via the Internet. Since the ILE makes only standard C and X Window function calls, any system capable of X Window System development should be capable of compiling and operating the environment.

V. CREATING AN ADA TUTOR

One of the stated benefits of the intelligent learning environment is the ease with which tutorial developers can transform their prepared instructional material into the appropriate format to be used with our system. In order to substantiate this claim we have prepared an Ada tutorial application. The instructional material was obtained from the ITS Ada Tutorial developed by DeLooze [Delo91]. This material consisted primarily of screen presentations, with the format each screen designed to fit on a standard 80 x 25 (column by row) display. In addition to the presentation of instructional material, the original work also included several pseudo programming assignments, which attempted to evaluate the student's current level of understanding. The evaluation process had direct influence over the presentation of instructional material by requiring students to successfully complete the programming assignments before allowing them to view further material.

One of the criticisms of this tutorial is that the evaluation of student's solutions consisted of direct comparison to an "expert's" solution, and that a single specific response was the only correct answer. An alternative method offered by Hoppe is to incorporate an expert system shell which can then be used to evaluate the code. He developed the Clips Ada Evaluator (CAE), an expert system shell designed to enhance the evaluation capabilities of the ITS Ada system. [Hopp92]

A. DEVELOPING A PROTOTYPE TUTORIAL

The first step in the transformation process is to modify the existing instructional material so that it is in the proper HTML format. Since the original material was in ascii format, we were able to experiment with one of the tools available which automatically transforms ascii files into HTML files. While this tool did accomplish the task as advertised, a great deal of work was still required in order to clean up the presentation and properly interconnect our files in the exact manner we desired. Since the task of transforming ascii files into HTML format is so straightforward, we decided to abandon the tool and do most of the work by hand. An example of the formatting commands needed to

transform a text file into the hypertext format is shown below. Figure 7 shows a small section of a lesson as it would appear in a standard ascii file. Figure 8 shows this same information with the embedded HTML formatting codes. Note that this file is still an ascii file, and can be edited and updated with a standard text editor.

```
Lesson 8-1 Record Types

We have seen that by using arrays we can describe data objects that contain many elements.
One limitation of arrays is that all of the components of an array must be the same type.
Array types cannot be used to represent real world objects that are made up of different
types. For this capability, we need what is known as a record type. A record provides the
means of encapsulating different types of related information into a single logical unit.
The declaration of a record type is as follows:
type Person is record
    First : string(1..15);
    Last  : string(1.15);
    Age   : integer;
    Wage : float;
end record;
```

Figure 7. Sample Lesson in Ascii Format

```
<H2>Lesson 8-1 Record Types</H2><P>
<P>We have seen that by using arrays we can describe data objects that contain many
elements. One limitation of arrays is that all of the components of an array must be the same
type. Array types cannot be used to represent real world objects that are made up of
different types. For this capability, we need what is known as a record type. A record
provides the means of encapsulating different types of related information into a single
logical unit.<P>
<P>The declaration of a record type is as follows:
<pre>
type Person is record
    First   : string(1..15);
    Last    : string(1.15);
    Age     : integer;
    Wage    : float;
end record; </pre>
```

Figure 8. Sample Lesson in HTML Format

Figure 9 shows how this formatted section of a lesson would actually appear to the user. Note the distinct appearance provided by the formatting commands for the various parts of this document. Although a discussion of the HTML formatting commands is beyond the scope of this thesis, additional information is provided in the system help section of the learning environment.

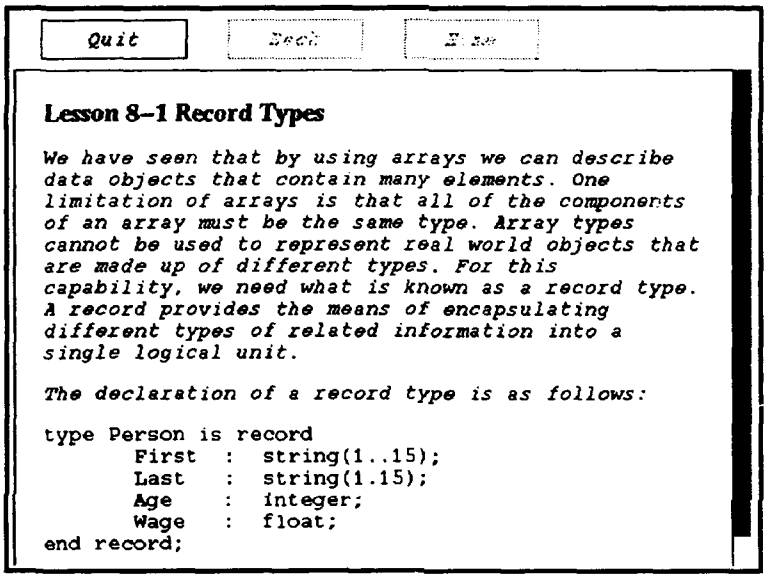

Figure 9. User's View of Sample Lesson

Once all of the files were in the HTML format, we then needed to modify the lessons to exploit the new screen presentation capabilities. While the previous format mandated that all pertinent information appear on a single screen, the scrolling capabilities provided by the X Window System provided much more flexibility. Several examples could be provided to illustrate each point, and sufficient comments could be added to explain key aspects of syntax and programming issues.

Upon completion of this task, we had a working prototype for the tutorial. While we still did not have the student evaluation mechanisms in place, the basic instructional material was in place and ready to go.

B. ADDING CODE EVALUATION MECHANISM

In order to simplify the development process, our initial intention was to incorporate the work that Hoppe had done in extending DeLooze's ITS Ada tutorial [Hopp92]. The potential benefits of this code-reuse was the primary motivating factor in this decision. Although the promise of time-savings through code-reuse was extremely attractive, we also recognized that this would still represent considerable effort. The expert system shell used in the original work, CLIPS version 5.0, would need to be modified to work with version 6.0, in part to replace obsolete features, but more importantly to exploit the five major enhancements of this newer version [Nasa93a].

Another significant modification required involved restructuring of the interface. One of the major shortcomings of the CAE is that it does not present the problem statement to the user. Once users start the program, they must remember which problem they wish to solve, as well as the specifics of the problem itself. We decided that this is an unnecessary burden to place upon users, especially when considering the level of detail and the significance of the specification of the problem statement. Furthermore, since the purpose of developing the intelligent learning environment using the X Window System is to exploit the graphical user interface components it provides, developing a subsystem for this environment that interacted with the user but did not use these capabilities would produce an inconsistent interface and would be a disservice to our users. The effort required to produce a simple, front-end interface to our code evaluators is minimal, and dramatically increases the flexibility of this subsystem.

1. Evaluator Interface

The evaluator interface is a stand-alone X Window application. It requires a single command line parameter, which designates the filename of the problem statement to be used. The current implementation of this program has been designed for single problem solving sessions. Users may make multiple attempts at solving this problem during a single

session, but may not load another problem once started. The problem statement is read in from the file, and displayed in the top section of a divided window (see Figure 10).

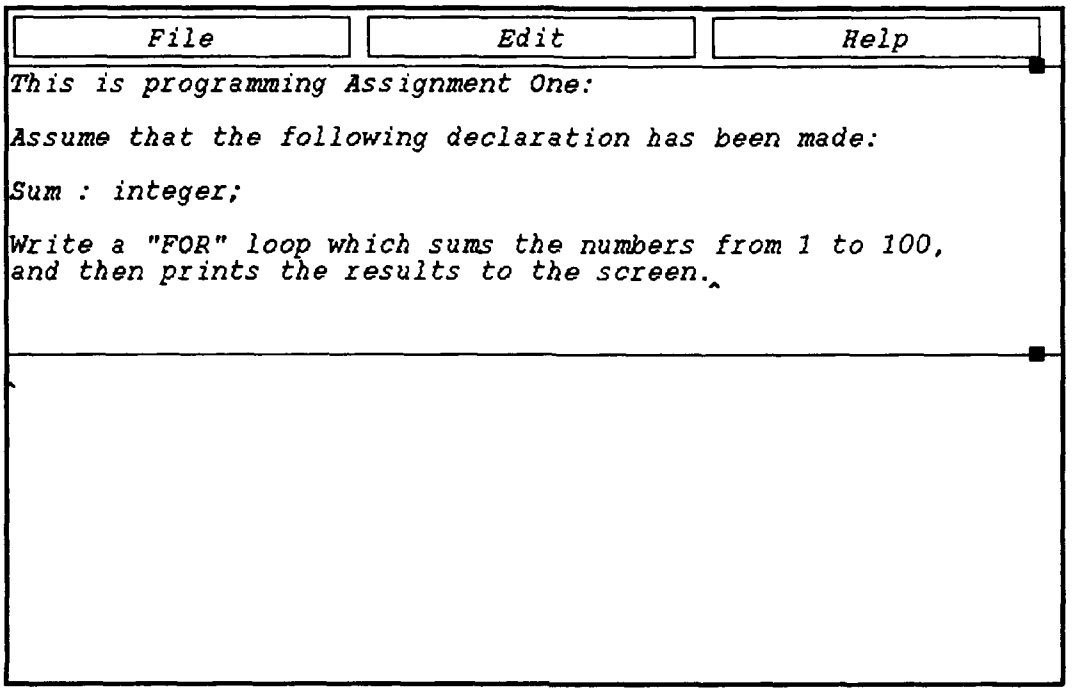

Figure 10. Initial Screen of Evaluator Interface

The "Edit" menu allows users to complete the assignment in an editor they are most familiar with out of one of the following three editors: Vi, Emacs, and Xedit. While they are entering their solution the problem statement remains accessible, and the window can be moved or resized to accommodate user's preferences. Once the user has completed and saved the assignment, then the evaluator selection of the "File" menu is enabled. Selecting this item causes the user's solution to be passed to the actual expert system subshell for evaluation, and the results of the analysis are displayed in the lower section of the divided window (see Figure 11). Users may then review the analysis, and reiterate the process as often as they desire.

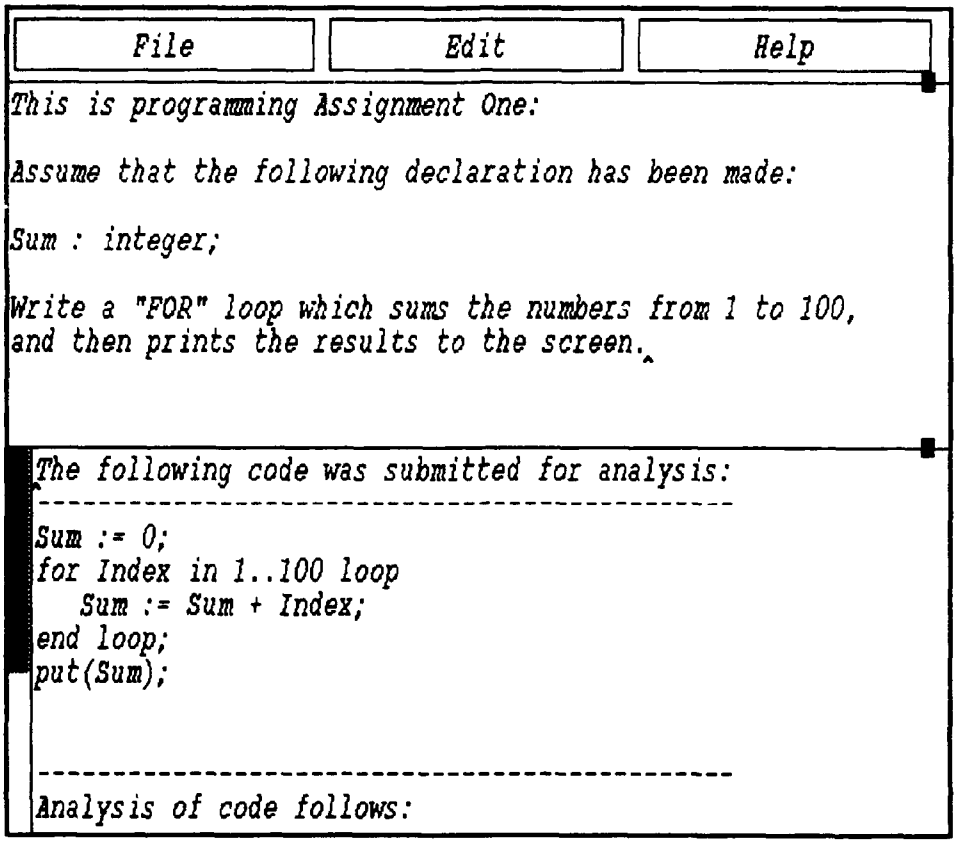

Figure 11. Analysis of User's Solution

Utilizing a front-end interface to the code evaluator offers distinct advantages in addition to providing a consistent interface throughout the learning environment. As we have demonstrated here, a prototype can be easily developed to ensure proper operation of the system. The actual expert system shell used in the evaluation of the user's code can be developed and tested independently, and can be modified or replaced without affecting operation of the rest of the system. For instance, during the validation process it may become evident that the forward chaining mechanism provided by the Clips inference engine is inadequate and that a backward chaining mechanism is more appropriate to doing this kind of analysis. The entire expert system shell can then be replaced by another tool that directly implements backward chaining within its inference engine, such as those based on the PROLOG programming language [GiRi94].

2. Expert System Code Evaluator

We began testing and validation of the CAE after successfully converting the original code into Clips version 6.0 compliant code. During this phase we discovered that the principle limitation cited with the original version of ITS Ada, that it could not properly identify multiple correct solutions to a given problem, was a limitation of the CAE as well [Hopp92]. Although both of these programs attempted to sufficiently restrict the scope of the problem domain in order to reduce the number of possible variations, neither of them could account for all of the possible solutions that remained. This can be attributed to the inherent ambiguity of the language used in the specification (English), as well as the expressiveness typically provided by higher order programming languages. These factors are directly responsible for the emergence of the software testing field as a separate discipline within computer science.

Upon closer inspection of the problem statements, however, it was discovered that these assignments did not actually represent programming assignments at all. The scope of the problem domain had been reduced so drastically that it had become an exercise in creating a syntactically correct programming construct. These problems are more suitable to quiz questions, where a single, most correct answer can be identified from a set of possible solutions. The typical user misconceptions that have been documented in bug catalogs can be used as distractors, which when combined with the explanatory capabilities provided in the quiz system would greatly enhance the learning process.

In view of these severe limitations we have decided to leave the evaluation portion of our tutorial implementation in its prototype form, which simulates evaluation of student code. We have converted all of the "programming" assignments into quiz questions and have incorporated them into the instructional material. Development of the code evaluation subsystem of this tutorial can continue independently, a built-in feature of the intelligent learning environment. Follow-on work can focus on the specific problem of code analysis, since the interface mechanisms have already been completed.

VI. A DETAILED EXAMPLE

A. SAMPLE USER SESSION

The intelligent learning environment is typically started from the command line prompt. The user is provided a greeting message and the control center menu, as shown in Figure 12. The purpose of the greeting is to advise users that several of the tutorial applications may need to save information into the user's directory, and that it would be best to have a specific working subdirectory created for this purpose. Although this is not necessary to run the program, it does prevent the user's home directory from becoming unnecessarily cluttered, and helps prevent different tutorial applications from interacting with each other's files.

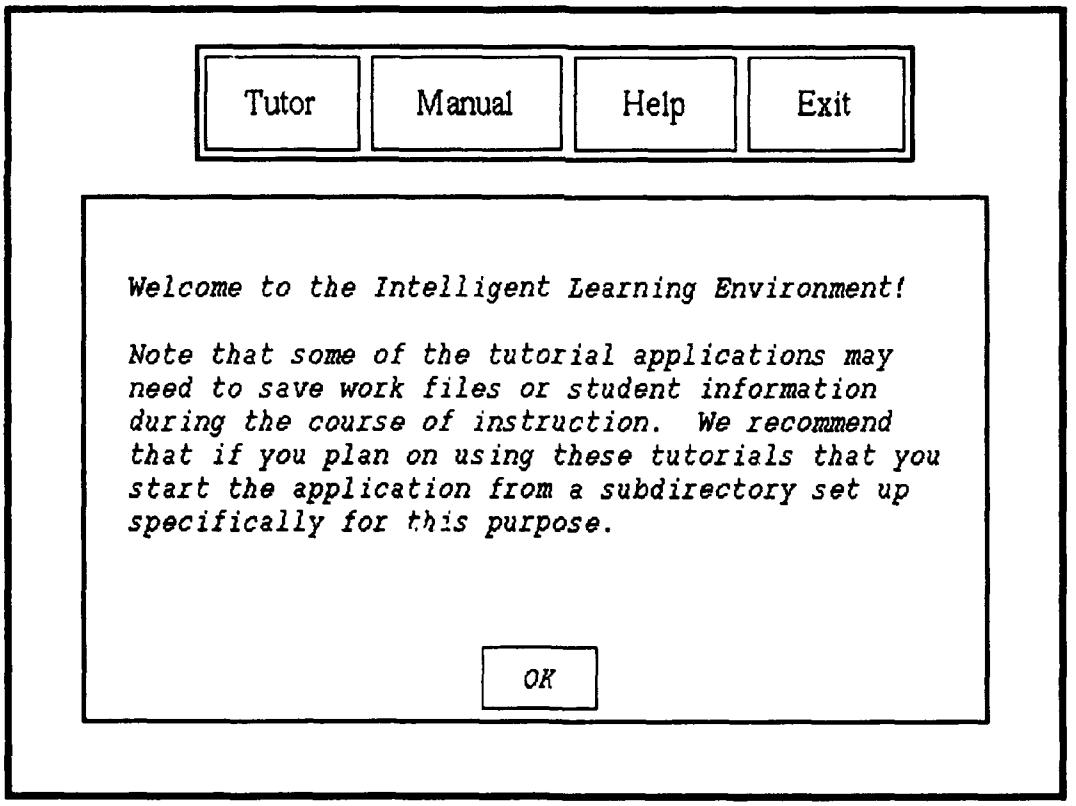

Figure 12. Opening Screen

The menu entries for each of the subsystems are contained in popup menus which are activated by selecting one of the control center buttons. The popup menu associated with the "Tutor" button appears in Figure 13. Remember that the submenu entries are dynamically determined from the global configuration file, "ilemenu.dat". These menus are spring-loaded, and have been designed so that moving the cursor away from the menu closes the menu. This helps prevent novice users from inadvertently starting an unwanted application.

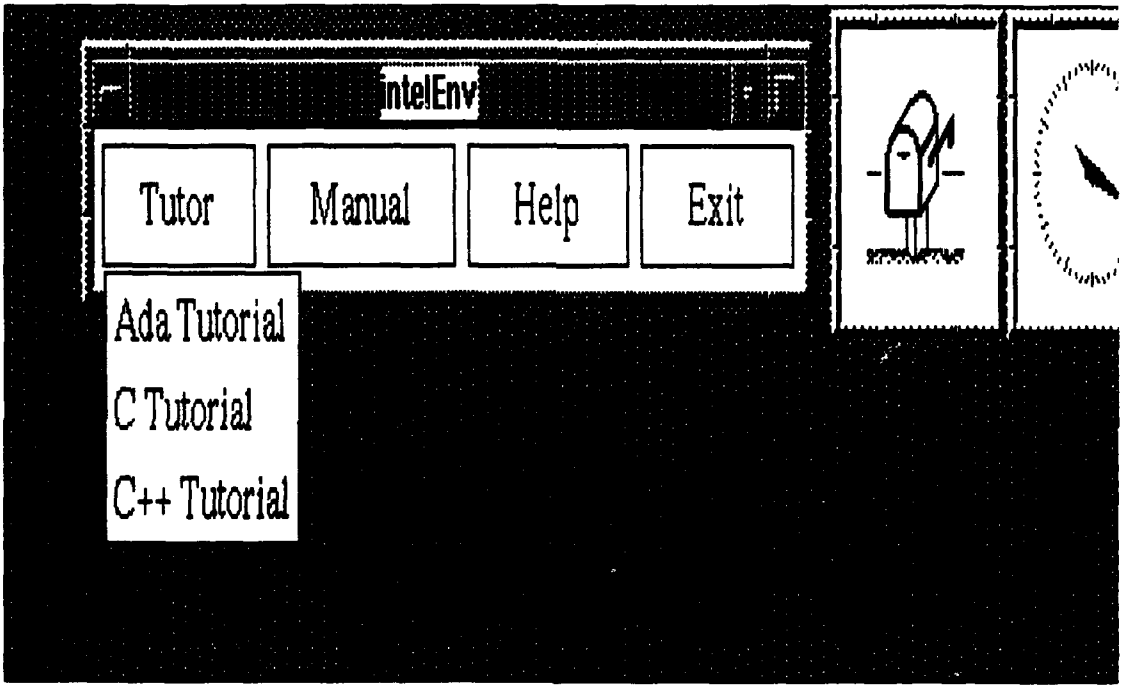

Figure 13. Subsystem Menus

1. Selecting the Ada Tutorial

Selecting a tutorial application opens the window for the application and loads the first page of instruction (see Figure 14). Note that the Tutor selection from the main menu is no longer available while a tutorial application is running. Users may simultaneously open both a help window and a tutor window, although typically only one is expected to be necessary.

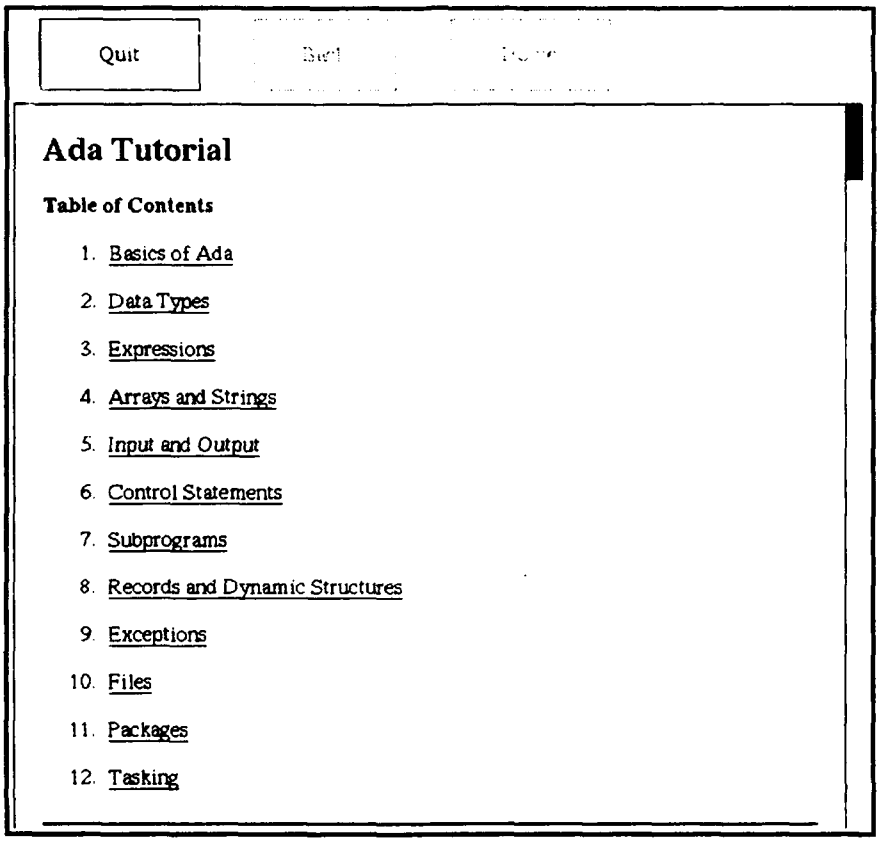

Figure 14. Starting the Ada Tutorial

The results of selecting the "Ada Tutorial" menu item for the Tutor menu is shown in Figure 14. The underlined entries represent hypertext links, and are selected by depressing the mouse button while the cursor is over any portion of these underlined entries. The buttons at the top of the screen are navigational control mechanisms provided in addition to the hypertext capabilities and the scrollbar positioned on the right side of the window. The grayed or dimmed appearance of the "Back" and "Home" buttons indicate that these selections are not currently available. This is because these provide backtracking mechanisms to the user, and no jumps have been taken to backtrack from.

Once the user has selected a hypertext link, the appearance of these button will automatically change, signalling the user that these capabilities have become available.

Figure 15 shows the results of the user selecting the "Basics of Ada" hypertext link from the table of contents, as well as the appearance of the enabled backtracking buttons.

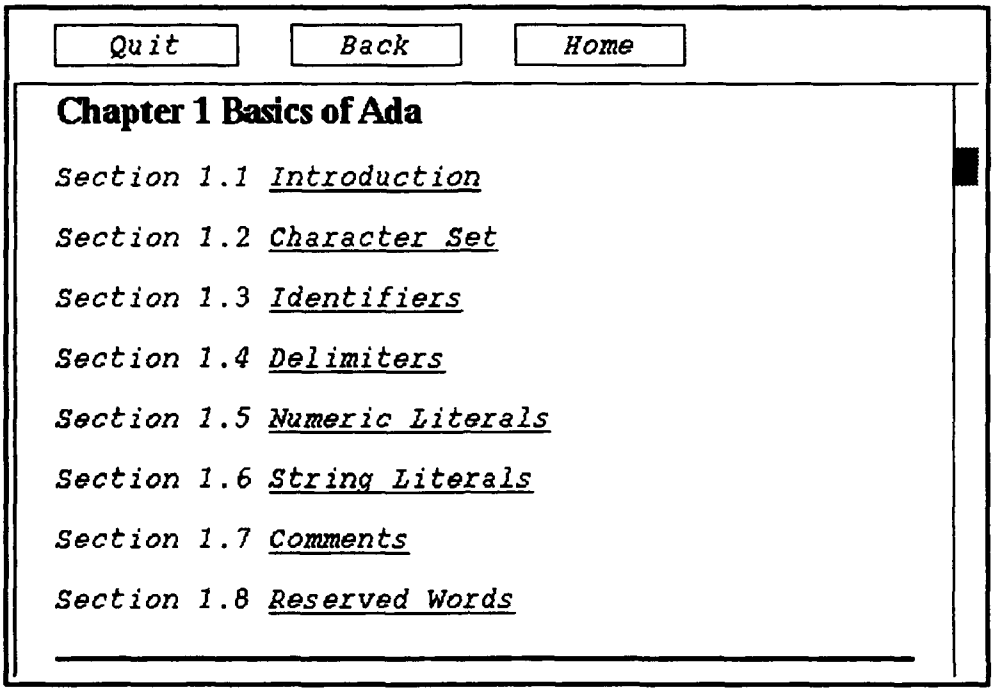

Figure 15. Results of Hypertext Jump

The previous two listings provide a glimpse of the degree of control provided to users. Each of the major topics has an expanded subtopic listing, allowing users to immediately proceed to the specific topic they are most interested in viewing. This allows users already using the tutorial to return to the exact lesson where they left off. Figure 16 shows the results of selecting a specific subtopic from this listing, as well as the general appearance of the tutor lessons. The user can resize the window to the appropriate dimensions that provides the most readable presentation for the particular monitor and screen layout.

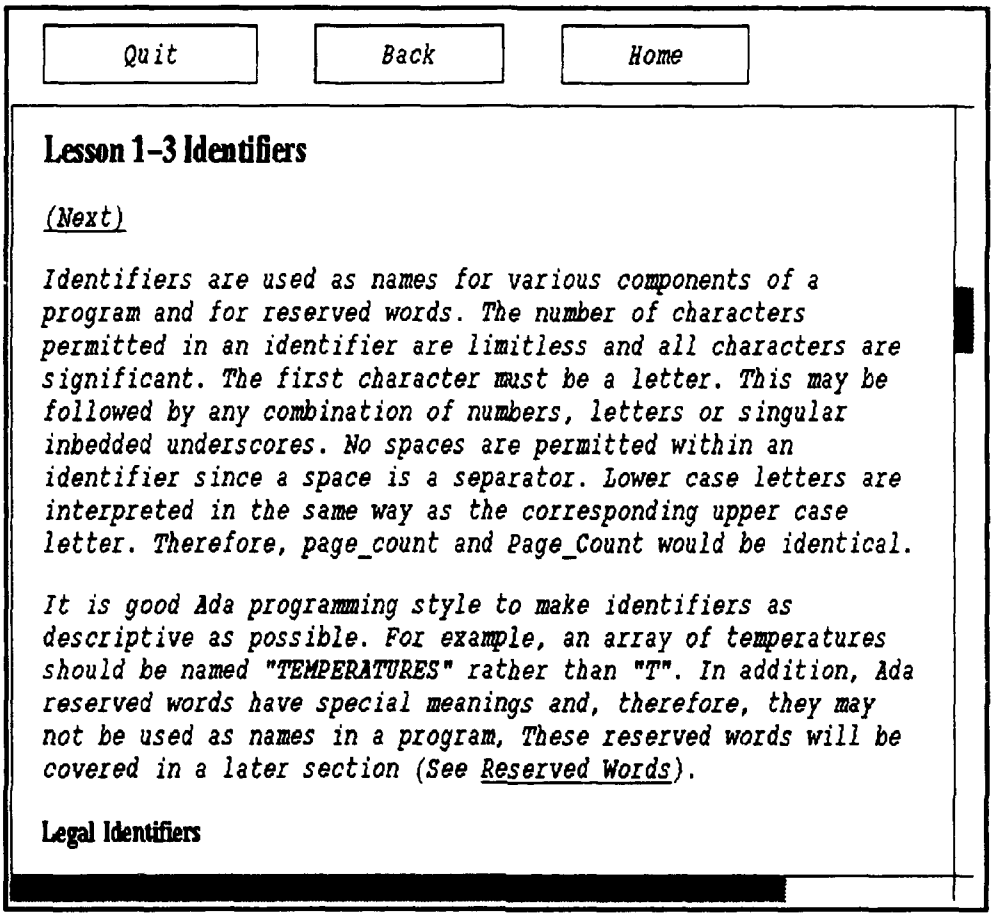

Figure 16. Selecting a Specific Topic

Although the scrollbar located on the right side of the window represents the normal method of navigating through the lessons, Figure 16 illustrates several other control mechanisms that are available. There are two hypertext links that can be executed from this view, one at the top of the screen labelled "Next", and the other located near the bottom labelled "See Reserved Words". The backtracking capabilities can be utilized through selection of the push buttons located at the top of the window.

The results of selecting the "Home" button from this or any subsequent window will cause the information contained in Figure 14 to be reloaded, effectively restarting the application. This would also clear the history list of jumps, and would disable both the

"Home" and "Back" buttons. Selecting the "Back" button, however, places the hypertext link that was last executed at the top of the viewing area, as shown in Figure 17. Note that this button remains enabled while backtracking until the history list is exhausted.

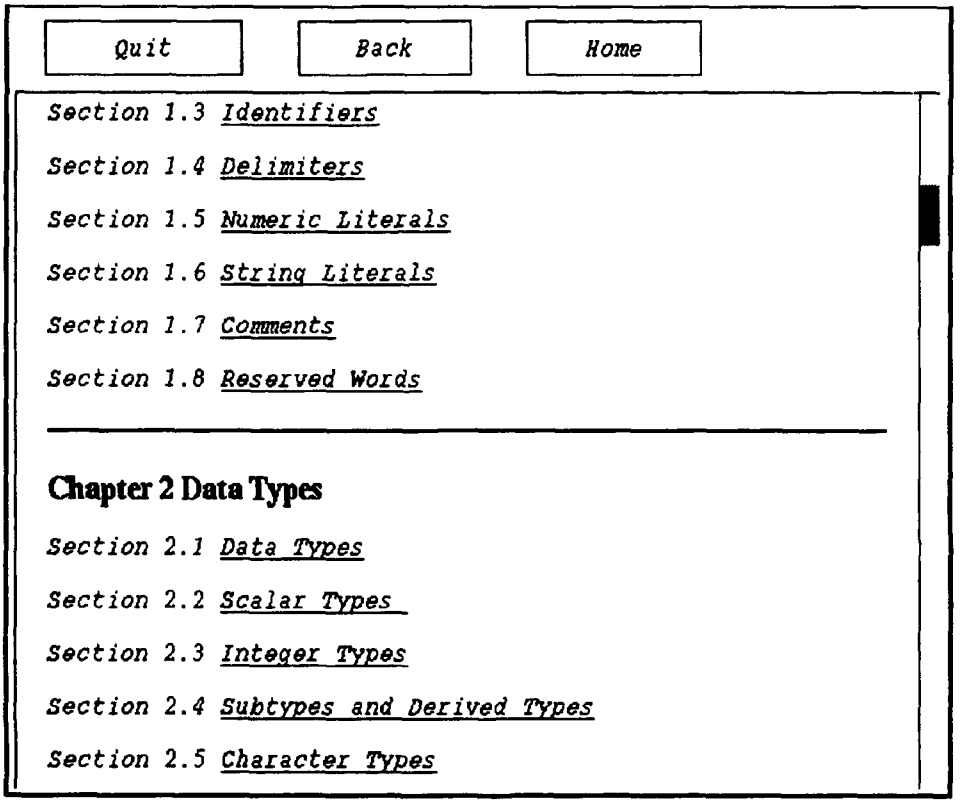

Figure 17. Selecting the Back Button

The hypertext link named "Next" that appears at the beginning of every lesson (see Figure 16) will cause the contents of the following lesson to be displayed. This has been incorporated to provide browsing capabilities to the users. While this link enables users to sequentially skip through the lessons, a far more useful non-linear access method is also provided. This method allows immediate access to related topics that are referenced from within a particular lesson. An example of this is provided by the link labelled "See Reserved Words" (see Figure 16), which causes the lesson containing that specific topic to be displayed, as shown in Figure 18.

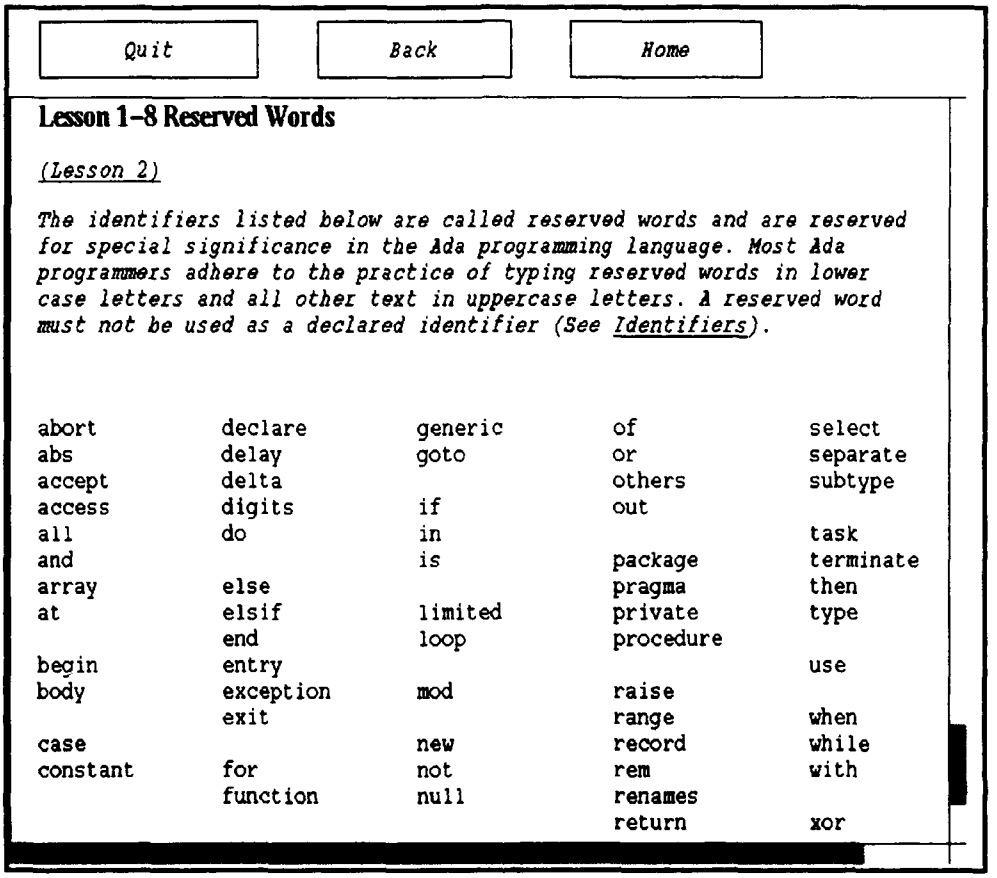

Figure 18. Jumping to Related Topic

Often the related topic will contain a hypertext link to the original document. This is precisely the case for the example provided. The lesson on identifiers referenced the lesson on reserved words, since the lexical requirements of the programming language precluded using reserved words as identifiers. It is intuitively appealing that the lesson on reserved words would also reference this rule. Rather than unnecessarily digressing and interrupting the flow of either of the lessons, the user is provided a means of accessing this information on his or her own when it is necessary.

Once a user reaches the end of a particular chapter several options become available (see Figure 19). The last two entries perform as one would expect, replacing the current document with either the table of contents or with the lesson material for chapter two.

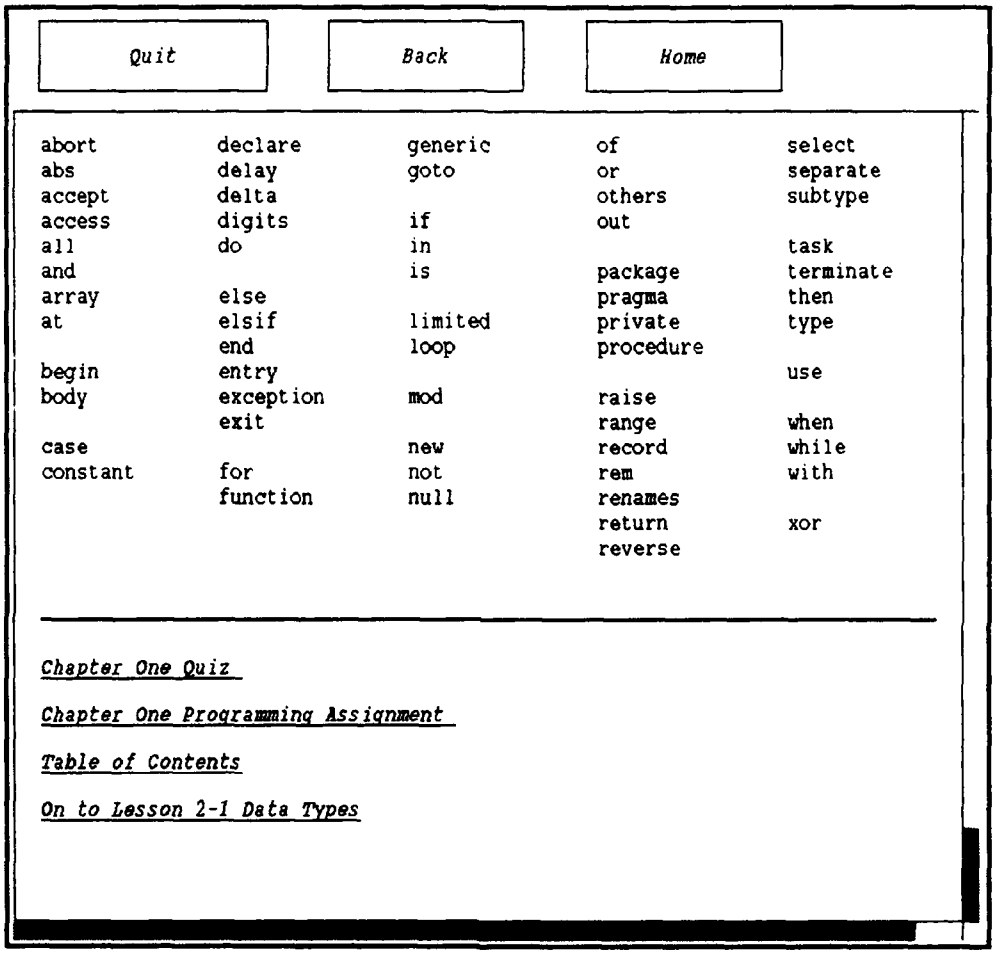

Figure 19. Quiz and Program Options

The first two entries, however, are what we have been referring to as special links. Selecting the first entry will cause a quiz application to be started. If the configuration file contains an entry in the quiz field other than "NONE", the name of the program in that field will be started; otherwise the default quiz will be started. Since their is no default code

evaluator provided, the second entry will either start the program specified in the configuration file or ignore the request.

2. Activating Special Links

Since no external quiz has been specified, selecting the quiz entry causes the default quiz application to be started, and its appearance is shown in Figure 20. Note that the answers to the quiz questions appear to the user as hypertext links, but their meaning is obvious from the context in which they appear.

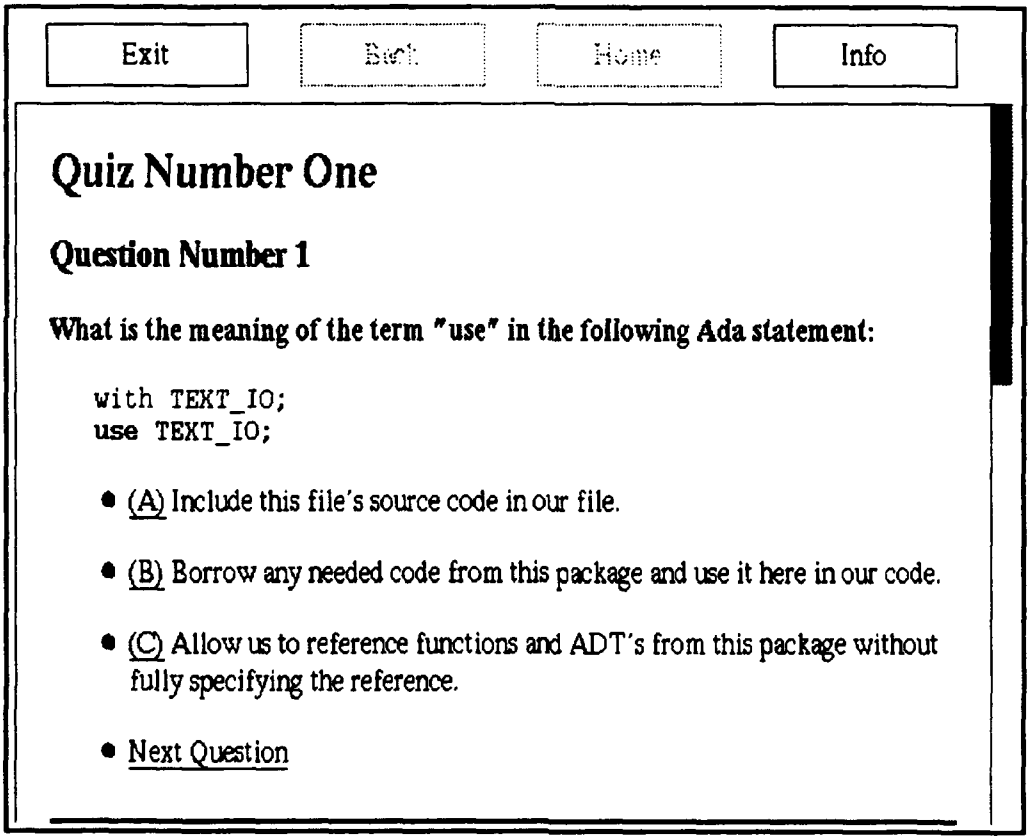

Figure 20. Activating the Default Quiz

In a similar manner, selecting a programming link would launch the code evaluator interface specified in the configuration file. The appearance of this application's main

window is shown in Figure 21. A complete description of this subsystem is contained in the previous chapter.

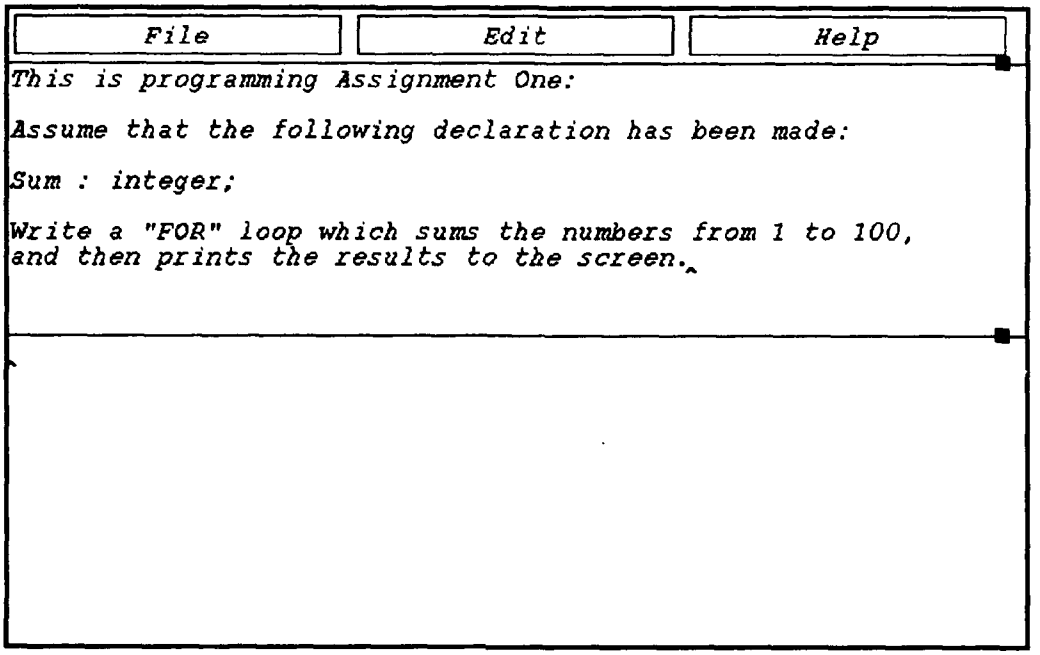

Figure 21. Activating the Code Evaluator

B. EXTENDING THE APPLICATION

The key to universally extending the application is the configuration file located in the directory identified by the environment variable "ILEHOME". Modification of this file will normally be done by system administrators, since they are responsible for maintaining centrally located files and subdirectories. This also provides them the flexibility to modify directory structures and relocate entire hierarchical structures as they see fit, especially as new applications are developed. A typical directory consists of several subdirectories branching from a central tutorial directory, as described in the implementation section. If an artificial intelligence instructor decides to develop a tutorial for Prolog, Lisp, or Clips that may not currently exist, a means of incorporating it into the ILE is to simply modify

the configuration file to include the necessary information. This would modify the system for all users, since this file determines the basic setup. If the instructor wished to restrict access to only those students currently enrolled in the class, the tutorial could be placed in a class account and then the students could modify their personal configuration file as described in the next section.

Assuming that universal access is desired, here is how the system could be extended. Suppose that files are located in the following directory: "/tmp/local/cs0101", that the home page is "introToCS.htm", and that no special quiz, code evaluators, or other extensions have been developed thus far. Then the configuration file would simply need the following entry appended to it:

Intro to CS;TUTOR;/tmp/local/cs0101/;introToCS.htm;NONE;NONE;

This would cause "Intro to CS" to appear in the "Tutor" submenu of the control center. Instructions for making entries in the configuration file are provided in the comment section of the configuration file, although the previous entries necessary for the current setup can serve as an example when adding new applications.

C. EXTENDING THE APPLICATION BY THE USER

It is relatively easy for advanced users to extend the functionality of the learning environment by adding their own help and reference material. The system is set up to accept a command-line configuration file that any user can provide. The format for this file is identical to that of the main configuration file, which may be copied and modified. An example is also provided in the general help section of the environment which can be cut and pasted into a text editor of the users choosing. Since this configuration file is passed as a command-line argument, one obvious requirement is that the file must either be located in the current directory, or else the path must be fully specified when calling the program.

In order to provide the hypertext capabilities for the user's own help and reference material a basic understanding of HTML is required. Again, since the goal of this system is to empower its users, a basic introduction and description of the language is included in

the basic help section. This will allow users without such knowledge the same opportunity as is provided users already familiar with the language.

It would be reasonable for an individual taking a programming language course or a data-structures course to desire a convenient mechanism to access syntax and code examples for class programming assignments. The user could create an ascii file containing the example code and data structures, and then insert the necessary control information for the hypertext widget. Alternatively, the user may have access to a help file that another student has created. Simply by creating a configuration file similar to that shown in Figure 22, the user will have created a quick reference hypertext help file that can be accessed from within the learning environment.

```
# Sample configuration file for the Intelligent Learning Environment
# Note that spaces are significant and preserved
# Format of semi-colon separated fields is as follows:
# Name;Type;Full Path;Home Page;Special App1;Special App2;
# The last two fields apply to tutorials only, and should be "NONE" for all others
# Second Field must be one of: "TUTOR", "MANUAL", or "HELP".
# Example:  Ada Tutor;TUTOR;/root/ITSENV/adaDir/;contents.htm;NONE;NONE;
Unix Help;HELP;/users/work1/delani/helpDir/;Unixhelp.htm;NONE;NONE;
X Windows Help;HELP;/users/work1/delani/helpDir/;xwinHelp.htm;NONE;NONE;
Data Structures;HELP;/users/work1/delani/cs3300/;datastruct.htm;NONE;NONE;
```

Figure 22. Sample User Configuration

D. ADDING SOFTWARE ENGINEERING TOOLS

As mentioned in the section describing the detailed implementation, hypertext links typically have an "htm" or "html" extension. This is not required, however, and developers can exploit the built in mechanisms by following alternative conventions. The system is currently set up to identify sound files, bitmap or image files, and postscript files as being special files and defers processing or these files to the system. It also recognizes filename extensions that begin with 'q' or 'y' as being hypertext links that require special handling

as well. The first field after the home page in the configuration file indicates the special quiz handler. If "NONE" is specified, then the system assumes that either a simple hypertext quiz exists or that the tutorial designer had no intention of using quizzes. If, however, a special program is indicated, then this program will be called and the information contained in the hypertext link will be passed to the program. This way developers who wish to create elaborate student models that track student's responses may do so.

Similarly, the next field indicates any special code evaluator programs that may exist for the tutorial. This could include an elaborate front-end interface which provides syntax directed editing capabilities, special formatting capabilities, or any other special preparation that the particular code evaluators may need. For instance, earlier versions of the ITS Ada tutorial included an expert system shell which processed certain programming assignments. By specifying the name of the program in this field, this separate application is started, and the information contained in the hypertext link is passed as a command line argument. This information may indicate a particular problem assignment or other information necessary to intelligently begin this program. User input can then be obtained, passed to the evaluator portion, and feedback provided to the student. Once the student is completed with this section control can be returned to the learning environment.

Although these mechanisms have been set up to facilitate the incorporation of quiz and code evaluation tools, there is nothing restricting their use to these functions. In the discussion of the implementation of the Ada tutorial we showed how we incorporated a front-end interface to a code evaluation system. As tutorial developers, we can change the purpose of this system to whatever we desire. If we only want to provide access to a syntax directed editor, we can either use the editor as our interface, or keep the level of indirection already provided and let this front-end serve as an integration mechanism to the syntax directed editor. Having this integration mechanism also allows us to incorporate other software engineering and software testing tools as well. This would be especially valuable for those tools which are not designed to interact with users. The interface could do all of

the pre-processing necessary for the tool, and then interpret the feedback and prepare it for the user to review.

Note that the enhanced functionality of these tools is not necessary to provide a useful tutorial. We have merely provided the mechanisms to enable tutorial developers to expand the power and usefulness of their own applications. These evaluators may also remain completely separate from the tutoring environment, allowing developers to concentrate on their own work and not on providing multiple interfaces to their students. This is exactly what we have done with our code evaluator subsystem. Until it has been properly tested and validated it should not be a part of the learning environment. Once we are satisfied with its quality, we can easily incorporate it into the system by modifying the configuration files.

VII. CONCLUSIONS

A. ACCOMPLISHMENTS

We have developed an integrated, extensible, intelligent learning environment designed to be used on the X Window System. We used this platform to exploit the rich graphical user interface it provides as well as its hardware and operating system independence. This learning environment provides the framework for producing hypertext based tutorials and reference materials without requiring explicit programming skills in any language.

We have incorporated a tutorial for the Ada programming language to exemplify the ease with which tutorials can be developed from existing course work. This tutorial also provides a template for tutorial developers to facilitate rapid development of prototype tutorial systems.

We have provided a means of modifying existing tutorials, enhancing their functionality and usefulness, or replacing them altogether without requiring modification to the learning environment or recompilation. Expert system code evaluators and syntax directed editors may developed separately from the tutorial, and then integrated once adequate testing has been performed.

We have created a mechanism for incorporating additional on-line reference manuals and help systems that can exploit the advantages of the non-linear access provided by the hypertext facility.

We have provided users with a tool that can be used in many different ways. In addition to using the environment for its tutorial applications, users can also exploit the functionality of the help and reference sections to facilitate program development for software projects. Users are free to utilize the editors they are most familiar with, and can even cut and paste template or example source code from reference material into their own work, greatly improving efficiency and productivity.

We have also provided users with a means of extending the power and usefulness of the learning environment as their programming prowess develops. Users can easily create their own help and reference sections (or use those created by colleagues or associates), and can incorporate the material into the learning environment by preparing a personal configuration file and passing the name of the configuration file as a command line parameter.

B. FUTURE WORK

We have mentioned the possibility of incorporating syntax directed editors throughout this discussion. This is one capability which we feel would add tremendous value to any programming language tutorial. In addition to guaranteeing syntactically correct structures, it can also be used to teach or enforce particular programming styles. The scope of this additional effort would have to be limited to a particular language tutorial, and would not apply to the environment universally.

Another area for possible future development that would necessarily be language specific is the code evaluation subsystem. In order to be of any particular value this feature would have to be able to analyze code segments which are substantially more complex than simple constructs. The methodology used to provide this capability might then be used as a framework in development efforts for other programming languages as well.

APPENDIX A. SYSTEM ADMINISTRATION

This file contains information necessary for creating tutorial applications for use with the Intelligent Learning Environment. The environment has been designed to integrate a variety of tutorial applications, reference manuals, and hypertext help systems. The hypertext help facilities were provided by the HTML widget, developed at NCSA (National Center for Supercomputer Applications). Since the environment has been developed for use with the Athena Widget Set, the HTML library must be compiled for this widget set as well. Although this is the default compiler option, systems that use these libraries for their Mosaic and WWW systems may have compiled the library using the Motif compiler options. If this is the case, then a separate library will need to be prepared for this system. Note that these libraries are also stored in non-standard subdirectories, which may cause problems during installation of the system. The makefile included with the system is set up to look in the current directory for both the HTML header file as well as the precompiled HTML library. This entry will need to be modified to reflect the actual directory location for these files.

The hypertext markup language is a subset of the standard generalized markup language (SGML). These languages consist of embedded format commands, similar to those used for pretty printing and formatted display. The format of all of the instructional and reference material needs to be in this format in order to work properly. Examples that may be useful are the tutorial applications provided with the environment (itsada##.htm), as well as the various help and manual files included. Several tools are available for converting ascii code into HTML code, but the process is very straightforward. Information on how to complete this process is included with the help files incorporated into the system.

The following is a sample setup file necessary for proper operation of the system. Users will need to source this file prior to starting the Intelligent Learning Environment.

EXAMPLE SETUP FILE

```
# This file sets up the environment variables for proper execution of the Intelligent
# Learning Environment. This file can be sourced from users' .cshrc file in order to allow
# them to access all the program files
setenv ILEHOME /usr/local/tutors
set path= (/usr/local/tutors  $path)
# In addition to the above line, the path must also be set for any special quiz programs or
# code evaluators indicated in the configuration file, itstutor.dat, which must be located in
# the ILEHOME directory
# For Example:  Suppose itstutor.dat contained following entry:
# CAPS Tutor;TUT;/usr/local/tutors/CAPS/;CAPS_00.htm;C_Quiz;C_Edit;
# The final two entries represent special quiz program and code evaluator programs.  We
# would need to include the path where the executables were stored in order to provide
# access from our program.  This may look something like the following:
# set path=(/root/CAPS/quizDir   /root/CAPS/editorDir  $path)
set path=(/users/work1/delani/UtilDir  $path)
setenv XADAHELP /usr/local/tutors
```

This file is used to enable the dynamic creation of control center menus when the system is first started. This also allows system administrators to add new subsystems or reconfigure the existing directory structure without affecting multiple files. This directory is designed to provide a central location for the system help files and the system configuration file, "ilemenu.dat". Although the executable program will normally be located in this same directory, this is not necessary, since administrators may want to centrally locate all executable binary files. The system will still function properly as long as the path statement properly reflects the directory in which the executable is stored.

The last two entries of this setup file pertain to special applications that can be incorporated into the system. As the comments explain, these correspond to the last two fields of particular tutor or reference entry in the configuration file. This will be explained in more detail during the discussion of this file.

The system configuration file and the system help files must be located in the directory identified by the "ILEHOME" environment variable. The configuration file must be named "ilemenu.dat", and the system help files must not be renamed either. Entries in

the configuration file must follow the format explicitly as described in the comment section of that file. We have included an example here for completeness and as a reference in the event that the original file becomes corrupted.

EXAMPLE CONFIGURATION FILE

```
# Configuration file for Intelligent Tutoring System. The format of this file is fairly rigid,
# as we did not want to have to do a complicated parse of the file.
# NOTES:
#       -- Spaces ARE significant and preserved
#       -- ';' semicolon is field separator
#       -- Assumes no blank lines or comments once entries start
#       -- Assumes all of fields are present (have entries)
#        -- The only valid entries for 2nd field are "TUTOR", "MANUAL", and "HELP",
#           which correspond to Menu Buttons
#       -- Fields 5 and 6 specify separate, executable applications and should have
#           environment variable settings already set up or be fully specified. If NO
#           separate applications are provided, these fields must be "NONE".
# FIELDS and example line:
# Name; TYPE; Full Path; Home Page; Quiz App; Program Tool;
# Ada Tutor;TUTOR;/root/ITS_ENV/ada_tutor/;contents.htm;NONE;NONE;
# Note that file itsSetup contains information regarding proper setup of environment
# variables for separate executables in fields 5 & 6
# ------------------------------------------------------------------------------
<Begin Data>
Ada Tutorial;TUTOR;/usr/local/tutors/itsAda; itsada00.htm; NONE; xAdaEval;
C Tutorial;TUTOR;/usr/local/tutors/itsCeedir/;itscee00.htm;NONE;NONE;
C++ Tutorial;TUTOR;/usr/local/tutors/itsCxxdir/;itscxx00.htm;NONE;NONE;
Using Unix;HELP;/usr/local/helpdir/UsingUnix/;jxxl_00.htm;NONE;NONE;
Clips Help;HELP;/usr/local/helpdir/itsClips/;clips00.htm;NONE;NONE;
Ada LRM;MANUAL;/usr/local/manual/Ada9xLRMdir/;rm9x_00.htm;NONE;NONE;
C Reference;MANUAL;/usr/local/manual/CeeREFdir/;ceeref00.htm;NONE;NONE;
C++ Reference;MANUAL;/usr/local/manual/CxxREFdir/;cxxref00.htm;NONE;NONE;
<EOF>
```

Please note that the <EOF> flag is shown here just for illustrative purposes. It should not appear when the file is viewed. The comments included in this file explain the format requirements as well as the significance of each field. The last two entries of each field correspond to the special applications we mentioned during the discussion of the system setup file. An entry of "NONE" indicates that no special application is currently

available. Any other entry represents a separately executable program that will be used when special jumps are encountered. Since these entries represent executable programs, the necessary path information must be properly set up in order for them to work when called from users' directories. Although these fields are currently set up for a quiz and code evaluator, there is nothing to enforce that they be used as such. Hypertext links that have an extension beginning with the letter 'q' indicate calls to the application identified in field five, while links that have an extension beginning with 'y' indicate calls to the application identified in field six. Tutorial developers are free to change the functionality and semantics of these applications as they see fit. Furthermore, these hypertext links appear as normal links to the user of the application (see Figure 19 on page 50), and developers could provide proper indications to the user with respect to what selecting the link will actually accomplish.

Also note that if the paths for these subsystems becomes sufficiently large it will cause the line to be wrapped to the following line. This is perfectly acceptable; do NOT place linefeeds or end of line markers in the file to improve the appearance. These will be erroneously interpreted as extra delimiters by the C function strtok(), which is being used to parse this file.

The makefiles for all of the system elements are as follows:

- MakeILE - For the Intelligent Learning Environment
- MakeEval - For the Ada Code Evaluator Interface
- MakeHman - For a utility HTML file browser, HyperMan.

HyperMan is a convenience utility that was created during the development of the ILE. The functionality provided by the reference subsystem seemed quite desirable to have in a single, stand-alone application. This program allows tutorial and reference material developers to view their work in its display format while they are developing it (without having to modify the global setup for the ILE). This is also useful just for browsing help files located in the current directory.

Field five of the configuration file has been identified as corresponding to a special quiz application provided with the system. If this entry is "NONE", then a default quiz application becomes available to tutorial developers. The default quiz has been provided merely as a convenience, and is not designed to create student models or actually evaluate a student's understanding. It utilizes the hypertext capabilities of the HTML widget in order to provide immediate feedback to users in the form of a pop-up window containing explanations about quiz distractors. Typical user misconceptions and potential areas of confusion can be presented to users as distractors to quiz questions, and when the user selects these as the response to the quiz question, he or she is provided an explanation as to why the answer is correct or incorrect. The default quiz application included with the environment expects two files to be present in order to administer quizzes. The question file is designated with an extension of "quiz", and the solution file is designated with "soln". Note that both files MUST have the same base filename - as in following example:

Quiz question file => section1.quiz

Quiz solution file => section1.soln

The format for the question file is similar to that of a standard HTML file, with the exception that quiz answers are designated with the following anchor tag:

<A HREF=##1_A>

The "##" identifies the link as an answer to a question, and the "1_A" identifies the response as being "Question Number 1, Response A". This differs from standard anchor tags, which appear as the following:

- <A HREF=#TargetAnchor>
- <A HREF="filename.htm">
- <A HREF="filename.htm#TargetAnchor>

The first entry represents a hypertext link to a target anchor in the current file. The second link is to a file, and since no target anchor is specified the beginning of the file will be loaded. The final entry is to a specific target anchor in another file. This will cause the contents of that file to be loaded into memory, and then the specific anchor identified to be

positioned at the top of the viewing area. Note that the default quiz is currently implemented with single file capabilities only. Any attempts to jump to alternative files will be ignored, mainly due to the close correlation required between quiz question and solution files and their associated overhead.

EXAMPLE QUIZ FILE FORMAT

```
<HEADER><TITLE> Intelligent Tutoring System </TITLE></HEADER>
<A NAME=TopOfPage> <P><H1>Quiz Number One</H1><P>
<A NAME=Quest1> <P> <h2>Question Number 1</h2><P>
<H3>What is the meaning of the term "use" in the following Ada statement:</H3>
<pre>
    with TEXT_IO;
    <b>use</b> TEXT_IO;
</pre>
<ul>
<li><A HREF=##1_A>(A)</A>Include this file's source code in our file.<P>
<li><A HREF=##1_B>(B)</A>Borrow any needed code from this package and use it
here in our code.<P>
<li><A HREF=##1_C>(C)</A>Use this package to resolve any references to functions or
data types that are not defined in the current file. This allows us to access these functions
and ADT's without fully specifying the reference.<P>
<li><A HREF=#Quest2>Next Question</A><P>
</ul>
```

The last HREF listed is a standard HTML jump to an anchor within the same file. Currently the default quiz only allows same file jumps (single quiz). The format for the solution file is fairly rigid, in that each question has an explicit beginning and ending, as well as each of the answers to a particular question. The current implementation allows up to ten questions, and each question may have up to four possible answers. The quizzes may be incorporated anywhere within the instructional material, and provide a context sensitive testing capability. Although the default quiz does not create a student model, the source code can be used as an example for developers who wish to incorporate such capability, and then the name of the new quiz can be entered in field five to override the default.

EXAMPLE SOLUTION FILE FORMAT:

```
# Solution file for Quiz Number One.  Note the strict format requirements.
QUEST_1 = C
ANS_A
Explanation why A is incorrect answer goes here
END_ANS
ANS_B
Explanation why B is incorrect answer goes here
END_ANS
ANS_C
Explanation as to why this is correct answer goes here
END_ANS
ANS_D
Explanation why D is incorrect answer goes here
END_ANS
END_QUEST
```

NOTE1: Each question begins with "QUEST_NUM = ANSWER", where Answer must be single character which designates the correct response to the question. The whitespace between different responses and questions is ignored, but the whitespace within an answer (between 'ANS_A' and very next 'END_ANS') is considered important. Note that certain text editors may cause the explanation to have awkward appearance in the popup window.

NOTE2: This format is only required if developers want to use the default quiz. Quizzes are started when a hypertext link in the tutorial has a ".quiz" extension. If no quizzes are desired, there should be no links to a ".quiz" file. If developers want to provide their own quiz application, this can be done by providing an entry other than "NONE" in the fifth field of the configuration file entry for that tutorial. For example, the following entry will cause a separate quiz application named "MY_QUIZ" to be loaded whenever a ".quiz" jump is encountered:

```
Sample Tutor;TUTOR;/some/full/path/specification/;HomePage.htm;MY_QUIZ;NONE:
```

The entry in the last field is "NONE", which indicates that "Sample Tutor" currently does not have a separate code evaluator application. An entry in this field would be called when the tutorial encountered a hypertext link to a file with a ".ylee" extension. Since there is no default code evaluator provided, an entry of "NONE" in this field causes jumps with the ".ylee" extension to be ignored.

It is important to remember that developers do not have to follow the conventions established here and use fields five and six as quiz and code evaluators. They are free to incorporate their own tools and applications, and exploit the built in capability that the instructional material will automatically call these applications when a hypertext link with a special extension is encountered.

APPENDIX B. SOURCE CODE

The following is the source code listing of the files that comprise the Intelligent Learning Environment. They are included here for completeness and as a reference for future development work. The code has been written to exploit the graphical user interface capabilities of the X Window System, and specifically the Athena Widget Set which is included with the standard distribution from MIT. The hypertext capabilities are provided by the HTML Widget developed at NCSA for use with their Mosaic system, and is the same widget set used by most world wide web (WWW) browsers. The following program files are included:

- globals.h
- main5.c
- parse5.h
- parse5.c
- tutor5.h
- tutor5.c
- help5.c
- utils5.h
- utils5.c
- quiz5.h
- quiz5.c

GLOBALS.H

```
/*****************************************************************************
Name:   globals.h   Global Header File for Intelligent Learning Environment
Author: James Delani

Description: This is the header file for the Intelligent Learning Environment. The main program is the only
source file that uses this header file. The executable program consists of the following files (and their header
files) :   main5.c, tutor5.c, help5.c, utils5.c, quiz5.c, parse5.c
NOTE: The environment variable, ILEHOME, must be set to the directory that the configuration file,
"ilemenu.dat", and the system help files are located. The program depends on this to dynamically create the
menu entries. The program uses the hypertext widget developed at NCSA to provide the hypertext
capabilities. The makefiles assumes this library has been compiled for use with the Athena Widget set.
*****************************************************************************/
#ifndef _GLOBALS_H_
#define _GLOBALS_H_

#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
```

```
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Xaw/MenuButton.h>
#include <X11/Xaw/SimpleMenu.h>
#include <X11/Xaw/SmeBSB.h>
#include <X11/Xaw/SmeLine.h>

#include "tutor5.h"
#include "parse5.h"

/*
 * These variables are necessary to allow dynamic creation of menus. Executable  program needs to be able
 * to find the configuration file from whatever directory it is started from. Environment variable and path
 * settings must be properly set up before running this program.
 */
char * SYSHOME = "ILEHOME";
char * SYS_ConfigFileName = "ilemenu.dat";
char * SYS_HelpFileName = "syshlp00.htm";

/* These are currently the only valid menu entry types */
char * TutorType = "TUTOR";
char * ManualType = "MANUAL";
char * HelpType = "HELP";


static String fallback_resources[] = {
  "*.Font: *courier-medium*14*iso8859-1",
  "*FixedFont: -adobe-courier-medium-r-normal-*-14-*-*-*-*-*-iso8859-1",
  "*FixedboldFont: -adobe-courier-bold-r-normal-*-14-*-*-*-*-*-iso8859-1",
  "*FixeditalicFont: -adobe-courier-medium-o-normal-*-14-*-*-*-*-*-iso8859-1",
  "*AnchorUnderlines:              1",
  "*VisitedAnchorUnderlines:       1",
  "*DashedVisitedAnchorUnderlines:    True",
  "*.name: Intelligent Tutoring Environment", NULL, };

/* The following are predefined strings for the popup dialog boxes in main program */
String ExitConfirm = "Do you really want to exit?";

String AboutVersion =
"Preliminary Release of the \n\
Intelligent Learning Environment \n\n\
    Copyright 1994 \n
     Jim Delani";

String StartUpMessage =
"Welcome to the Intelligent Learning Environment!\n\n\
Note that some of the tutorial applications may\n\
```

```
need to save work files or student information\n\
during the course of instruction. We recommend \n\
that if you plan on using these tutorials that you\n\
start the application from a subdirectory set up\n\
specifically for this purpose.\n\n";

/* Callback functions needed in main function */
extern void exit_cb();
extern void greetUser();
extern void about_cb();
extern void really_exit_cb();
extern void cancel_cb();
extern void StartTutorCB();
extern void StartHelpCB();

#endif
```

MAIN5.C

```
/*****************************************************************************
Name:    main5.c  Main program for Intelligent Learning Environment
Author:  Jim Delani

Description:  This program provides an intelligent learning environment whichenables a user to learn a
variety of languages. The tutors and help systems incorporate hypertext by utilizing the HTML widget
developed by NCSA for Mosaicand other WWW browsers. The system is configured by an ascii file,
"ilemenu.dat", which resides in the the ILEHOME directory. This environment variable must be set in order
for the system to work properly, especially since the first thing the program does is look for the configuration
file in that directory! The format for the config file is fairly rigid, and is used to provide system administrators
with flexibility in setting up their own systems. It would seem natural to place all of the tutorials and reference
manuals in subdirectories of ILEHOME, but this is not required. As new tutorials and references are created,
it is easy to incorporate them into the program simply by modifying this configuration file. Advanced users
may create their own help files and include them in the help portion by passing the appropriate command line
arguments. The format of the file and the command line switches are included in the basic help available
from within the system.
*****************************************************************************/
#include "globals.h"

Widget TopLevel, tutorB, pshell;
TutorList tutorialPtr = NULL;
TutorList userInput = NULL;
TutorList tempPtr = NULL;

/* ARGSUSED */
int main(argc, argv)
 int argc;
 char *argv[];
{/* begin main */
 int j;
```

```
XtAppContext app_context;
/* NEVER call a Widget variable "exit"! */
Widget  buttonBar, manualB, helpB, infoB, exitB;
Widget  tutorMenu, manualMenu, helpMenu, menuEntry;
TutorStruct  genericHelp;
char SYS_Config[65];
char * UserConfig;
char * Sys_Home;

Sys_Home = getenv(SYSHOME);
strcat(Sys_Home, "/");
strcpy(SYS_Config, Sys_Home);
strcat(SYS_Config, SYS_ConfigFileName);

/*************************************************************************
 NOTE:  Environment variable does NOT include trailing '/' for path info.  All of the code in the
tutor and help files assume that the path includes the trailing '/' and that the filename consists of
only  the actual name, ie "syshelp00.htm"   NOT "/syshlp00.htm"
*************************************************************************/
/* Note that structure string elements are fixed size arrays */
 strcpy(genericHelp.Name, "Help on Help");
 strcpy(genericHelp.Type, HelpType);
 strcpy(genericHelp.Path, Sys_Home);
 strcpy(genericHelp.Home, SYS_HelpFileName);
 strcpy(genericHelp.Quiz, "NONE");
 strcpy(genericHelp.Program, "NONE");
 genericHelp.next = NULL;
 tutorialPtr = parseTutorList(SYS_Config, tutorialPtr);

 /* Parse configuration file passed in by user, append info to menu list */
 if(argc > 1)
  {
   UserConfig = strdup(argv[1]);
   tutorialPtr = parseTutorList(UserConfig, tutorialPtr);
  }

 TopLevel = XtVaAppInitialize(&app_context, "athenaILE", NULL, 0,
                                  &argc, argv, fallback_resources, NULL);

 XtVaSetValues(TopLevel, XtNname, "Intelligent Tutoring System",
                    XtNwidth, 350, XtNheight,40, NULL);

 buttonBar = XtVaCreateManagedWidget("buttonBar", formWidgetClass, TopLevel,
                                          XtNheight, 75, NULL);

 tutorB  = XtVaCreateManagedWidget("Tutor", menuButtonWidgetClass, buttonBar,
                                       XtNlabel, " Tutor ", XtNmenuName, "tutorMenu",
                                       XtNleft, XtChainLeft, XtNtop, XtChainTop,
                                       XtNbottom, XtChainBottom, NULL);
tutorMenu = XtCreatePopupShell("tutorMenu", simpleMenuWidgetClass, tutorB, NULL, 0);
```

```
manualB  = XtVaCreateManagedWidget("Manual", menuButtonWidgetClass, buttonBar,
                                   XtNlabel, " Manual ", XtNfromHoriz, tutorB,
                                   XtNmenuName, "manualMenu", XtNtop, XtChainTop,
                                   XtNbottom, XtChainBottom, NULL);
manualMenu = XtCreatePopupShell("manualMenu", simpleMenuWidgetClass, manualB, NULL, 0);

helpB = XtVaCreateManagedWidget("Help", menuButtonWidgetClass, buttonBar,
                                XtNlabel, " Help ", XtNmenuName, "helpMenu",
                                XtNfromHoriz, manualB, XtNtop, XtChainTop,
                                XtNbottom, XtChainBottom, NULL);
helpMenu = XtCreatePopupShell("helpMenu", simpleMenuWidgetClass, helpB, NULL, 0);

menuEntry = XtVaCreateManagedWidget("generic", smeBSBObjectClass, helpMenu,
                                    XtNlabel, "Using ILE", NULL);
XtAddCallback(menuEntry, XtNcallback, StartHelpCB, &genericHelp);

tempPtr = tutorialPtr;
while (tempPtr != NULL)
 {
  if (!strcmp(tempPtr->Type, TutorType))
    {
     menuEntry = XtVaCreateManagedWidget(tempPtr->Name, smeBSBObjectClass,
                                         tutorMenu, XtNlabel, tempPtr->Name, NULL);
     XtAddCallback(menuEntry, XtNcallback, StartTutorCB, tempPtr);
    }
  else if (!strcmp(tempPtr->Type, ManualType))
    {
     menuEntry = XtVaCreateManagedWidget(tempPtr->Name, smeBSBObjectClass,
                                         manualMenu, XtNlabel, tempPtr->Name, NULL);
     XtAddCallback(menuEntry, XtNcallback, StartHelpCB, tempPtr);
    }
  else if (!strcmp(tempPtr->Type, HelpType))
    {
     menuEntry = XtVaCreateManagedWidget(tempPtr->Name, smeBSBObjectClass,
                                         helpMenu, XtNlabel, tempPtr->Name, NULL);
     XtAddCallback(menuEntry, XtNcallback, StartHelpCB, tempPtr);
    }
  tempPtr = tempPtr->next;
 }

 menuEntry = XtVaCreateManagedWidget(tempPtr->Name, smeBSBObjectClass,
                                     helpMenu, XtNlabel, "About...", NULL);
 XtAddCallback(menuEntry, XtNcallback, about_cb, NULL);

exitB = XtVaCreateManagedWidget("Exit", commandWidgetClass, buttonBar,
                                XtNlabel, " Exit ", XtNfromHoriz, helpB,
                                XtNright, XtChainRight, XtNtop, XtChainTop,
                                XtNbottom, XtChainBottom, NULL);
XtAddCallback(exitB, XtNcallback, exit_cb, NULL);

greetUser();
```

```
  XtRealizeWidget(TopLevel);
  XtAppMainLoop(app_context);
}

/*
 * Callback associated with tutor menu which activates our Tutorials.  This function is located in tutor5.c,
 * and uses information passed to disable calling widget (tutorB), and set up environment of the tutorial.
 * Need to allow tutor to disable button since we lose control from this side once we call StartTutor
 */
void StartTutorCB(widget, client_data, call_data)
  Widget widget;
  XtPointer client_data, call_data;
{
  TutorList tPtr = (TutorList)client_data;
  StartTutor(tutorB, tPtr, call_data);
}

/* Callback associated with both the Manual Menu and Help Menu. */
void StartHelpCB(widget, client_data, call_data)
  Widget widget;
  XtPointer client_data, call_data;
{
  TutorList tPtr = (TutorList)client_data;
  StartHelp(tPtr, call_data);
}

/* Popup dialog to welcome user and advise about directories */
void greetUser()
{
  Widget okB, welcome_dialog;
  Position x, y;
  Dimension Rwidth, Rheight;
  int Pwidth = 500, Pheight = 350;

  pshell  = XtVaCreatePopupShell("Welcome!", transientShellWidgetClass, TopLevel,
                                 XtNwidth, Pwidth, XtNheight, Pheight, NULL);

  welcome_dialog = XtVaCreateManagedWidget("Welcome", dialogWidgetClass, pshell,
                                           XtNlabel, StartUpMessage, NULL);

  okB = XtVaCreateManagedWidget("  OK  ", commandWidgetClass, welcome_dialog,
                                XtNhorizDistance, 220, NULL);

  XtAddCallback(okB, XtNcallback, cancel_cb, NULL);

  XtVaSetValues(pshell, XtNx, 120, XtNy, 100, NULL);
  XtPopup(pshell, XtGrabExclusive);
}
```

```
/*
 * Callback to provide user with opportunity to confirm or cancel EXIT.  Exit application does not have
 * menu and may be triggered inadvertently, so should provide user opportunity to abort the exit
 */
void exit_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data;
  XtPointer call_data;
{
  Widget okB, cancelB, exit_dialog;
  Position x, y;
  Dimension width, height;

  pshell = XtVaCreatePopupShell("confirm", transientShellWidgetClass, TopLevel,
                                XtNwidth, 300, XtNheight,150, XtNlabel, ExitConfirm, NULL);

  exit_dialog = XtVaCreateManagedWidget("Confirm", dialogWidgetClass, pshell,
                                        XtNlabel, ExitConfirm, NULL);

  okB = XtVaCreateManagedWidget(" Exit ", commandWidgetClass, exit_dialog,
                                XtNhorizDistance, 40, NULL);
  XtAddCallback(okB, XtNcallback, really_exit_cb, NULL);

  cancelB = XtVaCreateManagedWidget("Cancel", commandWidgetClass, exit_dialog,
                                XtNhorizDistance, 40, XtNright, XtChainRight, NULL);
  XtAddCallback(cancelB, XtNcallback, cancel_cb , NULL);
  XtVaGetValues(TopLevel, XtNwidth, &width, XtNheight, &height, NULL);
  XtTranslateCoords(TopLevel, (Position) width/2, (Position) height/2, &x, &y);
  XtVaSetValues(pshell, XtNx, x - (width/2) , XtNy, y + height, NULL);

  XtPopup(pshell, XtGrabExclusive);
}

/* Callback associated with our Info button which provides popup version dialog */
void about_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data;
  XtPointer call_data;
{
  Widget about_dialog, okB;
  Position x, y;
  Dimension width, height;

  pshell = XtVaCreatePopupShell("About ...", transientShellWidgetClass, TopLevel,
                                XtNwidth, 300, XtNheight,200, NULL);

  about_dialog = XtVaCreateManagedWidget("about", dialogWidgetClass, pshell,
                                         XtNlabel, AboutVersion, NULL);

  okB = XtVaCreateManagedWidget(" OK ", commandWidgetClass, about_dialog,
                                XtNhorizDistance, 120, NULL);
```

```
  XtAddCallback(okB, XtNcallback, cancel_cb, NULL);

  XtVaGetValues(TopLevel, XtNwidth, &width, XtNheight, &height, NULL);
  XtTranslateCoords(TopLevel, (Position) width/2, (Position) height/2, &x, &y);
  XtVaSetValues(pshell, XtNx, x - (width/2), XtNy, y + height, NULL);

  XtPopup(pshell, XtGrabExclusive);
}

/* Callback used to remove any of the popub dialogs attached to pshell */
void cancel_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data, call_data;
{
 XtPopdown(pshell);
}

/* Callback associated with OK button of exit_cb.  Go ahead and close up shop  */
void really_exit_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data, call_data;
{
 exit(0);
}
```

PARSE5.H

```
/*****************************************************************************
Name:  parser5.h  Configuration file parser header file
Author: James Delani

Description:  This file parses the configuration file passed as a parameter and creates a linked list of
TutorStructs.  Allows dynamic creation of menus based on an Ascii file.  NOTE: the format of the
configuration file is very rigid in order to simplify parsing.  All of the fields are expected to have entries, and
some fields only allow specific entries.  The following information is also contained in the system
configuration file to facilitate development.
  -- Spaces ARE significant and preserved
  -- ';' semicolon is field separator
  -- Assumes no blank lines or comments once entries start
  -- Assumes all of fields are present (have entries)
  -- The only valid entries for 2nd field are 'TUTOR', 'MANUAL' and 'HELP', correspond to Menu Buttons
  -- Fields 5 and 6 specify separate, executable applications and should have environment variable settings
     already set up or be fully specified.  If NO separate applications are provided, then these fields must have
     'NONE' as their entry.
  -- The application indicated in Field 5 is automatically called when a hypertext link has an extension
     beginning with "q", as in "section1.quiz".  (Standard HTML extensions are ".htm" or ".html")
  -- The application indicated in Field 5 is automatically called when a hypertext link has an extension
     beginning with "x", as in "section1.xlec".  This is considered a special programming link
```

```
Field specifications::
Field 1: Name as it will appear in Control Center Menu
Field 2: Type of subsystem (which menu it goes in)
Field 3: Full path to where html files are located  (NOTE:  MUST have trailing slash '/')
Field 4: Home Page of subsystem  (First html file to load)
Field 5: Special Application  (Currently for Quizzes)
Field 6: Special Application  (For Code evaluator interface)

Example Line:
Ada Tutor;TUTOR:/root/ITS_ENV/ada_tutor/:contents.htm:NONE:NONE:
Remember: spaces ARE significant, every field must have entry, and fields  2, 5, and 6 are special fields
*******************************************************************************/
#ifndef _CONFIG_PARSE_
#define _CONFIG_PARSE_
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>

/*
 * For comparison within main program  Will cause default quiz to be loaded in case of "q" jumps, and
 * and will cause "y" jumps to be ignored (no default evaluator is provided)
 */
#define NONE "NONE"

/* Needed to store results as we parse the configuration file */
typedef struct _TutorStruct {
  char  Name[25];       /* For listing in Menus */
  char  Type[10];       /* either 'TUTOR', 'MANUAL', or 'HELP'*/
  char  Path[50];      /* directory of html files */
  char  Home[25];       /* first file to load = home page */
  char  Quiz[25];      /* Special Quiz application, or NONE */
  char  Program[25];    /* Special Code evaluator, or NONE */
  struct _TutorStruct * next;
  } TutorStruct, * TutorList;

/*  PUBLIC functions.  prnt:TutorList is provided to facilitate debugging */
extern TutorList parseTutorList();
extern void prntTutorList();

#endif
```

PARSE5.C

```
/*****************************************************************************
Name:  parse5.c  Configuration file parser version 5
Author: James Delani

Description: This file parses the configuration file passed as a parameter and reates a linked list of
TutorStructs. Allows dynamic creation of menus based on an Ascii file. Note that the format of the
configuration file is very rigid in order to simplify parsing. All of the fields are expected to have entries, and
some fields only allow specific entries. Refer to header file and sample configuration file for more info.
*****************************************************************************/
#include "parse5.h"
/*
 * Utility function which allows developers to view what was actually read in from the configuration file.
 * Can be used in development * to ensure that entries were properly read.
 */
void printTutorList(tutorPTR)
 TutorList tutorPTR;
{
 TutorList temp;

 if (tutorPTR != NULL)
  {
  temp = tutorPTR;
  printf("Name        Type\n");
  printf("%s \t", temp->Name);
  printf("%s \n", temp->Type);
  while (temp->next != NULL)
    {/* want to print the contents of the record */
    temp = temp->next;
    printf("%s \t", temp->Name);
    printf("%s \n", temp->Type);
    }
  }
}

/*
 * Creates a linked list of TutorStructs as the information is parsed by the parseTutorList function. After
 * each line is read and interpreted this function is called to add the new structure to the linked list.
 */
TutorList AddToTutorList(tListPtr, tempTRec)
 TutorList tListPtr;
 TutorStruct* tempTRec;
{
  TutorList new, temp;

  new = malloc(sizeof(TutorStruct));
  if (new == NULL)
    printf("Error allocating memory to new\n");
  *new = *tempTRec;
```

```
  new->next = NULL;

/* tListPtr must always point to the head of the list */
  if (tListPtr == NULL)  /* empty list */
     {
     tListPtr = new;
     }
  else
     {
     temp = tListPtr;
     while(temp->next != NULL)
        temp =(*temp).next;
     temp->next = new;
     }
  new = NULL;
  return(tListPtr);
}

/*
 * Function which parses the configuration file and converts into a linked list of TutorStruct's.  Note that
 * numFields depends on the definition of the structure as it appears in the header file.  Any changes to
 * these fields will have enormous inpact on this function, and must be accounted for here.
 */
TutorList  parseTutorList(filename, tListPtr)
  char * filename;
  TutorList tListPtr;
{
  FILE * fp;
  TutorStruct tempTRec;
  char line[160];
  int numFields = 6;
  char * seps = ":\t\n";  /* Spaces are significant! */
  char * field;
  int numRecs= 0;

  fp = fopen(filename, "r");
  if (fp == NULL)
    printf("File open failed!\n");

  while  ( (fgets(line, sizeof(line), fp) != NULL) && (line[0] =='#') )
   | /* For now -- Don't do anything */ ; |

  /* can process this line , then comes tutor and manual */
  while  ( (fgets(line, sizeof(line), fp) != NULL) && (line[0] !='#') )
   {
   /* First call to strtok require passing string to be converted  which is used to set up function for subsequent
     calls. Uses strncpy in all record assignments since arrays need to have AsciiZero termination!!! */
   if ( (field = strtok(line,seps)) != NULL)
     {
     /* Field number 1 is the name as it will appear in MENU */
     strncpy(tempTRec.Name, field, sizeof(tempTRec.Name) - 1);
```

```
        numRecs++;
        }
      /* Subsequent calls to strtok require NULL as first parameter */
      if ( (field = strtok(NULL,seps)) != NULL)
        {
        /* Field number 2 is for 'REF' type or 'TUT' type */
        strncpy(tempTRec.Type, field, sizeof(tempTRec.Type) - 1);
        numRecs++;
        }
      if ( (field = strtok(NULL,seps)) != NULL)
        {
        /* Field number 3 is path where files are located */
        strncpy(tempTRec.Path , field, sizeof(tempTRec.Path) - 1);
        numRecs++;
        }
      if ( (field = strtok(NULL,seps)) != NULL)
        {
        /* Field number 4 is first file to load or Home Page */
        strncpy(tempTRec.Home , field, sizeof(tempTRec.Home) - 1);
        numRecs++;
        }
      if ( (field = strtok(NULL,seps)) != NULL)
        {
        /* Field number 5 is special quiz application assoc with tutor*/
        strncpy(tempTRec.Quiz, field, sizeof(tempTRec.Quiz) - 1);
        numRecs++;
        }
      if ( (field = strtok(NULL,seps)) != NULL)
        {
        /* Field number 6 is special code analysis program */
        strncpy(tempTRec.Program, field, sizeof(tempTRec.Program) - 1);
        numRecs++;
        }
      /* Ensure that line has been parsed properly.  Any errors in config file cannot be tolerated, since would
        cause unpredictable errors */
      if (numRecs != numFields)
        {
        printf("Error in file format\n");
        printf("Trying to parse %s \n", filename);
        exit(0);
        }
      numRecs = 0;
      tListPtr = AddToTutorList(tListPtr, & tempTRec);
      }
    fclose(fp);
    return(tListPtr);
}
```

TUTOR5.H

```
/*****************************************************************************
Name:    tutor5.h  Header for Tutor subsystem of ILE, version 5
Author:  Jim Delani

Description: Header file that is needed for proper execution of tutor and reference subsections of the
Intelligent Learning Environment. Defines structures and public functions, as well as serving as central
"include" file for most of X11 and Xaw include files.
*****************************************************************************/
#ifndef HTML_TUTOR__
#define HTML_TUTOR__

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Xaw/Text.h>
#include <X11/Xaw/AsciiText.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Box.h>

#include "HTML.h"
#include "parse5.h"

/* For use with fseek();  These should NOT be necessary but the Solbournes generated undefined errors
   (sporadically) for some of these. */
#ifndef SEEK_SET
#define SEEK_SET 0
#endif
#ifndef SEEK_CUR
#define SEEK_CUR 1
#endif
#ifndef SEEK_END
#define SEEK_END 2
#endif

/*
 * Public functions for the ITS Tutorial.  Each opens up its own window and loads the designated file into
 * the HTML widget.  StartTutor obviously starts the Intelligent tutor portion, and has extra facilities (quizzes
 * and program evaluator capabilities) built in.  StartHelp is for reference manuals and help files, and merely
 * exploits the simple hypertext mechanism provided for by the HTML W. ' 'et written by NCSA for Mosaic
 * and WWW browsers
 */
void StartTutor();
void StartHelp();
```

```
/*
 * HTML Anchor Tags are designated with HREF, and widget itself uses href to designate the name of the
 * anchor. Following the convention of widget code and also style used in Jeremy Payne's example code.
 * This structure is used to maintain history of * hypertext jumps. This facilitates backtracking.
 */
typedef struct _href {
 char        *href;
 int         CallingId;
 struct _href  *next;
 } HREF_STRUCT;

#endif
```

TUTOR5.C

```
/*****************************************************************************
Name:   HTML Help Window / Reference Manager
Author: James Delani

Description: Uses the Athena Widget set and the HTML widget set created by NCSAto provide hypertext
help capabilities for an intelligent tutoring environment. Entry point is StartTutor, which expects a pointer to
a TutorStruct as its second parameter. This is used to set up the proper setting to correctly utilize the tutorial.
The user is provide backtracking capabilities as well as the ability to bailout/clear and return to the first or
home page. Because of the multiple environment settings as well as the possible interaction with other
facilities (code evaluators, syntax directed editors, etc.), once the user has decided to open a tutorial the Tutor
button on the main window is disabled. In order to start another tutorial the user must first properly close the
tutorial currently open. Note that the help facility may be opened concurrently with a tutorial, and that no
such restriction is currently placed on the help section. The rational is that it is quite feasible to desire to move
among different help, reference, or manual sections, and requiring a user to reposition the window would be
an unnecessary inconvenience. However, it is assumed that once a tutorial is opened the user intends to
complete a lesson or work within a single application.
Special Acknowledgement: Incorporation of the HTML widget into this project was made considerably
easier because of the help of Jeremy Payne at NCSA. He provided example code which showed how to
perform the initialization of the widget as well as how to maintain a history list of hypertext jumps. The basic
functionality provided in the on-line help systems is modelled after his work
*****************************************************************************/
#include "tutor5.h"
#include "utils5.h"
#include "quiz5.h"

extern Widget TopLevel;
extern Display *display;
static Widget TutorShell, TutorForm, ExitTutor, BackButton;
static Widget TutorHome, TutorW, menu_box;
static Widget CallingWidget;

static char *tutorText=NULL;
static char *current_href=NULL;
static char TutorPath[80];
static char HomePage[100];
```

```
static char *QuizApp=NULL;
static char *ProgApp=NULL;

static HREF_STRUCT  *BackList=NULL;

/* Callback functions needed in help section */
static void Anchor_callback();
static void Back_callback();
static void Home_Callback();
static void Quit_callback();
/* Utility functions needed to manage widgets and history */
static void StartITS_Tutor();
static void ShutDownTutor();
static Boolean Jump();
/* Utility functions to provide quiz and code evaluation mechanism */
static void callProblem();
static void callQuiz();

/*
 * Starting point of tutorial as far as main procedure is concerned. Sets up environment and path variables,
 * since all jumps are local to the directory the file is located in.  Also disables calling widget,  which should
 * be tutor button, so that user cannot clobber all of our  information and evaluator setup.
 */
void StartTutor(widget, tutorPtr , call_data)
 Widget widget;
 TutorList tutorPtr;
 XtPointer call_data;

{
  /* TutorPath, HomePage, QuizApp, ProgApp, and CallingWidget are global variables since need to be
     accessed in independent callbacks.  Fields of TutorStruct listed in conparse.h  */
  char TEMP[100];
  char *NoEntry = "NONE";
  TutorStruct tutor;

  TEMP[0] = '\0';
  tutor = *tutorPtr;
  strcat(TEMP, tutor.Path);
  strcpy(TutorPath, TEMP);
  strcat(TEMP, tutor.Home);
  strcpy(HomePage, TEMP);

  if(strcmp(tutor.Quiz, NoEntry))
    QuizApp=strdup(tutor.Quiz);

 if(strcmp(tutor.Program, NoEntry))
    ProgApp=strdup(tutor.Program);

  CallingWidget = widget;
  StartITS_Tutor(HomePage);
}
```

```
/* Creates the window for the tutorial.  Provides user with Exit, Back and Home buttons */
static void StartITS_Tutor(filename)
 char *filename;
{
 int n;
 int menuBoxHt = 70;
 int horizDist = 20;
 int minHeight= 300;
 int minWidth = 400;
 Widget FormB, FormW;

 if(!loadAsciiFile(filename, &tutorText))
  {
  PrintStartError(filename);
  return;
  }
 else
  {
  XtVaSetValues(CallingWidget, XtNsensitive, False, NULL);
  current_href=strdup(filename);

  TutorShell=XtVaCreatePopupShell("ITS Tutor", topLevelShellWidgetClass, TopLevel,
                                  XtNallowShellResize, True, XtNminHeight, minHeight,
                                  XtNminWidth, minWidth, NULL);

  TutorForm=XtVaCreateManagedWidget("TutorForm", formWidgetClass, TutorShell,
                                    XtNborderWidth, 0, NULL);

  menu_box = XtVaCreateManagedWidget("menuBox", formWidgetClass, TutorForm,
                                     XtNborderWidth, 0, XtNheight, menuBoxHt,
                                     XtNwidth, minWidth, XtNleft, XtChainLeft,
                                     XtNtop, XtChainTop, XtNright, XtChainRight, NULL);

   ExitTutor=XtVaCreateManagedWidget(" Quit ", commandWidgetClass, menu_box,
                                     XtNhorizDistance, horizDist, XtNleft, XtChainLeft,
                                     XtNtop, XtChainTop, XtNbottom, XtChainBottom, NULL);
  XtAddCallback(ExitTutor, XtNcallback, Quit_callback, NULL);


  BackButton=XtVaCreateManagedWidget(" Back ", commandWidgetClass, menu_box,
                                     XtNhorizDistance, horizDist, XtNfromHoriz, ExitTutor,
                                     XtNtop, XtChainTop, XtNbottom, XtChainBottom,
                                     XtNsensitive, False, NULL);
  XtAddCallback(BackButton, XtNcallback, Back_callback, NULL);

  TutorHome=XtVaCreateManagedWidget(" Home ", commandWidgetClass, menu_box,
                                     XtNhorizDistance, horizDist, XtNfromHoriz, BackButton,
                                     XtNtop, XtChainTop, XtNbottom, XtChainBottom,
                                     XtNsensitive, False, NULL);
  XtAddCallback(TutorHome, XtNcallback, Home_Callback, filename);
```

```
    TutorW=XtVaCreateManagedWidget("TutorW", htmlWidgetClass, TutorForm,
                                        XtNfromVert, menu_box, XtNbottom, XtChainBottom,
                                        XtNheight, minHeight, XtNwidth, minWidth, NULL);
    XtAddCallback(TutorW, WbNanchorCallback, Anchor_callback, (XtPointer)0);

    XtRealizeWidget(TutorShell);
    /*  Parameters to HTML widget function call are as follows:
     (Widget, string, Header, Footer, WidgetID, Target Anchor, Cached Info */
    HTMLSetText(TutorW, tutorText, "\0", "\0", 0, NULL, NULL);

    XtPopup(TutorShell, XtGrabNone);
    }
} /* End StartITS_Tutor Function */

/*
 * Format for href is filename#targetanchor  If no target anchor has been specified then the beginning of the
 * file is displayed.  Else, the target anchor will be positioned at top of screen.
 */
static Boolean Jump(new_href)
 char *new_href;
{
 char *new_txt, *c;
 char *target_anchor=NULL;

 /* Returns ZERO if EQUAL!! */
 if(strcmp(new_href, HomePage))
    {
     XtVaSetValues(TutorHome, XtNsensitive, True, NULL);
     XtVaSetValues(BackButton, XtNsensitive, True, NULL);
    }
 /* strrchr returns pointer to the Last occurence of char in string */
 if((c=(char *)strrchr(new_href, '#'))!=NULL)
   {
    target_anchor=(char *)(c+1);
    *c='\0';
   }
 if(!loadAsciiFile(new_href, &new_txt))
   {
    PrintLinkError(new_href);
    return;
   }
 if(tutorText!=NULL)
   free(tutorText);
 tutorText=new_txt;

 /*  Parameters to HTML widget function call are as follows:
   (Widget, string, Header, Footer, WidgetID, Target Anchor, Cached Info */
 HTMLSetText(TutorW, tutorText, "\0", "\0", 0, target_anchor, NULL);

 return(True);
}
```

```
/* Function that responds to user selecting BACK button from options. */
static void Back_callback(widget, client_data, call_data)
 Widget widget;
 XtPointer client_data, call_data;
{
 if(BackList==NULL)
   return;
 else
  {
   if(!strcmp(BackList->href, current_href))
     HTMLGotoId(TutorW, BackList->CallingId);
   else if(Jump(BackList->href))
    {
     free(current_href);
     current_href=strdup(BackList->href);
     HTMLGotoId(TutorW, BackList->CallingId);
    }
   PopBacklist(&BackList);
   if (BackList == NULL)
    XtVaSetValues(BackButton, XtNsensitive, False, NULL);
  }
}

/*
 * Callback function to respond to user clicking on the HOME button at the top of the HTML form. Currently
 * clears the entire BackList, similar to a reset or start over. Loads the first page visited (Home Page)
 */
static void Home_Callback(widget, client_data, call_data)
  Widget widget;
  XtPointer client_data, call_data;
{
  char *target = strdup(HomePage);

  /* Note that even if same file currently loaded need to reread file
    since HomePage does not have CallingID associated with it */
  if(Jump(target))
    {
     ClearBacklist(&BackList);
     free(current_href);
     current_href=target;
    }
  else
   free(target);

  /* We need to disable the Home and Back buttons now */
  XtVaSetValues(TutorHome, XtNsensitive, False, NULL);
  XtVaSetValues(BackButton, XtNsensitive, False, NULL);
}
```

```
/*************************************************************************
 Fields of the data parameter passed to Callback function:
XEvent*event
intelement_id  => The ID of Anchor selected
char*text      => The String used to display anchor
char*href      => Name of target file & anchor to jump to
*************************************************************************/
/*
 * Callback needed to be provided when instantiating HTML widget Currently a filename extension of 'q'
 * or 'y' indicates special jump and needs to be handled accordingly.  A jump which begins with a '#' indicates
 * a jump within the same file, and needs to be handled a little differently than fully specified jumps.
 */
static void Anchor_callback(widget, client_data, data)
 Widget widget;
 XtPointer client_data;
 WbAnchorCallbackData *data;
{
 char *new_href=strdup(data->href);
 char *c, temp[100];
 int CallingID;

 temp[0] = '\0';
 CallingID = (*data).element_id;

 if(*new_href == '#')
  {  /* HTML-legal local jump within same file */
   int newID;
   c = new_href;
   c++;
   strcat(temp, c);
   newID = HTMLAnchorToId(TutorW, temp);
   if(!newID)
    {
     printf("Unable to determine new Widget ID\n");
    }
   HTMLGotoId(TutorW, newID);
   if(!PushBacklist(&BackList, current_href, CallingID))
     PrintMemoryWarning();
   XtVaSetValues(BackButton, XtNsensitive, True, NULL);
   XtVaSetValues(TutorHome, XtNsensitive, True, NULL);
   return;
  }
 else
  {
  /* First need to set up path information for file */
  strcat(temp, TutorPath),
  strcat(temp, new_href);
  free(new_href);
  new_href = strdup(temp);
```

```
 /* Next need to check for special file extensions strchr returns pointer to 1st occurence */
  if((c = (char *)strchr(new_href, '.')) != NULL)
    {
    c++;
    if (*c =='q')
      {
      callQuiz(new_href);
      return;
      }
    else if (*c =='y')
      {
      callProblem(new_href);
      return;
      }
    else if ((*c =='j') || (*c == 'g') || (*c == 'x'))
      {
      CallPictureViewer(new_href);
      return;
      }
    else if ((*c =='a') || (*c == 'w'))
      {
      CallAudioPlayer(new_href);
      return;
      }
    }  /* end if strrchr */
  if(Jump(new_href))
    {
    if(!PushBacklist(&BackList, current_href, CallingID))
      PrintMemoryWarning();
    free(current_href);
    current_href=new_href;
    }
  else
    free(new_href);
  }  /* end else not local jump */
}

/* Clears out back list of href's, frees allocated memory, and shuts down tutorial *
static void ShutDownTutor()
{
 ClearBacklist(&BackList);
 if(tutorText!=NULL)
   {
   free(tutorText);
   tutorText=NULL;
   }
 if(current_href!=NULL)
   {
   free(current_href);
   current_href=NULL;
   }
```

```
 if(QuizApp!=NULL)
   {
   free(QuizApp);
   QuizApp=NULL;
   }

 if(ProgApp!=NULL)
   {
   free(ProgApp);
   ProgApp=NULL;
   }

 XtPopdown(TutorShell);
 XtDestroyWidget(TutorShell);
 XtVaSetValues(CallingWidget, XtNsensitive, True, NULL);
}

/*
 * Callback function for Quit button.  Provide level of indirection is in case we decide to provide
 * popup confirm dialog box
 */
static void Quit_callback(widget, client_data, call_data)
 Widget widget;
 XtPointer client_data, call_data;
{
 ShutDownTutor();
}

/*  Function which handles hypertext jumps to quiz questions.  */
static void callQuiz(command_line)
 char * command_line;
{   /* Returns ZERO if EQUAL!!! */
 if(QuizApp != NULL)
   {
   char sysCommand[100];
   sysCommand[0]='\0';
   strcat(sysCommand, QuizApp);
   strcat(sysCommand, " ");    /* To ensure separation for cmd line */
   strcat(sysCommand, command_line);
   system(sysCommand);
   }
 else
   {
   StartQuiz(command_line);
   }
}
```

```
/*
 * Function which handles jumps to program evaluator topics.  Several ways to handle this, for now we will
 * just call an evaluator front end and pass the filenam which contains the problem statement.
 */
static void callProblem(command_line)
 char * command_line;
{
 if(ProgApp != NULL)
   {
   char sysCommand[100];
   sysCommand[0]='\0';
   strcat(sysCommand, ProgApp);
   strcat(sysCommand, " ");    /* To ensure separation for cmd line */
   strcat(sysCommand, command_line);
   system(sysCommand);
   }
 else
   {
   printf("Attempted hypertext link to unspecified program.\n");
   printf("Unable to complete request.\n");
   }
}
```

HELP5.C

```
/*****************************************************************************
Name:  help5.c   Help and Manual Subsystem Version 5
Author:  James Delani

Description:  Uses the HTML widget set created by NCSA to provide hypertext help capabilities.  This code
is based on the Athena Widget Set which is included in the standard distribution of the X Window System
available from MIT.  Entry point is StartHelp, which expects a pointer to a TutorStruct as its second
parameter.  Of particular concern to this function is the path and filename to load into the HTML widget (The
other entries are mainly for the tutorial, and possibly for expansion later on).  The user is provide backtracking
capabilities as well as a bailout/clear capability to return to the opening screen.  Once the user sets up the
window he/she may select other help files to view without having to first close the window.  This enables users
to customize their own programming environment, and possibly switch between several different reference
files
.Special Acknowledgement:  Incorporation of the HTML widget into this project was made considerably
easier because of the help of Jeremy Payne at NCSA.  He provided example code which showed how to
perform the initialization of the widget as well as how to maintain a history list of hypertext jumps.  The basic
functionality provided in the on-line help systems is modelled after his work
*****************************************************************************/
#include "tutor5.h"
#include "utils5.h"

extern Widget TopLevel;
extern Display *display;
static Widget HelpShell, HelpForm, ExitHelp, BackButton;
static Widget HelpHome, HelpW, menu_box;
```

```
static Boolean HelpWindowOpen;
static char *helpText=NULL;
static char *current_href=NULL;
static char HomePage[100];
static char HelpPath[80];
static HREF_STRUCT *BackList=NULL;
/* Callback functions needed in help section */
static void Anchor_callback();
static void Back_callback();
static void Home_Callback();
static void Quit_callback();
static void StartITS_Help();
static void ShutDownHelp();
static Boolean Jump();

/*
 * Starting point of help section as far as main procedure is concerned. Sets up environment and path
 * variables, since all jumps are local to the directory the file is located in. Note that (unlike tutorial) we
 * allow user to set up a help window and jump between different reference files. Although selecting new
 * help file will clobber all previous info, it does make it easier for user to manage environment, especially if
 *  using as programmers reference tool. This also allows having single file for both Manual and Help
 * selections.
 */
void StartHelp(tutorPtr , call_data)
 TutorList tutorPtr;
 XtPointer call_data;
{
  /* HelpPath, HomePage, and HelpWindowOpen are global variables */
  char TEMP[100];
  TutorStruct tutor;

  TEMP[0] = '\0';
  tutor = *tutorPtr;
  strcat(TEMP, tutor.Path);
  strcpy(HelpPath, TEMP);
  strcat(TEMP, tutor.Home);
  strcpy(HomePage, TEMP);

  if(!HelpWindowOpen)
    StartITS_Help(HomePage);
  else
    {
    ClearBacklist(&BackList);
    XtVaSetValues(HelpHome, XtNsensitive, False, NULL);
    if(Jump(HomePage))
      {
      free(current_href);
      current_href = strdup(HomePage);
      }
    }
}
```

```
/* Creates the help window.  Provides user with  Exit, Back and Home buttons for navigation and control.*/
static void StartITS_Help(filename)
 char *filename;
{
 int n;
 int menuBoxHt = 70;
 int horizDist = 20;
 int minHeight= 300;
 int minWidth = 400;

 Widget FormB, FormW;
 if(!loadAsciiFile(filename, &helpText))
  {
  PrintStartError(filename);
  return;
  }

 HelpShell=XtVaCreatePopupShell("ITS Help", topLevelShellWidgetClass, TopLevel,
                                XtNallowShellResize, True, XtNminHeight, minHeight,
                                XtNminWidth, minWidth, NULL);

 HelpForm=XtVaCreateManagedWidget("helpform", formWidgetClass, HelpShell,
                                  XtNborderWidth, 0, NULL);

 menu_box = XtVaCreateManagedWidget("menuBox", formWidgetClass, HelpForm,
                                    XtNborderWidth, 0, XtNheight, menuBoxHt,
                                    XtNwidth, minWidth, XtNleft, XtChainLeft,
                                    XtNtop, XtChainTop,  XtNright, XtChainRight, NULL);


 ExitHelp=XtVaCreateManagedWidget(" Quit ", commandWidgetClass, menu_box,
                                  XtNhorizDistance, horizDist, XtNleft, XtChainLeft,
                                  XtNtop, XtChainTop, XtNbottom, XtChainBottom, NULL);
 XtAddCallback(ExitHelp, XtNcallback, Quit_callback, NULL);


 BackButton=XtVaCreateManagedWidget(" Back ", commandWidgetClass, menu_box,
                                    XtNhorizDistance, horizDist, XtNfromHoriz, ExitHelp,
                                    XtNtop, XtChainTop, XtNbottom, XtChainBottom,
                                     XtNsensitive, False, NULL);
 XtAddCallback(BackButton, XtNcallback, Back_callback, NULL);

 HelpHome=XtVaCreateManagedWidget(" Home ", commandWidgetClass, menu_box,
                                  XtNhorizDistance, horizDist, XtNfromHoriz, BackButton,
                                  XtNtop, XtChainTop, XtNbottom, XtChainBottom,
                                  XtNsensitive, False, NULL);
 XtAddCallback(HelpHome, XtNcallback, Home_Callback, NULL);

 HelpW=XtVaCreateManagedWidget("HelpW", htmlWidgetClass, HelpForm,
                               XtNfromVert, menu_box, XtNbottom, XtChainBottom,
                               XtNheight, minHeight, XtNwidth, minWidth, NULL);
```

```
 XtAddCallback(HelpW, WbNanchorCallback, Anchor_callback, (XtPointer)0);

 XtRealizeWidget(HelpShell);

 /*  Parameters to HTML widget function call are as follows:
   (Widget, string, Header, Footer, WidgetID, Target Anchor, Cached Info) */
 HTMLSetText(HelpW, helpText, "\0", "\0", 0, NULL, NULL);

 current_href=strdup(filename);
 XtPopup(HelpShell, XtGrabNone);
 HelpWindowOpen = TRUE;

} /* End StartITS_Help Function */

/*
 * Format for href is filename#targetanchor  If no target is specified we need to have beginning of file
 * displayed in window, otherwise will require target anchor to be positioned at top of window.
 */
static Boolean Jump(new_href)
 char *new_href;
{
 char *new_txt, *c;
 char *target_anchor=NULL;
 /* Returns ZERO if EQUAL!! */
 if(strcmp(new_href, HomePage))
    {
     XtVaSetValues(HelpHome, XtNsensitive, True, NULL);
     XtVaSetValues(BackButton, XtNsensitive, True, NULL);
    }
 if((c=(char *)strrchr(new_href, '#'))!=NULL)
   {
    target_anchor=(char *)(c+1);
    *c='\0';
   }

 if(!loadAsciiFile(new_href, &new_txt))
   {
    PrintLinkError(new_href);
    return;
   }

 if(helpText!=NULL)
    free(helpText);
 helpText=new_txt;
 /*  Parameters to HTML widget function call are as follows:
   (Widget, string, Header, Footer, WidgetID, Target Anchor, Cached Info */
 HTMLSetText(HelpW, helpText, "\0", "\0", 0, target_anchor, NULL);

 return(True);
}
```

```
/* Function that responds to user selecting BACK button from options. */
static void Back_callback(widget, client_data, call_data)
 Widget widget;
 XtPointer client_data, call_data;
{
 if(BackList==NULL)
   return;
 else
  {
  if(!strcmp(BackList->href, current_href))
    HTMLGotoId(HelpW, BackList->CallingId);
  else if(Jump(BackList->href))
    {
    free(current_href);
    current_href=strdup(BackList->href);
    HTMLGotoId(HelpW, BackList->CallingId);
    }
  PopBacklist(&BackList);
   if (BackList == NULL)
     XtVaSetValues(BackButton, XtNsensitive, False, NULL);
  }
}

/* Callback function corresponding to HOME button.  Acts as a  Clear/ Start Over button */
static void Home_Callback(widget, client_data, call_data)
  Widget widget;
  XtPointer client_data, call_data;
{
  char *target=strdup(HomePage);

  if(Jump(target))
    {
    ClearBacklist(&BackList);
    free(current_href);
    current_href=target;
    }
  else
   free(target);

  /* We need to disable the Home and Back buttons now */
  XtVaSetValues(HelpHome, XtNsensitive, False, NULL);
  XtVaSetValues(BackButton, XtNsensitive, False, NULL);
}

/*************************************************************************
 Fields of the data parameter passed to Callback function:
   XEvent *event
   int   element_id  => The ID of Anchor that was selected
   char   *text      => The string used to display anchor
   char   *href      => The name of target file & anchor to jump to
*************************************************************************/
```

```
/* Callback needed to be provided when instantiating HTML widget */
static void Anchor_callback(widget, client_data, data)
 Widget widget;
 XtPointer client_data;
 WbAnchorCallbackData *data;
{
 char *new_href=strdup(data->href);
 char *c;
 char temp[100];
 int CallingID;
/* First check if hypertext link is to a specially formatted file, ie jpeg, giff, etc. */
 if((c = (char *)strchr(new_href, '.')) != NULL)
  {
  c++;
  if ((*c =='j') || (*c == 'g') || (*c == 'x'))
   {/* filename.jpeg or filename.gif or filename.xbm */
    CallPictureViewer(new_href);
    return;
    }
     else if ((*c =='a') || (*c == 'w'))
    {/* filename.au or filename.wav */
    CallAudioPlayer(new_href);
    return;
    }
  } /* end if strrchr */
/* If we get this far then it should be standard hypertext jump */
 temp[0] = '\0';
 CallingID = (*data).element_id;
 /* HTML => local jump within same file doesn't require filename */
 if(*new_href == '#')
  {
   int newID;
   c = new_href;
   c++;
   strcat(temp, c);
   newID = HTMLAnchorToId(HelpW, temp);
   if(!newID)
    {
     printf("Anchor didn't create ID\n");
     newID = HTMLAnchorToId(HelpW, new_href);
     if(!newID)
      printf("Didn't work with original href either!\n");
    }
   HTMLGotoId(HelpW, newID);
   if(!PushBacklist(&BackList, current_href, CallingID))
     PrintMemoryWarning();
   XtVaSetValues(HelpHome, XtNsensitive, True, NULL);
   XtVaSetValues(BackButton, XtNsensitive, True, NULL);
   return;
  }
 else
```

```
    { /* else target anchor is to specific file */
     strcat(temp, HelpPath);
     strcat(temp, new_href);
     free(new_href);
     new_href = strdup(temp);

     if(Jump(new_href))
       {
       if(!PushBacklist(&BackList, current_href, CallingID))
         PrintMemoryWarning();
       free(current_href);
       current_href=new_href;
       }
     else
       free(new_href);
     }
}

/* Clears out back list of href's and shuts down tutorial */
static void ShutDownHelp()
{
 if(helpText!=NULL)
   {
   free(helpText);
   helpText=NULL;
   }
 ClearBacklist(&BackList);
 if(current_href!=NULL)
   {
   free(current_href);
   current_href=NULL;
   }

 XtPopdown(HelpShell);
 XtDestroyWidget(HelpShell);
 HelpWindowOpen = FALSE;
}

/*
 * Callback function for Quit button.  Level of indirection is in case want to incorporate popup dialog
 * with exit confirmation message
 */
static void Quit_callback(widget, client_data, call_data)
 Widget widget;
 XtPointer client_data, call_data;
{
 ShutDownHelp();
}
```

UTILS5.H

```
/*****************************************************************************
Name:    utils5.h    Header file for Utility section of ILE
Author:  James Delani

Description: Provides utility functions needed to facilitate backtracking capabilities and for loading contents
of text files into the hyper text widget. This header file is included by the following programs: tutor5.c,
help5.c, and quiz5.c. Note that the type 'Boolean' is defined in the Xt Intrinsics toolkit. If modify system so
that doesn't use this then will have to define type or change to type integer. Provide several functions for
displaying error messages to stderr. PrintMemoryWarning designed to be used when PushBacklist cannot
allocate memory to add new item to backlist. PrintStartError for use when the system cannot read or properly
load the filenames needed to start one of the subsystems causing an abort of start operation. PrintLinkError
for use when hypertext link requested cannot be performed, so system is ignoring request.
*****************************************************************************/
#ifndef _UTILS_H_
#define _UTILS_H_
/* Public Functions provided */
Boolean PushBacklist();
void PopBacklist();
void ClearBacklist();
Boolean loadAsciiFile();
void PrintStartError();
void PrintMemoryWarning();
void PrintLinkError();
/* For handling special files designated in hypertext link */
void CallAudioPlayer();
void CallPictureViewer();
void CallPostScriptViewer();
#endif
```

UTILS5.C

```
/*****************************************************************************
Name:    utils5.c    Utility functions for Intelligent Learning Environment
Author:  James Delani

Description: Provides utility functions needed to facilitate backtracking capabilities and for loading contents
of text files into the hyper text widget. Used by tutor5.c, help5.c, and quiz5.c.
*****************************************************************************/
#include "tutor5.h"
#include "utils5.h"

/*  Reads in the text file and loads it into string for the HTML widget */
Boolean loadAsciiFile(filename, txt_ptr)
 char *filename, **txt_ptr;
{
 char * buffer;
 int bytes_read=0;
 FILE *filePtr;
 struct stat file_info;
```

```
 if((filePtr=fopen(filename, "r"))==NULL)
   return(False);
 if( stat(filename, &file_info) != 0)
   {
    fclose(filePtr);
    return(False);
   }
 buffer=(char *)malloc(file_info.st_size + 5);
 if(buffer == (char *) NULL)
   {
    fclose(filePtr);
    return(False);
   }
 bytes_read = fread( buffer, 1, file_info.st_size, filePtr);
 fclose(filePtr);
 buffer[file_info.st_size -1] = '\0';  /* truncate buffer*/
 if (bytes_read < file_info.st_size)
   {
    free(buffer);
    return(False);
   }
 *txt_ptr = strdup(buffer);
 free(buffer);
 return(True);
}

/*
 * Prints error message in the event that could not load html file into string.
 * Informs user why the link was not executed.
 */
void PrintLinkError(filename)
 char * filename;
{
 printf("WARNING!\nCould not read filename: %s \n", filename);
 printf("Unable to execute hypertext link so ignoring request\n");
}

/*
 * Prints error message in the event that could not load HomePage when starting up application.
 * Normally means path info incorrect or problem with file system.
 */
void PrintStartError(filename)
 char * filename;
{
 printf("WARNING!\nCould not read filename: %s \n", filename);
 printf("Unable to start subsystem application as requested.\n");
}
```

```
/*
 * Utility function that adds the current href to the backlist prior to  making a hypertext jump.  This allows
 * us to provide simple backtracking mechanism for user to single step back through jumps
 */
Boolean PushBacklist(back_list, href, widgetID)
 HREF_STRUCT  **back_list;
 char *href;
 int widgetID;
{
 HREF_STRUCT *new;

 new=(HREF_STRUCT *)malloc(sizeof(HREF_STRUCT));
 if (new == (HREF_STRUCT *)NULL)
   return False;
 else  /* for semantic correctness */
   {
   new->href=strdup(href);
   new->CallingId=widgetID;
   new->next=*back_list;
   *back_list = new;
   return True;
   }
}

/* Utility function to pop single entry from Backlist (history of jumps). */
void PopBacklist(back_list)
 HREF_STRUCT  **back_list;
{
 HREF_STRUCT *tempPtr;
 /* Safety check before we do anything */
 if(*back_list==NULL)
   return;
 else  /* for semantic correctness */
   {
   tempPtr=*back_list;
   *back_list=tempPtr->next;
   free(tempPtr->href);
   free(tempPtr);
   }
}

/* Utility function to clear Backlist and deallocate memory /
void ClearBacklist(back_list)
 HREF_STRUCT  **back_list;
{
 while(*back_list != NULL)
   PopBacklist(back_list);
}
```

```
/* Warning Message for allocation error duing PushBacklist */
void PrintMemoryWarning()
{
 printf("WARNING: Error encountered while trying to dynamically allocate memory.\n");
 printf("This failure indicates system memory resources are dangerously low,\n");
 printf("which could be caused by a number of reasons.  You may wish to exit\n");
 printf("the application and try again later...\n");
}

/*
 * Function which handles jumps to jpeg, giff and xbm formatted files
 * Passes filename as command line argument to xview
 */
void CallPictureViewer(filename)
 char * filename;
{
 char sysCommand[100];

 sysCommand[0]='\0';
 strcat(sysCommand, "xv ");
 strcat(sysCommand, filename);
 system(sysCommand);
}

/* Function which handles hypertext links to sound files */
void CallAudioPlayer(filename)
 char * filename;
{
 printf("Audio capabilities currently not implemented.\n");
 printf("Unable to execute link as requested.\n");
  }

/*
 * Function which will handle jumps to postscript files.  (Currently not implemented)
 * Passes filename as command line argument to ghostview
 */
void CallPostScriptViewer(filename)
 char * filename;
{
 char sysCommand[100];

 sysCommand[0]='\0';
 strcat(sysCommand, "gv ");
 strcat(sysCommand, filename);
  system(sysCommand);
}
```

QUIZ5.H

```
/*****************************************************************************
Name: quiz5.h  Header File for default quiz application for ILE
Author: Jim Delani

Description: Header file for default quiz portion provide with the intelligent learning environment. This does
not create an elaborate student model, nor maintain any information on the student. Simply provides a
mechanism for tutorial authors to add quiz functionality without having to do work themselves. Requires
both a question file and an answer file in order to work properly. Question files must have an extension of
"quiz", as in "filename.quiz", while the solution file must be have the same base filename, but an extension
of "soln", as in the following example:
   Quiz question file ==> section1.quiz
   Quiz solution file ==> section1.soln

Hypertext jumps that indicate answers to question must be labeled as follows: <A HREF=##1_A> , where
the '##' flags link as a quiz answer, and the '1_A' is interpreted as meaning "User has selected Question 1,
Answer A". Currently allows up to 10 questions, and each question can have up to four distractors. Note
that questions and answers should match exactly, and we do not guarentee results of trying to evaluate non-
existent questions. The quiz function opens up a separate window and takes exclusive control over the system.
Quiz questions and answers appear as normal hypertext jumps and are only distinguishable by the context in
which they appear.
*****************************************************************/
#ifndef HTML_QUIZ__
#define HTML_QUIZ__

/*
 * Public function for starting default quiz. Note that function expects fully specified filename (must include
 * path), and looks for solution file in same directory.
 */
void StartQuiz();

/* Callback functions needed from within quiz portion */
static void SignalAnswer();
static void exit_cb();
static void about_cb();
static void really_exit_cb();
static void cancel_cb();

/* For maintaining information on each question */
typedef struct _Quiz_Struct {
  char Question[10];
  char Correct;
  String MessageA;
  String MessageB;
  String MessageC;
  String MessageD;
} QUIZ_STRUCT;

static String ExitQuizConfirm =
"Do you really want to exit?\n\n";
```

```
static String AboutQuizVersion =
"Preliminary Release of \n\
  the QUIZZER!\n\n\
Created by Jim Delani\n";

#endif
```

QUIZ5.C

```
/*************************************************************************
Name:    quiz5.c  Default Quiz Application for ILE
Author:  Jim Delani

Description:  Provides a Quiz interface mechanism in support of the intelligent  learning environment. Uses
the hypertext capabilities of the HTML widget to provide explanations as to why answers are correct or
incorrect. Starting point of quiz function is StartQuiz, which expects fully specified filename as its only
parameter. The path information is critical, since the function uses this information in conjunction with the
filename information to read the solution file as well. For example, a filename of "/root/tmp/section1.quiz"
will have a corresponding solution file of "/root/tmp/section1.soln" (or else this function will abort the quiz
application. Note that the format of the solution file is fairly rigid, which was inteneded to make the parsing
process a little easier.  Format of solution file is as follows:
----------  Beginning of  Example  ----------
QUEST_1 = C
ANS_A
message explaining why A not correct goes here
END_ANS
ANS_B
message explaining why B not correct goes here
END_ANS
ANS_C
message explaining why C IS CORRECT goes here
END_ANS
ANS_D
message explaining why D not correct goes here
END_ANS
END_QUEST
----------  End of Example  ----------
The number of questions is limited to 10, and each question may have up to four possible answers. Note that
line 1 of the example states that Reply C is the correct answer to this question.  All of the QUEST_, ANS_,
and END_ANS tokens must appear in the leftmost column in order to be recognized (beginning of line)
*************************************************************************/
#include "tutor5.h"
#include "utils5.h"
#include "quiz5.h"
#define ARRAYSIZE 10

extern Widget TopLevel;
extern Display *display;

static Widget QuizShell, QuizForm, ExitQuiz, QuizInfo;
```

```
static Widget QuizW, QuizHome, QuizBack, menu_box;
static Widget CallingWidget, pshell;
static int CallingID, NumWrong=0, NumRight=0;
static char *quizText=NULL;
static char HomePage[100];
static char * current_href;
static HREF_STRUCT *BackList=NULL;
static QUIZ_STRUCT CurrentInfo;
static QUIZ_STRUCT * MessageArray[ARRAYSIZE];

static void Home_Callback();
static void Back_Callback();
static int NumQuestions = 0;

/* Parses the solution file and loads into the Message array */
static Boolean load_solution(filename)
 char *filename;
{
 int arrayIndex=0, bytes_read = 0;
 FILE *filePtr;
 long int startPos, endPos, AnswerLength, tempPos;
 char *c, *NewAnswer, MsgNum;
 char line[100];
 char * seps = " \t\n=";
 char * field;
 QUIZ_STRUCT * tempAnswerPtr;
 Boolean Done = False;

 if((filePtr=fopen(filename, "r"))==NULL)
  return(False);
 while ( (fgets(line, sizeof(line), filePtr) != NULL) &&  (strncmp(line, "QUEST", 5)) )
   { ; /* Ignore any comments or prolog */ }
 if (feof(filePtr))
  return False;

while (!Done)
 {
  tempAnswerPtr = (QUIZ_STRUCT *)malloc(sizeof(QUIZ_STRUCT));
  if(tempAnswerPtr == NULL)
    {
     PrintMemoryError();
     return False;
    }

 /* Now need to determine question number and correct answer.  Currently do not use this, but will be
  necessary if desire to create elaborate student model.  Note that if correct answer is not provided there is
  no way a student can match the correct answer! */
  if((c = strchr(line, '=')) != NULL)
    {
     *c = '\0';
     c++;
```

```
  while(*c == ' ')
    c++;
  tempAnswerPtr->Correct = *c;
  }
else
  tempAnswerPtr->Correct = '\0';

strcpy(tempAnswerPtr->Question,line);
fgets(line, sizeof(line), filePtr);

/* This loop processes all of the answers to a particular question.  note that a maximum of four answers
   is allowed for any question.*/
while (strncmp(line, "END_QUEST", 9))   /* Returns Zero if EQUAL */
  {
  /* Skip over any blank lines until get to start of answer*/
  while (strncmp(line, "ANS", 3))   /* Returns Zero if EQUAL */
    {
    fgets(line, sizeof(line), filePtr);
    }
  /* Format for answer is "ANS_X", where X represents answer */
  c = (char *)strchr(line, '_');
  c++;
  MsgNum = *c;
  /* We are now at beginning of answer */
  startPos = ftell(filePtr);
  fgets(line, sizeof(line), filePtr);
  /* Now need to find out how big it is */
  while (strncmp(line, "END_ANS", 7))   /* Returns Zero if EQUAL */
    {
    endPos = ftell(filePtr);
    fgets(line, sizeof(line), filePtr);
    /* Note that we consume EndAns flag */
    }
  tempPos = ftell(filePtr);
  AnswerLength = endPos - startPos;
  NewAnswer = (char *)malloc(AnswerLength + 5);
  if (NewAnswer == (char *)NULL)
    {
    PrintMemoryError();
    return False;
    }
  fseek(filePtr, startPos, SEEK_SET);
  bytes_read = fread(NewAnswer, 1, AnswerLength, filePtr);
  NewAnswer[AnswerLength -1] = '\0';
  if (bytes_read < AnswerLength)
    {
    printf("Unknown read error occured while reading solution file.\n");
    return False;
    }
```

```
    switch (MsgNum)
        {
        case 'A': tempAnswerPtr->MessageA = (String)NewAnswer; break;
        case 'B': tempAnswerPtr->MessageB = (String)NewAnswer; break;
        case 'C': tempAnswerPtr->MessageC = (String)NewAnswer; break;
        case 'D': tempAnswerPtr->MessageD = (String)NewAnswer; break;
        default:  ;
        }

      fseek(filePtr, tempPos, SEEK_SET);
      startPos = endPos = tempPos;

   fgets(line, sizeof(line), filePtr);
   /* Looking for one of two Keys here, and need to consume whitespace */
   while ( (strncmp(line, "ANS", 3)) && (strncmp(line, "END_QUEST", 9)) )
     {
      fgets(line, sizeof(line), filePtr);
     }
    } /* while  NOT EndQuest */
   /* Note that EndQuest loop consumes Flag it is looking for */
  MessageArray[arrayIndex] = tempAnswerPtr;
  tempAnswerPtr = NULL;
  arrayIndex ++;

  fgets(line, sizeof(line), filePtr);
  while ( (!feof(filePtr)) && (strncmp(line, "QUEST", 5)) )
    {
     fgets(line, sizeof(line), filePtr);
     /* ignore all blank spaces , need to get to next header */
    }
  if  (feof(filePtr) || (arrayIndex >= ARRAYSIZE) )
    Done = TRUE;

 } /* while (!Done)  */
fclose(filePtr);
return True;
}

/*
 * Callback needed to be provided when instantiating HTML widget  NOTE: Currently Quiz only allows
 * jumps within same filename.  Maintaining two pointers, c and new_href, since may want to allow multiple
 * files, then will have to maintain filename info for back list.  Easier to implement if already set up.
 */
static void Anchor_callback(widget, client_data, data)
 Widget widget;
 XtPointer client_data;
 WbAnchorCallbackData *data;
{
 int newID, Index;
 char temp[100];
 char * c, Solution, UserAnswer;
```

```
char * new_href = strdup(data->href);

temp[0] = '\0';
/* For the quiz, selectiag an answer to a question returns an anchor  with two '#' symbols.  Any other
   anchor represents a legitimate  jump, whether to another question, top of page or what have you.
   Currently the ONLY legal jumps are within the same page. */
if(strncmp(new_href, "##", 2)==0)
 {
 /* format of answer = ##Num_Letter  */
 c = (char *)strrchr(new_href, '_');
 *c = '\0';
 UserAnswer = *(c+1);
 c--;
 Index = (atoi(c) - 1);
 Solution = (MessageArray[Index]->Correct);

 if(Solution  == UserAnswer)
  {
   NumRight++;
   SignalAnswer(True, Index, UserAnswer);
   return;
  }
 else
  {
  NumWrong++;
  SignalAnswer(False, Index, UserAnswer);
  return;
  }
 }
else if(*new_href == '#')
  {
  c= new_href;
  c++;
  strcat(temp, c);
  newID = HTMLAnchorToId(QuizW, temp);
  if(!newID)
    {
     printf("Error attempting local jump. \n");
     printf("Unable to complete hypertext link as requested.\n");
    }
  else
    {
     HTMLGotoId(QuizW, newID);
     CallingID = (*data).element_id;
     if (!PushBacklist(&BackList, current_href, CallingID))
       PrintMemoryWarning();
     XtVaSetValues(QuizHome, XtNsensitive, True, NULL);
     XtVaSetValues(QuizBack, XtNsensitive, True, NULL);
     return;
    }
  }
```

```
 else
   printf("Quiz currently has single file capabilities only.\n");
   printf("Ignoring request to load external file.\n");
}

/*Function that responds to user selecting BACK button from options. */
static void Back_Callback(widget, client_data, call_data)
 Widget widget;
 XtPointer client_data, call_data;
{
  if(BackList==NULL)
    return;
  else
    {
    HTMLGotoId(QuizW, BackList->CallingId);
    PopBacklist(&BackList);
    if (BackList == NULL)
      XtVaSetValues(QuizBack, XtNsensitive, False, NULL);
    }
}

/*
 * Callback function to respond to user clicking on the HOME button at the  top of the HTML form.
 * Currently clears the entire BackList, similar to a reset or start over.  Loads Home Page (first page viewed
 */
static void Home_Callback(widget, client_data, call_data)
  Widget widget;
  XtPointer client_data, call_data;
{
  char *target = strdup(HomePage);
  char *textPtr;

  /* Note that even if same file currently loaded need to reread file  since HomePage does not have
     CallingID associated with it */
  if(loadAsciiFile(HomePage, &textPtr))
    {
    ClearBacklist(&BackList);
    HTMLSetText(QuizW, quizText, "\0", "\0", 0, NULL, NULL);
    }
  else
    free(target);

  /* We need to disable the Home and Back buttons now */
  XtVaSetValues(QuizHome, XtNsensitive, False, NULL);
  XtVaSetValues(QuizBack, XtNsensitive, False, NULL);
}

/*
 *  Free up allocated memory and exit application  NOTE: attempting to free memory for Messages
 * causes segmentation errors when re-starting the quiz application.  Appears to reclaim the memory on
 * its own during preliminary testing.
```

```
*/
void ShutDownQuiz()
{
 QUIZ_STRUCT * tempAnswerPtr;
 int i;

 /* Free up Quiz questions */
 if(quizText!=NULL)
  {
   free(quizText);
   quizText=NULL;
  }
 if(current_href!=NULL)
  {
   free(current_href);
   current_href=NULL;
  }

 XtPopdown(QuizShell);
 XtDestroyWidget(QuizShell);
}

/*
 * Starting point of quiz application. Calls Load text and solution functions prior to creation of
 * application shells.
 */
void StartQuiz(filename)
 char *filename;
{
 int n;
 char * SolutionFile;
 char *c, temp[100];
 int horizDist = 20;
 int MinHeight = 300;
 int MinWidth = 400;
 int ButtonWidth = 70;
 Widget FormB, FormW;
 if(!loadAsciiFile(filename, &quizText))
  {
   PrintStartError(filename);
   return;
  }
 else
  strcpy(HomePage, filename);

 /* Need to maintain current href for compatibility with other subsystems.
    May also be helpful for future expansion */
 current_href=strdup(filename);

 strcpy(temp, filename);
 if( (c=(char *)strrchr(temp, '.')) != NULL)
```

```
  {
  c++;
  *c = '\0';
  strcat(temp, "soln");
  if(!load_solution(temp))
    {
     PrintStartError(temp);
     return;
    }
  }
else
  {
  PrintStartError(temp);
  return;
  }

QuizShell=XtVaCreatePopupShell("ITS Quiz", topLevelShellWidgetClass, TopLevel,
                                XtNallowShellResize, True, XtNminHeight, MinHeight,
                                XtNminWidth, MinWidth, NULL);

QuizForm=XtVaCreateManagedWidget("htmlform", formWidgetClass, QuizShell, NULL);

menu_box = XtVaCreateManagedWidget("formB", formWidgetClass, QuizForm,
                                    XtNheight, 50, XtNborderWidth, 0, XtNwidth, MinWidth,
                                    XtNleft, XtChainLeft, XtNtop, XtChainTop,
                                    XtNright, XtChainRight, NULL);

ExitQuiz=XtVaCreateManagedWidget("Exit", commandWidgetClass, menu_box,
                                  XtNhorizDistance, horizDist, XtNwidth, ButtonWidth,
                                  XtNlabel, " Exit ", XtNleft, XtChainLeft,
                                  XtNtop, XtChainTop, XtNbottom, XtChainBottom, NULL);
XtAddCallback(ExitQuiz, XtNcallback, exit_cb, NULL);

QuizBack=XtVaCreateManagedWidget("Back", commandWidgetClass, menu_box,
                                  XtNsensitive, False, XtNhorizDistance, horizDist,
                                  XtNfromHoriz, ExitQuiz, XtNwidth, ButtonWidth,
                                  XtNlabel, " Back ", XtNtop, XtChainTop,
                                  XtNbottom, XtChainBottom, NULL);
XtAddCallback(QuizBack, XtNcallback, Back_Callback, NULL);

QuizHome=XtVaCreateManagedWidget("Home", commandWidgetClass, menu_box,
                                  XtNsensitive, False, XtNhorizDistance, horizDist,
                                  XtNfromHoriz, QuizBack, XtNwidth, ButtonWidth,
                                  XtNlabel, " Home ", XtNtop, XtChainTop,
                                  XtNbottom, XtChainBottom, NULL);
XtAddCallback(QuizHome, XtNcallback, Home_Callback, NULL);

QuizInfo=XtVaCreateManagedWidget("Info", commandWidgetClass, menu_box,
                                  XtNhorizDistance, horizDist, XtNfromHoriz, QuizHome,
                                  XtNwidth, ButtonWidth, XtNlabel, " Info ",
                                  XtNtop, XtChainTop, XtNbottom, XtChainBottom, NULL);
```

```
 XtAddCallback(QuizInfo, XtNcallback, about_cb, NULL);

 QuizW=XtVaCreateManagedWidget("html", htmlWidgetClass, QuizForm,
                                        XtNfromVert, menu_box, XtNbottom, XtChainBottom,
                                        XtNheight, 300, XtNwidth, 400, NULL);
 XtAddCallback(QuizW, WbNanchorCallback, Anchor_callback, (XtPointer)0);

 XtRealizeWidget(QuizShell);

 HTMLSetText(QuizW, quizText, "\0", "\0", 0, NULL, NULL);
 XtPopup(QuizShell, XtGrabExclusive);
} /* End Start Quiz Function */

/*
 * Callback to provide user with opportunity to confirm or cancel EXIT.  Exit application does not have
 * menu and may be triggered inadvertently, so should provide user opportunity to abort the exit
 */
static void exit_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data;
  XtPointer call_data;
{
  Widget okB, exit_dialog,cancelB;
  Position x, y;
  Dimension width, height;

  pshell  = XtVaCreatePopupShell("confirm", transientShellWidgetClass, TopLevel,
                                      XtNwidth, 300, XtNheight,150, XtNlabel, ExitQuizConfirm, NULL);

  exit_dialog = XtVaCreateManagedWidget("Confirm", dialogWidgetClass, pshell,
                                            XtNlabel, ExitQuizConfirm, NULL);

  okB = XtVaCreateManagedWidget(" Exit ", commandWidgetClass, exit_dialog,
                                     XtNlabel, " OK ", XtNhorizDistance, 40, NULL);
  XtAddCallback(okB, XtNcallback, really_exit_cb, NULL);

  cancelB = XtVaCreateManagedWidget("Cancel", commandWidgetClass, exit_dialog,
                                          XtNhorizDistance, 40, XtNright, XtChainRight, NULL);
  XtAddCallback(cancelB, XtNcallback, cancel_cb , NULL);

  XtVaGetValues(QuizW, XtNwidth, &width, XtNheight, &height, NULL);
  XtTranslateCoords(QuizW, (Position) width/2, (Position) height/2, &x, &y);
  XtVaSetValues(pshell, XtNx, x - (width/2) , XtNy, y - (height/2), NULL);

  XtPopup(pshell, XtGrabExclusive);
}

/* Popup dialog with message explaining why answer chosen is correct or incorrect. */
static void SignalAnswer(Correct, Index, Alpha)
  Boolean Correct;
  int Index;
```

```
  char Alpha;
{
  Widget okB, info_dialog, formW, explanation;
  String Message;
  Position x, y;
  Dimension width, height;

 switch (Alpha)
    {
    case 'A': Message = MessageArray[Index]->MessageA; break;
    case 'B': Message = MessageArray[Index]->MessageB; break;
    case 'C': Message = MessageArray[Index]->MessageC; break;
    case 'D': Message = MessageArray[Index]->MessageD; break;
    default: ;
    }

  pshell  = XtVaCreatePopupShell("answer", transientShellWidgetClass, TopLevel,
                                      XtNwidth, 320, XtNheight,230, NULL);

  formW = XtVaCreateManagedWidget("AnsForm", formWidgetClass, pshell,
                                          XtNborderWidth, 0, XtNwidth, 320, XtNheight, 200, NULL);

  explanation = XtVaCreateManagedWidget("Explain", asciiTextWidgetClass, formW,
                                                XtNscrollVertical, XawtextScrollWhenNeeded,
                                                XtNwrap, XawtextWrapWord, XtNstring, Message,
                                                XtNheight, 180, XtNwidth, 320,
                                                XtNleft, XtChainLeft, XtNright, XtChainRight, NULL);

  okB = XtVaCreateManagedWidget("  OK  ", commandWidgetClass, formW,
                                       XtNfromVert, explanation, XtNhorizDistance, 120, NULL);
  XtAddCallback(okB, XtNcallback, cancel_cb, NULL);

  XtVaGetValues(QuizW, XtNwidth, &width, XtNheight, &height, NULL);
  XtTranslateCoords(QuizW, (Position) width/2, (Position) height/2, &x, &y);

  if(Correct)
    {
    XtVaSetValues(pshell, XtNx, x , XtNy, y - (height/2), NULL);
    }
  else
    {
    XtVaSetValues(pshell, XtNx, x - (width/2), XtNy, y - (height/2), NULL);
    }
  XtPopup(pshell, XtGrabExclusive);
}

/*
 * Callback associated with our Info button which provides user with popup dialog describing our
 * implementation.  Typical "Look what I did" type dialog
 */
static void about_cb(w, client_data, call_data)
```

```
  Widget w;
  XtPointer client_data;
  XtPointer call_data;
{
  Widget okB, about_dialog;
  Position x, y;
  Dimension width, height;
  pshell  = XtVaCreatePopupShell("About...", transientShellWidgetClass, TopLevel,
                                    XtNwidth, 300, XtNheight,200, NULL);

  about_dialog = XtVaCreateManagedWidget("about", dialogWidgetClass, pshell,
                                                XtNlabel, AboutQuizVersion, NULL);

  okB = XtVaCreateManagedWidget(" OK ", commandWidgetClass, about_dialog,
                                        XtNhorizDistance, 120, NULL);
  XtAddCallback(okB, XtNcallback, cancel_cb, NULL);

  XtVaGetValues(QuizW, XtNwidth, &width, XtNheight, &height, NULL);
  XtTranslateCoords(QuizW, (Position) width/2, (Position) height/2, &x, &y);
  XtVaSetValues(pshell, XtNx, x - (width/2), XtNy, y - (height/2), NULL);

  XtPopup(pshell, XtGrabExclusive);
}

/* Callback used to remove any of the popub dialogs attached to pshell */
static void cancel_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data, call_data;
{
 XtPopdown(pshell);
}

/* Callback associated with OK button of exit_cb.  Go ahead and close up shop  */
static void really_exit_cb(w, client_data, call_data)
  Widget w;
  XtPointer client_data, call_data;
{
 XtPopdown(pshell);
 ShutDownQuiz();

}
```

REFERENCES

[ABCL90] Anderson, J., Boyle, C., Corbett, A., and Lewis, M., "Cognitive Modeling and Intelligent Tutoring", in W. Clancey and E. Soloway (Ed.), *Artificial Intelligence and Learning Environments*, The MIT Press, 1990.

[BiLe91] Bielawski, L., and Lewand, R., *Intelligent Systems Design*, John Wiley and Sons, Inc., 1991.

[Blum92] Blum, B., *Software Engineering: A Holistic View*, Oxford University Press, 1992.

[Camp90] Campbell, L., *An Intelligent Tutor System for Visual Aircraft Recognition*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1991.

[Clan87] Clancey, W., *Knowledge Based Tutoring: The Guidon Program*, The MIT Press, 1987.

[Delo91] DeLooze, L., *ITS Ada: An Intelligent Tutoring System for the Ada Programming Language*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1991.

[GaWe89] Gause, D., and Weinberg, G., *Exploring Requirements: Quality Before Design*, Dorset House Publishing, 1989.

[GiRi94] Giarratano, J., and Riley, G., *Expert Systems: Principles and Programming*, PWS-Kent Publishing Co., 1994.

[Hamm93] Hamming, R., EC4000 and OS3001 Lecture Series, Naval Postgraduate School, Monterey, CA, 1993-1994.

[HeFe92] Heller, D., and Ferguson, P., *XMotif Programming Manual (for OSF/Motif Release 1.2)*, O'Reilly & Associates, Inc., 1992.

[Hill77] Hill, N., *Success Through a Positive Mental Attitude*, Prentice-Hall, Inc., 1977.

[HiHa93] Hix, D., and Hartson, H., *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley and Sons, Inc., 1993.

[Hopp92] Hoppe, W., *Cognitive Modeling and the Evolution of the Student Model in Intelligent Tutoring Systems*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1992.

[JoRe92] Johnson, E., and Reichard, K., *X Window Applications Programming*, Management Information Services, Inc., 1992.

[Kear87] Kearsley, G., *Artificial Intelligence and Instruction*, Addison-Wesley Publishing Company, 1987.

[LeMB92] Levine, J., Mason, T., and Brown, D., *Lex & Yacc*, O'Reilly and Associates, Inc., 1992.

[Nasa93a] NASA, *CLIPS Reference Manual, Volume I: Basic Programming Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX, 1993.

[NyOr92] Nye, A., and O'Reilly, T., *X Toolkit Intrinsics Programming Manual (OSF/ Mofif 1.2 Edition)*, O'Reilly & Associates, Inc., 1992.

[Pape80] Papert, S., *Mindstorms*, Basic Book, 1980.

[Reps89] Reps, T., and Teitelbaum, T., *The Synthesizer Generator: A System for Constructing Language-Based Editors*, Springer-Verlag New York Inc., 1989.

[Skin86] Skinner, M., *Design and Evaluation of a Computer-Assisted Instructional Program on Concepts in Applied Behavioral Analysis*, University Microfilms International, 1986.

[Weng87] Wenger, E., *Artificial Intelligence and Tutoring Systems*, Morgan Kaufman Publishers, Inc., 1987.

[Wool92] Woolf, B., "AI in Education", in S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence, Second Edition*, John Wiley and Sons, Inc., 1992.

[YoPe92] Young, D., and Pew, J., *The X Window System: Programming & Applications with Xt, (Open-Look Edition)*, Prentice-Hall, Inc., 1992.

[Your89] Yourdon, E., *Modern Structured Analysis*, Prentice-Hall Inc., 1989.

BIBLIOGRAPHY

[Bark91] Barkakati, N., *X Window System Programming*, Sams Publishing, 1991.

[Barn89] Barnes, J., *Programming in Ada*, Third Edition, Addison-Wesley Publishing Co., 1989.

[Beiz90] Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, 1990.

[BoGa91] Boose, J., and Gaines, B., *The Foundations of Knowledge Acquisition*, Academic Press Limited, 1991.

[Brum94] Brumbaugh, D., *Object-Oriented Development: Building CASE Tools with C++*, John Wiley and Sons, Inc., 1994.

[CoNi93] Coad, P., and Nicola, J., *Object Oriented Programming*, Prentice-Hall, Inc., 1993.

[CoYo91] Coad, P., and Yourdon, E., *Object Oriented Analysis*, Prentice-Hall Inc. 1991.

[CuGR92] Cutler, E., Gilly, D., and O'Reilly, T., *The X Window System in a Nutshell*, O'Reilly and Associates, Inc., 1992.

[Ecke93] Eckel, B., *C++ Inside and Out*, Osborne McGraw-Hill, Inc., 1993.

[Fire92] Firesmith, D., *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, John Wiley and Sons, Inc., 1992.

[FiLe91] Fisher, C., and LeBlanc, R., *Crafting a Compiler With C*, Benjamin/ Cummings Publishing Company, Inc., 1991.

[Flan92] Flanagan, D., *X Toolkit Intrinsics Reference Manual (for X11 Release 4 and Release 5)*, O'Reilly & Associates, Inc., 1992.

[Giar93] Giarratano, J., *CLIPS User Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX, 1993.

[Gonz91] Gonzalez, D., *Ada Programmer's Handbook and Language Reference Manual*, Benjamin Cummings Publishing Company, Inc., 1991.

[Gonz93] Gonzalez, A., *The Engineering of Knowledge Based Systems*, Prentice-Hall Inc., 1993.

[Good92] Goodwin, M., *User Interfaces in C and C++*, MIS:Press, 1992.

[HaSt91] Harbison, S., and Steele, G., *C: A Reference Manual*, Prentice-Hall, Inc., 1991.

[Lafo91] Lafore, R., *Object Oriented Programming in Turbo C++*, Waite Group Press, 1991.

[Lee89] Lee, Y., *A Knowledge Based Approach to Program Debugging*, Technical Paper, NPS52-89-060, Naval Postgraduate School, Monterey CA, 1989.

[LiDe89] Liebowitz, J., and DeSalvo, A., *Structuring Expert Systems*, Prentice-Hall Inc., 1989.

[Loca93] Local Proceedings, AADEBUG'93, 1st International Workshop on Automated and Algorithmic Debugging, Linkoping University, Linkoping, Sweden, 3-5 May, 1993.

[Luca92] Lucas, J., *The C++ Programmer's Handbook*, Prentice-Hall, Inc., 1992.

[Nasa93b] NASA, *CLIPS Reference Manual, Volume II: Advanced Programming Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX, 1993.

[Nasa93c] NASA, *CLIPS Reference Manual, Volume III: Interfaces Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX, 1993.

[Niel90] Nielson, J., *Hypertext and Hypermedia*, Academic Press, Inc., 1990.

[NyOr93] Nye, A., and O'Reilly, T., *X Toolkit Intrinsics Programming Manual (for Version 11)*, O'Reilly & Associates, Inc., 1993.

[Paga91] Pagan, F., *Partial Computation and the Contruction of Language Processors*, Prentice-Hall Inc., 1991.

[Perr83] Perry, W., *A Structured Approach to Systems Testing*, QED Information Sciences, Inc., 1983.

[Prat91] Prata, S., *C++ Primer Plus: Teach Yourself Object Oriented Programming*, Waite Group Press, 1991.

[Schi92] Schildt, H., *The Craft of C: Take-Charge Programming*, Osborne McGraw-Hill, 1992.

[ScKe89] Schneiderman, B., and Kearsley, G., *Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information*, Adison-Wesley Publishing Company, 1989.

[Stit92] Stitt, M., Debugging: *Creative Techniques and Tools for Software Repair*, John Wiley and Sons, Inc., 1992.

[Stit93] Stitt, M., *Building Custom Software Tools and Libraries*, John Wiley and Sons, Inc., 1993.

[Stro91] Stroustrup, B., *The C++ Programming Language*, Addison Wesley Publishing Company, 1991.

[Vall92] Valley, J., *C Programming for UNIX*, SAMS Publishing, 1992.

[Youn92] Young, D., *Object Oriented Programming with C++ and OSF/Motif*, Prentice-Hall Inc., 1992.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5101	2
3.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Director, Training and Education MCCDC, Code C46 1019 Elliot Road Quantico, VA 22134-5027	1
5.	Professor Yuh-jeng Lee, Code CS/Le Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	5
6.	Professor Mantak Shing, Code CS/Sh Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	2
7.	Captain James M. Delani Jr. Marine Corps Systems Command 2033 Barnett Ave., Suite 315 Quantico, VA 22134-5010	2