A menu driven, user friendly interface to UNIX

Mary Hayward
A MENU DRIVEN, USER FRIENDLY INTERFACE TO UNIX*

Mary Hayward

a thesis submitted to the faculty
of the School of Computer Science and Technology,
Rochester Institute of Technology
in partial fulfillment of the
requirements of the degree of
Master of Science

Approvals: 

Peter H. Lutz (chairman)

Henry A. Etlinger

Warren R. Carithers

* UNIX is a Trademark of Bell Telephone Laboratories, Inc.
Title of Thesis: A MENU DRIVEN, USER FRIENDLY INTERFACE TO UNIX

I _______________________ hereby grant permission to the Wallace Memorial Library, of R.I.T., to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. Peter E. Lutz for his continued interest and assistance in this thesis project, and for his guidance throughout the entire master's degree program.

Appreciation is also extended to Mr. Henry A. Etlinger and Mr. Warren R. Carithers for their assistance in the completion of this project.

I would like to thank my husband, Craig and my children who have supported and encouraged me in my educational endeavors.
Abstract

A menu-driven, user-friendly interface to UNIX

An interface was developed to provide an environment where an inexperienced UNIX user can work effectively within the UNIX operating system. A by-product of the interface results in the user learning a subset of commonly used UNIX commands. The interface incorporates a menu structure which presents the available UNIX functions to the user. When the user selects a function, the interface prompts him for the required arguments, echoes the corresponding UNIX command and executes the command.
TABLE OF CONTENTS

1. Introduction .......................................................... 1
   1.1. Demand for user friendly interfaces ............ 1
   1.2. Description of the UNIX operating system .......... 3
   1.3. Recluse nature of UNIX ......................... 7
   1.4. Statement of thesis project .................. 8

2. Review of user friendliness ................................. 9
   2.1. Importance of user friendly software ...... 9
   2.2. Principles of human communication applied to man-machine dialogue ............ 10
       2.2.1. Expectation, prediction, and familiarity ........ 11
       2.2.2. Implication ............................. 13
       2.2.3. Experimentation ......................... 15
       2.2.4. Motivation ............................. 16
   2.3. Design features found in user friendly interfaces ......................... 17
       2.3.1. Mental models .............................. 17
       2.3.2. User expertise ............................. 20
       2.3.3. Mental load .............................. 22
       2.3.4. Response time ............................ 25
       2.3.5. User feedback ............................ 26
CHAPTER 1

INTRODUCTION

1. Introduction

1.1. Demand for user friendly interfaces

Man must become the prime focus of system design. The computer is there to serve him, to obtain information for him, and to help him do his job. The ease with which he communicates with it will determine the extent to which he uses it. ... To be effective, systems will have to be designed from the outside in [MAFT73].

Computer usage has increased dramatically over the past few years. We can expect an even greater integration of the computer into man's life, both in his work and home activities. As the number of computer users increases the characteristics of the user will diversify. Computer usage is no longer solely associated with programmers and technicians. Instead, individuals from all walks of life are actively using computers. It is no longer reasonable to expect a user to have computer training to become an effective computer user.

The man-machine interface is becoming a critical area in computer systems. The interaction between man and the computer gains significance not only in applications software but also in operating system design, programming...
languages and documentation. 'In today's complex world, man and machine work interactively. The 'system' is a combination of both' [SIMPE2]. An interactive computer system must be designed to meet the functional requirements of the user and yet remain efficient and easy to learn. A powerful functional capability has little value if it is so difficult that the user chooses to disregard it [MORA68].

The software a user interacts with can be seen as having two orientations: system-oriented software and user-oriented software. System-oriented software includes commands which perform file maintenance, commands for accessing utilities, and commands for use of assemblers, compilers and editors. It hides the many interface details from the user in order to make the system easier to access and use. The user must initiate the commands needed and remains responsible for the results he initiates. In contrast, user-oriented software is concerned with the user and his needs rather than the system and its needs.

The operating system is the interface between the hardware and the user. An individual who solely runs applications software such as payroll packages or computer aided instruction, will have minimal interaction with the operating system. His knowledge may be limited to the
commands for logging into his system and running his programs. For other users the interaction with the operating system will be of greater magnitude. For these users the dialogue between the user and the operating system is a significant factor in their productivity and enjoyment of the computer system.

1.2. Description of the UNIX operating system

The UNIX operating system was developed at Bell Laboratories by Ken Thompson and Dennis Ritchie. Thompson and Ritchie originally designed UNIX for their own use as an aid in software research. Word spread quickly regarding the power and flexibility of the UNIX operating system. 'The general sense of the word power is usually something like 'being able to do more with less effort.' Applied to software such as an operating system it translates into convenience of use'' [SWAIS2]. Research and academic institutions were interested in implementing UNIX. In 1973 Western Electric made UNIX available to universities and other nonprofit organizations. UNIX has since been updated by Bell Laboratories and other organizations [GRFDE81]. UNIX growth has been rapid; according to some, it may even become the standard operating system for mini and micro computer systems [SWAIS2].
UNIX is powerful and flexible without resorting to a complex system. An experienced programmer can quickly understand the entire operating system. The following UNIX design goals described by Greenburg provide a basis for a discussion of the strengths and weaknesses of UNIX:

1. "Support a basic functionality within the operating system itself, relying on the normal user programs to provide sophistication. Features such as login/ logout and line printer queuing are implemented as normal user programs instead of operating system functions. This approach reduces the complexity of the system.

2. "Generality - having a single method serve a variety of purposes." This generality enables one to program easily along with providing flexibility and extensibility.

3. "Accomplish large tasks by combining several smaller tasks whenever possible." Again this design goal encourages a modular approach to complex programs and keeps the underlying operating system simpler [GREE81].

It is also beneficial to note several of the features of UNIX as described by one of its designers Ken Ritchie.

1. File structure

The UNIX operating system is not tied down to the notion of a record. Each file is regarded as a featureless, randomly accessible sequence of bytes. The system conceals physical properties of the device on which the file is stored. File space is not preallocated and system calls to read and write have only one form.
2. Structure of the file system

"The UNIX file system is a recursive structure operating from a root directory. The root directory contains the names of files and subdirectories; the subdirectories contain names of files and other subdirectories, etc." [GREE81]. The structure has the form of a tree. It is inexpensive to implement and efficient in search time.

3. Input/output devices

UNIX has tried to remove differences between disk files and I/O devices including terminals. The user's terminal is important in inter-user communication and redirection of input and output to other files.

4. User interface

UNIX uses a 'shell' as a means of communication between the user and the operating system. The shell interprets user commands. The shell is not part of the operating system. It can easily be replaced with another shell program thus providing a new interface to the user. Individual command structure is simple and regular. Command arguments are separated by white spaces and symbols such as '<', '>', or '>>'. The shell parses the commands and passes them on as separate strings, thus the user is not required to learn a complicated syntax involving commas, semicolons and parenthesis. One of the most important aspects of UNIX programming is the concept of pipes. A pipe is an open file connecting two processes. The shell provides a notation for pipes that facilitates their use and enables interconnection of programs to be a commonplace activity.

5. Environment of a process

"New processes are created by the fork operation, which creates a child whose code and data are copied from the parent. The child inherits the
open files of the parent and executes asynchronously with it unless the parent explicitly waits for termination of the child." When the shell executes a command, each command becomes a separate process. Communication between processes is handled by sending "signals".

6. Reliability

UNIX reliability has generally been good, however, running out of swap space or an irrecoverable I/O error during swapping will cause the system to crash. A reliable system should not lose or corrupt files if the system crashes. UNIX does not take any special precautions to keep user files intact. UNIX is not very tolerant to hardware inconsistencies and problems.

7. Security

UNIX was designed in a open environment where security was not an issue. Consequently security problems do exist within UNIX. Security shells have been developed for applications that require greater security [SWAI82].

8. Use of a high-level language

UNIX is written primarily in the C language. Because the operating system is written in a high level language it is easy to modify it to meet an installation's unique needs. Of even greater significance is the portability that a high level language offers. Users are given the option of an operating system independent of the hardware on which it runs. "As the price of software rapidly out paces that of computers, the need to increase software productivity and reduce duplication of effort has become paramount" [GRIE81]. Application programs can be written to run on diverse hardware configurations [RITC78].
1.3. **Recluse nature of UNIX**

Most UNIX users are enthusiastic and pleased with the positive features UNIX includes. However, there exists a common area of concern regarding the user interface and the ease of use. Ritchie has said:

Both input and output of UNIX programs tend to be terse. This can be disconcerting especially to the beginner. The command interpreter does not remark loudly each time a program finishes normally, or announce how much time and space it took. Likewise, commands seldom prompt for missing arguments; instead, if the argument is not optional, they give at most a one line summary of their usage and terminate [RITC78].

The following examples illustrate the terse and often confusing interface of UNIX:

1. Often command names do not reflect the task to be accomplished; e.g., "cat" does not suggest by its name that it can be used to list contents of a file.

2. There are inconsistencies in abbreviations for commands; e.g., the four letter command 'date' is not abbreviated but the four letter word "copy" is abbreviated to "cp".

3. Commands exist that produce output without labels or column headings; e.g., "who" returns a one line description of users currently logged into the system but uses no column headings.

4. UNIX tends to be silent; a user does not receive recognition from the system as to his ex-
istence unless he makes an error.

The terseness of UNIX may be frustrating to the new or casual user; however, the experienced user finds it desirable. Once an individual gains experience with UNIX he appreciates the user interface. He is able to communicate quickly and efficiently and is not bothered with excess noise sent to him by the operating system.

1.4. Statement of thesis project

The above discussions of the need for user friendly interfaces, the features of UNIX and the recluse character of UNIX are brought together in this thesis project. The purpose of the thesis project was to design and implement a user friendly interface to UNIX. The interface is a program, or "shell" that enables inexperienced UNIX users to function comfortably within the UNIX environment. It also teaches the user appropriate UNIX commands to complete his desired tasks.
CHAPTER 2

REVIEW OF USER FRIENDLINESS

2. Review of user friendliness

2.1. Importance of user friendly software

Advances in hardware development have been rapid. The computer of today is a vastly different machine from the early computers. The new technology becomes an integral part of daily activities. Software too has advanced over the years; however, there remains a void in guidelines for optimal man-machine interface. "We have been so bound by technology (removing obvious annoyances of existing interfaces and getting the bandwidth up, all within the bounds of economy), that we have not addressed substantive issues of man-computer communication" [ROBES81].

Despite the current interest in user-computer interfaces the design of a good interface remains to a great extent an art, with much argument over guidelines and principles for interface design. Pertinent information, scattered throughout the literature of psychology, graphic design, linguistics, hardware design and under the general umbrella of computer science is only gradually being gathered together into survey publications for applications in computer science [ELFS81].
A common thread throughout the literature expresses a concern and realization that the man-computer interface is a fertile area for further research [PLAC81]. There is a desire to see interactive dialogue move from an art to systematic dialogue engineering [GAIN76]. Interactive systems need guidelines for dialogue enabling the system designers to avoid software that operates effectively for the designer but leaves the end user in a confused and frustrated state. The following discussion will look into the literature with the goal of better understanding man-machine interfaces. An emphasis will be placed on the principles directly related to the thesis project of the user friendly interface to UNIX.

2.2. Principles of human communication applied to man-machine dialogue

An interactive man-computer system involves a dialogue or communication between man and the computer system. "To be truly usable, a system must be compatible not only with the characteristics of human perception and action but more critically also with users' cognitive skills in communication, memory and problem solving" [CLAR81]. When human dialogue is examined, strategies and procedures are revealed. Often man-computer dialogue is designed incorporating a near-English vocabulary and syntax hoping the result is a user friendly interface. A near-English
vocabulary is not enough as it does not include the underlying strategies of human communication. Four underlying principles of human communication and how they are vital to man-machine dialogue will be examined [JONE78].

2.2.1. Expectation, prediction, and familiarity

When two people meet for the first time they incorporate past experience. Person A does not expect person B to bite him. Person C does not look at person D and then hide in a corner. Instead A, B, C, and D expect a normal introduction and greeting. It should be noted that a conversation will take place without consulting a manual describing the people involved. When we look at computer dialogue the opposite is true. Often the first thing a user encounters is a formidable manual which proceeds to show him how new and how different the computer system is; perhaps training courses covering use of the system will even be required. Human beings enter any experience with expectations which should be accounted for in system design.

Experimental research has affirmed the need for using prior real world knowledge during communication [PLACS1]. Man-computer dialogue should allow for these expectations and use them in a positive manner. Once an individual has used a particular dialogue technique, he will be able to
predict how a similar model will work. 'A good standard dialogue technology has the same impact as a good standard programming language - users can transfer experience from one situation to another' [GAIN78]. When a given technique or model for dialogue is adopted, the dialogue should consistently follow the model to take advantage of the user's predictability. If a user is accustomed to seeing error messages at the bottom of the screen he will expect feedback in that location. System designers can anticipate the users response and program accordingly.

A designer must carefully consider the naive user's past knowledge, determine if it is congruent with an experienced user's model, and use caution when the two models do not coincide. In 1979, Bott conducted an experiment which demonstrated a situation where a user's prior knowledge was detrimental to his understanding in a man-computer interaction. Bott presented naive computer users with instructions on how to use the print command in the text editor of the UNIX operating system. Each subject responded with his interpretation of each instruction after it was presented. When the subjects were given the simple introductory line of 'How to print text', the responses were far from the experienced user's interpretation (displaying some written material on an output device such as a CRT screen). The subjects prior knowledge of
"print" referred to printing presses, "text" referred to a book and 'how' suggested a set of directions for doing a task. The subjects interpreted the line as procedures for printing a book on a printing press. In this instance the subjects' real world knowledge resulted in an incorrect interpretation [BCTT79]. Perhaps if real world knowledge had been considered when designing UNIX this type of misunderstanding could have been avoided. It must be recognized; however, that UNIX was not designed with the naive user in mind, but rather for Thompson's and Ritchie's personal convenience.

2.2.2. Implication

The set of interactions prior to a given statement make up the context of that statement. Human beings imply much from the context of a statement. The use of implication is seen in today's computer systems. The user is not surprised to encounter a different set of commands as he moves from the editor to the operating system. He realizes that he is in a different context or mode. It is reasonable to expect the user to make implications in a man-computer dialogue on the basis of context. However, a designer must be cautious when relying on the user's implications. "Consistency and compatibility are generally acknowledged to be basic ergonomic principles. A system which provides the user with a consistent
representation within a particular type of knowledge is likely to be easier to learn and less prone to error than an inconsistent one [BARNE81]. Consistency is not always possible and the designer must rely on the user's capabilities to make implications. To illustrate, suppose a decision has been made to position a line number as the first argument in any command in which it is required. A command would be of the form '10d' which, in a text editor, might mean "delete the 10th line of text". In another mode it may be best to form commands to coincide with natural language. Such a command would be of the form 'delete <entity> <line number>', i.e., 'delete name 10' where name is deleted from line 10 [BARNE81].

Consistency within a mode should be a high design priority. Distinct modes within a computer system may require different command structures. The increase in clarity resulting from different command structures must be weighed against the decrease in consistency. If a system contains more than one mode, each mode should be clearly identified. Commands should be unique to each mode, thus reducing the potential of error if a command is used in an incorrect mode [NORM83]. These guidelines will allow the user to make correct implications regarding the appropriate commands for a given mode.
2.2.3. **Experimentation**

Human beings do not accept a list of rules once and for all and then never break these rules. People make mistakes, both deliberate and accidental, and then learn from their mistakes. "Errors in dialogue are not essentially key problems to be avoided at all costs. The naive user should feel free to make errors as part of their exploration of the system" [GAIN78]. If users are given the option of trial and error several prerequisites must exist:

1. Penalties for errors must not be excessive.

2. Cost in time, money and labor must be low enough to allow for several trials.

3. Results of each trial must indicate changes to be made in later trials.

4. Results of a trial should be given quickly.

These prerequisites require cheap and plentiful computing power which has not been available in the past. Because computer power has decreased in cost it is now reasonable to expect the user to experiment with his computer system. Effective computer dialogues will inform the user if his trial results in an irrecoverable state and will also give
him estimates to the cost of his requests, thus allowing the user to experiment intelligently.

2.2.4. Motivation

Although computers do not have feelings toward humans, humans definitely have feelings towards computers. "Few people who have interacted personally with a computer have lukewarm feelings about the experience. Most often people find the interaction either highly enjoyable or totally distasteful" [HEIN75]. Man-machine dialogue must be designed in such a way as to nurture a positive attitude toward the computer. The user has direct contact with his terminal and treats the remainder of the system as a black box. He is concerned that he obtain the appropriate responses to his requests in a reasonable amount of time. One way the user measures the quality of service is by the features found within his dialogue with the terminal and the ease with which his dialogue takes place [HEBD78]. The user's interaction with his terminal can be viewed as the highest level virtual machine in a hierarchy of virtual machines. It is critical that this virtual machine present itself as understandable and sympathetic to its users [GAIN78]. As a user communicates with a computer his confidence in it should increase. This can be done through respectful responses to his commands, a feeling that his data is safe because of
systematic, predictable and consistent dialogues and by being provided with pertinent information as to how his requests are progressing.

2.3. **Design features found in user friendly interfaces**

The previous discussion of underlying principles of human communication provide a basis for the discussion of specific principles which can be incorporated in a user friendly interface.

2.3.1. **Mental models**

System designers and applied psychologists are increasingly coming to believe that people deal with complex interactive devices by making use of a conceptual model of the device" [YOUNG].

A user's conceptual model is a set of concepts the user assembles to explain the organization and behavior of his system. Often the model is an analogy to a real life activity or object. The Star Information System by Xerox incorporates into its system a strong analogy to the office environment with which its user is familiar. The display screen shows the common office furniture: a desk, mail box, file cabinet, etc. A user sees the result of his commands on the screen as functions are executed [SMIT].
Research has shown that the development of mental models does affect learning and memory. In 1979 Mayer concluded that advance organizers (introductory materials presented in advance of learning) have an effect on the learning of technical material [MAYF79]. A user's model is a form of advance organizer and will have an effect on the user's relation to a computer system. In 1975 Mayer conducted an experiment where subjects learned a subset of the BASIC programming language. The subjects were divided into two groups. One group was given a model which made analogies between the four major functional units that are present in programming languages: a ticket window analogy for input, a note pad for output, an erasable score board for memory and a shopping list for executive control. The second group was not given a model. Both groups read a booklet that defined and gave illustrations of seven BASIC commands. The results of the study showed that the group possessing a mental model was better able to determine what tasks a given sequence of commands would perform [MAYF79].

If a user is not presented with a model of his system he will formulate one himself. The formation of the model may not be a conscious effort by the user. The human mind constantly searches for patterns of cause and effect. A danger exists when the user generates his own model. If
the user formulates a false or superstitious model his effectiveness will decrease. Designers can guide the user in formulating correct models by avoiding non-deterministic behavior within a system and by clearly showing the user the consequences of his actions.

A transparent system that is totally open and understandable will also contribute to a correct model. The user should know why the system is doing what it is doing. If a novice user has a broad understanding of how the components of his system relate to each other and an understanding of the overall system structure, he feels confident as he navigates throughout the system. In contrast, a user with knowledge limited to the specific commands needed to execute a given task will be fearful of the consequences, if he deviates from these commands [SELAS2].

The user should feel the system is controllable and non-mysterious [ROBES1]. For example, when more than one level of dialogue is included in a system, if the user has control over the dialogue, the changes in interaction will not come as a surprise nor seem mysterious or unpredictable.

The designer may choose to present a model to the user or may choose to create an atmosphere in which the user forms a valid model independently. There have not
been extensive guidelines developed to assist designers in the invention of models. Further investigation is also needed to study the variance between models of the same system by different users [NICKS1].

2.3.2. User expertise

"The design of many current computer systems is based on the assumption that there is a single ideal system which can satisfy all users, but it is doubtful that any single system is best for any and all users" [ELAC81]. A user friendly system must acknowledge the varying abilities of the users. Generally users can be divided into two groups: novice or infrequent users and expert or frequent users. What users expect from a system and the way a system is used will vary greatly between the two groups.

Flexibility within a system is required to meet the continuum of abilities among its users. Dialogue formats such as menu displays and form filling are easy to learn and provide rigid structure for the novice user. On the other hand, programming style dialogues provide rapid interaction which is desirable for the expert user. If a system will be used by both groups provisions must be made for each group. Hall describes a library retrieval system which incorporates both menus and a command language. There are two distinct paths for retrieving information.
The user is allowed to select his approach and is able to switch between the two paths at will. The novice user is not overwhelmed with a terse set of commands and the expert is not frustrated by being required to traverse a sequence of menus to accomplish a task [HALL78].

"A truly friendly system must afford both intimacy as well as distance" [WFRN82]. A novice user should have interaction maximized. He should be presented brief messages and be required to make simple responses. A continuous interaction reinforces his ability to function successfully within the system. An expert user would prefer a quieter system that allows him to progress quickly through a task with a minimum amount of interruption.

The designer must step outside his own experience and view the system from the user's eyes, hence examining the system from the outside in [DWE81b]. Consider the input statement in BASIC. A user can be presented with either of the following statements:

```
INPUT THE INTEREST RATE PER YEAR
?
```

```
WHAT IS THE INTEREST RATE PER YEAR?
```

A designer feels completely at ease with the first
interaction because he has a clear understanding of the word 'input'. The naive user may be confused with the word and unsure of what action is expected of him. Clearly the second interaction relates more closely to a naive user's concept of data entry [HEIN75].

2.3.3. Mental load

Men and computers each have their own strengths and limitations. Man is skillful in selecting goals, observing patterns, and handling unforeseen circumstances; however, he is slow, forgetful and prone to error. In contrast the computer is extremely fast, accurate and does not forget, but it does not have man's reasoning ability. The designer must carefully balance the strengths of both man and machine. A computer can be used to complement the user by minimizing the mental load on the user. Because the computer's memory is better than man's, the designer should rely on the computer memory as much as possible [SIMP82]. Four methods for reducing mental load are described below.

Mental load is reduced in a system which makes use of default options where the user is given the most likely response to an interaction. The user is then free to concentrate on the exceptions he encounters. Defaults can also be incorporated into the operating system. The RSTS
operating system automatically makes a backup of each file when it is edited. The user is not required to create his own safeguard if he does irrecoverable damage to his file while editing. The user does not have to express his desire to have his editing changes written to the file at the end of the editing session. UNIX, however, requires the user to give a write command before he finishes editing. If the write command did not have to be explicitly given by the user, the editor would be friendlier to the naive user. It is likely that a new UNIX user will be frustrated at least once by neglecting to use this command.

A second technique for reducing the mental load is to provide prompts that are brief, specific, and show the desired entry format. If the user must enter a date, he should be shown the appropriate form of the date. He should not be left to guess the expected entry format.

The use of menus in man–machine dialogue reduces mental load. The user is not required to memorize special commands or names; rather he must only recognize the desired task from a list of options presented to him. Care must be taken to ensure only relevant options are presented to the user. Too many options or ambiguous options will confuse the user and increase his mental load.
Fourthly, there are applications that can be structured such that the user's response consists of only "yes" or "no" to a series of questions. This type of interaction is effective in medical diagnosis systems where the user may have no computer experience.

It is not always expedient to eliminate user memorization. Research has shown that when humans have memorized information, they are better able to use that information to make inferences relating to it than when the information is available but not memorized. If a system is a tool which will be used consistently by the user over a period of time, the user may use the system more effectively if he has committed pertinent information to long term memory [BLAC81].

Mental load to the user will be increased when his system is non-uniform or inconsistent. If the dialogue between user and computer are similar throughout a system except for a few tasks, or if help is available for most tasks but not all, the system reduces its friendliness. As the user becomes familiar with the system, certain features he expects may not exist.

This not only puts an additional memory load on the user but can also be devastating psychologically when a naive user has at last gained confidence in his familiarity with a system and then suddenly finds that his trust in himself and the system is misplaced [GAIN78].
2.3.4. **Response time**

System response time is a vital issue relating to interactive systems. Nagel presents the following guidelines for acceptable response time: [NAGF78].

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 15 sec</td>
<td>Intolerable</td>
</tr>
<tr>
<td>&gt; 4 sec</td>
<td>Too long in most cases - Possibly tolerable after termination of major work step.</td>
</tr>
<tr>
<td>&gt; 2 sec</td>
<td>Too long for high concentration work</td>
</tr>
<tr>
<td>&lt; 2 sec</td>
<td>Necessary for work consisting of more than one step to be bridged mentally.</td>
</tr>
<tr>
<td>&lt; .1 sec</td>
<td>Immediate reaction to key stroke</td>
</tr>
</tbody>
</table>

Long delays in system response are frustrating to the user and will reduce his efficiency. If delays exceed 15 seconds the system no longer provides conversation-like interactions. The length of delay is not the only consideration in determining acceptable response times. "A long delay may be more tolerable if one can predict when it will end than if one cannot" [NICK81]. A user friendly system should warn the user when a task will take a longer than the usual period of time to execute. For example, if a user requests an archived file, he should be given an estimate of the time needed to retrieve the file. In UNIX a program compilation will take longer than a simple request for the current date and time. The user will not
be alarmed by a delay if he expects one. Also if a user believes that his task is causing the delay, he will be more tolerant of the delay.

User expertise contributes to the definition of acceptable response time. A naive user requires time to carefully read dialogues and formulate the appropriate responses. He would be less likely to notice a delay in response time. In contrast, the expert user who is familiar with the system dialogues will be less tolerant of delays.

Response time will contribute significantly to the degree in which a user will experiment within a system. If there is a low response time a user will be more likely to explore and discover the full potential of his system. The Zog man-machine interface was designed to provide the user 'instantaneous' results. Zog is a menu driven interface to large networks of data. The user is presented with a new frame instantly upon requesting an option within a menu. He can easily scan a new menu frame and not be penalized with a lengthy time delay [ROEFSI].

2.3.5. User feedback

Closely related to response time is the concept of user feedback. "It is disastrous to the user's model when he invokes an action and the system does nothing in
response' [SMIT82]. If the user does not see a response to his commands he is likely to repeat his previous actions because he is not sure whether the system has heard him. Such interaction could lead to undesirable results.

A user should be sure of his orientation within a system. The user will be more confident when he knows where he is in relation to the total system as he progresses in his work. "If the user internalizes the system's structure, he will feel more comfortable working with the system" [SELAR82]. The tcp level menu provides a sense of direction and the ability to reach a place of stability within a menu driven system.

The system feedback a user receives can be used to modify his behavior. If the user is presented with positive phrases acknowledging his responses, the user is likely to repeat his correct actions. Words such as 'OK' and 'GOOD' can be used as rewards. Such rewards are most effective when they are presented immediately. On the other hand, computer systems which only acknowledge incorrect actions will increase the user's sense of failure, thus decreasing his effectiveness. System feedback should not be harsh or demeaning [DWYES1a], [DWYES1b].
2.3.6. Error handling

Error messages reveal characteristics of a system. Messages such as 'FATAL ERROR, RUN ABORTED' confuse and frustrate users. A novice user may quickly develop a negative image of his computer system if the error messages are hostile or obscure [SHNF82]. A user friendly system presents the user with polite and factual error messages. These messages should not be designed to humor, punish, or intimidate the user.

Error handling is a three step process. First the user must be told an error exists. A designer may choose to alert the user by sounding a beep or blinking the screen. These techniques are only effective if they are not an integral part of the normal system activity. It is also effective to reserve an area of the screen for error messages. When text appears in the area the user is alerted to an error. Secondly, the system must identify the errors. When describing error it is best to use non-computer jargon. The designer should select words that are understood by the user. The error message should be as specific as possible. If a user enters a letter when a number is expected he should be told that a letter is not an appropriate response instead of the vague message - 'invalid entry'. Lastly the user should be directed to the correct response for the error situation. In the
previous example the user should be told that only numbers are acceptable entries [EFIN75].

Galitz presents the following techniques which lead to easy, correct and fast message interpretation [GALI81].

Do not use abbreviations
Do not use contractions or short forms
Use brief, simple sentences
Use affirmative statements
Use the active voice
Order words chronologically

Again, the expertise of the user dictates how explicit error messages should be. The naive user needs lengthy error messages whereas a cryptic reminder may be sufficient for the experienced user. It would be desirable for a system to contain a multilevel approach to error messages. The user should control the amount of detail he is given on an error condition.

Extensive error checking at data entry will minimize the number of errors within a system. Data syntax should definitely be checked; however, care must be used when rejecting data which deviates too far from the norm. A designer may not be able to predict all acceptable responses but he is able to warn users if values are
suspicious.

Tasks which result in major changes to a system should require revalidation of the task requested. In a database system a user should be informed of the consequences when he requests deletion or purging of a file. He should be asked to verify his request, thus safeguarding himself against costly errors.

A further extension pertaining to errors is a system which identifies errors and then corrects them without user interaction. Such a system would be difficult to implement, for often times it would be difficult to determine a user's intention. The following two examples illustrate situations where error correction would be desirable. Naive users may know what they want to do but are unsure of the appropriate commands. If they are able to remember part of a command the system could supply the remainder. A "C" programmer can easily forget to include a semicolon at the end of a line. If the system inserted a semicolon as a default the programmer would be relieved of another detail.

Error handling is costly in terms of system design and implementation. Layered error messages and error correction increase run time overhead. The system designer must evaluate the optimal balance between cost
Error handling is one of the most important aspects of the man-machine interface, yet it is often neglected. Designers must give error messages a high priority. A well thought-out system of error messages coupled with improved displays will increase the quality of a user interface [FROW83].

2.3.7. Help facilities

All users of a system need assistance in using their system. The amount and type of help required is dependent on the level of user expertise. A novice user is anxious to do something useful to him as quickly as possible. "Introductory training material should be designed to bring the beginning user to the point of accomplishing something of interest quickly" [NICK81]. The new user does not need to know complex details which show him how to use the full power of his system. As his level of expertise increases further training and documentation with greater detail become significant and interesting to the user. Once a groundwork of fundamental capabilities is mastered the user is in a position to discover and learn advanced tasks.

It is desirable to have an on-line help facility. A large manual is formidable to both the new and experienced
user. When the user has a question he appreciates an answer available on demand placed at his screen. An excessive amount of material on a screen, however, can distract the user and discourage him from reading the material. A layered approach within a help system would be effective in relieving the information overload. The top layer in the help system should present short messages which briefly list appropriate responses. If the user desires more detailed help he can continue to ask for help, thus being lead through additional layers within the help structure [GAIN78]. The top layer of help system should be extremely sensitive to the new user in its presentation. It is most frustrating to require help to understand the help message.

A system which logs user responses can be integrated into the help subsystem. The records can be analyzed to see the system capabilities the user has employed, the types and frequency of errors, as well as the general flow of dialogue. A supervisor could use the log to encourage users and offer guidance. A more sophisticated system could make use of a log to initiate suggestions to the user. A user may lack the knowledge required to even ask for information. A user is unlikely to explore and search for a more efficient method to complete a task once he is comfortable with an established method. A log could be
utilized in system suggestions to the user for alternate methods.

The existence of on-line help systems does not eliminate the need for hard copy documentation. Some information, e.g. diagrams, is presented more effectively in hard copy. A well organized user's manual is often easier to read than text presented at a terminal. Long descriptions to which the user must frequently refer would be valuable in hard copy.

2.3.8. Screen design

The screen layout affects the user effectiveness and enjoyment within a system. "As a general rule, access screens by paging, not by scrolling. People find it easier to read stationary pages than moving pages" [SIMP82]. A designer should use his capabilities to clear the screen when moving from frame to frame within a system.

Galitz offers several general guidelines for effective screen design. Screens should be presented in an orderly manner. The mind quickly identifies a cluttered screen and will either consciously or unconsciously exert effort by mentally reorganizing it into a meaningful form. A screen density of about 15% is easy to discern as compared to hard copy text that allows for a 40% density.
The eye is drawn to the upper left corner of the screen, thus the designer should use the upper left corner as the starting position. The designer should center titles and attempt to balance information on either side of the vertical axis. The screen can be divided into sections through the use of color, reverse images, lines, etc. The screen, however, should not be divided into too many small windows which gives the screen a cluttered appearance. If both upper and lower case are available on the terminal, both should be utilized. Lower case is less forbidding than upper case. Upper case is effective for labels and captions as it reduces search time. Attention getters such as blinking, reverse video, higher intensity and changes in character size must be used in moderation to be effective. Overuse of attention getters can be distracting.

2.4. Cost considerations

The design of a user friendly interface is an ongoing process. As users interact with a system they will express comments, complaints and suggestions. The system designer will eventually break his ties with a system. Before he does, he will probably be asked to make changes in the man-machine interface. Careful preliminary design will make changes easier to incorporate. The system design will contribute to the ease of maintenance as its
use continues.

A user friendly system will take more time to design and implement but this should not be an excuse for less than optimal interactive systems. Project milestones and financial constraints may force a designer to allocate insufficient time to study the user's point of view. Although there is an immediate savings in cost, the savings may result in long term dissatisfaction.

We must not lose sight of the end objective of the programming activity; to provide an effective implementation of the user's requirements. The conflict arises not because programmers do not want to achieve that objective, but features which make marginal improvements to the usability of a dialogue may appear to require a disproportionate amount of development effort. The problem then becomes one of how you measure the benefit feature and financial terms [EEED78].

As computers are increasingly utilized in the work force the ease with which they interact will directly affect job satisfaction. Consequently, a user friendly system will have an effect on the financial concerns of the institutions in which they are incorporated.
3. Design and implementation of user friendly interface to UNIX

3.1. Design goals

3.1.1. Primary objectives

The user friendly interface to UNIX was written with two primary objectives. First the interface must be easy to use. An individual should be able to interact with the UNIX operating system without prior UNIX instruction. Secondly, the interface must provide the user an opportunity to learn UNIX commands. The first objective of providing an environment where the user can function effectively could be accomplished without informing the user of the corresponding UNIX commands. However, if the user is notified of the appropriate UNIX commands for each task he requests, a by-product of the man-machine interaction will be the user learning UNIX commands. The training aspect of the interface is important to the new user who will continue to use UNIX for an extended period of time.
3.1.2. **Secondary objectives**

A secondary design objective addresses the flexibility of the interface. The process of selecting the type of dialogue and the implementation of that dialogue included the consideration of adapting the interface to other applications. It was desirable to design an interface that could be easily modified and be incorporated into other systems where user friendliness is a priority.

The size of the interface program was a significant design consideration. It was desirable to keep the interface as small as possible. It was projected that the interface would be used by many users simultaneously. A small interface would place less demand on the computer system, increasing overall system response time.

3.2. **Audience using the interface**

A major design decision addressed the audience using the interface to UNIX. Three classes of potential interface users were established:

1) New UNIX users, in particular beginning programming students

2) Experienced UNIX users who have been away from UNIX for an extended period of time
3) Casual users, individuals who do not use UNIX regularly.

The primary audience has no UNIX experience. This class can be divided into two subclasses. One subclass consists of individuals who have prior computer experience. These users have an understanding of the tasks they want completed, but do not know how to have them performed in the UNIX environment.

The second subclass has no computer experience. These users require support in developing a model for a computer system. They need help in understanding fundamental concepts such as what is a file, what does it mean to edit, what is a directory, etc. The interface written for this project is not a tutorial on computer systems. It does not attempt to define or provide a model for underlying concepts. Instead, it does provide a means of functioning effectively within the UNIX environment. This interface is beneficial to the beginning programming student in that his anxiety related to learning a computer system is reduced; thus, the beginning student is able to appropriate more time and effort toward developing programming skills. Classroom experience will provide the underlying concepts and models.
The second class of users consists of experienced UNIX users who have been away from UNIX for an extended period of time. Such a user has a series of tasks to be completed but is unsure of the syntax of the appropriate UNIX commands. This user will benefit from seeing the UNIX commands as his tasks are performed. The user will accomplish his tasks and at the same time be reminded of correct UNIX commands. An experienced user will spend a brief time within the interface. A quick reminder will enable him to use UNIX effectively once again.

The third class of users includes individuals whose experience with UNIX is limited. These users may occasionally want to use UNIX but will not use it frequently enough to warrant their learning the system. This class of users will not be particularly interested in the training aspect of the interface. They will, however, appreciate the ease with which their tasks are completed.

3.3. Length of time spent using the interface

The interface is not designed for long term continuous use. As an individual becomes familiar with UNIX he will appreciate its terseness. An interface between the user and UNIX would become a nuisance over an extended period of time. The interface should, however, provide for the transition period when the user needs the security
of the interface while at the same time would like the freedom to initiate a subset of commands on his own.

3.4. **Scope of the interface**

The scope of the interface is limited to common UNIX functions (see figure 1). The tasks included reflect the activities a beginning programming student uses. Once an individual has spent time using the UNIX interface, he will be better equipped to understand UNIX documentation found in manuals. The user will have established a foundation of common commands and their usage. He will then be able to build on this foundation independently. The user will be able to compare and relate less frequently used functions to the ones with which he is familiar.

3.5. **Model for interface**

The TENIX 8560 Multi-User Software Development system by Tektronics includes a program named Guide. Guide helps the user learn about the TENIX operating system while he uses the system. Guide incorporates a menu structure. The TENIX system provided a model for the interface implemented by this thesis project [TEKT81].

3.6. **Dialogue selection**

The dialogue requirements within the interface can be classified into two types. First, the user must
1) **Introduction to menu shell**

2) **Introduction to terminal**

3) **Change level of prompt**

4) **File manipulation**
   1) Return to top level menu
   2) Create a file
   3) Edit a file
      1) Display contents of file
      2) Display a screen of text beginning with a given line
      3) Display a given line
      4) Append after a given line
      5) Insert before a given line
      6) Delete a given line
      7) Exit the editor
   4) Copy a file
   5) Move a file (change its name)
   6) Delete a file
   7) Display a file at terminal
      1) Return to previous level
      2) Scrolled display of a file
      3) Text displayed one screen at a time
      4) Octal display of a file
      5) Display non-printing characters in a file
      6) Temporarily exit menu shell
   8) Display a file at printer
   9) List names of files
  10) Temporarily exit menu shell

5) **Directory manipulation**
   1) Return to top level
   2) Create a directory
   3) Delete a directory
   4) List contents of a directory
   5) Change directory
   6) Path identification
   7) Temporarily exit menu shell

6) **Other system functions**
   1) Return to top level
   2) Current date and time
   3) Who is working on the system
   4) Send mail
   5) Receive mail
   6) Write to another user
7) Display UNIX manual
8) Change password
9) Temporarily exit menu shell

7) Program compilation
   1) Return to top level
   2) Pascal program
   3) C program
   4) FORTRAN program
   5) Temporarily exit menu shell

8) Temporarily exit menu shell
9) Exit

Figure 1. Scope of menu shell

communicate the specific task he wishes to have completed. Secondly, the user must supply the pertinent information that a particular command demands. A combination of menu selection and question-answer dialogues was selected to accommodate the communication flow between the user and UNIX. A series of menus satisfies the first class of communication. The user is lead through a sequence of menus resulting in the selection of the task to be completed. Once a task is selected, the second type of communication is satisfied with a question-answer dialogue. The question-answer technique was selected due to the diverse nature of tasks available to the user within the UNIX interface. The information required for compiling a program differs from the information required for sending mail, printing a file, changing directory, etc. Question-answer dialogue provides the flexibility needed
to gather the broad range of responses required for the available tasks.

The literature supports the ease with which a menu system is used. Menu dialogues are the archetype of computer-initiated dialogue [RAMS79]. A menu system presents the user with a list of alternatives. The user then selects one of these alternatives. A menu system is especially helpful to the new user in that it provides a list of available options. Many times the new user may not have sufficient knowledge to understand what are and what are not appropriate requests. The computer is responsible for the majority of the man-computer interaction, thus the user's participation in the dialogue is minimal. The passive role of the user contributes to the rapid speed in which a menu system can be learned.

Menu dialogues have a major drawback. Once a user is familiar with a system, the menus which once provided a pleasant environment become a nuisance. For this project, the cumbersome characteristic of a menu system becomes a positive feature. The objective to teach UNIX commands is enhanced because the user is motivated to learn UNIX commands and thus be able to work independently of the menu structure. The user realizes he is able to initiate a command on his own faster than he is able to using the menu interface. The frequent UNIX user is encouraged to
become self-sufficient.

3.7. **Implementation of the interface**

A menu dialogue naturally lends itself to a hierarchical structure. The first menu presented to a user corresponds to the root in a tree structure. As the user makes selections he traverses the tree. When the user reaches a leaf node, he is prompted for the information related to his task, and finally his task is executed.

3.7.1. **Language selected** The program driving the menu structure is written in "C". The C language provides a mechanism that allows one process to create another process, wait until the new process completes execution, and then continue in its own execution. The ability to 'fork' a child process is particularly suited to this project. Most of the UNIX operating system is written in C. The close ties between UNIX and C also point toward C as the language for the interface.

The leaf node programs are written as shell scripts. Shell scripts are used because of their error checking capabilities. Increased error checking within the shell script programs prevents the user from receiving obscure error messages from UNIX. The shell script programs also increase the flexibility of the interface. If a question-answer dialogue must be changed, the file
containing the shell program can be modified without recompiling the entire interface.

There are two exceptions to the use of shell scripts at the leaf nodes. The option which edits a file required a program which was able to spawn a child process. The edit option was written in C. Thus, the process was able to fork the UNIX editor as a child process. The edit program simulates the menu selection procedure found in the driving program.

The second exception was the change directory option. In the C language when a child process returns control to the parent process, the child's environment is replaced with the parent's environment. A shell script program could be written to change directories; however, that directory change would be nullified when the process completed. When control returned to the parent, the child's environment would be lost and the current directory would remain the parent's current directory. When a user requests the change directory option, a simple shell script program is executed. The single function of the shell script is to exit with a status code. This exit status signals the driving program to call a function which performs a directory change. This function follows the dialogue format found in the shell script programs. Because the directory change is executed within the
driving program, the new directory remains the current directory until another change of directory is requested.

3.7.2. Structure of the interface The program driving the menu structure is short and straightforward. Figure 2 describes the flow of the driving program.

Each option within a menu display is associated with a separate file. As the user makes a selection from a menu display, the driving program opens the file associated with that option. If the file is another menu display file, the driving program repeats the cycle of menu display and registering user response. If the file

1) A brief introduction welcoming the user and describing the appropriate response to a menu display.

2) Repeat until "exit menu" is requested
   a) Display menu
   b) Register response
      If a leaf node is selected then
         execute the file associated with leaf node
         return to previous menu
      else
         retrieve associated menu file

Figure 2. Program flow of driving program
is a leaf node, the program named within the file is executed. At the completion of a leaf node task, the user is presented with the most recent menu.

The path taken within the menu structure is recorded in an array representing a stack. The array consists of pointers to the names of the files accessed in the path from the root (top level menu) to the current position in the menu structure. The stack is used to determine the return path.

The menu display system consists of 3 major activities.

1) Initialization
2) Option menu processing
3) Leaf node processing

The initialization activity presents the introductory statements to the user. The stack pointer to the root or top level menu is also initialized (see figure 3).

The option processing activity navigates a series of menus until a leaf node is reached. The user must respond with a valid response to the menu options. He is instructed as to the appropriate responses if he gives an invalid response. This activity also maintains the stack array representing the path taken (see figure 4).
Initialization

User introduction

Place root menu on stack

Option processing

Figure 3. Initialization activity
Figure 4. Option processing activity
The leaf node activity initiates the request to fork the executable file associated with the menu option selected. The exit code from the child process is evaluated and the appropriate action is taken (see figure 5).

3.6. Description of files The files accessed by the driving program are in two forms. A menu display file is shown in figure 6. A menu display file consists of the number of available options in the menu, the menu title, the list of menu options and the file associated with each option.

The form of a leaf node is shown in figure 7. A leaf file uses the exclamation mark to distinguish it as a leaf file. 'File name' represents an executable file which prompts the user for pertinent information and initiates the appropriate UNIX commands.

Communication between executable files and the driving program is accomplished with the exit status of the executable programs. Most programs exit normally with a status of 0. The user is presented with the most recent menu and option processing continues. The following non-zero exit status have special meanings.
Figure 5. Leaf node processing activity
Figure 6. Example of menu display file

Figure 7. Example of leaf node file

1. Change directory
2. Return to previous level menu
4. Change to verbose prompt
5. Change to terse prompt
6. Exit menu interface

A status code of 3 does not have a special meaning. If a
user breaks out of a shell script, the exit status is 3. In this case the user is presented with the most recent menu and option processing continues, thus no special action is required.

3.9. Significant features of the interface

The interface is flexible. In this application the driving program initiates execution of shell script programs. The only restriction the driving program places on the leaf files is that the files name executable programs. The interface can easily be modified for other applications. Appropriate menu options must be specified in menu files and the corresponding executable programs must exist; however, the executable programs may be written in any language and may use the dialogue technique most appropriate to the new application.

The interface provides a means for the user to learn UNIX commands. Each time a program initiates a UNIX command, the interface displays the command at the terminal. Each command is preceded by three exclamation marks, thus, the user is able to recognize UNIX commands from other dialogue. For example, if a user requests the "current date and time" option, he is presented with the following command immediately before the date and time are displayed.
Many UNIX commands require arguments. There are two levels of prompting users for these arguments. Level 1 is a verbose prompt. The user is presented with complete questions which reinforce the nature of the task requested. Level 2 is a terse prompt; user prompts are limited to key phrases. For example, if a user requests the "delete a file" option, the verbose prompt asks:

What is the name of the file you wish to delete?

The terse prompt simply asks:

File name?

The interface initially presents the verbose level of prompt. A menu option allows the user to change the level of prompt from verbose to terse and vice versa. When a user first interacts with the UNIX interface, the verbose prompt is appropriate. Once a user is familiar with the tasks available, the terse prompt will increase the speed of interaction between the user and the interface.

The user is in control of the level of prompt. The literature supports the concept of user control [ROE81]. If the user makes the decision to change the prompt level, his sense of control over the system is increased. If the
prompt level changes without explanation or user request, the system will seem confusing or mysterious. The user will formulate a reason for the change in dialogue. His conclusions may not be accurate and his confidence in the system may be damaged by an inaccurate model.

The leaf programs include a limited help facility. The user is asked questions pertinent to the task he has selected. If a question is preceded with an "*", the user has the option of entering a question mark as his response. A question mark results in the presentation of additional comments regarding the question the user was asked and the nature of the task requested. The responses to the question mark option are written into the executable programs. The new user will find the additional comments helpful. We would be wise to make use of the question mark option. However, as the user becomes familiar with the commands, he will not require this support. The additional explanations are not part of the basic dialogue. The help dialogue would reduce user efficiency once the user becomes familiar with a given task. The help messages are readily available; however, the user must request them.

Most users will use the interface on a temporary basis. A user will quickly learn a subset of UNIX commands and will want to try these commands independently.
At the same time, he will want the security of the menu interface for those commands of which he is unsure. The interface provides for the transition period by including a temporary exit option in each menu. The temporary exit option accepts the UNIX commands a user requests and executes these commands. The interface does no error checking in relation to the commands, and the user sees the normal UNIX response to his commands. The user is able to return to the menu dialogue at any time. This option allows the user to experiment on his own. He is not threatened by the syntax errors he makes because he realizes he is able to complete his task by returning to the menu structure if he is unsuccessful on his own. When the user enters the temporary exit option, he must take responsibility for his commands. For example, if the user wishes to copy a file and reverses the input and output arguments, his task will be completed and he must accept the consequences of his command. Because the temporary exit option is included in each menu, it is readily available at any time. It is easy to request the option and easy to return to the menu structure, thus the user is encouraged to initiate UNIX commands independently.

Execution time varies between UNIX commands. Program compilation and execution require significantly more time to complete than other UNIX commands such as file display
and listing the contents of a directory. The interface includes a warning message to the user when a task will have longer than usual execution time. These warning messages will reduce user anxiety. The user will expect a delay in response and will be more tolerant of the delay.

The interface dialogue is straightforward and polite. Complete sentences are used whenever messages are printed. The words and phrases used in messages are meaningful to a new user; however, UNIX terminology was not avoided. Most users will eventually interact with UNIX independently; they must become acquainted with UNIX terminology.

Both upper and lower case characters are used in the interface. Menu displays use upper case characters to list available menu options. Leaf programs primarily use lower case characters which coincide with normal English text. All menus and leaf program have titles. The titles help the user verify his current location within the menu structure.

The above features illustrate how many of the characteristics of a user friendly interface were incorporated into this interface. The interface is easy to use; however, each time a user makes a response, he must press the 'return' key. Other techniques such as cursor movement and cursor position could have been used. Such techniques
would initially be easier for the user; however, they do not prepare the user for future UNIX terminal interaction.
4. Extensions

This thesis project accomplished the goal of providing an interface to UNIX enabling an inexperienced UNIX user to function comfortably within the UNIX environment while at the same time teaching a subset of UNIX commands to the user. There are improvements to be made which would increase the quality of user friendliness and would increase the scope of the project. Extensions to the project are discussed below.

The interface was written for use on non-graphic terminals. The user is responsible for knowing his position within the menu structure. This is not difficult in the present application as the menu structure is only four levels deep. A graphics display window showing the current position in the menu structure in relation to the entire structure would increase user orientation [SIIAS2].

The issue of user orientation becomes more critical as the depth of the menu structure increases. As the number of levels increases in a hierarchical structure, the cost in time of returning to the top level increases. A visual display of the path taken within the menu structure would
eliminate the need for a user returning to the top level menu as a means of restoring orientation.

An option that returns control directly to the top level menu also becomes important as the depth of the menu structure increases. This option would be valuable on each menu display. The user would no longer be required to retrace his path, one level at a time in order to return to the top level menu.

The help facility available within the interface is limited. The leaf programs provide brief help messages related to the questions the users must answer. There is also a menu option which provides access to the UNIX manual command. However, the interface does not provide a comprehensive help system which is tutorial in nature. A help option could be incorporated into each menu display. Such a help facility should give a general description of the options available within that menu along with an explanation of underlying concepts related to the menu options. An extended help facility would assist the user in formulating a correct model of the system.

The interface includes an edit a file option. The editor is consistent with the menu design in that it is a menu driven editor. The edit option contains line oriented edit commands. It would be desirable to expand
the editing capabilities to include many of the character oriented edit commands. A screen oriented editor would also increase the friendliness of the edit option.

The temporary exit option within the interface provides a means for the user to work independently. When the user returns to the menu structure, he returns to his position prior to his temporary exit. The interface could be enhanced by paralleling the user's position within the menu structure while he works independently. The user would then return to the position within the menu structure associated with his most recent independent action. A parallel system of menu structure and command language would increase the mobility of the user and provide a mechanism for user short cuts. Hall describes a library administrative example which incorporates a dual interface of menu selection and command language and allows the user to move freely between the two techniques [HALL76]. A dual approach is effective in an environment which serves a varied population comprised of novice and expert users.

The subset of UNIX commands selected for this project was oriented towards a beginning programming student. Additional subsets of commands could be selected for various categories of users. The interface could be modified to implement these new subsets of commands. A subset could be written for an office environment. This
interface would place a greater emphasis on commands such as electronic mail, text formatting, and file manipulation. Another subset could be written exclusively for program compilation, debugging and execution.
CHAPTER 5

SAMPLE INTERACTION

5. Sample Interaction

Significant features of the menu shell are illustrated in the following sample interaction. Tasks have been selected from each of the top menu options with the exception of the "introduction to menu shell" and "introduction to terminal" options. The tasks included in the sample interaction demonstrate a user's movement within the shell structure as well as the dialogue he encounters.
WELCOME TO THE MENU SDFLI

A menu of available options will be displayed at your terminal. You must type the number which precedes your choice of the menu options and press the return key.
TOP LEVEL MENU

1) INTRODUCTION TO MENU SHELL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORY MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) COMPILK AND EXECUTE PROGRAMS
8) TEMPORARILY EXIT MENU SHELL
9) EXIT

Enter your choice of the menu options: 4
FILE MANIPULATION MENU

1) TEMPORARY EXIT FILE MENU
2) LIST NAMES OF FILES
3) PRINT A FILE AT PRINTER
4) DISPLAY A FILE AT TERMINAL
5) DELETE A FILE
6) MOVE A FILE (CHANGE ITS NAME)
7) COPY A FILE
8) EDIT A FILE
9) CREATE A FILE
10) RETUN TO TOP LEVEL MENU

Enter your choice of the menu options.
What is the name of the file you wish to create?  

A file name is limited to 14 characters. Most characters may be used in a file name, however until you are familiar with the special characters and their meanings you should use only letters, numbers and the period.

What is the name of the file you wish to create? sample

After seeing the UNIX command, enter the contents of your file. You are able to make corrections on the current line you are typing, however once you press the return key for a given line, corrections must be done at a later time by choosing the "EDIT A FILE" menu option.

TO INDICATE THAT YOU HAVE FINISHED ENTERING YOUR FILE PRESS THE "CONTROL" KEY AND THE 'D' KEY SIMULTANEOUSLY.

!!! cat > sample

This is a sample file. Its purpose is to demonstrate the "CREATE A FILE" option in the menu shell.

Press return to continue
FILE MANIPULATION MENU

1) RETURN TO TOP LEVEL MENU
2) CREATE A FILE
3) EDIT A FILE
4) COPY A FILE
5) MOVE A FILE (CHANGE ITS NAME)
6) DELETE A FILE
7) DISPLAY A FILE AT TERMINAL
8) PRINT A FILE AT PRINTER
9) LIST NAMES OF FILES
10) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 9
LIST NAMES OF FILES

Do you wish a brief or long listing (b or l)?

A brief listing will only give the names of the files whereas a long listing gives the mode, number of links, owner, size in bytes and time of last modification in addition to the file names.

Do you wish a brief or long listing (b or l)?

You must enter "b" for a brief listing or "l" for a long listing. (b or l)? b

III l

sample test.c seesample

Press return to continue
FIIF MANIPULATION MENU

1) RETURN TO TOP LEVEL MENU
2) CREATE A FILE
3) EDIT A FILE
4) COPY A FILE
5) MOVE A FILE (CHANGE ITS NAME)
6) DELETE A FILE
7) DISPLAY A FILE AT TERMINAL
8) PRINT A FIIF AT PRINTER
9) LIST NAMES OF FILES
10) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 7
DISPLAY A FILE AT TERMINAL MENU

1) RETURN TO PREVIOUS LEVEL
2) SCROLL PD DISPLAY OF A FILE
3) TEXT DISPLAYED ONE SCREEN AT A TIME
4) OCTAL DISPLAY OF A FILE
5) DISPLAY NON-PRINTING CHARACTERS IN A FILE
6) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 5
DISPLAY NON-PRINTING CHARACTERS IN A FILE

Your file will be displayed with the non-printing characters in a readable format. The following formats are used.

`\t` = tab
`\?` = delete
`\` = end of line

What is the name of the file you wish to display? samplesee

File samplesee does not exist!
DISPLAY A FILE AT TERMINAL MENU

1) RETURN TO PREVIOUS LEVEL
2) SCROLLID DISPLAY OF A FILE
3) TFXT DISPLAYED ONF SCREEN AT A TIME
4) OCTAL DISPLAY OF A FILE
5) DISPLAY NON-PRINTING CHARACTERS IN A FILE
6) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 5
DISPLAY NON-PRINTING CHARACTERS IN A FILE

Your file will be displayed with the non-printing characters in a readable format. The following formats are used:

^I = tab
~? = delete
\# = end of line

What is the name of the file you wish to display? 

You must enter the name of an existing file.

What is the name of the file you wish to display? seesample

!!! see -v -e -t seesample

This file demonstrates:
the tab non-printing character;
and the end of line non-printing character.

Press return to continue
DISPLAY A FILE AT TERMINAL MENU

1) RETURN TO PREVIOUS LEVEL
2) SCROLLED DISPLAY OF A FILE
3) TEXT DISPLAYED ONE SCREEN AT A TIME
4) OCTAL DISPLAY OF A FILE
5) DISPLAY NON-PRINTING CHARACTERS IN A FILE
6) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 1
FILE MANIPULATION MENU

1) RETURN TO TOP LEVEL MENU
2) CREATE A FILE
3) EDIT A FILE
4) COPY A FILE
5) MOVE A FILE (CHANGE ITS NAME)
6) DELETE A FILE
7) DISPLAY A FILE AT TERMINAL
8) PRINT A FILE AT PRINTER
9) LIST NAMES OF FILES
10) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 1
TOP LEVEL MENU

1) INTRODUCTION TO MENU SHELL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORy MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) COMPILe AND EXECUTE PROGRAMS
8) TEMPORARILY EXIT MENU SHELL
9) EXIT

Enter your choice of the menu options: 6
OTHER SYSTEM FUNCTIONS

1) RETURN TO TOP LEVEL MENU
2) CURRENT DATE AND TIME
3) WHO IS WORKING ON THE SYSTEM
4) SEND MAIL
5) RECEIVE MAIL
6) WRITE TO ANOTHER USER
7) DISPLAY UNIX MANUAL
8) CHANGE PASSWORD
9) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 3
WHO IS WORKING ON THE SYSTEM

The following command lists the login name, terminal name and login time for each user currently on the system.

!!! who

rxf0947 tty02 Jun 10 14:12
rkd1649 tty07 Jun 10 15:12
rwf6565 tty09 Jun 10 15:09
jms2990 tty20 Jun 10 15:06
mkk1141 tty34 Jun 10 12:43

Press return to continue
OTHER SYSTEM FUNCTIONS

1) RETURN TO TCP ILEVEL MENU
2) CURRENT DATE AND TIME
3) WHO IS WORKING ON THE SYSTEM
4) SEND MAIL
5) RECEIVE MAIL
6) WRITE TO ANOTHER USER
7) DISPLAY UNIX MANUAL
8) CHANGE PASSWORD
9) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 4
SEND MAIL.

*To whom do you wish to send mail? ?

You must enter the login name of the person to whom you wish to send mail.

*To whom do you wish to send mail? mkh1141

*Is the message you wish to send contained in a file (y or n)? ?

You are able to send mail in two ways:

1) Create a file whose contents is the message you wish to send.
2) Enter the message to be sent at the terminal after the mail command is initiated.

*Is the message you wish to send contained in a file (y or n)? t

You must enter 'y' for yes and 'n' for no.

*Is the message you wish to send contained in a file (y or n)? n

After you see the UNIX command enter your message one line at a time. When you have finished entering your message press the control and 'd' keys simultaneously.

III mail mkh1141

Hello mkh1141
This is a small message to demonstrate the send mail option.
Good bye

Press return to continue
OTHERS SYSTEM FUNCTIONS

1) RETURN TO TOP LEVEL MENU
2) CURRENT DATE AND TIME
3) WHO IS WORKING ON THE SYSTEM
4) SEND MAIL
5) RECEIVE MAIL
6) WRITE TO ANOTHER USER
7) DISPLAY UNIX MANUAL
8) CHANGE PASSWORD
9) TEMPORARILY EXIT USER MENU SHHELL

Enter your choice of the menu options: 1
TOP LEVEL MENU

1) INTRODUCTION TO MENU SHELL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORY MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) Compile and Execute Programs
8) Temporarily Exit Menu Shell
9) Exit

Enter your choice of menu options: 7
COMPILE AND EXECUTE MENU

1) RETURN TO TOP LEVEL
2) PASCAL PROGRAM
3) C PROGRAM
4) FORTRAN PROGRAM
5) TEMPOAPILY EXIT MENU SHELL

Enter your choice of the menu options: 3
C PROGRAM

You should expect a delay when your program is compiled and executed. Program compilation and execution requires more time than most of the tasks found in the menu shell.

“What is the name of the C source file you wish to compile? ?

You must enter the name of a file which contains a C program. The file name must end in "c"

“What is the name of the C source file you wish to compile? test.c

This option will compile your C file.

“Do you also want your program executed (y or n)? ?

You will not want to have your program executed until it is free of errors. You should answer "n" to this question until you have an error free compilation.

This option will compile your C file.

‘Do you also want your program executed (y or n)? y

When a C program is compiled the following options are available:

c suppress the loading phase
   (Do not select this option if you wish to have your program executed.)

f floating point simulation
   (Do not select this option if you are working on a VAX.)

O Optimize object code

w suppress warning diagnostics

To see the additional options use the MANUAL menu option found in the OTHER SYSTEM FUNCTIONS menu. You should ask for the ‘cc’ command.

Enter the options you want one at a time followed by a return. When you have finished entering options simply press return.

Option (c f O w): O

Option (c f O w):

‘Do you have an input file (y or n)? ?

A program which expects data to be input from the terminal can have
the data redirected from an input file. If your program is expecting data from standard input and the actual data is in a file answer "y" to this question.

*Do you want the output placed in a file (y or n)? y

Any program results which are output to the terminal may be redirected to an output file. If you name an existing file as the output file the contents of that file will be replaced by the new output of your program.

*Do you want the output placed in a file (y or n)? y

What is the name of the output file? testsee

!!! cc -0 test.c
!!! a.out > testsee

Press return to continue
COMPILE AND EXECUTE MENU

1) RETURN TO TOP LEVEL
2) PASCAL PROGRAM
3) C PROGRAM
4) FORTRAN PROGRAM
5) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 5
TEMPORARILY EXIT MENU SHELL

You are able to return to the menu shell any time by pressing the return key. Proceed to enter the UNIX commands you wish to execute.

$ cat testsee
Output from program which demonstrates "COMPILF AND EXECUTE"
option in the menu shell.

a=3 b=4
The product c = 12

$
COMPILE AND EXECUTE MENU

1) RETURN TO TOP LEVEL
2) PASCAL PROGRAM
3) C PROGRAM
4) FORTRAN PROGRAM
5) TEMPORARILY EXIT MENU SHPIII

Enter your choice of the menu options: 1
TOP IFVII MENU

1) INTRODUCTION TO MENU SHILL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORY MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) COMPILATE AND EXECUTE PROGRAMS
8) TEMPORARILY EXIT MENU SHELL
9) EXIT

Enter your choice of the menu options: 3
CHANGE LEVEL OF PROMPT

*What level of prompt do you wish (1 or 2)?

There are two levels of prompting. "1" is a verbose prompt which prompts you with questions which help explain the desired task. "2" is a terse prompt which can be used once you are familiar with the available tasks.

*What level of prompt do you wish (1 or 2)? 2
TOP LEVEL MENU

1) INTRODUCTION TO MENU SHELL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORY MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) COMPILIE AND EXECUTE PROGRAMS
8) TEMPORARILY EXIT MENU SHELL
9) EXIT

Enter your choice of the menu option: 5
DIRECTORY MANIPULATION MENU

1) RETURN TO TOP LEVEL
2) CREATE A DIRECTORY
3) DELETE A DIRECTORY
4) LIST CONTENTS OF DIRECTORY
5) CHANGE DIRECTORY
6) PATH IDENTIFICATION
7) TEMPORARILY EXIT MENU SHELL

Enter your choice of menu options: 2
CREATE A DIRECTORY

"Directory name?  ?

You must enter the name of the directory you wish to make. A directory name is limited to 14 characters. Most characters may be used in a directory name, however, until you are familiar with the special characters and their meanings you should use only letters, numbers and the period.

"Directory name?  newdirect

III mkdir newdirect

Press return to continue
DIRECTORY MANIPULATION MENU

1) RETURN TO TOP LEVEL
2) CREATE A DIRECTORY
3) DELETE A DIRECTORY
4) LIST CONTENTS OF A DIRECTORY
5) CHANGE DIRECTORY
6) PATH IDENTIFICATION
7) TEMPORARILY EXIT MENU SHELL

Enter your choice of the menu options: 1
TOP IFVFI MENU

1) INTRODUCTION TO MENU SHELL
2) INTRODUCTION TO TERMINAL
3) CHANGE LEVEL OF PROMPT
4) FILE MANIPULATION
5) DIRECTORY MANIPULATION
6) OTHER SYSTEM FUNCTIONS
7) COMPILF AND EXECUTE PROGRAMS
8) TEMPORARILY EXIT MENU SHELL
9) EXIT

Enter your choice of the menu options: 9
BIBLIOGRAPHY


611, September, 1982.


APPENDIX A

PROGRAM LISTINGS
This program is a driver program to the Menu shell. The Menu shell is a menu driven interface to the UNIX operating system.

Program structure:

1. Initialization process
   a. User introduction
   b. Open root menu and place on stack maintaining path

2. Option processing
   Repeat until exit option is selected
   a. Display menu
   b. Accept user response
   c. Maintain path stack
   d. If leaf node - leaf node processing
   e. Retrieve most recent menu

3. Leaf node processing
   a. Execute program associated with menu option
   b. Evaluate exit status

All tasks are performed in separate shell programs with the exception of changing directories. A function is included within the main program which changes directories. It was necessary to include this in the main to assure that the directory change would be permanent. A separate shell program would return to the environment of the parent program and the directory change would be nullified.

The main program calls the following functions:

- intro: presents user introduction
- choose_menu_op: determines user response
- changedir: performs the change directory menu option
- clear: clears the screen
#include <stdio.h>
#include <signal.h>

#define ARRAYLEN 14
#define PATHLEN 50
#define COMMANDLEN 65
#define NUMOPT 12
#define MENUDEPTH 15
#define ROOT ".level-1"
#define rootpath "/acct/sli/mkh1141/menu/
#define QUESTION ?

main()
{
    char name[ARRAYLEN];
    char full_name[PATHLEN];
    char level[ARRAYLEN];
    char prog[ARRAYLEN];
    char command[COMMANDLEN];
    char *fileptr[NUMOPT];
    char *stackptr[MENUDEPTH];
    char *root_name;
    char *p;
    char *malloc();

    FILE *fopen();
    FILE *fpa[MENUDEPTH];

    int c;
    int i=1;
    int j=1;
    int nonmenu;
    int choice;
    int leaf=0;
    int cont=0;
    int prompt=1;
    int status;

    /* file name array */
    /* full file name array - including path from root */
    /* file name associated with a menu option */
    /* name of executable program associated with menu option */
    /* command array to forked as a child process */
    /* pointers to the files associated with a menu display */
    /* pointers maintaining path within menu structure */
    /* root menu */

    FILE *fopen();
    FILE *fpa[MENUDEPTH];

    int c;
    int i=1;
    int j=1;
    int nonmenu;
    int choice;
    int leaf=0;
    int cont=0;
    int prompt=1;
    int status;

    /* a character read and displayed */
    /* fileptr subscript */
    /* stackptr subscript */
    /* number of options within a given menu */
    /* menu option selected */
    /* leaf node flag: 0=non-leaf, 1=leaf */
    /* exit menu shell flag: 0=do not exit, 1=exit */
    /* prompt level */
    /* exit status when child process terminates */
/* Ignore the break and delete key, thus preventing user from a non-standard exit from menu shell. */

signal(SIGINT, SIG_IGN);

/* Initialization process */
/* User introduction */

clear();
intro();

/* Open root menu and place on stack maintaining menu path */

root_name = ROOT;
fpav[j] = fopen(root_name, "r");
p = malloc(ARRAYLEN);
strcpy(p, root_name);
stackptr[j] = p;

/\
/* Option processing */
while (cont == 0)
{
    /* Display menu */
    clear();
    while (leaf == 0)
    {
        c = getc(fpa[j]);
        if (c != 'l')
        {
            while (c != EOF)
            {
                if (c == '#')
                    /* read number of menu options */
                    fscanf(fpa[j], "%d", &nomenu);
                else if (c == '+')
                    /* read file name associated with menu option */
                    fscanf(fpa[j], "%s", &level);
                    p = malloc(ARRAYLEN);
                    strcpy(p, level);
                    fileptr[i++] = p;
                    /* record file associated with menu option */
                else
                    /* display menu one character at a time */
                    putchar(c);
                c = getc(fpa[j]);
            }
        }
        /* Accept user response */
        choice = choose_menu_op(nomenu);
        sprintf(full_name, "%s", rootpath, fileptr[choice]);
        fpa[++j] = fopen(full_name, "r");
        /* open file associated with menu choice */
        p = malloc(ARRAYLEN);
        strcpy(p, fileptr[choice]);
        stackptr[j] = p;
        l = 1;
        clear();
    }
    else
        leaf = 1;
}
/*
/* leaf node processing */

fscanf(fpa[j], "\%s", proc);
printf(command, \%s %d, rootpath, proc, promt);

/* Execute leaf file */

if (fork() == 0)
{
    signal(SIGINT, SIG_DFL);
    execl("/bin/sh", "sh", "-c", command, NULL);
}
wait(&status);

/* Evaluate exit status of child process */

switch (status)
{
    case 256:
        chdir(prompt);
        break;
    case 512:
        fclose(fpa[--j]);
        break;
    case 768:
        /* no action taken */
        /* If the break or delete key is pressed within an executable leaf node program, */
        /* the program will exit with a status of 3 (768). The driving program takes */
        /* menu display. */
        break;
    case 1024:
        /* verbose prompt */
        prompt = 1;
        break;
    case 1280:
        /* terse prompt */
        prompt = 2;
        break;
    case 1536:
        /* exit menu shell */
        cont = 1;
        break;
    default:
        break;
}

leaf = 0;

/* Retrieve previous menu file */

fclose(fpa[j--]);
fclose(fpa[j]);
sprintf(full_name, \%s\%s', rootpath, stackptr[j]);
fpa[j] = fopen(full_name, "r");
}
```c
int intro()
{
    long double x;  /* dummy declarations */

    printf("WELCOME TO THE MENU SHELL\n\n");  /* margin left */
    printf("A menu of available options will be displayed at your terminal.\n\n");  /* margin right */
    printf("You must type your choice of the options and press the return\n\n");  /* margin left */
    printf("key.\n\n");  /* margin right */
    sleep (10);

    return 0;
}
```
choose_menu_op(nomenu)

/****************************************************************************
** Function choose_menu_op()
** Purpose: To prompt user for his choice of the available menu options.
** Input parameters:
** nomenu - number of available options in current menu
** Output parameters:
** choice - the number of the menu option the user has selected
** Error checking:
** This function continues to prompt the user for his menu response until he has responded with a numeric value within the range acceptable for the current menu.
/****************************************************************************/

int nomenu;
{
    char garbage[50];
    int choice=0;

    while (choice <= 0 || choice > nomenu)
    {
        printf("Enter your choice of the menu options: ");
        if (scanf("%d", &choice) == 0)
        {
            scanf("%s", garbage);
            printf("%s only digits are valid menu options! \n"), garbage);
        }
        else if (choice <= 0 || choice > nomenu)
        {
            printf("%s is not a valid option\n", choice);
        }
    }
    return choice;
}
changedir (prompt)

/************************************************************
/* Function changedir(prompt)                           */
/* Purpose: To provide a means of changing directories. */
/* Input parameters:                                    */
/*          prompt - the level of prompt                */
/* Error checking:                                      */
/* This functions forks a shell program names          */
/* xcheckdir which checks to see if the named          */
/* directory does exist.                               */
/* This function follows the dialogue found in the shell*/
/* script programs.                                    */
/************************************************************

int prompt;
{
    char *direct_name;    /* new directory name */
    char command[COMMANLEN]; /* command to check if directory exists */
    char s[];              /* accepts response to continue */
    int dstatus;          /* status returned after directory is validated */

direct_name = malloc(COMMANLEN); /* Initially set directory name to ? */
    strcpy(direct_name, QUESTION);
    /* Initially set directory name to ? */
/* Present change directory dialog */

printf("\n
CHANGE DIRECTORY\n\n");
while (strcmp(direct_name, QUESTION) == 0)
{
    switch (prompt)
    {
    case 1:
        printf("\n\nWhat is the name of the directory to which you wish to change? ");
        break;
    case 2:
        printf("\n\nDirectory name? ");
        break;
    }
    scanf("%s", direct_name);
    if (strcmp(direct_name, QUESTION) == 0)
    {
        printf("\n\nYou must enter the full path name of the directory you wish to change\n");
        printf("\n\nTo with the following exceptions:\n");
        printf("\n\n1) A child of the current directory requires only the directory name.\n");
        printf("\n\n2) The parent of the current directory can be reached by entering \"../\"\n");
        printf("\n\n3) A sibling of the current directory can be reached by entering\n\n\n\n\n..\directory name\n");
    }
}

/* Check to see if the directory name exists */

sprintf(command,"%sxcheck_dir %s",rootpath,direct_name);
if (fork() == 0)
    execl("/bin/sh",sh,-c,command,NULL);
wait (&dstatus);

switch (dstatus)
{
    case 256:
        printf("\n\n!!!cd %s\n",direct_name);
        chdir(direct_name);
        printf("\n\nPress return to continue ");
        gets(s);
        gets(s);
        break;
    case 512:
        printf("\n\nDirectory does not exist!\n\n");
        sleep (2);
        break;

    /* directory does exist */

    /* directory does not exist */

}
```c
clear()

/*****************************************************************************/
/*                                                                           */
/* Function clear()                                                          */
/*                                                                           */
/*     Purpose: To clear the screen.                                         */
/*                                                                           */
/*****************************************************************************/
{
    system("clear");
}
```
Medit.c simulates the menu structure found in the Menu shell. It is a simple line oriented editor. It provides the following options:

1. Display contents of a file
2. Display a screen of text beginning with a given line
3. Display a given line
4. Append after a given line
5. Insert before a given line
6. Delete a given line
7. Exit the editor

Medit.c spawns the UNIX editor. The user is prompted for his choice of menu options. Medit also prompts for the appropriate arguments for each menu option. Medit sends the appropriate commands and arguments to the editor via the pipe that was opened when the editor process was spawned.

The main program calls the following functions:

- spawned: returns the filepointer of the file used for communication between medit.c and the editor process
- get_number: returns the number of the edit option selected by the user. It includes error checking to insure that the user enters a valid menu choice.
- dis_all_lines: displays the contents of a file
- dis_screen: displays a screen of text beginning with a given line number.
- singleln: displays a given line of a file
- append: append after a given line
- insert: insert above a given line
- delete_line: deletes a given line
- exit_edit: exits the editor
- cont: used to accept users response to continue edit menu
- clear: clears the screen
#include <stdio.h>
#include <signal.h>

#define ARRAYLEN 65 /* maximum length of file name to be edited */
#define MAXLINE 255 /* maximum length of line to be inserted or appended */
#define QUESTION "?" /* maximum length of line to be inserted or appended */
#define LOT ".

main()
{
    char *euit_file; /* file to be edited */
    FILE *spawnd(); /* function to spawn editor */
    FILE *fptr; /* file between medit.c and UNIX editor */

    int j=0; /* flag to continue in editor: 0=edit, 1=quit editor */
    int choice; /* number of edit option selected */
    int pid; /* process id */

    /* Ignore break and delete keys while in the editor. This is an */
    /* exception from the other leaf node programs. Ignoring the */
    /* interrupts will protect the user from exiting from the */
    /* without writing out the changes he has made. */

    signal (SIGINT,SIG_IGN);

}
/* Prompt user for the name of the file to be edited */

edit_file = QUESTION;
printf ("\n\nEDIT A FILE\n\n" );
printf ("Do not be alarmed if there is a delay in response. The edit options\n"");
printf ("require more time than most of the tasks found in the Menu shell.\n\n" );
while (strcmp (edit_file,QUESTION) == 0 )
{
    printf("\n\nWhat is the name of the file you wish to edit? ");
    scanf( "%s", edit_file );
    if (strcmp (edit_file,QUESTION) == 0 )
    {
        printf("\n You must enter the name of an existing file. If you give a new \n" );
        printf("file name you will see UNIX respond with a \? new name\%. Here you have \n"");
        printf("two options: \n" );
        printf("1) Quit the editing session by choosing menu option 7.\n"");
        printf("2) Append to the file by choosing menu option 4. This will\n"");
        printf("create a new file with the new name you have given.\n"");
    }
}

/* Spawn the editor */

printf("\n\n!!! edit %s\n" , edit_file);
if ((fp = spawnedit ( edit_file, &pid )) == NULL )
{
    puts("Cannot spawn the editor. Try the edit option again.\n"");
    exit (0 );
}
sleep (2 );

/* Present user with edit options until user wishes to exit */
/* editor */
while (j == 0)
{
    clear();
    fflush(stdout);

    printf("\n\n                EXIT MENU\n\n") ;

    printf("   1) DISPLAY CONTENTS OF FILE\n\n");
    printf("   2) DISPLAY A SCREEN OF TEXT BEGINNING WITH A GIVEN LINE\n\n");
    printf("   3) DISPLAY A GIVEN LINE\n\n");
    printf("   4) APPEND AFTER A GIVEN LINE\n\n");
    printf("   5) INSERT BEFORE A GIVEN LINE\n\n");
    printf("   6) DELETE A GIVEN LINE\n\n");
    printf("   7) EXIT THE EDITOR\n\n\n") ;
    printf("Enter your choice of the menu options: ");

    /* Accept user response to edit menu */
    choice = get_number();
    clear();

    /* Call function associated with menu option */
    switch (choice)
    {
        case 1:
            dis_all_lines(fptr);
            break;

        case 2:
            dis_screen(fptr);
            break;

        case 3:
            singlein (fptr);
            break;

        case 4:
            append (fptr);
            break;

        case 5:
            insert (fptr);
            break;

        case 6:
            delete_line(fptr);
            break;

        case 7:
            exit_edit(fptr);
            i = 1;
            break;
        default:
            break;
    }
}
flush(fptr);
sleep (0);
exit (0);
FILE *spawned(file, pid)

/**************************************************************************
/* Function FILE *spawned(file, pid)
/*
/* Purpose: To spawn the edit process.
/*
/* Input parameters:
/* file - file to be edited
/* pid - process id
/*
/* Output parameter:
/* fdes[1] - file descriptor for medit.c write end of pipe
/*
/**************************************************************************

char *file;  /* file to be edited */
int *pid;   /* process id */
{

int tmp0;    /* temporary file descriptor */
int fdes[2]; /* file descriptor array */
tmp0 = dup(0);
close(0);
pipe(fdes);

switch(*pid = fork())
{
    case -1:
        close(fdes[0]);
        close(fdes[1]);
        dup(tmp0);
        close(tmp0);
        return(NULL);
    case 0:
        close(fdes[1]);
        execcl("/bin/ed", "ed", file, 0);
        exit(1);
    default:
        close(fdes[0]);
        dup(tmp0);
        close(tmp0);
        return(fopen(fdes[1], "w"));
}
}

/*
/*
cont ()

/***************************************************************************/
/* Function cont () */
/* */
/* Purpose: To determine whether user is ready to continue exit */
/* process. */
/* */
/***************************************************************************/

{
    char s[];
    printf("\n\nPress return to continue ");
    gets(s);
} /*
dis_all_lines(fpftr)

/****************************************************************************
/* Function dis_all_lines(fpftr)                                      */
/* Purpose: To issue the command to the editor to display the       */
/* contents of a file.                                               */
/* Input parameter:                                                 */
/*    fpftr - file descriptor of pipe to editor                      */
/* Timing is controlled with the sleep command. This should be     */
/* modified to accommodate system load. The synchronization is      */
/* difficult to control. Best results are obtained if very small    */
/* files are displayed.                                             */
/****************************************************************************

FILE *fpftr;
{
    printf("\n\nDISPLAY CONTENTS OF FILE\n\n"");
    printf("It is possible to stop and restart the display of your file. Remember\n");
    printf("you can temporarily stop the screen enabling you to read the display. You\n");
    printf("can learn how to stop the display by choosing \"INTRODUCTION TO TERMINAL\"\n");
    printf("menu option at the top level menu.\n\n");
    fprintf(fpftr, 1, $fpftr)
    sleep (20);
} */
dis_screen(fp)

FLICTS

* Function dis_screen(fp)
* Purpose: To display one screen of text beginning with a given
  line number. The function prompts the user for the
  line number with which he wishes to begin the display.
* Input parameter:
  fp - file descriptor of pipe to editor
* Timing is controlled with the sleep command. This should be
  adjusted to accommodate system load.

FILE *fp;
{
  int lineno;
  char s[];

  printf("\n
  DISPLAY A SCREEN OF TEXT\n\n");
  printf("What line number do you wish to begin the display with? ");
  lineno = get_number();
  printf("\n%zd\n", lineno);
  printf(fp, "%zd\n", lineno);
  flush(fp);
  sleep (20);
  gets(s);
  cont();
}
*/
singleln(fpstr)

/**************************************************************************/
/* Function singleln(fpstr)                                             */
/* Purpose: To display a single line of text. The user is prompted for the */
/* line number he wishes to have displayed.                              */
/* Input parameter:                                                     */
/* fpstr - file descriptor of pipe to editor                            */
/* Functions called:                                                   */
/* get_number: returns an integer value                                 */
/* cont: determines when user is ready to continue the process          */
/**************************************************************************/

FILE *fpstr;
{
    int lineno;
    char s[];

    printf("DISPLAY A GIVEN LINE\n\n");
    printf("What line number do you wish to display?\n\n");
    lineno = get_number();
    printf("\n\n%*dp\n", lineno);
    fprintf(fpstr,多种方式，\n    lineno);
    fflush(fpstr);
    sleep (2);
    gets (s);
    cont ();
}
/*
append(fptr)

/*********************************************/
/* Function append(fptr) */
/* */
/* Purpose: To append text after a given line. The user is */
/* prompted to enter the line number to which he wishes to */
/* append text. He is also given instructions on how */
/* to end the append option. */
/* */
/* Input parameter: */
/* fpotr - file descriptor of pipe to editor */
/* */
/* Functions called: */
/* get_number: returns an integer value */
/* cont: determines when user is ready to continue the */
/* edit process */
/* */
/*********************************************/

FILE *fpotr;
{
  int lineno;

  char cmd[MAXLINE];

  printf("\n\n  APPEND AFTER A GIVEN LINE\n\n");
  printf("What line number do you wish to append after?  ");
  lineno = get_number();
  printf("\n\nAfter you see the UNIX command enter the lines you wish to add after\n");
  printf("\n\n\nline %d. When you have finished appending to your file you MUST enter \n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n}\n
sleep (2);
cont ();
} /*
```c
insert(fptr)

/**********************************************/
/** Function insert(fptr) */
/** 
** Purpose: To insert text before a given line number. The user is prompted for the line number above which he wishes to insert text. He is also given instructions on how to end the insert option. 
** 
** Input parameter: 
** fptr - file descriptor of pipe to editor 
** 
** Functions called: 
** get number: returns an integer value 
** cont: determines when user is read to continue the edit process 
** */
/***********************************************/

FILE *fptr;
{
    int lineno;
    char cmd[MAXLINE];

    printf("\n\n INSERT BEFORE A GIVEN LINE\n\n");
    printf("What line do you wish to insert before? ");
    lineno = get number();
    printf("\n\nAfter you see the UNIX command enter the lines you wish to insert before\n");
    printf("line %d. When you have finished inserting in your file, you MUST enter\n",lineno);
    printf("a line which contains only a \".\" The \".\" signals the editor that you\n");
    printf("have finished inserting.\n\n");
    printf("!! %d\n",lineno);
    fprintf(fptr,"%d
",lineno);
    gets(cmd);
    while (strcmp(cmd,IOT) !=0)
    {
        gets (cmd);
        fprintf(fptr,"%s
",cmd);
        fflush(fptr);
    }
    sleep (1);
    cont ();
}
/**
delete_line(fp)

*************************************************************************
/* Function delete_line(fp) */
/* Purpose: To delete a given line of text. The user is prompted */
/* for the number of the line he wishes to delete. */
/* Input parameter: */
/* fp - file descriptor of pipe to editor */
/* Functions called: */
/* get_number: returns an integer value */
/* cont: determines when user is ready to continue the */
/* edit process */
*************************************************************************

FILE *fp;
{
    int lineno;
    char s[];

    printf("\n\n DELETE A GIVEN LINE\n\n");
    printf("What is the number of the line you wish to delete? ");
    lineno = get_number();
    printf("\n\n!!! \%dd\n",lineno);
    fprintf(fp,\"%dd\n",lineno);
    fflush(fp);
    sleep (2);
    gets(s);
    cont();
}
exit_ed(fptr)

/******************************************************************************
 /* Function exit_ed(fptr)                                                */
 /* Purpose: To close the edit session.                                    */
 /* Input parameter:                                                      */
 /*     fptr - file descriptor of pipe to editor                          */
 /******************************************************************************/

FILE *fptr;
{
    int status;

    printf("\n\n");                  EXIT THE EDITOR\n\n);
    printf("!!! w\n");
    printf("!!! q\n");
    fprintf(fptr,"w\n");
    fprintf(fptr,"q\n");
    fflush(fptr);
    wait(&status);
    sleep (2);
} /*
get_number()

/****************************************************************************
/* Function get_number() */
/* Purpose: To accept an integer response from the user. */
/* Error checking: The user must enter a number. The function continues */
/* to prompt the user for a digit until the user responds */
/* accordingly. */
/****************************************************************************

{ char garbage[];
  int number;
  number = -1;
  while (number == -1)
  {
    if (scanf("%d", &number) == 0)
    {
      scanf("%s", garbage);
      printf("Please enter a digit: ");
    }
  }
  return number;
}*/
clear()

/*****************************

function clear()

Purpose: To clear the screen.

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

{
    system("clear");
}

file

line level source

1  0  #
2  0  #
3  0  #
4  0  #
5  0  #
6  0  #
7  0  #
8  0  #
9  0  #
10 0  #
11 0  #
12 0  #
13 0  #
14 0  #
15 0  #
16 0  #
17 0  #
18 0  #
19 0  question?
20 0  yes
21 0  s=?
22 0  t=?
23 0  u=?
24 0  v=?
25 0  w=?
26 0  x=?
27 0  y=?
28 0  z=?
29 0  echo C PROGRAM
30 0  
31 0  
32 0  
33 0  
34 0  
35 0  
36 0  
37 0  
38 0  
39 0  
40 0  
41 0  #
42 0  # Request source file
43 0  #
44 0  #
45 0  #
46 0  #
47 0  #
48 0  #
49 0  #
50 0  read

Purpose - To provide a means to compile and execute a "C" program.

Method - The user must name the "C" program he wishes to compile. If the user presses return when first asked the file name, the program will exit. The user is given 4 compile options, the option to execute the program after compilation and the option for input and output files.

Error checking - If the user enters the name of a nonexistent source file or a nonexistent input file, the user is notified that the file does not exist. The program then exits.

You should expect a delay when your program is compiled and executed. Program compilation requires more time than most of the tasks found in the Menu shell.

# Request source file

while /bin/test "$x" = "$question"
  do
echo 
  case $1 in
    1) echo "What is the name of the C source file you wish to compile? ";
    2) echo "What is the name of the C source file? ";
    esac
  esac
read x
if /bin/test "$x"
then
  if /bin/test "$x" = "$question"
  then
    echo "You must enter the name of a file which contains a C program. The file name must end in ".c""
  fi
else
  exit 0
  sleep 2
fi
done

# Determine whether program will be executed

if /bin/test -f "$x"
then
  while /bin/test "$t" = "$question"
  do
    echo "This option will compile your C file."
    echo "This will compile your C file."
    case $1 in
      1) echo -n "Do you also want your program executed (y or n)? ";
        echo "You will not want to have your program executed until it is free of errors. You should answer yes to this question until you have an error free compilation.";;
        t=y;;
      n | N) t=n;;
      *) t=?;;
        echo "";;
    esac
    read t
    case $t in
      y | Y) t=y;;
      n | N) t=n;;
      \) echo "You will not want to have your program executed until it is free of errors. You should answer yes to this question until you have an error free compilation.";;
        t=?;;
      *) t=?;;
        echo "";;
    esac
    done
  esac
else
  echo "File "$x" does not exist!"
fi
exit 0
fi
file
line
level
source

101  
  # Request compile options

102  
  #
  echo
  echo . When a C program is compiled the following options are available:
  echo . c suppress the loading phase
  echo . (Do NOT select this option if you wish to have
  echo . your program executed.)
  echo .

106  
  echo . f floating point simulation
  echo . (Do NOT select this option if you are working on
  echo . a VAX.)
  echo .

108  
  echo . O optimize object code
  echo . w suppress warning diagnostics
  echo .

113  
  echo . To see the additional options use the MANUAL menu option found
  echo . in the OTHER SYSTEM FUNCTIONS menu. You should ask for the "cc"
  echo . command.
  echo .

121  
  echo .
  echo "Enter the options you want one at a time followed by a return. When"
  echo "you have finished entering options simply press return."
  while /bin/test "$s"
  do
  echo .
  echo -n "Option (c f o w): 
    read $s
    case $s in
    c) c=c;
      s=??;
    f) f=f;
      s=??;
    o) 0=0;
      s=??;
    w) w=w;
      s=??;
    esac
    done
  
127  
  # Request input file
  #
  if /bin/test "$t" = "yes"
  then
    while /bin/test "$u = "question"
    do
      echo .
      case $1 in

```bash
1) echo -n "Do you have an input file (y or n)? " ;
esac
2) echo -n "Input file (y or n)? " ;
read u

case $u in
  y | Y) echo ";n What is the name of the input file? ";
      read v;
      if /bin/test -f "$v"
        then u=y
        else
          echo ". File $v does not exist! 
          sleep 2
          exit 0
          fi;
      n | N) v= "" ;
esac
 ogłos A program which expects data to be input from the terminal or a
data redirected from an input file. If your program is expecting
"data from standard input and the actual data is in a file answer
"to this question. ;"

  u=y; .
  echo " 
  esac
done

# Request output file

# while /bin/test ""$z = "$question"
do
  echo 
  case $1 in
    1) echo -n "Do you want the output placed in a file (y or n)? 
      esac
    2) echo -n "Output file (y or n)? " ;
      esac
  esac
  read z

  case $z in
  y | Y) while /bin/test ""$y = "$question"
    echo 
    echo -n "What is the name of the output file? 
    read y
    if /bin/test $y
```
if /bin/test "$v" = "$yes"
then
    echo -n '< $v'
    fi
if /bin/test "$z" = "$yes"
then
    echo -n ' > $y'
fi
if /bin/test "$v" = "$yes"
then
    if /bin/test "$z" = "$yes"
    then
        a.out < "$v" > "$y"
        else
            a.out < "$v"
        fi
    else
        if /bin/test "$z" = "$yes"
        then
            a.out > "$y"
            else
                a.out
        fi
fi
fi
fi
echo
read
exit
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xchagedir</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xchagedir</td>
<td>2</td>
<td>0</td>
<td>#xchagedir</td>
</tr>
<tr>
<td>xchagedir</td>
<td>3</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>4</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>5</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>6</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>7</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>8</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>9</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xchagedir</td>
<td>10</td>
<td>0</td>
<td>exit 1</td>
</tr>
</tbody>
</table>

Purpose - To provide a means for changing directories.

Method - This program has only one function. It exits with a status of 1 which signals the main program to call the change directory function.
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xchangelevel</code></td>
<td>1</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>2</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>3</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>4</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>5</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>6</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>7</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>8</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>9</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>10</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>11</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>12</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>13</td>
<td>0</td>
<td># <code>xchangelevel</code></td>
</tr>
<tr>
<td><code>xchangelevel</code></td>
<td>14</td>
<td>0</td>
<td>exit 2</td>
</tr>
</tbody>
</table>

**Purpose** - To provide a means for returning to the previous level in the menu structure.

**Method** - This program has only one function. It exits with a status of 2 which signals the main program to move one position within the array which represents the menu structure.
Purpose - To provide a means for changing the level of prompt.

Method - When the user selects a level of prompt the program will exit with a status of 4 for the verbose prompt and 5 for the terse prompt. If the user presses return the first time he is asked for the level of prompt no change in prompt will be made.

Error checking - The user must respond with a "1", "2" or "null". If he responds with any other character, he will be reminded of the appropriate responses.

File line level source

```
xchangeprompt 1 0 # xchangeprompt
xchangeprompt 2 0 #
xchangeprompt 3 0 #
xchangeprompt 4 0 #
xchangeprompt 5 0 #
xchangeprompt 6 0 #
xchangeprompt 7 0 #
xchangeprompt 8 0 #
xchangeprompt 9 0 #
xchangeprompt 10 0 #
xchangeprompt 11 0 #
xchangeprompt 12 0 #
xchangeprompt 13 0 #
xchangeprompt 14 0 #
xchangeprompt 15 0 #
xchangeprompt 16 0 question=
  x=
  y=
  echo
  CHANGE LEVEL OF PROMPT
  while /bin/test "$x" = "$question"
    echo " "
    echo " "
    case $1 in
      1) echo -n "What level of prompt do you wish (1 or 2)? ";
      2) echo -n "Prompt level? ";
        esac
      read x
      if /bin/test "$x"
        then
          if /bin/test "$x" = "$question"
            then
              echo " "
              echo " There are two levels of prompting. "1" is a verbose prompt while"n echo " prompts you with questions which help explain the desired task. "2"n echo " a terse prompt which can be used once you are familiar with the available"n echo " tasks."
            else
              exit 0
            fi
          else
            exit 0
          fi
      done
```
while /bin/test "$y" = "$question"

do

case $x in

1) exit 4;;

2) exit 5;;

) echo ";

echo "You must enter either 1 for verbose prompts or 2 for terse prompts."

echo -n "(1 or 2)? ";

read x;;

esac

done

exit 0
file line level source
xcheck_dir 1 0 # xcheck_dir
xcheck_dir 2 0 #
xcheck_dir 3 0 #
xcheck_dir 4 0 # Purpose - This program is called by the change directory function which is called by the main program. Xcheck_dir verifies that the directory to which the user wishes to change does exist.
xcheck_dir 5 0 #
xcheck_dir 6 0 #
xcheck_dir 7 0 #
xcheck_dir 8 0 #
xcheck_dir 9 0 # Method - The program uses the "test" command to verify that the directory named in the first parameter does exist. If the directory does exist the program ends with a status of 1 and if the directory does not exist the program exits with a status of 2.
xcheck_dir 10 0 #
xcheck_dir 11 0 #
xcheck_dir 12 0 #
xcheck_dir 13 0 #
xcheck_dir 14 0 #
xcheck_dir 15 0 #
xcheck_dir 16 0
xcheck_dir 17 0 if /bin/test -d $1
xcheck_dir 18 0 then
xcheck_dir 19 0
xcheck_dir 20 0 else
xcheck_dir 21 0 exit 1
xcheck_dir 22 0 exit 2
xcheck_dir 23 0 exit 2

file  line  level  source

```bash
  1  0  # xcopyfile
  2  0  # Purpose - To provide a means for copying a file.
  3  0  #
  4  0  # Method - The user must enter the name of the file he wishes
  5  0  # to have copied and the name of the file where the
  6  0  # copy will be placed. If the user presses return
  7  0  # when first asked for the source file, the program
  8  0  # will exit.
  9  0  #
 10  0  # Error checking - If the user gives the name of a nonexistent
 11  0  # file to be copied, he will be told the file does not
 12  0  # exist and the program will exit.
 13  0  #
 14  0  #
 15  0  #
 16  0  question=?
 17  0  x=?
 18  0  y=?
 19  0  #
 20  0  echo 'COPI A FILE'
 21  0  #
 22  0  while /bin/test "${x}" = "${question}"
 23  0  do
 24  0  
 25  0  echo " "
 26  0  case $1 in
 27  0  *#what is the name of the file you wish to copy? " ; ;
 28  0  1) echo -n "Source file? " ; ;
 29  0  2) echo -n "Destination file? " ; ;
 30  0  esac
 31  0  read x
 32  0  if test "${x}" then
 33  0  then
 34  0  echo " "
 35  0  echo " "
 36  0  You must enter the name of an existing file. If the file is not
 37  0  in the current directory the entire path name must be entered:
 38  0  fi
 39  0  fi
 40  0  fi
 41  0  done
 42  0  if /bin/test -f ${x}
 43  0  then
```
while /bin/test "\$y = "$question"
do
  echo ""
case $1 in
    1) echo -n "*Where do you wish to copy the file?  ";
      2) echo -n "*Destination file?  ";
esac
read y
if /bin/test "$y"
then
  if /bin/test "$y" = "$question"
  then
    echo ""
    echo "You must enter a new file name. If you enter the name of 
    echo "file the contents of that file will be lost. The file will be 
    echo "with the contents of "$x."
  else
    f1
  echo ""
  y=?
  echo "You must enter the name of the file where "$x will be copied."
  f1
done
else
  echo ""
  echo "!!! cp $x $y" 
cp $x $y
  echo ""
  echo "Press return to continue " 
  read z
else
  echo ""
  echo "file "$x does not exist!" 
sleep 2
fi
fi
exit 0
# createdir

Purpose - To provide a means for creating a directory.

Method - The user must name the directory he wishes to create. If the user presses return when first asked for the directory name, the program will exit.

Error checking - If the user enters the name of an existing directory, he will be told the directory exists and the program will exit.

```bash
question=

while /bin/test "$x" = "$question"
do
echo "".

if /bin/test "$x"
then
  if /bin/test "$x" = "$question"
    then
  echo "You must enter the name of the directory you wish to make. A dir name is limited to 14 characters. Most characters may be used in a directory name, however, until you are familiar with the special chara
echo "and their meanings you should use only letters, numbers and the period

else
  exit 0
fi

else
  done
fi

if /bin/test -d "$x"
then
```

```bash
```
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xcreatedir</td>
<td>51</td>
<td>0</td>
<td>echo ' '</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>52</td>
<td>0</td>
<td>echo &quot;Directory $x already exists!&quot;</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>53</td>
<td>0</td>
<td>sleep 2</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>54</td>
<td>0</td>
<td>echo ':'</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>55</td>
<td>0</td>
<td>echo &quot;!!! mkdir $x&quot;</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>56</td>
<td>0</td>
<td>mkdir $x</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>57</td>
<td>0</td>
<td>echo ':'</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>58</td>
<td>0</td>
<td>echo -n &quot;Press return to continue &quot;</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>59</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>60</td>
<td>0</td>
<td>fl</td>
</tr>
<tr>
<td>xcreatedir</td>
<td>61</td>
<td>0</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
```bash
# xcreatefile

```
<table>
<thead>
<tr>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>0</td>
<td>sleep 2</td>
</tr>
<tr>
<td>52</td>
<td>0</td>
<td>else</td>
</tr>
<tr>
<td>53</td>
<td>0</td>
<td>echo 'After seeing the UNIX command, enter the contents of your file. You'</td>
</tr>
<tr>
<td>54</td>
<td>0</td>
<td>echo 'are able to make corrections on the current line you are typing, however.'</td>
</tr>
<tr>
<td>55</td>
<td>0</td>
<td>echo 'once you press the return key for a given line, corrections must be done'</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>echo 'at a later time by choosing the &quot;EDIT A FILE&quot; menu option.'</td>
</tr>
<tr>
<td>57</td>
<td>0</td>
<td>echo 'TO INDICATE THAT YOU HAVE FINISHED ENTERING YOUR FILE PRESS THE &quot;CONTROL&quot;'</td>
</tr>
<tr>
<td>58</td>
<td>0</td>
<td>echo 'KEY AND THE &quot;D&quot; KEY SIMULTANEOUSLY.'</td>
</tr>
<tr>
<td>59</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>echo 'lll cat &gt; $x'</td>
</tr>
<tr>
<td>61</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>62</td>
<td>0</td>
<td>cat &gt; $x</td>
</tr>
<tr>
<td>63</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>65</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>66</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>67</td>
<td>0</td>
<td>echo '                               '</td>
</tr>
<tr>
<td>68</td>
<td>0</td>
<td>echo -n 'Press return to continue '</td>
</tr>
<tr>
<td>69</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>70</td>
<td>0</td>
<td>f1</td>
</tr>
<tr>
<td>71</td>
<td>0</td>
<td>exit 0</td>
</tr>
<tr>
<td>72</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Purpose: To provide a means for printing the current date and time.

Method: A brief explanation of the "date" command is given. The shell then issues the "date" command.

CURRENT DATE AND TIME

The following command returns the current date and time:

```bash
date
```

```bash
echo 'Press return to continue'
```

```bash
read z
```

```bash
exit 0
```
file

deletedir 1 0
  #xdeletedir
xdeletedir 2 0
  # Purpose - To provide a means for deleting a directory.
xdeletedir 3 0
  # Method - The user must name the directory he wishes to
deletedir 4 0
  # to delete. If the user presses return when
xdeletedir 5 0
  # first asked for the directory name, the program
xdeletedir 6 0
  # will exit.
xdeletedir 7 0
  # Error checking - If the user enters the name of a nonexistent
xdeletedir 8 0
  # directory, the program will tell him it does not
xdeletedir 9 0
  # exist and then exit.
xdeletedir 10 0
  #
xdeletedir 11 0
  # question=?
xdeletedir 12 0
  # x=?
xdeletedir 13 0
  echo

DELETED A DIRECTORY

  while /bin/test "$x" = "$question"
    do
      echo 
        case $1 in
          1) echo -n "What is the name of the directory you wish to delete? ");
          2) echo -n "Directory name? ");
        esac
        read x
        if /bin/test "$x"
          then
            if /bin/test "$x = $question"
              then
                echo "You must enter the name of an existing directory. The directory"
                echo "must be empty before it can be deleted."
              else
                exit 0
              fi
            done
          else
            echo "!!!rm -d $x"
        fi
  fi
xdeletedir 20 0
  .
xdeletedir 21 0
xdeletedir 22 0
  while /bin/test "$x" = "$question"
    do
      echo  
        case $1 in
          1) echo -n "What is the name of the directory you wish to delete? ");
          2) echo -n "Directory name? ");
        esac
        read x
        if /bin/test "$x"
          then
            if /bin/test "$x = $question"
              then
                echo "You must enter the name of an existing directory. The directory"
                echo "must be empty before it can be deleted."
              else
                exit 0
              fi
            done
          else
            echo "!!!rm -d $x"
        fi
xdeletedir 23 0
xdeletedir 24 0
xdeletedir 25 0
xdeletedir 26 0
  case $1 in
    1) echo -n "What is the name of the directory you wish to delete? ");
    2) echo -n "Directory name? ");
  esac
xdeletedir 27 0
xdeletedir 28 0
xdeletedir 29 0
xdeletedir 30 0
xdeletedir 31 0
xdeletedir 32 0
  if /bin/test "$x"
    then
      if /bin/test "$x = $question"
        then
          echo "You must enter the name of an existing directory. The directory"
          echo "must be empty before it can be deleted."
        else
          exit 0
        fi
      done
    else
      exit 0
    fi
xdeletedir 33 0
xdeletedir 34 0
xdeletedir 35 0
xdeletedir 36 0
xdeletedir 37 0
xdeletedir 38 0
xdeletedir 39 0
xdeletedir 40 0
xdeletedir 41 0
xdeletedir 42 0
xdeletedir 43 0
xdeletedir 44 0
xdeletedir 45 0
xdeletedir 46 0
  if /bin/test -d $x
xdeletedir 47 0
    then
      echo
xdeletedir 48 0
xdeletedir 49 0
xdeletedir 50 0
  echo "!!!rm -d $x"
xdeletedir Fri Jul 15 10:43:00 1983 Page 2

file line level source
xdeletedir 51 0      
xdeletedir 52 0      
xdeletedir 53 0      
xdeletedir 54 0      
xdeletedir 55 0      
xdeletedir 56 0      
xdeletedir 57 0      
xdeletedir 58 0      
xdeletedir 59 0      
xdeletedir 60 0      
xdeletedir 61 0      
xdeletedir 62 0      
xdeletedir 63 0      
xdeletedir 64 0      
xdeletedir 65 0      
xdeletedir 66 0      
xdeletedir 67 0      

rm -r $x

echo 

echo -n "Press return to continue"

read z

if [ ? ]; then
  echo "Directory $x does not exist!"
  sleep 2
fi

exit 0
file  line  level  source

# xdeletefile
# Purpose - To provide a means for deleting a file.
# Method - The user must name the file he wishes to delete.
#          If the user presses return when first asked for
#          name of the file he wishes to delete, the program
#          will exit.
# Error checking: If the user asks to delete a nonexistent
# file, the program will tell him it does not exist
# and exit.

# question=7
x=7

echo 'DELITE A FILE

while /bin/test "\$x" = \"\$question"
do
echo '

case $1 in
  1) echo -n "What is the name of the file you wish to delete? \";;
  2) echo -n 'File name? \";;
  esac
echo '
read x

if /bin/test "\$x"
then
  if /bin/test "\$x" = "\$question"
  then
echo "You must enter the name of an existing file. The file you name"
echo "will be permanently removed. Remember if you change your mind and"
echo "no longer wish to delete a file, simply press the return key."
  fi

else
  exit 0
fi

done

f1

else
  exit 0
fi

done

if /bin/test -f $x
then
echo

else
  exit 0
fi

exit 0
```bash
file line level source
xdeletefile 51 0
xdeletefile 52 0
xdeletefile 53 0
xdeletefile 54 0
xdeletefile 55 0
xdeletefile 56 0
xdeletefile 57 0
xdeletefile 58 0
xdeletefile 59 0
xdeletefile 60 0
xdeletefile 61 0
xdeletefile 62 0
xdeletefile 63 0
xdeletefile 64 0
xdeletefile 65 0
xdeletefile 66 0
xdeletefile 67 0
xdeletefile 68 0

echo "!!! rm $x"
rm $x
echo :
echo -n "Press return to continue"
read z
read z
echo ..
echo "File $x does not exist!"
sleep 2
f1
exit 0
```
file
line  level  source

# xdisplayfile
 Purpose - To provide a means for a scrolling display of a file.
 Method - The user enters the name of the file he wishes to display. If the user presses return when first asked for the name of the file to be displayed, the program will exit.
 Error checking - If the user enters the name of a nonexistent file, the program will tell him it does not exist and exit.

```
while /bin/test "$x" = "$question"
do
    case $1 in
        1) echo -n "What is the name of the file you wish to display? ";
        2) echo -n "File name? ";
    esac
    echo ";"
    read x
    if /bin/test "$x" then
        if /bin/test "$x" = "$question" then
            echo ";" You must enter the name of an existing file. Remember you are:
            echo "to stop the screen enabling you to read the display. You can learn:
            echo "how to stop the display by choosing the \"INTRODUCTION TO TERMINAL\" menu option at the top level menu.
            echo fi
        else
            exit 0
        fi
    done
    if /bin/test -f "$x" then
then
```
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdisplayfile</td>
<td>51</td>
<td>0</td>
<td>echo &quot;&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>52</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>53</td>
<td>0</td>
<td>echo &quot;!! cat $x&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>54</td>
<td>0</td>
<td>echo &quot;&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>55</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>56</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>57</td>
<td>0</td>
<td>cat $x</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>58</td>
<td>0</td>
<td>echo &quot;&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>59</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>60</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>61</td>
<td>0</td>
<td>echo -n &quot;Press return to continue &quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>62</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>63</td>
<td>0</td>
<td>else</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>64</td>
<td>0</td>
<td>echo &quot;&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>65</td>
<td>0</td>
<td>echo &quot;File $x does not exist!&quot;</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>66</td>
<td>0</td>
<td>sleep 2</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>67</td>
<td>0</td>
<td>fi</td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>68</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xdisplayfile</td>
<td>69</td>
<td>0</td>
<td>exit 0</td>
</tr>
<tr>
<td>file</td>
<td>line</td>
<td>level</td>
<td>source</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>xexit</td>
<td>1</td>
<td>0</td>
<td># xexit</td>
</tr>
<tr>
<td>xexit</td>
<td>2</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>3</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>4</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>5</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>6</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>7</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>8</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>9</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>xexit</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xexit</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xexit</td>
<td>12</td>
<td>0</td>
<td>exit 6</td>
</tr>
</tbody>
</table>

**Purpose** - To provide a means for exiting the Menu shell.

**Method** - This program has only one function. It exits with a status of 6 which signals the main program to exit the Menu shell.
file

line level source

```fortran
1 0
xfortran 2 0 #xfortran
xfortran 3 0 #
Purpose - To provide a means to compile and execute a fortran program.
xfortran 5 0 #
Method - The user must enter the name of the fortran program he wishes to compile. If he presses return when first asked, the program will exit. The user is given the option to execute the program after compilation. He is also given the option for 3 compile options. The user may use input and output files.
xfortran 10 0 #
xfortran 11 0 #
xfortran 12 0 # Error checking - if the user names a nonexistent fortran source file or a nonexistent input file, the user will be notified that the file does not exist and the program will exit.
xfortran 13 0 #
xfortran 14 0 #
xfortran 15 0 #
xfortran 16 0 #
xfortran 17 0 #
xfortran 18 0 #
xfortran 20 0 question=+</xfortran
xfortran 21 0 yes=y
xfortran 22 0 s=? # Compile citations flag
xfortran 23 0 t=? # Program execution flag
xfortran 24 0 u=? # Input file flag
xfortran 25 0 v=? # Input file name
xfortran 26 0 x=? # Source file name
xfortran 27 0 y=? # Output file name
xfortran 28 0 z=? # Output file flag
xfortran 29 0 echo '

FORTRAN PROGRAM

You should expect a delay when your program is compiled and executed.
Program compilation requires more time than most of the tasks found in the menu shell.

# Request source file
xfortran 42 0 #
xfortran 43 0 #
xfortran 44 0 while /bin/test "$x" = "$question"
 xfortran 46 0
do
xfortran 47 0 echo " "
exfortran 48 0 case $1 in
1) echo -n "What is the name of FORTRAN source file you wish to compile? ";
xfortran 49 0 2) echo -n "Source file? ";
xfortran 50 0 esac
```
```fortran
read x
if /bin/test "x"
then
  if /bin/test "x" = "question"
  then
    echo "You must enter the name of a file which contains a FORTRAN program. The file name must end in \".f\"
    f1
  else
    exit 0
  f1
done
# Determine whether program will also be executed
# 1)
if /bin/test -f "x"
then
  while /bin/test "$t" = "question"
  do
    echo "This option will compile your FORTRAN file."
    case $t in
      [y]
        t=y;;
      [n]
        t=n;;
      [?]
        echo "You will not want to have your program executed until it is free of errors. You should answer 'no' to this question until you have an error free compilation."
        t=?;
        echo "";;
    esac
  esac
else
done
else
  echo "File \"x\" does not exist!"
  sleep
  exit 0
fi
```
file

line

level

source

xfortran 101 0
xfortran 102 0
xfortran 103 0
xfortran 104 0
  # Request compile options
xfortran 105 0
  echo '. When a FORTRAN program is compiled the following options are available:
  echo '  c) suppress the loading phase' (Do NOT select this option if you wish to have
  echo your program executed.)
  echo '  f) floating point simulation' (Do NOT select this option if you are working on
  echo a VAX.)
  echo '  w) suppress warning diagnostics
  echo To see the additional options use the MANUAL menu item in the'
  echo 'top level menu OTHER SYSTEM FUNCTIONS. You should ask for the \ f??\'
  echo 'command.'
  echo 'Enter the options you want one at a time followed by a return. When'
  echo 'you have finished entering options simply press return.'
  echo ' while /bin/test "$s" do
xfortran 123 0
  echo '  "'
  echo 'do
  echo '    echo -n "Option (c f w): "
  echo '    read s
  xfortran 128 0
    case $s in
  xfortran 129 0
      c)  c=c;
  xfortran 130 0
      s=s?;
  xfortran 131 0
      f)  f=f;
  xfortran 132 0
      s=s?;
  xfortran 133 0
      w)  w=w;
  xfortran 134 0
      s=s?;
  xfortran 135 0
    esac
  xfortran 136 0
  done
  xfortran 137 0
  # Request input file
xfortran 138 0
  #
  xfortran 140 0
  if /bin/test "$t" = "$yes"
  xfortran 142 0
    then
  xfortran 143 0
      while /bin/test "$u = "$question"
  xfortran 144 0
        do
  xfortran 145 0
          echo ' 1) echo -n "Do you have an input file (y or n)?
  xfortran 146 0
            2) echo -n "Input file (y or n)?
  xfortran 147 0
            esac
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfortran</td>
<td>161</td>
<td>0</td>
<td>read u</td>
</tr>
<tr>
<td>xfortran</td>
<td>162</td>
<td>0</td>
<td>case $u in</td>
</tr>
<tr>
<td>xfortran</td>
<td>163</td>
<td>0</td>
<td>y</td>
</tr>
<tr>
<td>xfortran</td>
<td>164</td>
<td>0</td>
<td>echo -n</td>
</tr>
<tr>
<td>xfortran</td>
<td>165</td>
<td>0</td>
<td>if /bin/test -f &quot;$v&quot; then</td>
</tr>
<tr>
<td>xfortran</td>
<td>166</td>
<td>0</td>
<td>u=y</td>
</tr>
<tr>
<td>xfortran</td>
<td>167</td>
<td>0</td>
<td>else echo</td>
</tr>
<tr>
<td>xfortran</td>
<td>168</td>
<td>0</td>
<td>&quot;File $v does not exist!&quot;</td>
</tr>
<tr>
<td>xfortran</td>
<td>169</td>
<td>0</td>
<td>sleep 2</td>
</tr>
<tr>
<td>xfortran</td>
<td>170</td>
<td>0</td>
<td>exit 0</td>
</tr>
<tr>
<td>xfortran</td>
<td>171</td>
<td>0</td>
<td>fi;</td>
</tr>
<tr>
<td>xfortran</td>
<td>172</td>
<td>0</td>
<td>n</td>
</tr>
<tr>
<td>xfortran</td>
<td>173</td>
<td>0</td>
<td>? echo &quot;A program which expects data to be input from the terminal echo &quot;The data redirected from an input file. If your program is exec</td>
</tr>
<tr>
<td>xfortran</td>
<td>174</td>
<td>0</td>
<td>echo &quot;data from standard input and the actual data is in a file answer echo &quot;to this question.&quot; ;;</td>
</tr>
<tr>
<td>xfortran</td>
<td>175</td>
<td>0</td>
<td>u=y; echo</td>
</tr>
<tr>
<td>xfortran</td>
<td>176</td>
<td>0</td>
<td>done</td>
</tr>
<tr>
<td>xfortran</td>
<td>177</td>
<td>0</td>
<td>esac</td>
</tr>
<tr>
<td>xfortran</td>
<td>178</td>
<td>0</td>
<td>Request output file</td>
</tr>
<tr>
<td>xfortran</td>
<td>179</td>
<td>0</td>
<td>while /bin/test &quot;$z&quot; == &quot;$question&quot; do</td>
</tr>
<tr>
<td>xfortran</td>
<td>180</td>
<td>0</td>
<td>echo ..</td>
</tr>
<tr>
<td>xfortran</td>
<td>181</td>
<td>0</td>
<td>case $z in</td>
</tr>
<tr>
<td>xfortran</td>
<td>182</td>
<td>0</td>
<td>1 echo -n &quot;Do you want the output placed in a file (y or n)? ;;</td>
</tr>
<tr>
<td>xfortran</td>
<td>183</td>
<td>0</td>
<td>2 echo -n &quot;Output file (y or n)? ;;</td>
</tr>
<tr>
<td>xfortran</td>
<td>184</td>
<td>0</td>
<td>esac</td>
</tr>
<tr>
<td>xfortran</td>
<td>185</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>xfortran</td>
<td>186</td>
<td>0</td>
<td>case $z in</td>
</tr>
<tr>
<td>xfortran</td>
<td>187</td>
<td>0</td>
<td>y</td>
</tr>
<tr>
<td>xfortran</td>
<td>188</td>
<td>0</td>
<td>echo ..</td>
</tr>
<tr>
<td>xfortran</td>
<td>189</td>
<td>0</td>
<td>echo -n &quot;What is the name of the output file?&quot;</td>
</tr>
<tr>
<td>xfortran</td>
<td>190</td>
<td>0</td>
<td>if /bin/test &quot;$y&quot; then</td>
</tr>
<tr>
<td>xfortran</td>
<td>191</td>
<td>0</td>
<td>z=y</td>
</tr>
<tr>
<td>xfortran</td>
<td>192</td>
<td>0</td>
<td>else</td>
</tr>
</tbody>
</table>
file
line  level  source

xfortran  201  0          ;
xfortran  202  0          fi
xfortran  203  0          done
xfortran  204  0          esac
xfortran  205  0          f1
xfortran  206  0
xfortran  207  0
xfortran  208  0          # Echo and execute compile commands
xfortran  209  0          #
xfortran  210  0          echo "
xfortran  211  0          echo -n "!!! f?? "
xfortran  212  0          if /bin/test "$c" then
xfortran  213  0          echo -n "-c "
xfortran  214  0          f1
xfortran  215  0          if /bin/test "$f" then
xfortran  216  0          echo -n "-f "
xfortran  217  0          f1
xfortran  218  0          if /bin/test "$w" then
xfortran  219  0          echo -n "-w "
xfortran  220  0          echo "$x"
xfortran  221  0
xfortran  222  0          if /bin/test "$c" then
xfortran  223  0          if /bin/test "$f" then
xfortran  224  0          if /bin/test "$w" then
xfortran  225  0          if /bin/test "$x"

xfortran  226  0          if /bin/test "$c" then
xfortran  227  0          if /bin/test "$f" then
xfortran  228  0          if /bin/test "$w" then
xfortran  229  0          if /bin/test "$x" then
xfortran  230  0          if /bin/test "$c" then
xfortran  231  0          if /bin/test "$f" then
xfortran  232  0          if /bin/test "$w" then
xfortran  233  0          if /bin/test "$x" then
xfortran  234  0          if /bin/test "$c" then
xfortran  235  0          if /bin/test "$f" then
xfortran  236  0          if /bin/test "$w" then
xfortran  237  0          if /bin/test "$x" then

file

251 else
252 f77 -$c $x
253 f1
254 else
255 if /bin/test "$f"
256 then
257 if /bin/test "$w"
258 then
259 f77 -$f -$w $x
260 f1
261 else
262 f77 -$f $x
263 f1
264 else
265 if /bin/test "$w"
266 then
267 f77 -$w $x
268 f1
269 else
270 f77 $x
271 f1
272 f1
273 # Echo command to execute program and execute program
274 #
275 h
276 #
277 if /bin/test "$t" = "$yes"
278 then
279 echo -n "$!la.out"
280 if /bin/test "$v" = "$yes"
281 then
282 echo "$v"
283 f1
284 if /bin/test "$z" = "$yes"
285 then
286 echo "$y"
287 f1
288 echo "$u" = "$yes"
289 then
290 if /bin/test "$z" = "$yes"
291 then
292 a.out < "$v" "$y"
293 f1
294 else
295 a.out < "$v"
296 f1
297 else
298 if /bin/test "$z" = "$yes"
299 then
300 a.out > "$y"
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfortran</td>
<td>301</td>
<td>0</td>
<td>else</td>
</tr>
<tr>
<td>xfortran</td>
<td>302</td>
<td>0</td>
<td>a.out</td>
</tr>
<tr>
<td>xfortran</td>
<td>303</td>
<td>0</td>
<td>f1</td>
</tr>
<tr>
<td>xfortran</td>
<td>304</td>
<td>0</td>
<td>f1</td>
</tr>
<tr>
<td>xfortran</td>
<td>305</td>
<td>0</td>
<td>f1</td>
</tr>
<tr>
<td>xfortran</td>
<td>306</td>
<td>0</td>
<td>echo ;</td>
</tr>
<tr>
<td>xfortran</td>
<td>307</td>
<td>0</td>
<td>;</td>
</tr>
<tr>
<td>xfortran</td>
<td>308</td>
<td>0</td>
<td>echo -n &quot;Press return to continue &quot;</td>
</tr>
<tr>
<td>xfortran</td>
<td>309</td>
<td>0</td>
<td>read y</td>
</tr>
<tr>
<td>xfortran</td>
<td>310</td>
<td>0</td>
<td>exit 0</td>
</tr>
<tr>
<td>file</td>
<td>line</td>
<td>level</td>
<td>source</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>xintroduction</td>
<td>1</td>
<td>0</td>
<td># xintroduction</td>
</tr>
<tr>
<td>xintroduction</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>4</td>
<td>0</td>
<td># Purpose - to display a file which contains an introduction</td>
</tr>
<tr>
<td>xintroduction</td>
<td>5</td>
<td>0</td>
<td># the system</td>
</tr>
<tr>
<td>xintroduction</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>7</td>
<td>0</td>
<td># Method - A file called introduction must exist. This file</td>
</tr>
<tr>
<td>xintroduction</td>
<td>8</td>
<td>0</td>
<td>is displayed using the more command, thus the contents</td>
</tr>
<tr>
<td>xintroduction</td>
<td>9</td>
<td>0</td>
<td>of the file is displayed one screen at a time.</td>
</tr>
<tr>
<td>xintroduction</td>
<td>10</td>
<td>0</td>
<td>The file introduction can be easily modified to</td>
</tr>
<tr>
<td>xintroduction</td>
<td>11</td>
<td>0</td>
<td>.reflect changes to the system.</td>
</tr>
<tr>
<td>xintroduction</td>
<td>12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>13</td>
<td>0</td>
<td>echo '------------------------------'</td>
</tr>
<tr>
<td>xintroduction</td>
<td>14</td>
<td>0</td>
<td>MENU SHELL INTRODUCTION</td>
</tr>
<tr>
<td>xintroduction</td>
<td>15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>16</td>
<td>0</td>
<td>cat introduction</td>
</tr>
<tr>
<td>xintroduction</td>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>18</td>
<td>0</td>
<td>echo -n &quot;Press return to continue&quot;</td>
</tr>
<tr>
<td>xintroduction</td>
<td>19</td>
<td>0</td>
<td>read x</td>
</tr>
<tr>
<td>xintroduction</td>
<td>20</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xintroduction</td>
<td>21</td>
<td>0</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
1) echo -n "Do you wish a brief or long listing (b or l)? "
2) echo -n "\(b\) or l)? "

read y
if /bin/test "$y" = "$question"
then
  echo " "
echo " A brief listing will only give the names of the files whereas a long listing gives the mode, number of links, owner, size in bytes and time of last modification in addition to the file names."
fi
while /bin/test "$z"
do
  echo ""
case "$y" in
    b) echo "!!! 1 $x"
      1 $x;
      z= ;
      l= 1s -l $x
      z= ;
      *) echo " You must enter '/b' for a brief listing or '/l' for a long listing:"
        echo -n "(b or l)? "
        read y;;
  esac
done
exit 0
file line level source

xlistdirfiles 1 0 #xlistdirfiles
xlistdirfiles 2 0 #
xlistdirfiles 3 0 # Purpose - To provide a means for displaying the files in a
xlistdirfiles 4 0 # directory.
xlistdirfiles 5 0 #
xlistdirfiles 6 0 #
xlistdirfiles 7 0 # Method - The user must name the directory of which he
xlistdirfiles 8 0 # wishes to see the contents. The user must
xlistdirfiles 9 0 # also specify "b" or "l" for a brief or long
xlistdirfiles 10 0 # listing respectively. If the user presses
xlistdirfiles 11 0 # return when first asked for the name of the
xlistdirfiles 12 0 # directory, the program will exit.
xlistdirfiles 13 0 #
xlistdirfiles 14 0 # Error checking - The user is notified if he enters the
xlistdirfiles 15 0 # an invalid option.
xlistdirfiles 16 0 #
xlistdirfiles 17 0 # question?
xlistdirfiles 18 0 #
xlistdirfiles 19 0 # x= ?
xlistdirfiles 20 0 #
xlistdirfiles 21 0 #
xlistdirfiles 22 0 # echo .
xlistdirfiles 23 0 #
xlistdirfiles 24 0 #
xlistdirfiles 25 0 #
xlistdirfiles 26 0 #
xlistdirfiles 27 0 #
xlistdirfiles 28 0 # while /bin/test "$x" = "$question"
xlistdirfiles 29 0 # do
xlistdirfiles 30 0 #   echo " ."
xlistdirfiles 31 0 #   case $1 in
xlistdirfiles 32 0 #        1) echo -n "$What is the name of the directory of which you wish to see the contents"
xlistdirfiles 33 0 #           2) echo -n "Directory name? ";
xlistdirfiles 34 0 # esac
xlistdirfiles 35 0 # read x
xlistdirfiles 36 0 #
xlistdirfiles 37 0 # if /bin/test "$x" = "$question"
xlistdirfiles 38 0 # then
xlistdirfiles 39 0 #   echo " ."
xlistdirfiles 40 0 #   echo " You may see the contents of any directory in your account. Enter"
xlistdirfiles 41 0 #   echo " the full path name of the directory whose contents you wish to see. If"
xlistdirfiles 42 0 #   echo " you wish to see the contents of the current directory simply press return."
xlistdirfiles 43 0 # fi
xlistdirfiles 44 0 #
xlistdirfiles 45 0 # done
xlistdirfiles 46 0 #
xlistdirfiles 47 0 # while /bin/test "$y" = "$question"
xlistdirfiles 48 0 # do
xlistdirfiles 49 0 # case $1 in

LIST CONTENTS OF A DIRECTORY
xlistfiles Fri Jul 15 10:45:00 1983 Page 1

file line level source

xlistfiles 1 0  # xlistfiles
xlistfiles 2 0  #
 Purpose - To provide a means for listing the files in the
current directory.
xlistfiles 4 0  #
 Method - The user must specify a brief or long listing of
files.
xlistfiles 6 0  #
 Error checking - The user must specify "b" or "l" for a
brief or long listing respectively. The user is
notified of an invalid option.
xlistfiles 14 0  #
 question= ?
xlistfiles 15 0  x = ?
xlistfiles 16 0  y = ?
xlistfiles 17 0  
 echo ' 

 LIST NAMES OF FILES

 while /bin/test "$x" = "$question" 
 do
 xlistfiles 25 0  
 echo ' 
 xlistfiles 27 0  case $1 in
 xlistfiles 28 0  1) echo -n "Do you wish a brief or long listing (b or l)? ";
xlistfiles 29 0  2) echo -n "s(t or l)? ";
xlistfiles 30 0  esac
 xlistfiles 31 0  
 read x
 xlistfiles 33 0  if /bin/test "$x"
 xlistfiles 34 0  then
 xlistfiles 35 0  if /bin/test "$x" = "$question"
 xlistfiles 37 0  then
 xlistfiles 39 0  echo ' 
 xlistfiles 40 0  A brief listing will only give the names of the files whereas a l
 xlistfiles 42 0  echo "listing gives the mode, number of links, owner, size in bytes and time
 xlistfiles 44 0  
 xlistfiles 46 0  done
 xlistfiles 47 0  while /bin/test "$y" = "$question"
 xlistfiles 49 0  do
 xlistfiles 50 0  

xlistfiles Fri Jul 15 10:50 1982 Fare 2

<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlistfiles</td>
<td>51</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>52</td>
<td>0</td>
<td>case $x in t) echo &quot;!!! 1&quot;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>53</td>
<td>0</td>
<td>l; y=0;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>54</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>55</td>
<td>0</td>
<td>1) echo &quot;!!! 1s -1&quot;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>56</td>
<td>0</td>
<td>ls -1;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>57</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>58</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>59</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>60</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>61</td>
<td>0</td>
<td>*) echo &quot;You must enter &quot;b&quot; for a brief listing or &quot;l&quot; for a long listing&quot;;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>62</td>
<td>0</td>
<td>echo -n &quot;(b or l)?&quot;;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>63</td>
<td>0</td>
<td>read x;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>64</td>
<td>0</td>
<td>esac</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>65</td>
<td>0</td>
<td>done</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>66</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>67</td>
<td>0</td>
<td>echo .</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>68</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>69</td>
<td>0</td>
<td>echo -n &quot;Press return to continue.&quot;</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>70</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>xlistfiles</td>
<td>71</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xlistfiles</td>
<td>72</td>
<td>0</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
file line level source

xmanual 1 0
xmanual 2 0 #xmanual
xmanual 3 0 #
xmanual 4 0 # Purpose - To provide a means of displaying a command
description found in the on-line manual.
xmanual 5 0 #
xmanual 6 0 # Method - The user must name the command he wishes to
have displayed. If he presses return when
first asked for the command, the program will
exit.
xmanual 10 0 #
xmanual 11 0 #
xmanual 12 0 question=?
xmanual 13 0 x=

DISPLAY UNIX MANUAL
The "man" command is used to display the appropriate section of
the UNIX manual for the UNIX command you specify.

while /bin/test "$x" = "$question"
do
  echo ' 
  case $1 in
    1) echo -n "What UNIX command do you wish to see? ";
    2) echo -n "UNIX command? ";
      esac
      read x
      if /bin/test "$x"
        then if /bin/test "$x" = "$question"
          then
            echo ' You must enter a UNIX command exactly as it is used. For'
            echo ' example if you wish to learn about the copy command enter "/cp"'
          fi
          else
            exit 0
          fi
        done
    echo ' 
  esac

xmanual 48 0 "man" $x
xmanual 49 0
xmanual 50 0
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmanual</td>
<td>51</td>
<td>0</td>
<td>man $x</td>
</tr>
<tr>
<td>xmanual</td>
<td>52</td>
<td>0</td>
<td>echo '</td>
</tr>
<tr>
<td>xmanual</td>
<td>53</td>
<td>0</td>
<td>echo -n &quot;Press return to continue</td>
</tr>
<tr>
<td>xmanual</td>
<td>54</td>
<td>0</td>
<td>read 2</td>
</tr>
<tr>
<td>xmanual</td>
<td>55</td>
<td>0</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
file

line

level

source

xmorefile

1

0

# xmorefile

xmorefile

2

0

#

xmorefile

3

0

#

xmorefile

4

0

#

xmorefile

5

0

#

xmorefile

6

0

#

xmorefile

7

0

#

xmorefile

8

0

#

xmorefile

9

0

#

xmorefile

10

0

#

xmorefile

11

0

#

xmorefile

12

0

#

xmorefile

13

0

#

xmorefile

14

0

#

xmorefile

15

0

#

xmorefile

16

0

question=?

xmorefile

17

0

x=

xmorefile

18

0

echo "

DISPLAY ONE SCREEN OF TEXT AT A TIME

while /bin/test "$x" = "$question"

do

echo ;

while /bin/test "$x"

do

if /bin/test "$x"

then

if /bin/test "$x" = "$question"

then

echo ""

else

f1

else

exit 0

done

f1

then

if /bin/test "$x"
```bash
file line level source
xmorefile 61 0   echo :

xmorefile 62 0   echo '!!! more -d $x'

xmorefile 63 0   echo ""

xmorefile 64 0   echo ""

xmorefile 65 0   echo "Press return to continue"

xmorefile 66 0   read z

xmorefile 67 0   else

xmorefile 68 0   echo ""

xmorefile 69 0   echo "File $x does not exist!"

xmorefile 70 0   sleep z

xmorefile 71 0   if

xmorefile 72 0   exit 0
```
Purpose - To provide a means for moving a file from one
name to another.

Method - The user is asked for the name of the file which
he wishes to move and the name of the file where
he wishes it moved to. If the user presses return
as the source file when first asked the program
will exit.

Error checking - If the user gives the name of a nonexistent
file as the source file, he will be told the file
does not exist and the program will exit.

MOVE A FILE (CHANGE ITS NAME)

while /bin/test ""$x"" = "$question"
    do echo .
    echo .
    case $1 in
        1) echo -n "What is the name of the file you wish to move? ";
           esac
        2) echo -n "Old file? ";
           read x
           if /bin/test "$x"
            then
                if test "$x" = "$question"
                    then
                    echo .
                    echo "You must enter the name of an existing file. If the file is not"
                    echo "in the current directory the entire path must be entered."
                else
                    fi
                    exit 0
                fi
            done
file line level source

`xmovefile` 51 0 if `/bin/test -f $x`
`xmovefile` 52 0 then
`xmovefile` 53 0 while test "" - $question
`xmovefile` 54 0 do
  `xmovefile` 55 0 echo ""
  case $1 in
                      1) echo -n "What is the name of the file where you wish to move $x?"
                                 ;;
                      2) echo -n "New file?"
                                 ;;
                      esac
  `xmovefile` 56 0 read y
`xmovefile` 57 0 if `/bin/test "$y"
then
  if test "$y" = "$question"
  then
    `xmovefile` 59 0 echo ""
    You must enter the new name for $x. The contents of $x
    will be moved to the file name you enter. If you enter the name
    of an existing file the contents of that file will be lost. The file
    $x will be replaced with the contents of $y."
  `xmovefile` 60 0 else
    `xmovefile` 61 0 echo ""
    You must enter the file name where $x will be moved.
    $y=?
  `xmovefile` 62 0 fi
`xmovefile` 63 0 fi
`xmovefile` 64 0 done
`xmovefile` 65 0 echo ""
`xmovefile` 66 0 echo "$mv $x $y"
`xmovefile` 67 0 mv $x $y
`xmovefile` 68 0 echo ""
`xmovefile` 69 0 echo -n "Press return to continue"
`xmovefile` 70 0 read z
`xmovefile` 71 0 echo ""
`xmovefile` 72 0 echo -n "File $x does not exist!"
`xmovefile` 73 0 sleep 2
`xmovefile` 74 0 fi
`xmovefile` 75 0 exit 0
Purpose - To provide a means for displaying a file in octal, ASCII, decimal, or hexadecimal form.

Method - The user must enter the name of the file he wishes to display. If the user presses return when first asked for the file name, the program will exit.

The user must specify the form of display he desires from the list of options he is given.

Error checking - If the user enters a nonexistent file, the program will tell him the file does exist and then exit. The program verifies that the user selects a valid format for display.

```
while /bin/test "\$x" = "$question"
  do
    echo "OCTAL DISPLAY OF A FILE"
    echo "FILE NAME: \$x"
    case $1 in
      1) echo -n "What is the name of the file you wish to display? "
         ;;
      2) echo -n "File name? "
         ;;
    esac
    read x
    if /bin/test "$x"
      then
        if /bin/test "$x" = "$question"
          then
            echo "You must enter the name of an existing file. If the file is not"
            echo "in the current directory the entire path name must be given."
            return
          fi
        else
          exit 0
        fi
    done
```
```bash
xoctalfileFri Jul 1 12:53:51 1983 Page 2

file line level source

xoctalfile 51 0 if /bin/test -f $x
xoctalfile 52 0 then
xoctalfile 53 0 while /bin/test "$y" = "question"
  do
  echo ""
  echo "Octal displays may be done in one of the following formats:"
  echo "  a) Interpret bytes in octal"
  echo "  b) Interpret bytes in ASCII"
  echo "  c) Interpret bytes in ASCII"
  echo "  d) Interpret words in decimal"
  echo "  e) Interpret words in octal"
  echo "  x) Interpret words in hex"
  echo ""
  echo -n "Enter \"b, c, d, o, w\" indicating the format you wish. "
  read y
  case $y in
    b|c|d|o|w) echo "!!! od -$y $x";;
    *) y=$y;;
  esac
xoctalfile 74 0 done
xoctalfile 75 0 echo "."

xoctalfile 76 0 od -$y $x
xoctalfile 77 0 echo ""

xoctalfile 78 0 else
xoctalfile 79 0 echo "" else
xoctalfile 81 0 echo "File $x does not exist!"
xoctalfile 83 0 echo "Press return to continue "
  read z
xoctalfile 84 0
xoctalfile 85 0 exit 0
```
file

# Purpose - To provide a means to compile and execute a Pascal program.
Method - The user the name the file he wishes to compile.
If the user presses return when first asked the file name, the program will exit. The user is given the option for an input file and listing file.
Error checking - If the user names a nonexistent input file or a nonexistent pascal source file, the user will be notified that the file does not exist.
The program will then exit.

question=y

#listing flag
# input file flag
# input file name
# source file name
# listing in a file flag
# listing file name

echo

PASCAL PROGRAM

You should expect a delay when your program is compiled and executed.
Program compilation requires more time than most of the tasks found in the Menu shell.

# Request source file

while /bin/test "$x" = "$question"
do
echo
case $1 in
  1) echo -n "What is the name of PASCAI source file you wish to run? ";;
  2) echo -n "Source file? ";;
esac
read x
if /bin/test "$x"
file       line   level   source

xascal    01    0 then if /bin/test "$x" = "$question"
                   then echo .
                   echo you must enter the name of a file which contains a PASCAL
                   echo program. The file name must end in ".p"
                   fi
                   else fi
                   done
xascal    02    0 # Request input file
xascal    03    0 # if /bin/test -f "$x"
xascal    03    0 then while /bin/test "$u" = "$question"
                   do echo .
                       case $1 in
                           1) echo -n "Do you have an input file (y or n)? ";
                           2) echo -n "Input file (y or n)? ";
                              esac
                       read u
                       case $u in
                           y | Y) echo .
                               echo -n "What is the name of the input file? ";
                               read v;
                               if /bin/test -f "$v"
                                   then echo
                               else echo
                                   echo 'File $v does not exist!' .
                                   exit 0
                               fi;
                           n | N) u= ;;
                           *) u= ;;
                              echo .
                       esac
                   esac
xascal    100   0 ^}
Determine if user desires a listing:

```bash
while /bin/test "$t = "$question
do
    echo ;
    case $1 in
        1) echo -n "Do you want a listing of your program (y or n)? 
        2) echo -n "Listing of program (y or n)? 
        esac
    read t
    case $t in
        y | Y) y=y;;
    esac
# Compile and execute program with no listing
n | N) echo ";
    if /bin/test "$u
    then
        echo "!!! pix $x < $v"
        pix $x < $v
    else
        echo "!!! pix $x"
        pix $x
    fi;
w=ij;
    \) echo "Any errors in the source file will be printed out, however:"
    echo "To get a complete listing of your program you must request one."
    echo ""
    esac
    esac
    esac
    esac
    done
else
    echo "File $x does not exist!"
exit 0
```

echo -n "Do you want the listing printed to a file (y or n)?  
read z

case $z in
  y | Y) z=y;;
  Compile and execute programs with listing at terminal
  n | N) echo T;
      if /bin/test "$u
            then
                  echo '!!! pic -l $x < $v'
                  pix -l $x < $v
            else
                  echo '!!! pic -1 $x'
                  pix -1 $x
            fi;
      wx$i:;"
      \? echo . The listing will be printed at the terminal unless you"
      echo "you request the listing to be placed in a file. The program"
      echo "output will also be placed in the file.";;
      *) z=?;
      echo ";;
    done esac
fi

# Request listing file name
if /bin/test "$z = "$yes
then
  while /bin/test "$w = "$question'
    do
      echo T
        case $w in
          1) echo -n 'What is the name of the listing file?  
            2) echo -n "listing file?  
            esac
      read w
      case $w in
        \? echo ; You must enter a new file name. If you enter the name of an
        echo "file the contents of that file will be lost. The file will be re
        echo "with the contents of the listing file.";;
        *) if /bin/test "$w
          then
file  line  level  source

\$pascal  201  0  echo
\$pascal  202  0  else
\$pascal  203  0  w=7
\$pascal  204  0  echo "",
\$pascal  205  0  fi;;
\$pascal  206  0  esac
\$pascal  207  0  done
\$pascal  208  0  fi
\$pascal  209  0
\$pascal  210  0 #
\$pascal  211  0 # Compile and execute program with listing file
\$pascal  212  0 #
\$pascal  213  0 if /bin/test $w"
\$pascal  214  0 then
\$pascal  215  0 echo "",
\$pascal  216  0 if /bin/test "$v
\$pascal  217  0 then
\$pascal  218  0 echo "$u pix -1 $x < $v > $w
\$pascal  219  0 pix -1 $x < $v > $w
\$pascal  220  0 else
\$pascal  221  0 echo "$v pix -1 $x > $w",
\$pascal  222  0 pix -1 $x > $w
\$pascal  223  0 fi
\$pascal  224  0 fi
\$pascal  225  0 echo
\$pascal  226  0
\$pascal  227  0 echo -n "Press return to continue"
\$pascal  228  0 read z
\$pascal  229  0
\$pascal  230  0 exit 0
file

<table>
<thead>
<tr>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpassword</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
| xpassword | 2 | 0 | #xpassword
| xpassword | 3 | 0 |
| xpassword | 4 | 0 | # Purpose - To provide a means to change the users password.
| xpassword | 5 | 0 |
| xpassword | 6 | 0 | # Method - A brief explanation of the "passwd" command is given. The shell then issues the "passwd" command.
| xpassword | 7 | 0 |
| xpassword | 8 | 0 | echo

**CHANGE PASSWORD**

You may change your password by using the "passwd" command. "passwd" will ask you for your current password and then ask for your new password two times. If your new password is less than 6 characters in length you will be asked for a longer password since a longer password is generally more secure; however if you persist a short password will be accepted.

```bash
$ passwd
```
**PATH IDENTIFICATION**

The UNIX operating system uses a tree structure of directories. It is possible to see the ancestors of your current directory by using the `pwd` command. The `pwd` command returns the full path name of your current working directory.

```bash
#!/bin/sh

# Purpose - To provide a means of printing the current path.
# Method - A brief explanation of the `pwd` command is given.
# The shell then issues the `pwd` command.

if [ -z "$1" ]; then
  echo 'PATH IDENTIFICATION

The UNIX operating system uses a tree structure of directories.
It is possible to see the ancestors of your current directory by
using the "pwd" command. The "pwd" command returns the full path
name of your current working directory.

!!! pwd

." 
pwd
echo ';
echo -n 'Press return to continue
read z

exit 0
```
file line level source

xprintfile 1 0 # xprintfile
xprintfile 2 0 # xprintfile
xprintfile 3 0 #
xprintfile 4 0 # Purpose - To provide a means of printing a file at a printer.
xprintfile 5 0 #
xprintfile 6 0 # Method - The user must enter the name of the file he wishes
to have printed. If the user presses return when
xprintfile 7 0 # first asked the name of the file the program will
xprintfile 8 0 # exit. The user is given the option to have the file
xprintfile 9 0 # printed with line numbers.
xprintfile 10 0 #
xprintfile 11 0 #
xprintfile 12 0 # Error checking - If the user enters the name of a nonexistent
xprintfile 13 0 # file, the program tells the user that the file does
xprintfile 14 0 # exist and exits.
xprintfile 15 0 #
xprintfile 16 0 # question=?
xprintfile 17 0 #
xprintfile 18 0 #
xprintfile 19 0 # echo 'PRINT A FILE AT PRINTER'

while /bin/test "$x" = "$question

do
echo

case $1 in
  1) echo -n "What is the name of the file you wish to print? ";;
  2) echo -n "File name? ";;

esac
read x

if /bin/test "$x" then
  echo "You must enter the name of an existing file. This file will be printed at the printer assigned to your system."
fi

else
debug
fi
debug

if /bin/test -f "$x"
then
debug
fi
echo -n "Do you wish to have the line numbers printed (y or n)? "
read y

echo 'case $y in
   y|Y) echo "!!! cpr $x | lpr";
   echo "!!! YOUR FILE IS BEING PRINTED";
   echo "!!! cpr $x | lpr";
   echo "!!! YOUR FILE IS BEING PRINTED";;
   *) echo "!!! lpr $x";
   echo "!!! lpr $x";;
   esac

esac

echo "Press return to continue"
read z

else
   echo "File $x does not exist!"
sleep 2
fi

exit 0
<table>
<thead>
<tr>
<th>line</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td># receivesmail</td>
</tr>
<tr>
<td>2</td>
<td>#</td>
</tr>
<tr>
<td>3</td>
<td># Purpose - To provide a means for receiving mail.</td>
</tr>
<tr>
<td>4</td>
<td># Method - A brief explanation of the &quot;mail&quot; command is given. If the prompt level is verbose,</td>
</tr>
<tr>
<td>5</td>
<td># a subset of the available mail options is given. The shell then issues the &quot;mail&quot; command.</td>
</tr>
<tr>
<td>6</td>
<td>echo '</td>
</tr>
<tr>
<td>7</td>
<td>RECEIVE MAIL</td>
</tr>
<tr>
<td>8</td>
<td>The &quot;mail&quot; command displays any mail you have received. The</td>
</tr>
<tr>
<td>9</td>
<td>messages are displayed in chronological order with the most recent</td>
</tr>
<tr>
<td>10</td>
<td>one first. After each message you will see a question mark. If you</td>
</tr>
<tr>
<td>11</td>
<td>type &quot;/?&quot;, return, you will see the valid responses to your mail. You</td>
</tr>
<tr>
<td>12</td>
<td>must respond to each message.</td>
</tr>
<tr>
<td>13</td>
<td>sleep 3</td>
</tr>
<tr>
<td>14</td>
<td>if /bin/test $1 = 1 then</td>
</tr>
<tr>
<td>15</td>
<td>echo 'The following responses to mail are frequently used:'</td>
</tr>
<tr>
<td>16</td>
<td>echo q - quit reading your mail&quot;</td>
</tr>
<tr>
<td>17</td>
<td>echo w [file] - write the mail message to the file name you specify&quot;</td>
</tr>
<tr>
<td>18</td>
<td>echo a - look at the next mail message (does not delete)&quot;</td>
</tr>
<tr>
<td>19</td>
<td>echo d - delete current mail message&quot;</td>
</tr>
<tr>
<td>20</td>
<td>fi</td>
</tr>
<tr>
<td>21</td>
<td>sleep 7</td>
</tr>
<tr>
<td>22</td>
<td>echo &quot;</td>
</tr>
<tr>
<td>23</td>
<td>echo &quot;!!! mail&quot;</td>
</tr>
<tr>
<td>24</td>
<td>mail</td>
</tr>
<tr>
<td>25</td>
<td>echo</td>
</tr>
<tr>
<td>26</td>
<td>echo -n 'Press return to continue</td>
</tr>
<tr>
<td>27</td>
<td>read z</td>
</tr>
<tr>
<td>28</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
file line level source

xseefile 1 0 # xseefile
xseefile 2 0 #

Purpose - To provide a means for displaying a file with the non-printing characters in a readable format.

xseefile 4 0 #
Method - The user must enter the name of the file he wishes to have displayed. If the user presses return when first asked for the file name, the program will exit.

xseefile 8 0 #

Error checking - If the user enters the name of a nonexistent file, the program will tell the user that the file does not exist and exit.

xseefile 11 0 #
xseefile 12 0 #
xseefile 13 0 #
xseefile 14 0 #
xseefile 15 0 #

xseefile 16 0 question=?
xseefile 17 0 x=?
xseefile 18 0 echo '

DISPLAY NON-PRINTING CHARACTERS IN A FILE

Your file will be displayed with the non-printing characters in a readable format. The following formats are used:

- = tab
? = delete
\$ = end of line

xseefile 24 0

while /bin/test "$x" = "$question"
  do
    echo ' 
    case $1 in
      1) echo -n "What is the name of the file you wish to display? ";
      2) echo -n "File name? ";
      esac

xseefile 29 0
read x

xseefile 33 0
if /bin/test "$x"
  then
    if /bin/test "$x" = "$question"
      then
        echo ' 
        echo 'You must enter the name of existing file. If the file is not'
        echo 'in the current directory the entire path name must be entered.'
      fi
    else
      exit 0
  fi

xseefile 49 0

xseefile 50 0
```bash
file     line    level     source
xseefile  1   0       fl
xseefile  2   0       done
xseefile  3   0
xseefile  4   0       if /bin/test -f "$x"
xseefile  5   0       then
xseefile  6   0       echo '
xseefile  7   0
xseefile  8   0       echo "Hi! see -v -e -t $x"
xseefile  9   0       echo
xseefile 10  0
xseefile 11  0
xseefile 12  0
xseefile 13  0       see -v -e -t $x
xseefile 14  0
xseefile 15  0       echo '
xseefile 16  0
xseefile 17  0       echo -n "Press return to continue ."
xseefile 18  0       read z
xseefile 19  0       else
xseefile 20  0       echo ""
xseefile 21  0       echo "File $x does not exist!"
xseefile 22  0       sleep 2
xseefile 23  0       f1
xseefile 24  0
xseefile 25  0       exit 0
```
Purpose - To provide a means for sending mail to another user.

Method - The user must enter the login name of the user to whom he wishes to send mail. If the user presses return, the program will exit. The user has the option of sending a previously existing file in the mail or sending a messages he composes at the terminal.

Error checking - If the user wants to send a nonexistent file, he is notified that the file does not exist. The program then exits.

```
xsendmail
1 0
xsendmail
2 0 #xsendmail
xsendmail
3 0 #
  #
xsendmail
4 0 #
  #
xsendmail
5 0 #
  #
xsendmail
6 0 #
  #
xsendmail
7 0 #
  #
xsendmail
8 0 #
  #
xsendmail
9 0 #
  #
xsendmail
10 0 #
  #
xsendmail
11 0 #
  #
xsendmail
12 0 #
  #
xsendmail
13 0 #
  #
xsendmail
14 0 #
  #
xsendmail
15 0 #
  #
xsendmail
16 0 #
  #
xsendmail
17 0 question=?
  #
xsendmail
18 0 x=?
  #
xsendmail
19 0 y=?
  #
xsendmail
20 0
  #
xsendmail
21 0
  #
xsendmail
22 0
  #
xsendmail
23 0
  #
xsendmail
24 0
echo 'SEND MAIL'
  #
xsendmail
25 0 while /bin/test "x" = "$question"
  #
xsendmail
26 0 do
  #
xsendmail
27 0 echo " "
  #
xsendmail
28 0 case $1 in
  #
  # 1) echo -n "To whom do you wish to send mail? "
  # 2) echo -n "To whom? 
  #
xsendmail
30 0 esac
  #
xsendmail
31 0 read x
  #
xsendmail
32 0 if /bin/test "$x"
  #
xsendmail
33 0 then
  #
xsendmail
34 0 if /bin/test "$x" = "$question"
  #
xsendmail
35 0 then
  #
xsendmail
36 0 echo "You must enter the login name of the person to whom you wish to"
  #
xsendmail
37 0 echo "send mail."
  #
xsendmail
38 0 fi
  #
xsendmail
39 0 else
  #
xsendmail
40 0 exit 0
  #
xsendmail
41 0 done
```
file
line  level  source
xsendmail  51  0                        while /bin/test "$y" = "$question"
xsendmail  52  0                        do
xsendmail  53  0                        echo "
xsendmail  54  0                        case $1 in
xsendmail  55  0                        1) echo -n "Is the message you wish to send contained in a file (y or n)? ";
xsendmail  56  0                        2) echo -n "Send a file (y or n)? ";
xsendmail  57  0                        esac
xsendmail  58  0                        read y
xsendmail  59  0                        case $y in
xsendmail  60  0                        \? echo . "
xsendmail  61  0                        echo "You are able to send mail in two ways:";
xsendmail  62  0                        echo " 1) Create a file whose contents is the message you wish to send."
xsendmail  63  0                        echo " 2) Enter the message to be sent at the terminal after the mail";
xsendmail  64  0                        echo "command is initiated.";
xsendmail  65  0                        y(Yin[N]) ?"
xsendmail  66  0                        y) echo "
xsendmail  67  0                        echo " You must enter \y\ for yes and \n\ for no."
xsendmail  68  0                        esac
xsendmail  69  0                        done
xsendmail  70  0                        case $y in
xsendmail  71  0                        y(Y) echo ""
xsendmail  72  0                        echo -n "What is the name of the file you wish to send? ";
xsendmail  73  0                        read y;
xsendmail  74  0                        if /bin/test -f "$y"
xsendmail  75  0                        then
xsendmail  76  0                        echo "
xsendmail  77  0                        echo "!!! mail $x < $y"
xsendmail  78  0                        echo "
xsendmail  79  0                        mail $x < $y
xsendmail  80  0                        else
xsendmail  81  0                        echo "File $y does not exist!"
xsendmail  82  0                        sleep 2
xsendmail  83  0                        exit 0
xsendmail  84  0                        fi;;
xsendmail  85  0                        n[N) echo .
xsendmail  86  0                        echo "After you see the UNIX command enter your message one line at a time.
xsendmail  87  0                        echo "a time. When you have finished entering your message press the"
xsendmail  88  0                        echo "CONTROL and \1\ keys simultaneously."
xsendmail  89  0                        echo .
xsendmail  90  0                        echo "!!! mail $x;"
<table>
<thead>
<tr>
<th>file</th>
<th>line</th>
<th>level</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsendmail</td>
<td>101</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>102</td>
<td>0</td>
<td>&quot;</td>
</tr>
<tr>
<td>xsendmail</td>
<td>103</td>
<td>0</td>
<td>mail $x;;</td>
</tr>
<tr>
<td>xsendmail</td>
<td>104</td>
<td>0</td>
<td>esac</td>
</tr>
<tr>
<td>xsendmail</td>
<td>105</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>106</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>107</td>
<td>0</td>
<td>echo '</td>
</tr>
<tr>
<td>xsendmail</td>
<td>108</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>109</td>
<td>0</td>
<td>echo -n &quot;Press return to continue</td>
</tr>
<tr>
<td>xsendmail</td>
<td>110</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>111</td>
<td>0</td>
<td>read z</td>
</tr>
<tr>
<td>xsendmail</td>
<td>112</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>xsendmail</td>
<td>113</td>
<td>0</td>
<td>exit 0</td>
</tr>
</tbody>
</table>
file file line level source
xtmexit 1 0 # xtmexit
xtmexit 2 0 # Purpose - To provide a means of temporarily exiting the menu shell.
xtmexit 3 0 #
xtmexit 4 0 #
xtmexit 5 0 #
xtmexit 6 0 #
xtmexit 7 0 # Method - The program consists of a loop which echos a prompt to the user and waits for input from the keyboard. The user's entry is executed. The loop continues until the user enters a null response.
xtmexit 8 0 #
xtmexit 9 0 #
xtmexit 10 0 #
xtmexit 11 0 #
xtmexit 12 0 #
xtmexit 13 0 #
xtmexit 14 0 #
xtmexit 15 0 #
xtmexit 16 0 x*?
xtmexit 17 0 echo
xtmexit 18 0
xtmexit 19 0
xtmexit 20 0
xtmexit 21 0
xtmexit 22 0
xtmexit 23 0
xtmexit 24 0
xtmexit 25 0
xtmexit 26 0
xtmexit 27 0 while /bin/test "$x"
xtmexit 28 0
done
xtmexit 30 0 echo
xtmexit 31 0 echo -n "$  
xtmexit 32 0 read x
xtmexit 33 0 eval "$x"
xtmexit 35 0
done
xtmexit 36 0
echo
xtmexit 37 0
xtmexit 38 0 exit 0
# xterminal

Purpose - To introduce the user to special keys on his terminal.

Method - Xterminal displays files which describe the special keys available on the terminals used at RIT. The files are displayed using the `more` command. This shell script uses the `tset` command to identify the current terminal. A case statement is used to display the appropriate file. The case statement can be expanded to accommodate additional terminals.

## INTRODUCTION TO TERMINAL

There are keys on your terminal that have special meanings. If you use these keys your time spent working at your terminal will become more productive.

This menu option recognizes the keys for the most common terminals used in the Computer Science Department at Rochester Institute of Technology.
file   line level source
xwho  1  0
xwho  2  0  #xwho
xwho  3  0  # Purpose - To provide a means for displaying the current
xwho  4  0  # system users.
xwho  5  0  # Method - A brief explanation of the 'who' command is
xwho  6  0  # given.  The shell then issues the 'who' command.
xwho  7  0  #
xwho  8  0  #
xwho  9  0  #
xwho 10  0  echo
xwho 11  0
xwho 12  0
xwho 13  0
xwho 14  0
xwho 15  0
xwho 16  0
xwho 17  0
xwho 18  0
xwho 19  0
xwho 20  0
xwho 21  0
xwho 22  0
xwho 23  0  sleep $'
xwho 24  0  who
xwho 25  0
xwho 26  0  echo .
xwho 27  0
xwho 28  0  echo -n 'Press return to continue
xwho 29  0  read x
xwho 30  0
xwho 31  0  exit 0
```
file line level source

write 1 0
write 2 0 #write
write 3 0 #
write 4 0 # Purpose - To provide a means to write to another user.
write 5 0 # Method - If the level of prompt is verbose, an explanation
write 6 0 # of the write command is given. The explanation
write 7 0 # includes a description of the protocol to be
write 8 0 # followed. The user must enter the login name of the
write 9 0 # user to whom he wishes to write. If the user enters
write 10 0 # a return, the program will exit.
write 11 0
write 12 0
write 13 0 question?:
write 14 0 x?=
write 15 0 echo
write 16 0
echo

WRITE TO ANOTHER USER

if /bin/test $1 = 1
then
  echo : The "write" command is used to communicate with another user.
  echo : Currently logged on to the system. The messages sent with "write"
  echo : will be displayed at the receiver's terminal immediately. He may wish
  echo : to respond to you by writing back, thus establishing two way communi-
  echo : cation. For this reason the following protocol has been established:
  echo : 1) When you see the UNIX command enter your message followed by
  echo : "-o" which indicates you have temporarily finished
  echo : writing.
  echo : 2) Wait for the receiver to answer back. You will know he is
  echo : finished when you see his "-o". If you do not expect
  echo : an answer, skip to step 5.
  echo : 3) Continue writing back and forth in the above manner.
  echo : 4) End your final message with "-oo" to indicate your communi-
  echo : cation has completed.
  echo : 5) Press the CONTROL and "da" keys simultaneously to discontinue
  echo : the writing process.

fi

while /bin/test "$x" = "$question"
do
write 41 0
write 42 0
write 43 0
write 44 0 while /bin/test "$x" = "$question"
do
write 45 0
echo
write 46 0
write 47 0 case $1 in
write 48 0 1) echo -n 'What is the login name of the person to whom you wish to write?';
write 49 0 2) echo -n 'To whom?';
write 50 0 esac
```
file

read x
if `/bin/test 'x'
then
    if `/bin/test 'x' = 'question'
    then
        echo "You must enter the login name of someone currently using the system."
    else
        exit 0
    fi
else
    done
    exit 0
fi

Remember - "CONTROL L" discontinues the write process!

!!! write x

write x

write x

write x

write x

write x

write x

write x

write x

write x

write x

write x
To temporarily stop the screen press the "CONTROL" key and the "S" key simultaneously. When you are ready to restart the screen movement, press the "CONTROL" and "Q" keys simultaneously.

The "TAB" key will move the cursor 8 positions to the right each time it is pressed. If your terminal does not have the "TAB" key, you can achieve the same result by pressing the "CONTROL" and "I" keys simultaneously.

If you make a mistake when entering a line there are two options for making corrections. These options only apply to errors in the current line.
If your terminal has a backspace key, simply press that key until the cursor has backed up to the error position. If you do not have a backspace key, press the "CONTROL" and "H" keys simultaneously to back up the cursor. Once the cursor is in the correct position, type in the corrected line.

Sometimes a line has many errors and you feel it would be easier to retype the entire line. To have the computer ignore the current line press the "CONTROL" and "U" keys simultaneously.

There will be times when you want to stop the current program. The "BREAK" key or the "DELETE" key can be used to stop the current program. If you do not see the prompt (the signal that the computer is ready for your next command) you may have to press the "RETURN" key to have the prompt appear.
To temporarily stop the screen press the "NO SCROLL" key.
when you are ready to restart the screen movement press 'NO
SCROLL' again.

The "TAB" key will move the cursor 8 positions to the right
each time it is pressed. If your terminal does not have the
"TAB" key, you can achieve the same result by pressing the "CON-
TROL" and '1' keys simultaneously.

If you make a mistake when entering a line there are two
options for making corrections. These options only apply to
errors in the current line.
If your terminal has a backspace key, simply press that key until the cursor has backed up to the error position. If you do not have a backspace key, press the "CONTROL" and "U" keys simultaneously to back up the cursor. Once the cursor is in the correct position, type in the corrected line.

Sometimes a line has many errors and you feel it would be easier to retype the entire line. To have the computer ignore the current line press the "CONTROL" and "U" keys simultaneously.

There will be times when you want to stop the current program. The "BREAK" key or the "DELETE" key can be used to stop the current program. If you do not see the prompt (the signal that the computer is ready for your next command) you may have to press the "RETURN" key to have the prompt appear.
Welcome to the Menu shell. It has been designed to make the UNIX operating system friendlier to you the user. UNIX is a powerful operating system which you will appreciate as you become familiar with it. UNIX does have a limitation in regard to the new user. UNIX is a very terse operating system and offers little help to its users. Once you are familiar with UNIX commands the limitation of terseness becomes a desirable feature.

The purpose of the Menu shell is threefold:

1. Allows you to function in the UNIX environment successfully without knowing any UNIX commands.
2. Serves as a tool for learning UNIX commands.
3. Serves as a review to users who have not used UNIX for an extended period of time.

The Menu shell includes an option which introduces you to your terminal. It would be beneficial to choose this option if
you are unfamiliar with the terminal you are using. This option will identify special keys and their meanings.

Whenever you see three exclamation marks (!!!) on the screen it means that the command which follows is the UNIX command to perform the particular task you asked for. For example:

```
!! date
```

indicates to you that "date" is a valid UNIX command. If you entered "date" at your terminal it would respond with the current date and time.

The Menu shell provides for two levels of prompting you regarding the tasks you wish to perform. The default level of prompt is verbose and the most beneficial to use in the beginning. After you have spent some time in the Menu shell and are familiar with the various tasks it offers, you may find the second level of prompting to be best. You are able to switch back and forth from the verbose and terse prompts by choosing the appropriate menu option.
You will be asked questions which are needed to perform your desired tasks. Most of the questions are preceded by an "*". Any time an "*" appears it is a signal to you that you can obtain more information by responding with a "?" Once you have seen the additional explanation you will be asked the same question again. It would be especially beneficial for you to respond with a "?" to the "*" question the first time you have a task performed.

If you select a menu option and realize that you really do not want that task performed, the best way to exit that task is to simply enter a "return" to the first question you are asked.
As you become familiar with UNIX you will find the option to
temporarily exit the Menu shell extremely useful. This option
allows you to use any of the UNIX commands you know on your own. You
will be able to quickly perform tasks without the menu structure.
When you wish to return to the menus again simply press the
return key.

GOOD LUCK!!