IMPLEMENTATION OF AN OFFICE INFORMATION SYSTEM FOR THE DEPARTMENT OF COMPUTER SCIENCE

by

JOHN SCOTT ANDERSON

B.S., Kansas State University, 1979

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

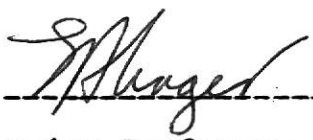
Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1983

Approved by:



Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

I would like to thank Mary Beth Cole for her input and patience in the Computer Science Department Office.

I would also like to thank Michael Terry for the design and organization of the "KOOL RECORDS SYSTEM".

I would also like to thank Carlos Qualls for his technical help and without whom this project couldn't have began, and thanks should also go to Harvard Townsend without whom this project couldn't have finished.

And most of all I would like to give special thanks to Dr. Elizabeth Unger, whom without her guidance and support the project could not have been started or completed.

CHAPTER ONE

INTRODUCTION

1.1 OBJECTIVE

The objective of this report is to describe the implementation of an office information system (OIS) for the Department of Computer Science. The intent of this system is to allow the department faculty and staff to monitor the progress of its applicants for graduate study and its graduate students. This office information system, termed the "KOOL RECORDS SYSTEM", consists of a menu-driven transaction processor interfacing with a database and report generator. The "KOOL RECORDS SYSTEM" is written in the C programming language and incorporates Unix operating system library software (2,4,5,7). The "KOOL RECORDS SYSTEM" will reside on the department's Perkin-Elmer 3220 minicomputer system and operate under the Unix operating system.

1.2 BACKGROUND

In the spring of 1983, Michael S. Terry completed a master's report detailing the design of an office information system consisting of a transaction processor coupled with a relational database and report generator (6). He used a functional approach to incorporate the already

existent routines of the Computer Science Department faculty and staff into the system that was intended to track the progress of graduate students and applicants.

An emphasis was placed on the system design to be as "user-friendly" as possible. This meant that the system must interface with the user in such a way as to be easily understandable, easy to use, and expeditious. For this to occur, Mr. Terry conducted extensive interviews with the appropriate personnel (the future system users) to determine the functions for the system to perform, and to develop an understanding of these functions. Once this was achieved, he then designed the system screens that were to be used for input and perusal of student information. (A screen can be defined simply as a form or format that appears to the user on a CRT terminal). These screens were to be used in a menu-driven fashion to interface with the relational database and with the report generator.

The relational database, as designed, consisted of two separate relational databases. One of these databases was to contain the information for each of the Computer Science Department's graduate students, while the other database contained the information for each of the department's graduate study applicants. These databases were designed to consist of normalized relations. Mr. Terry verified, with the use of Bernstein's Algorithm II, that these relations were in the third normal form. He also verified that the

database relations were in the Boyce-Codd normal form, as well as fourth and fifth normal forms.

It has been the intent of the author to implement the "KOOL RECORDS SYSTEM". Since Terry's methodology included working closely with the future system users, it is felt that his design adequately met with their requirements. Because of this, every attempt has been made to continue the communication with future users and to adhere strictly to the design during the implementation phase of the "KOOL RECORDS SYSTEM".

1.3 REPORT OUTLINE

This report will detail the entire implementation phase of the "KOOL RECORDS SYSTEM".

Chapter Two provides the components of the "KOOL RECORDS SYSTEM" and describes how these components are structured. This chapter includes a discussion of the system design and implementation. The data structures that are used are also discussed.

Chapter Three describes the "KOOL RECORDS SYSTEM" in a diagrammatic manner. Included is a brief introduction to a new software diagram methodology that is used to detail the interactions that occur between the user, the data, and the programs.

Chapter Four provides a critique of the implemented "KOOL RECORDS SYSTEM"
and suggests possible enhancements that could be added in the future.

In addition to the information contained in the above chapters, Appendix D contains a users manual that outlines the steps required of a user to perform the various functions that the "KOOL RECORDS SYSTEM" provides.

CHAPTER TWO

PROGRAM AND DATA STRUCTURES

2.1 OVERVIEW

The intent of this chapter is to describe, in detail, the architecture of the "KOOL RECORDS SYSTEM" along with the various data structures supported by this system. This description will include details of the implementation of this system. Numerous references will be made to the design phase of the "KOOL RECORDS SYSTEM" project. To aid the transition of the design phase to the implementation phase and to better illustrate the architecture and structures involved in both phases, Warnier/Orr diagrams will be employed throughout this chapter. The following section will briefly describe Warnier/Orr diagrams.

2.2 DESCRIPTION OF WARNIER/ORR DIAGRAMS

The Warnier diagram was first developed by Jean-Dominique Warnier and was discussed in "The Logical Construction of Programs" authored by Warnier (8). Kenneth Orr further modified the diagram to encompass both data flow and data structure-oriented design methodologies.

The Warnier/Orr diagram describes the logical organization of information in a hierarchical system.

Instead of the common vertical hierarchical block diagram, the Warnier/Orr diagram flows horizontally from left to right and involves the use of braces instead of boxes and arcs or lines. The diagram can be thought of as a mathematical set being divided into subsets (with the use of the brace) and each subset being further divided until the smallest minimal subset is reached.

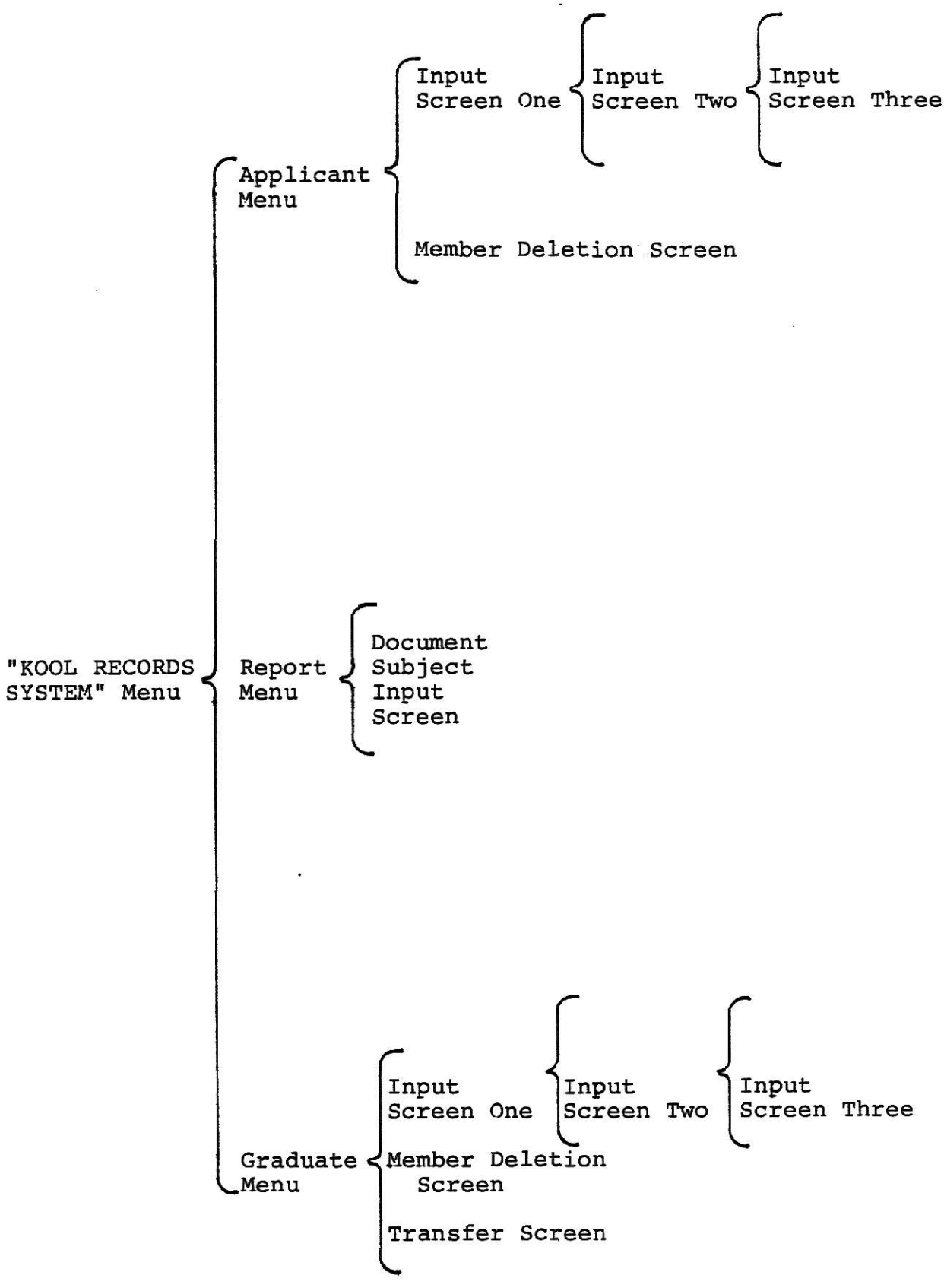
Warnier/Orr diagrams will be used to provide the overall structure of the "KOOL RECORDS SYSTEM" components.

2.3 TRANSACTION PROCESSOR STRUCTURE

The transaction processor, as designed by Michael Terry, includes a series of screens that are to allow information insertion to, deletion from, and perusal of the applicant and graduate databases. In addition, screen formats exist to allow the transfer of pertinent information from the applicant database to the graduate database as applicants are accepted into the graduate study program. Screens to allow the staff to print documents and reports as a function of the system's report generator are an additional part of the designed system.

The transaction processor's flow of control through these screens is illustrated in the Warnier/Orr diagram in Figure 2.1. This diagram depicts, in a hierarchical fashion, the order of these screens as the user navigates from a "main-

Figure 2.1


"KOOL RECORDS SYSTEM" Menu
Applicant Menu
Input Screen One
Input Screen Two
Input Screen Three
Member Deletion Screen
Report Menu
Document Subject Input Screen
Graduate Menu
Input Screen One
Input Screen Two
Input Screen Three
Member Deletion Screen
Transfer Screen

menu" to the "sub-menus" and finally to the input screens that act as the interface between the user and the applicant and graduate databases. Each of these input screens also allows information deletion, as well as editing and perusal. The member deletion screens were designed to allow complete removal of a student or applicant's record from the respective database. The transfer screen allows information to be copied from the applicant database to the graduate database. This diagram of the system screens also conveniently serves as a template for the structure of the various programs of the "KOOL RECORDS SYSTEM". This template served as an aid to the implementation phase of the project by demonstrating the modularization of the system's functions. The system's functional modules will be discussed further in later paragraphs.

The actual implementation of the transaction processor of the "KOOL RECORD SYSTEM" can be divided into eleven steps. These steps are as follows:

(1) generation of system screens

(2) development of routines to allow navigation through system screens

(3) development of routines to allow user input of information

(4) creation of temporary data structures to contain information entered by the user

(5) development of code to allow transfer of information from temporary data structures to databases

(6) development of code to allow retrieval of information from databases

(7) development of code to allow transfer of information from one database to the other

(8) development of record deletion function

(9) development of code for user verification for system security.

(10) development of faculty perusal function

(11) development of rejected applicant function

Each of these steps will be described in detail in the following sections including illustrations to demonstrate the structure of the transaction processor.

2.3.1 GENERATION OF "KOOL RECORDS SYSTEM" SCREENS

The first step of the implementation phase of the project was to devise a method to generate the designed screens on the CRT terminal. Various methods were attempted including placing the screen on the CRT, first in a "character by character" fashion, and then attempting a "string by string" approach. These methods proved to be too long and tedious, in the sense of the amount of code that was produced, and thus it was felt, these methods were unconstructive. Finally, a more expedient approach was discovered. Since these screens were already "written" on a Unix file during the design phase, they were already within the required eighty column, twenty-five line CRT terminal screen size. The screen generation technique simply involved reading and

writing the Unix file onto the CRT terminal screen. This procedure was delightfully effortless since it only required approximately five lines of code to generate each of the seventeen screens of the "KOOL RECORDS SYSTEM". The source code for this screen generation method can be found in Appendix A-1 under the program name "screen.c".

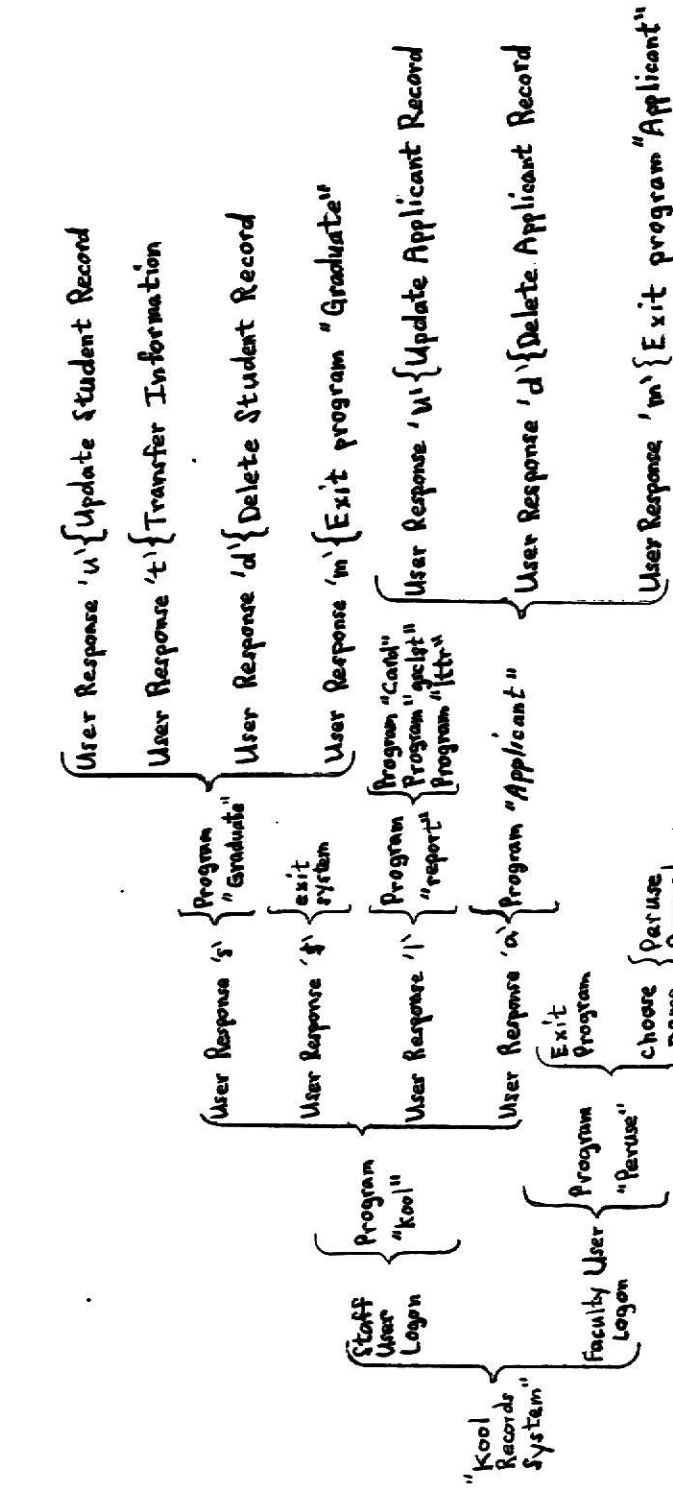
2.3.2 DEVELOPMENT OF SCREEN NAVIGATION ROUTINES

Once the code for the generation of system screens was implemented, routines were needed to allow the user to navigate from the "main menu" to each of the input screens and back again to the "main menu". Since this system was to be as "user-friendly" as possible, a minimum of user keystrokes was desired. The system screens, as designed by Mr. Terry, required only a single keystroke. Also, to promote user-friendliness, the appropriate key for the user to press was mnemonically designed. For example, a user response of 'm' will bring the user back to the menu or sub-menu of the current program (either graduate program, applicant program, transfer program, or deletion program). During the implementation phase, user-friendliness was also promoted by accepting either capitalized or uncapitalized letter commands as valid input.

Navigation through the system was actually implemented by allowing the user to initiate the desired program by simply pressing the appropriate key. If an incorrect key is pressed

by the user, a friendly error message appears stating that the user must make another attempt. The number of incorrect entries allowed is left to the user's discretion. The navigational flow through the system is illustrated in Figure 2.2. Tracing this flow, a log on to the system by the appropriate user initiates program "kool". From this point a user response of 's', for student, will initiate program "graduate", user response 'a' will initiate program "applicant", and user response 'l', for letter, will initiate program "report". Figure 2.2 then, depicts the required response of the user to navigate further into the system. Figure 2.2 can be compared to Figure 2.1 to correlate the screens associated with the various programs.

These user responses, as described in the above paragraph, are actually programmed as components of a control flow statement in the C language. This statement is called a switch statement in C and is analogous to the case statement in Pascal. Each user response is actually a "case" within this switch statement. Each case, then, either initiates a procedure within the current program, or allows a system call to allow execution of another separate program, as can be seen in Figure 2.2. These switch statements can be found throughout the "KOOL RECORDS SYSTEM" source code, conveniently and logically specifying the flow of control as dictated by the user.


"Kool Records System"
Staff User Logon
Program "kool"
Faculty User Logon
Program "Peruse"
User Response 's' Program "Graduate"
User Response 'x' exit system
User Response 'r' Program "report"
User Response 'a' Program "Applicant"
Exit Program
choose name
Peruse Record
User Response 'u' {Update Student Record
User Response 't' {Transfer Information
User Response 'd' {Delete Student Record
User Response 'm' {Exit program "Graduate"
Program "card"
Program "script"
Program "ttt"
User Response 'u' {Update Applicant Record
User Response 'd' {Delete Applicant Record
User Response 'm' {Exit program "Applicant"

2.3.3 DEVELOPMENT OF USER INPUT ROUTINES

The system allows a user to "flash" screens onto a CRT in a menu-driven fashion. The user can very simply follow the directions given on each screen to traverse to the next screen or to "log off" of the system. User input routines allow the user to input information on a student or applicant. The manner in which the screens are designed dictate that the user input of information is analogous to "filling in a paper form". Each screen contains labelled fields corresponding to the labelled fields on a paper form. When filling in a paper form, the user moves the pen or pencil to the first field and fills in the appropriate information. The user then moves the pen or pencil to the next field and fills in that field, etc. With the use of a C library screen package, the cursor can be directed to any point on the screen (1). This is accomplished using the following statement provided with the screen package:

```
mvcur(old_y,old_x,new_y,new_x);
```

where "old_y" and "old_x" are the coordinates of the old cursor position on the CRT screen ("where you are"), and "new_y" and "new_x" are the coordinates of the new cursor postion on the screen ("where you want to go"). In order for a C program to have access to this screen package the preprocessor command line: " include <curses.h>" must be

added to the program's declaration section. In the "KOOL RECORDS SYSTEM", cursor movement is triggered when the user presses the 'return' key. Once the user has directed the cursor to the desired field in a "top-to-bottom" fashion, information can be entered at that location. Simple editing procedures are performed by allowing the user to backspace to the mistake, correct it, and then retype the rest of the information. This can only be done, however, if the cursor is already located in the field containing the mistake.

2.3.4 TEMPORARY DATA STRUCTURES

Two main structures are used in the "KOOL RECORDS SYSTEM". One of these, found in Appendix A-12under the file name "appstrt.h", is termed "temp_rec". The other, "grad_rec", is found in Appendix A-13 under the file name "grstrt.h". These structures (a C language term) are analogous to records in Pascal. When the return key is pressed by the user to go from one field to another, the information just entered is copied into one of these temporary data structures, depending on which program is executing. These structures consist of fields (analogous to a record field in Pascal) that correspond to the field labels on the screen. For example, the screen field label "NAME" corresponds to the C language structure variable "temprec.name" in the applicant program, "app.c" found in Appendix A-3, or "gradrec.name" in the graduate program

"grad.c" found in Appendix A-4. Each field on the screen then has its corresponding structure field for information to be copied to as it is entered. These structure field names also correspond to the attribute names of the database relations in the "KOOL RECORDS SYSTEM" design.

2.3.5 INSERTION OF INFORMATION INTO THE DATABASES

Each time the user enters information into the temporary data structures through use of the input screens, this information must be filed into the appropriate database. This filing of information occurs as the user returns to the program's menu (either the applicant program or the graduate program). The filing procedure is accomplished with the use of C's "write" statement along with two other key statements, "open" and "lseek". The write statement, which is actually a system call, is a low level I/O Unix function. This call provides direct entry into the operating system. The write call has the following format:

```
n_written = write(fd,buf,n);
```

where "fd" is the file descriptor, "buf" is the data structure (either temprec or gradrec) and "n" is the number of bytes to be transferred. Each call returns the number of bytes transferred, which is "n_written" in the

above statement. The actual call format used in the "KOOL RECORDS SYSTEM" program "app.c" for example, is:

```
n_written = write(fd,&temprec,sizeof(struct temp_rec));
```

"&temprec" is the beginning address of the instance of the record structure temp_rec. The transfer of data will begin at this address and continue until the end of the structure is reached. The length if this structure is determined by the "sizeof()" function, which simply returns the size of whatever is inside the parentheses.

The "open" statement is also a system call found in C. This call opens the Unix file for either read access, write access, or read and write access. The type of access that is required at this point is write access. The format of the system call would then be as follows:

```
fd = open("database", 1);
```

where "fd" is the file descriptor, ""database"" is the name of the database, and "1" which only allows information to be written to the "database". If read access was required, a "0" would be used in place of "1". If both read and write access were required, the integer "2" would be used. Also, the file descriptor "fd" is the same "fd" found in the

"write" system call previously described.

The third major C statement used in the information transfer procedure is the "lseek" system call. "lseek" allows navigation within a Unix file without actually reading or writing. "lseek" can be used to pinpoint the location in the file of where reading or writing is to begin. The format of "lseek" is as follows:

```
lseek(fd, offset, origin);
```

The file descriptor "fd" is the same "fd" as returned by the "open" system call. The "lseek" call causes the current location within a file specified by "fd" to move to the location specified by "offset" beginning at a certain point "origin". This call is used in the "KOOL RECORDS SYSTEM" to find the specific point in a file to begin writing or reading information.

The following code segment, then, demonstrates the typical write procedure in the "KOOL RECORDS SYSTEM":

```
fd = open("database",1);
/* open database file for write only access */
lseek(fd,offset,0);
/* go to location offset starting at the beginning of the
file */
```

```
n_written = write(fd,&temprec,sizeof(struct temp_rec));
/* at location offset, write the temprec structure */
```

After the above routine is completed, the opened Unix file should be closed. This is accomplished using the following system call:

```
close(fd);
```

2.3.6 INFORMATION RETRIEVAL

Information retrieval from the databases is accomplished in much the same way as information "filing". The same system calls as described in section 2.3.5 are used except for the "write" call. This is replaced by the "read" system call. The typical program code segment to read information from the desired database is as follows:

```
fd = open("database",0);
lseek(fd, offset, 0);
n_read = read(fd,&temprec,sizeof(struct temp_rec));
```

The only difference here is that a zero is used in the open statement to provide read access only. And, of course, "read" and "n_read" replace "write" and "n_written",

respectively. Execution of this code, then, copies information from the Unix database file into the temporary record structure for subsequent display on the "KOOL RECORDS SYSTEM" screens.

2.3.7 DATABASE TO DATABASE INFORMATION TRANSFER

As an applicant is accepted into the graduate program, pertinent information from the applicant's files are needed. This information must be filed into the graduate files as this applicant becomes a graduate student. Because of this, a method was needed to transfer information from the applicant database to the graduate database. The following paragraphs describe this procedure.

The same Unix system calls described in section 2.3.6 are used for the information transfer procedure. To initiate these calls also requires user response. The basic steps that occur to transfer information from one database to the other are as follows:

(1) user input of names of new graduate students

(2) locate and read information on each entered name from the applicant database into a temporary applicant record structure.

(3) actual assignment of desired applicant information to a temporary graduate record structure

(4) locate correct position in graduate database and file the new graduate student's information.

This procedure is found in Appendix A-6 under the program name "trans.c". This procedure is fairly simple and can obviously save the user from reentering the same information twice, once into the applicant database and once into the graduate database, for example. This is just another example of the "user-friendliness" concept hard at work.

2.3.8 DATABASE RECORD DELETION

The record deletion function of the "KOOL RECORDS SYSTEM" allows a staff user to remove an entire record from either the graduate student database, the applicant database, or the rejected applicant database. The function is intended to be used when an applicant is either accepted for graduate study, or has been rejected. If acceptance is the case, the appropriate information should be transferred to the graduate student database prior to removal of the applicant's record. If the applicant has been rejected, the appropriate information is transferred to the rejected applicant database prior to deletion of the applicant from the applicant database. This function can also be used by the graduate management program as a student graduates, is withdrawn from the university, or is dismissed from the university.

The program that performs the deletion procedure is found in Appendix A-5 under the program name "assass.c". The program name is an abbreviation for assassin. This program

can be called from either the applicant management program or the graduate management program and will delete the desired records from the appropriate database depending on the parameters passed to it. This program was written with the intent of ensuring that the user is informed of exactly what he/she is doing. The screens were also designed so that the user is notified of the seriousness of the function he/she is about to perform. Also, if the user has accidentally called the deletion program, several "escape points" are available to allow return to the applicant or graduate menu. In addition, a record can only be deleted if the user has entered the correct name in the correct format.

The screen that allows the user to delete records consists of ten name fields. The user, then, can delete up to ten records during each screen session. The user can also ask for a second screen to delete ten more records if desired, and so on. To actually delete a record, the deletion program simply flags the record indicating that it can be written over when the user next "files" new records. This provides another safety This procedure involves filling in the name field of the record with pluses (ie., '+'). This provides another safety precaution in that the information is still in the database as long as no new records are added. In this way, the garbage collection function does not invalidate the records, but simply allows them to be written over.

2.3.9 SECURITY ENHANCEMENT THROUGH USER VERIFICATION

Security is and will always cause difficulties for "owners" of database information. A minimum security mechanism has been programmed into the "KOOL RECORDS SYSTEM".

For the Computer Science Department's staff, a prearranged password is required before access to the system's programs are allowed. These passwords must be written into the actual source code so that modification of them is somewhat difficult. The verification function allows a user two attempts at entering their password. If the correct password is not entered during these attempts, the program returns to Unix, immediately disallowing further program execution. The illegal user can, however, simply reenter the "KOOL RECORDS SYSTEM" and attempt to enter the correct password again. Other illegal forms of entry are also possible, but it has been assured by faculty and staff that maximum security of this system is not necessarily required at the time of this writing. The source code that contains this verification mechanism is shown in Appendix A-2 under the program name "super.c".

2.3.10 FACULTY PERUSAL OF STUDENT INFORMATION

The ability of the faculty to peruse a graduate student's record is perhaps one of the most important features of the

"KOOL RECORDS SYSTEM". Since the faculty members have access to CRT terminal in their own offices, student advising can be greatly enhanced by having the opportunity to immediately peruse the student's record during their meeting.

The perusal program, shown in Appendix A-7 under the program name "peruse", is similar to the graduate management program in that they both have access to the graduate student database and both have similar screen formats. The major difference is that the perusal program (as the name implies) only provides read access to the database. If any changes in the information content are desired by a faculty member, the Department's staff must be notified. This will allow data integrity to be better maintained, since only a single staff user will have write access to the graduate student database.

A second difference between the faculty perusal program and the staff's program is the manner in which a student record is "pulled" for use. The faculty perusal program presents a name list to the faculty member to choose the name of the student whose record is desired for perusal. With this added feature, the faculty user does not have to remember the exact, full name of the student. This alphabetized list is presented on as many screens as are required. Alphabetization of these names is done by implementation of a linked list data structure, as can be

seen in the source code in Appendix A-7.

2.3.11 THE REJECTED APPLICANT FUNCTION

After reevaluation of the "KOOL RECORDS SYSTEM" by the Computer Science Department faculty and staff, it was decided that an additional function was needed. This function was required so that the staff could monitor the applicants that were denied entry into the Graduate School. The development of this function required the addition of a third database that was to contain the rejected applicant's name along with the reason for rejection.

The rejected applicant database can actually be thouqht of as a sub-database of the applicant database. The entry of a name into the reject database is done after an 'x' is placed in the rejection field on the third applicant input screen. The name along with the reason for rejection is then filed into the reject database. The complete applicant's record is also still filed in the applicant database, however, and this record must be deleted from the applicant database by the user. Once deletion is completed, the user may enter the rejected applicant's name when prompted. The applicant program then searches the applicant database for the record. When the desired name is not found, the applicant program then searches the reject database. If the name is still not found, a message is displayed stating this.

Rejected applicant names may also be deleted from the reject database by entering the deletion program and following the directions provided on the deletion screens.

2.4 DATABASE STRUCTURE

The design of the "KOOL RECORDS SYSTEM" included relational databases consisting of fifth normal form relations. It was the original intent that these databases were to be implemented using the "Ingres" database system. The Computer Science Department has not been able to obtain this system at the time of this writing, however. Because of this, the author decided to develop the graduate and applicant databases in a manner which would allow ease of implementation with the intent of providing means by which a relational database could be added at a future date.

The actual implementation of the "KOOL RECORDS SYSTEM" databases is done with the use of a large "array" written into a Unix file. These "arrays" consist of records which are all exactly the same size, and exist in a contiquous fashion, with each record representing a single student's or applicant's record. There is no order by which these records are stored. The next new record is simply filed in the first "flagged" location (indicating a deleted record) or at the end of the file if no such locations currently exist. A record is found by using a linear search procedure. Since the probable size of the databases will not be large, the

search time should not prove to be a major factor in the system's efficiency. This type of database structure, it is felt, will allow simple and expedient transfer of data if a relational database is installed in the future.

Retrieval, addition, and modification of database information has been previously discussed in sections 2.3.5 and 2.3.6.

2.5 REPORT GENERATOR STRUCTURE

The two types of reports that currently exist are listings and letters. To generate a listing, the "KOOL RECORDS SYSTEM" pulls the desired information on each student or applicant into a linked-list structure that is alphabetically ordered. The contents of this linked-list are then sent to a Unix file to be printed out. Since these lists may become rather lengthy, they will be automatically printed at a high-speed line printer rather than allowing the user a choice of where to print out the list.

Each of the letters that can be printed using the "KOOL RECORDS SYSTEM" exist in a Unix file and are formatted using "nroff" (2). Nroff is a Unix tool that allows text formatting for typewriter quality output. A sample letter incorporating nroff commands is demonstrated in Figure 2.3.

The variables that are present in each letter, such as address and name, must be obtained from the appropriate database and placed into Unix files. These files are then

```
Oct 20 20:10 1983  admitnon.nr Page 1


.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
Welcome to K-State.  The Department Graduate Committee has recommended to
the Graduate School that your application be accepted.  You will
receive official notice from the Graduate School.
.sp
Enrollment for
.so /usr/scott/lttrinfo/semester
will be held
.so /usr/scott/lttrinfo/enrdates
and classes begin
.so /usr/scott/lttrinfo/classdate
\&. There will be general advising sessions for new graduate
students scheduled during the enrollment period.
.sp
We are not able to offer you support at this time.  However, it is
likely that you will be able to find support after having gained
experience in the Graduate Program.
.sp
We are looking forward to having you in the Department.
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger
Grad. Studies Committee
.sp
EAU:jsa
```

read using the nroff command ".so filename". This allows the name and address to be inserted into the letter at the appropriate locations. The other letter variables that are not obtained from the student or applicant records, such as the current date, are obtained from the user via user prompts. The actual source code for the report generator may be found in Appendix A-8 under the program name "report.c". The instructions that are to be followed for the report generator are found in the user's manual in Appendix D.

CHAPTER 3

LOGICAL DESIGN OF THE "KOOL RECORDS SYSTEM"

3.1 OVERVIEW

Although the Warnier/Orr diagram shown in Figure 2.2 adequately depicts the overall program structure of the "KOOL RECORDS SYSTEM", it was desired by the author to also demonstrate the flow of control as well as the flow of information. Various formal modelling techniques for software engineering have been described in the past, but each technique only demonstrates a particular aspect of a program system.

One of the modelling techniques that demonstrates most of the characteristics desired by the author, is the Information Control Net diagram (ICN) (3). This technique depicts the procedural flow of a system as well as the flow of information. This technique was successfully used by Michael Terry during the design phase of the "KOOL RECORDS SYSTEM" to present the various procedures that occur in the Computer Science Department office. These diagrams were used to also demonstrate which of these procedures could be automated.

Although the ICN diagram has proved to be an adequate modelling technique, the author required a different type of

diagram to adequately demonstrate procedural flow, information flow, and user control of the various "KOOL RECORDS SYSTEM" programs.

3.2 DESCRIPTION OF THE "KOOL DIAGRAM"

In order to fully demonstrate the "KOOL RECORDS SYSTEM" in an algorithmic and diagrammatic fashion, a new software engineering modelling technique has been developed and will be introduced in this report. This technique, henceforth termed the "KOOL Diagram", actually combines several modelling techniques into one, but more closely resembles the ICN diagram. As in the ICN, procedural flow is represented by a set of circles that are labelled with a program or sub-routine name, connected by solid, directed lines or arrows. Also, temporary data structures are represented by large inverted triangles and are labelled with the structure name. Data flow to and from these structures is depicted by a broken line. These broken lines are also used to depict the flow of data to and from permanent data structures, which are represented by squares labelled with the data structure name. Since the "KOOL RECORDS SYSTEM" is highly dependent upon menus and CRT terminal screens, a construct was needed to show the interaction with these screens, the user, and program data structures. These screens are represented in the "KOOL

Diagram" by rectangles labelled with the screen names. A broken line is used to represent how and when these screens are displayed.

To demonstrate the order of procedural flow in the "KOOL Diagram", a small numbered circle is used to label a flow line. This allows a sequential order of precedence to be shown if a single program or sub-routine calls several other programs or sub-routines. The line labelled with the smallest number, then, is the path that is followed first. A bidirectional line indicates, for example, that control of execution resumes in a calling program or sub-routine after the called program or subroutine has finished execution. These control lines should be considered as extensions of the calling program or sub-routine. Execution of the routine being called does not actually begin until a directed line meets the circled routine.

Also represented in the "KOOL Diagram" are separate variables depicted by a small triangle. These triangles can be labelled either with a variable name, such as V1, or with a user command, indicated by single quotes. For example, 'm' indicates that the user has entered an "m". If a variable is of type Boolean, the lines flowing from the variable are labelled with a small circle containing either a "T" or an "F". The "T" indicates the true path that is followed if the Boolean is true, while the "F" indicates the false path to follow if the Boolean is false. These variables or user

responses can be thought of as "gates". In other words, the flow of control through these gates is dependent upon the value of the variable.

Another "KOOL Diagram" component that should be discussed is the control dot or point. This is represented by a solid circle. The control dot can be used to depict control or data flow branching or joining. Lines flowing from the dot can also be numbered to indicate control flow order.

A summary of the various components of the "KOOL Diagram" are shown in Table 3.1.

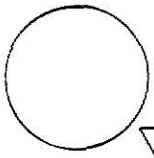
Since program systems can be very large, several diagrams may be required for a complete description of execution events. In order to keep these diagrams adequately organized, each diagram should be properly labelled. For example, if a single diagram is to be illustrated on several separate pages, they could be labelled as A.1 and A.2. Different programs in the same system can be lettered in ascending order, such as A.1, B.1, C.1, etc.

3.3 REPRESENTATION OF THE "KOOL RECORDS SYSTEM" USING THE "KOOL DIAGRAM"

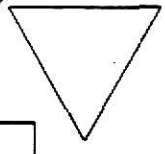
Each program of the "KOOL RECORDS SYSTEM" has been diagrammed using the "KOOL Diagram" modelling technique

Table 3.1

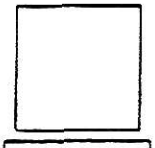
large circle: program or subroutine----------



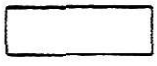
inverted triangle: temporary data structure----------



square: permanent data structure----------



rectangle: CRT terminal screen or output document----



small circle, numbered: order of flow----------

small triangle, letter only: variable----------

small triangle, letter and quotemarks: user response-

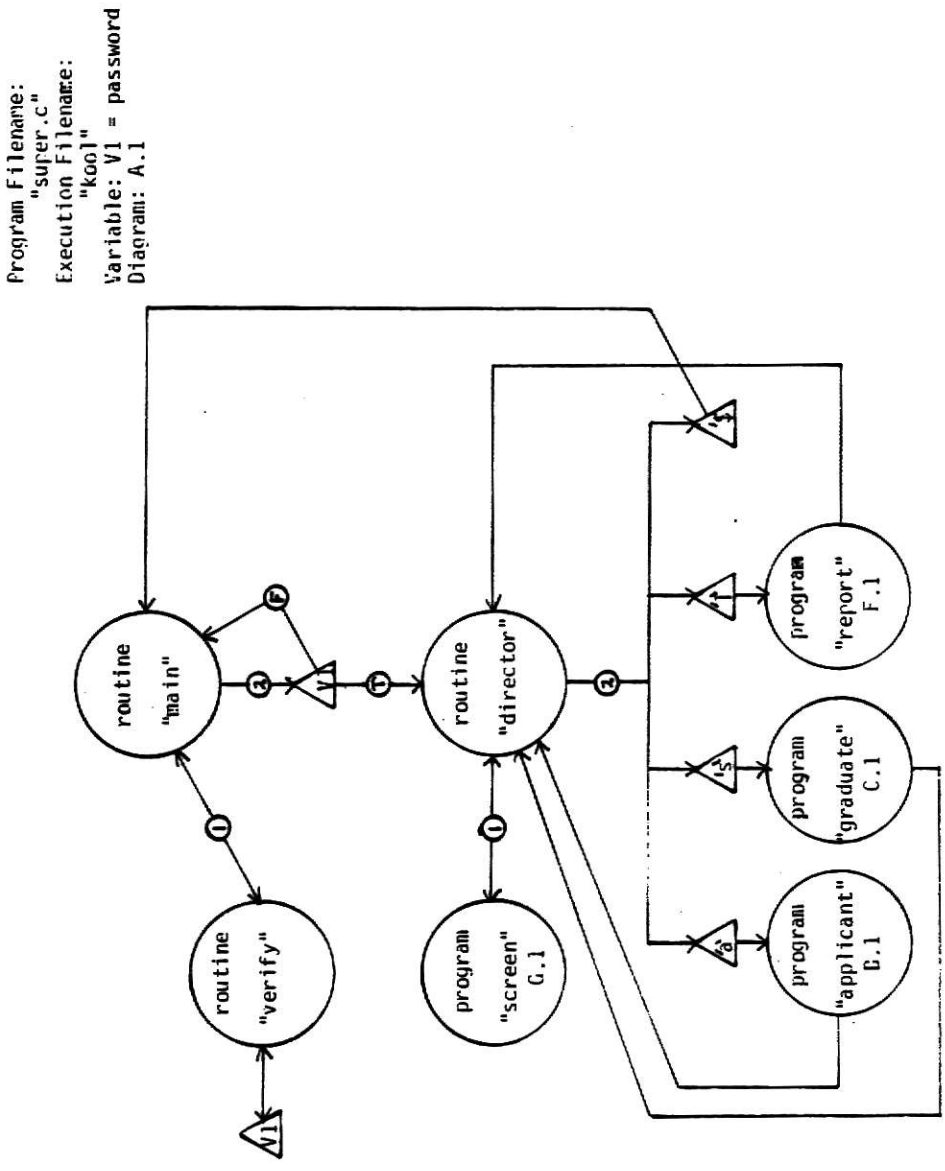
control dot: control branch or join----------

solid line: control flow----------

broken line: data flow----------

previously described. These diagrams are labelled with program file name and execution file name. These diagrams are shown in Figures 3.1 through 3.11 and are labelled from Diagram A.1 to H.1. Each diagram or set of diagrams representing a "KOOL RECORDS SYSTEM" program begins at the circle labelled: "routine main". This is where execution of the program begins. The programs that are modelled by these diagrams are contained in Appendix A.

Figure 3.1 illustrates the execution flow of the first program of the "KOOL RECORDS SYSTEM". The first routine that is called after program initiation is the routine "verify". This routine requires the user to enter the appropriate password, called V1 in the diagram. If an incorrect password is entered, control returns to the "main" routine and program execution terminates. This password variable, then, acts as a true-false gate. If the password is incorrect or false, execution stops. If the password is correct or true, execution continues. If program execution continues, the routine "director" is called. The first thing that "director" does is call the program "screen" to generate a CRT terminal screen, which in this case is the "KOOL RECORDS SYSTEM" main menu. This menu can be seen in Appendix B-1. Once this menu is displayed, program execution can continue in one of four possible directions, depending on the user response. For example, Diagram A.1 demonstrates that a user can enter an 'a' for the applicant management program, an 's' for the graduate student management program, an 'l' for


Program Filename:
"super.c"
Execution Filename:
"kool"
Variable: V1 = password
Diagram: A.1
routine
"main"
routine
"verify"
routine
"director"
program
"screen"
C.1
program
"applicant"
B.1
program
"graduate"
C.1
program
"report"
F.1
V1

the letter and report program, or a '$' which results in program termination. The programs that can be chosen to execute next, are represented by the circles labelled: "program applicant", "program graduate", and "program report". Also within the circles are the diagram letters that give the location of the diagram that models that particular program.

As can be seen in Diagram A.1, execution control returns to routine "director" in the program "super.c" when the chosen program has terminated. At this point, the program "screen" is called again to display the main menu. A user response is then required for further program execution, as before.

The second diagram of the "KOOL RECORDS SYSTEM", B.1, is shown in Figure 3.2. This diagram is fairly straightforward and execution occurs in the same manner as that in diagram A.1. This diagram illustrates the generation of the applicant management program menu (shown in Appendix B-2) and the routines involved with accepting an applicant's name and searching for this applicant's file. The routine "readbase" or "rejbase" (if the applicant has been rejected) is responsible for finding the name given and "pulling" the correct record. If the record is found it copies it into the temporary record structure "temprec".

The program "app.c" becomes more complicated in Diagram B.2, which is a continuation of Diagram B.1 and is shown in

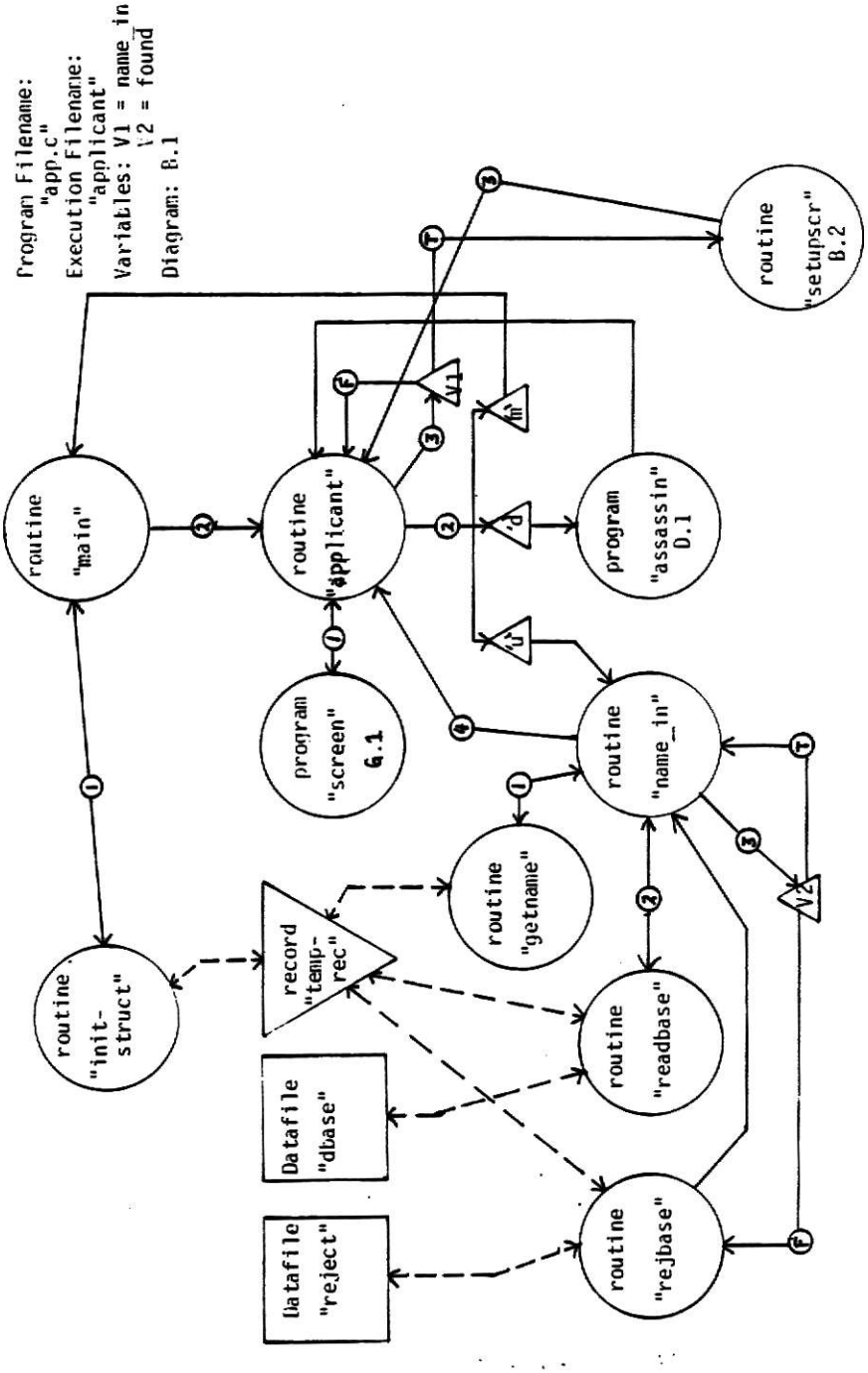
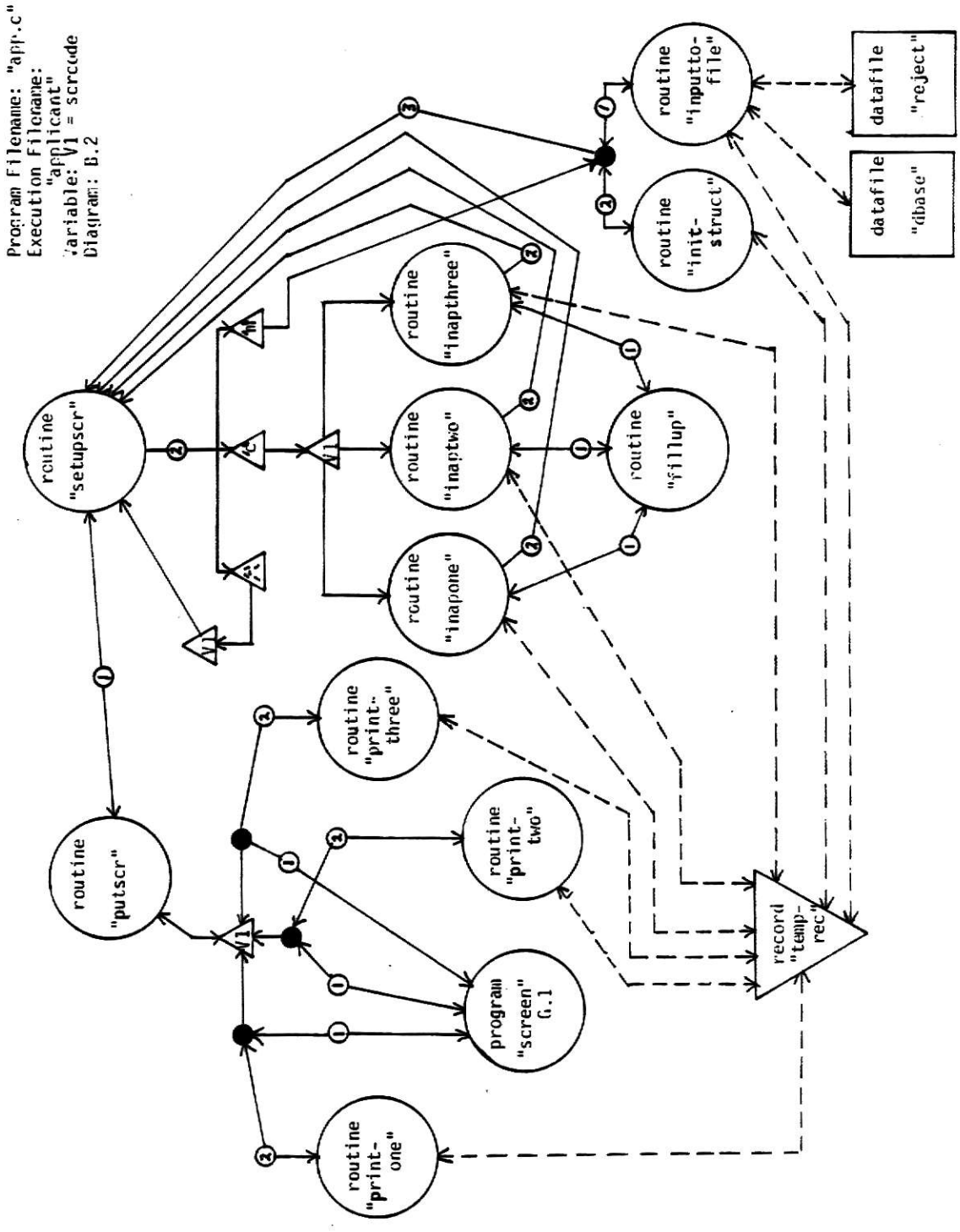

Program Filename:
"app.c"
Execution Filename:
"applicant"
Variables: V1 = name_in
V2 = found
Diagram: B.1
routine
"main"
routine
"applicant"
routine
"setupscr"
B.2
program
"assassin"
D.1
program
"screen"
G.1
routine
"name_in"
routine
"getname"
routine
"init-
struct"
record
"temp-
rec"
routine
"readbase"
Datafile
"dbase"
routine
"rejbase"
Datafile
"reject"

Figure 3.3. Diagram B.2 begins with routine "setupscr". This routine first calls the routine "putscr" which tells the program "screen" to display an applicant input screen. The screen that is displayed is dependent on the variable "scrcode" which is V1 in the diagram. This variable is initialized at the start of execution so that the first input screen "sone" (shown in Appendix B-3) is displayed. After initial display of "sone", "scrcode" may be modified by the user to display the second input screen, "stwo" (shown in Appendix B-4), by pressing the space bar. The third and last applicant input screen, "sthree" (shown in Appendix B-5), may then be reached from "stwo" by pressing the space bar, or from "sone" by entering a minus sign, "-". This part of the program, then, allows the user to reach any of the three input screens by pressing the space bar or entering a minus sign. Also, as can be seen by Diagram B.2, after an input screen has been displayed, the corresponding applicant information is displayed in the screen fields by a call to one of the three routines, "printone" for "sone", "printtwo" for "stwo", or "printthree" for "sthree", again depending on the variable "scrcode". These routines simply print the information present in the temporary record structure, "temprec" onto the screen.

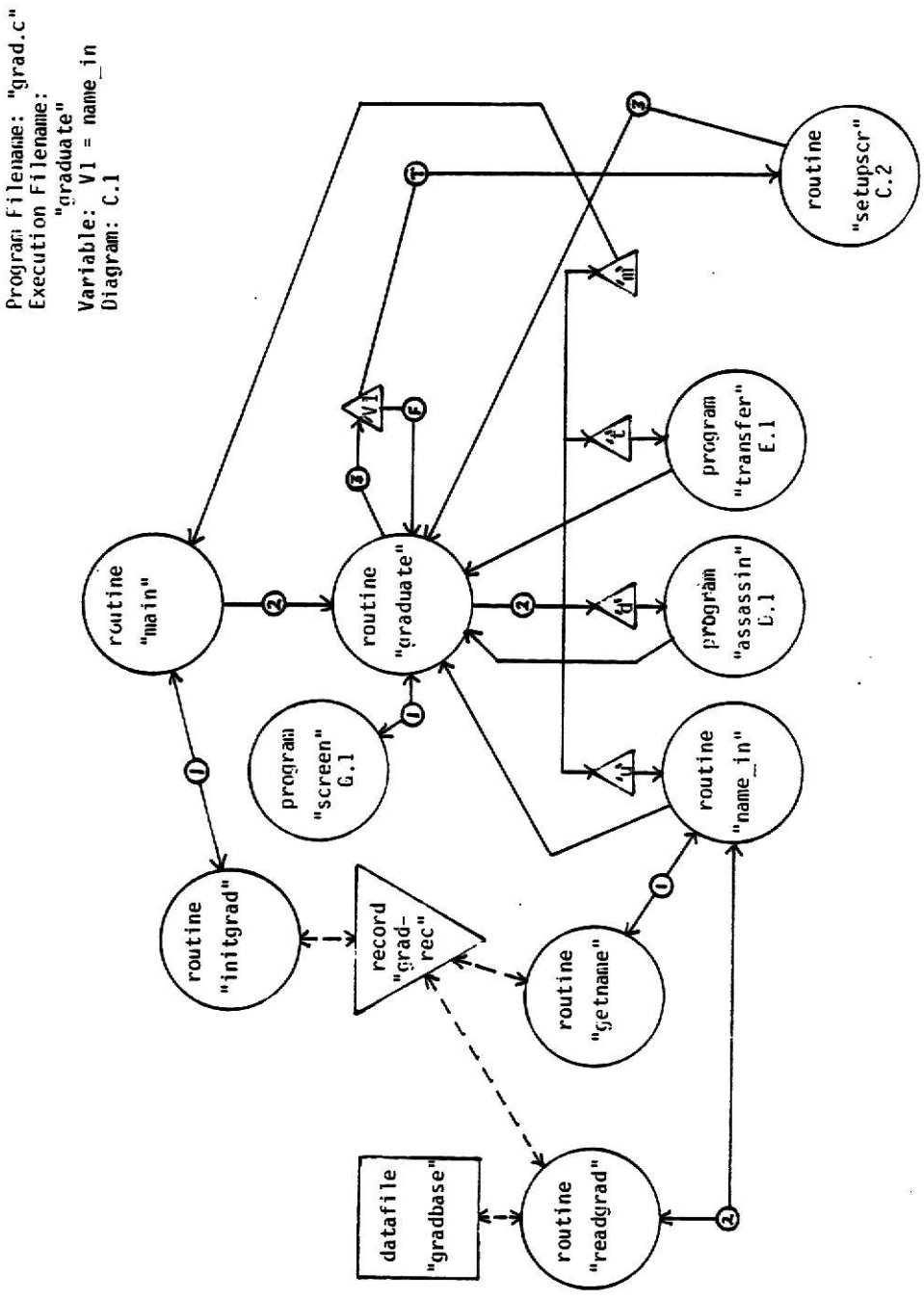
Once a screen and its corresponding information has been displayed for a particular applicant, the user may decide to add to or modify information by entering a 'c'. When this action occurs, one of three information input routines are

Figure 3.3


Program Filename: "appl.c"
Execution Filename:
"applicant"
Variable: V1 = screode
Diagram: B.2
routine "setupscr"
routine "putscr"
routine "print-one"
routine "print-two"
routine "print-three"
routine "inapone"
routine "inaptwo"
routine "inapthree"
routine "fillup"
routine "inputto-file"
routine "init-struct"
program "screen" 6.1
record "temp-rec"
datafile "reject"
datafile "dbase"
V1
V2
V3
V4

called, depending once again on the variable, V1, or "scrcode". These routines also have access to the temporary record structure, "temprec". The addition or modification of information is accomplished by the routine "fillup". When the user has navigated down to the command line that is present at the bottom of the input screens, execution control returns to the routine "setupscr", where further execution again depends on user response. At this point a user response of 'm' will first allow the routine "inputtofile" to copy the temporary record structure, "temprec" into the permanent datafile, "dbase". If the applicant has been rejected, the applicant's name and reason for rejection is copied to the "reject" datafile rather than "dbase". After one of these events occurs, control returns, first to "setupscr" and then to routine "applicant" in Diagram B.1.

The remaining diagrams demonstrate the flow of execution and the required user responses in the same manner as has been described. The various screens that can be generated by the "KOOL RECORDS SYSTEM" are shown in Appendices B-1 through B-22. The mechanism by which these screens are generated is shown in Diagram G.1 of Figure 3.9. In addition, the system's report generator is diagrammed in Figure 3.8. The letters generated by this program are illustrated in Appendices C-1 through C-8.


Program Filename: "grad.c"
Execution Filename:
"graduate"
Variable: V1 = name_in
Diagram: C.1
routine "main"
routine "graduate"
routine "setupscr" C.2
program "transfer" E.1
program "assassin" E.1
program "screen" G.1
routine "name_in"
routine "initgrad"
record "grad-rec"
routine "getname"
datafile "gradbase"
routine "readgrad"

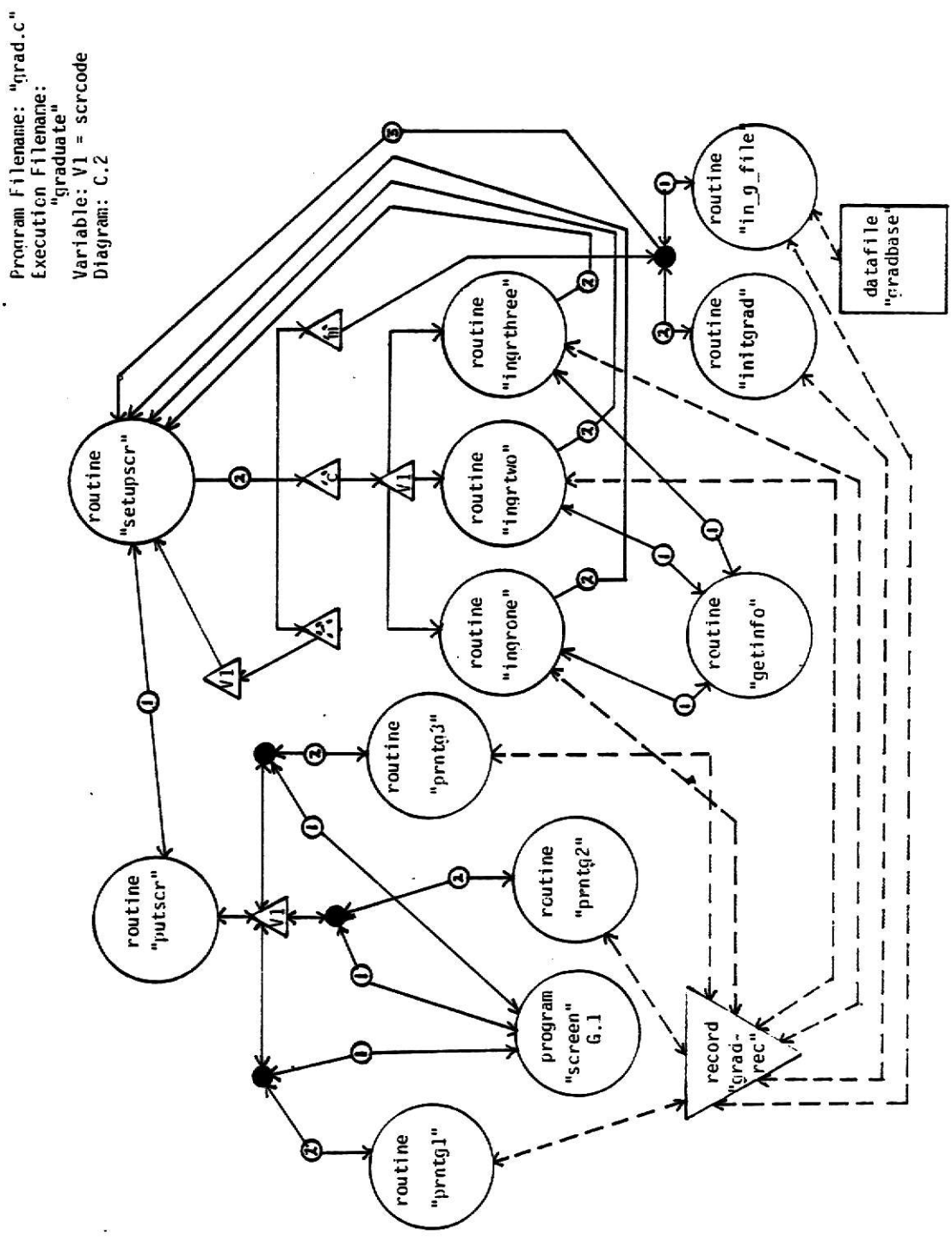
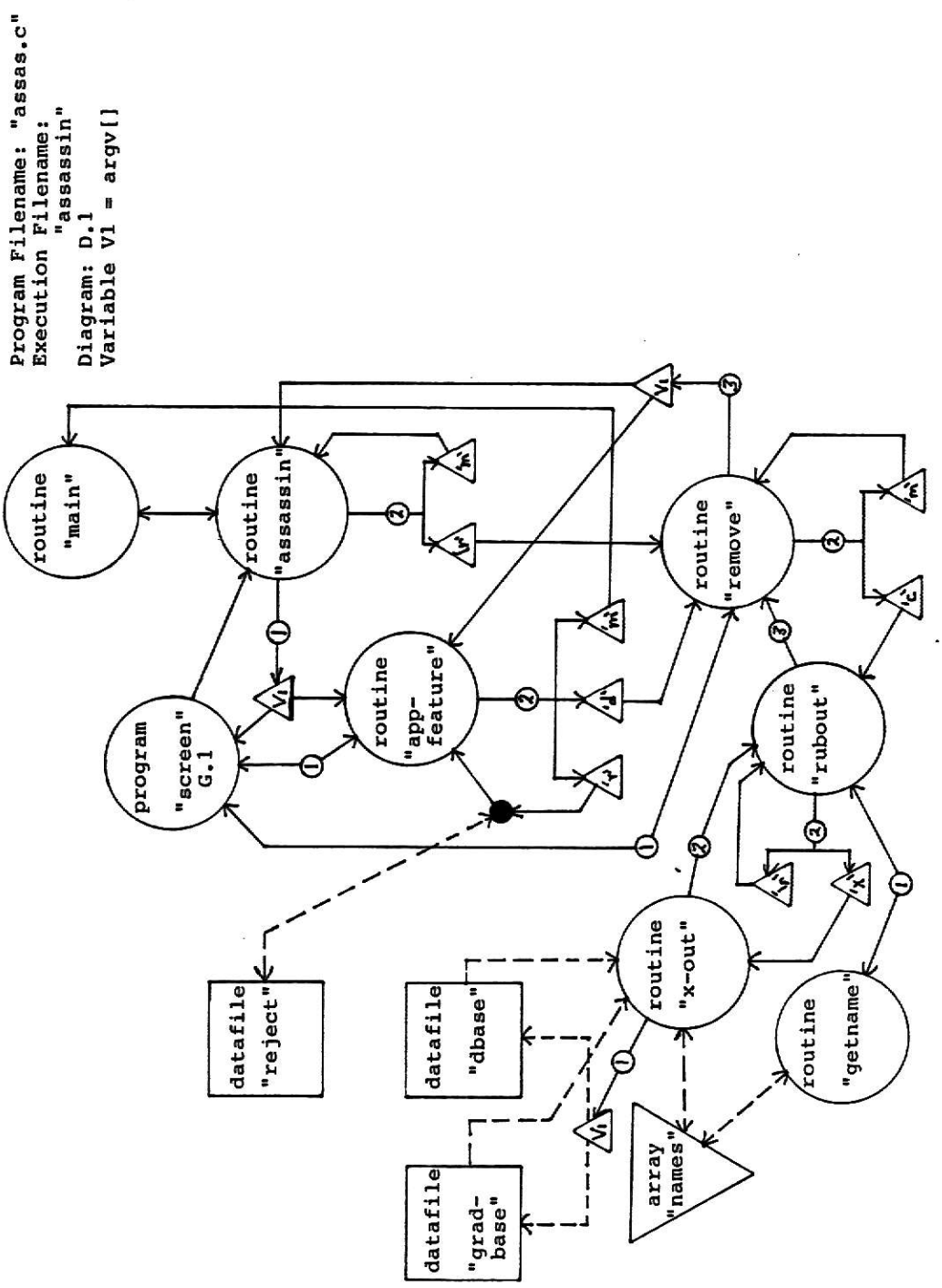

Program Filename: "grad.c"
Execution Filename:
"graduate"
Variable: V1 = scrcode
Diagram: C.2
routine "setupscr"
routine "putscr"
routine "prntg1"
routine "prntg2"
routine "prntg3"
routine "ingrone"
routine "ingrtwo"
routine "ingrthree"
routine "getinfo"
routine "initgrad"
routine "in_g_file"
program "screen" G.1
record "grad-rec"
datafile "gradbase"
V1
V1
V1
3
2
1

Figure 3.6


Program Filename: "assas.c"
Execution Filename:
"assassin"
Diagram: D.1
Variable V1 = argv[]
routine "main"
routine "assassin"
routine "remove"
routine "app-feature"
routine "rubout"
routine "x-out"
routine "getname"
program "screen" G.1
datafile "reject"
datafile "dbase"
datafile "grad-base"
array "names"

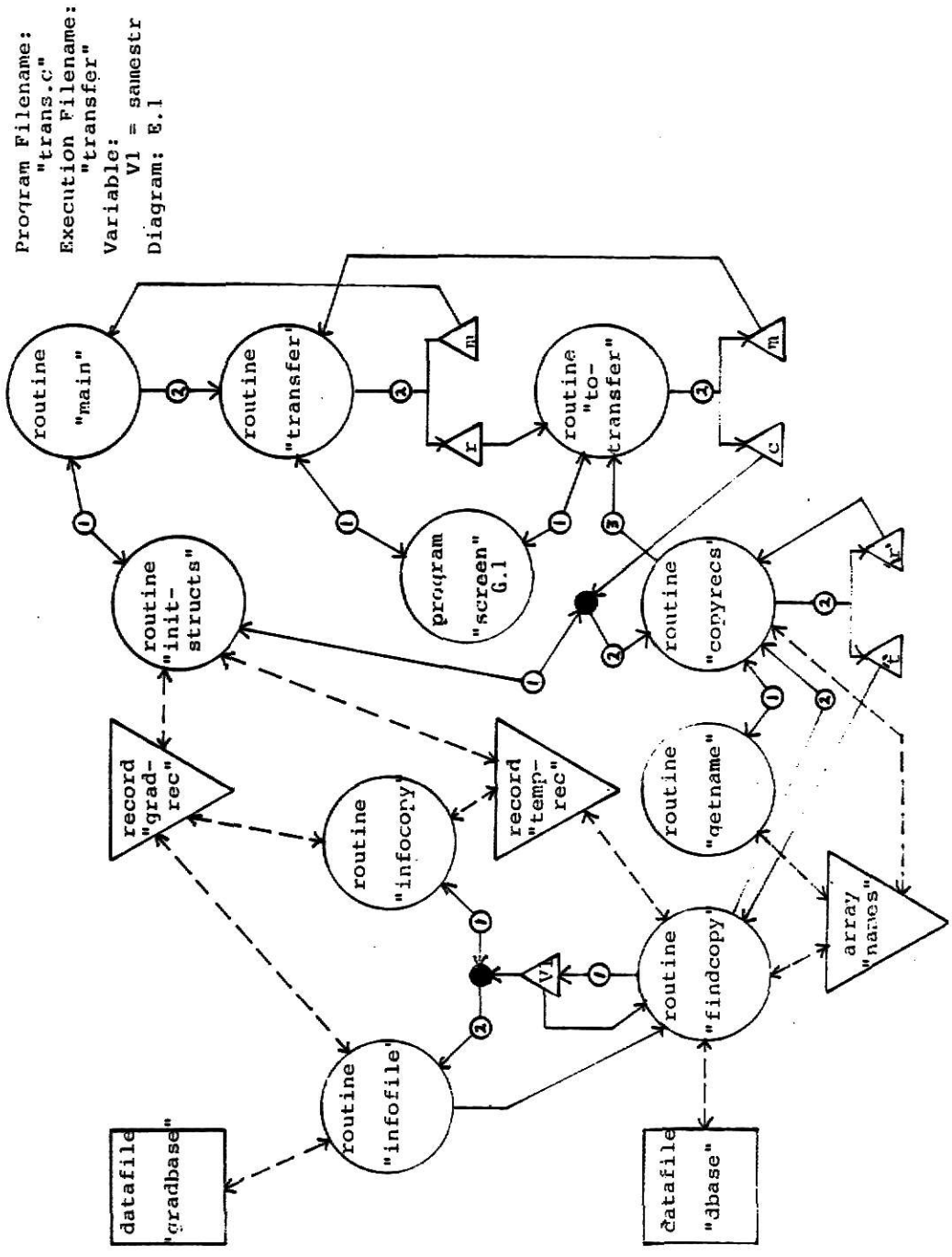
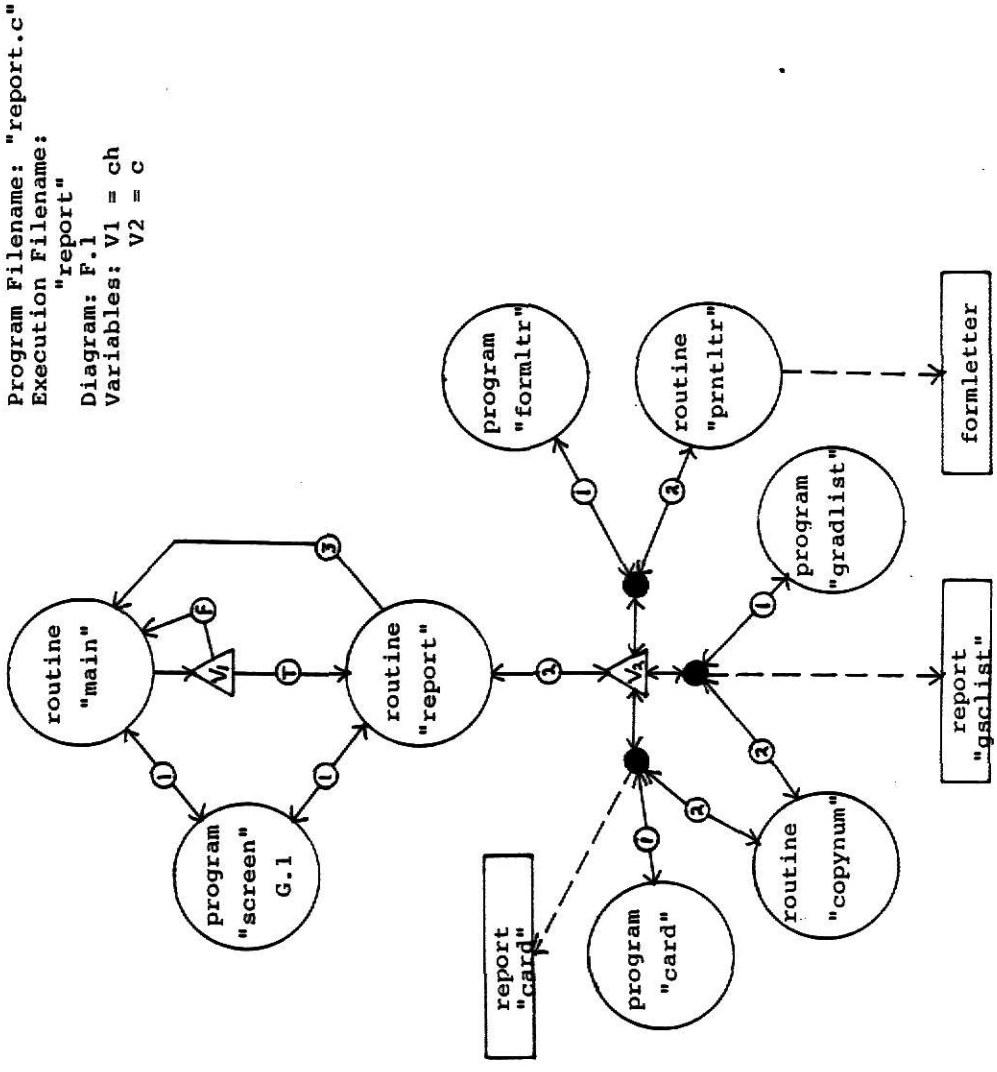
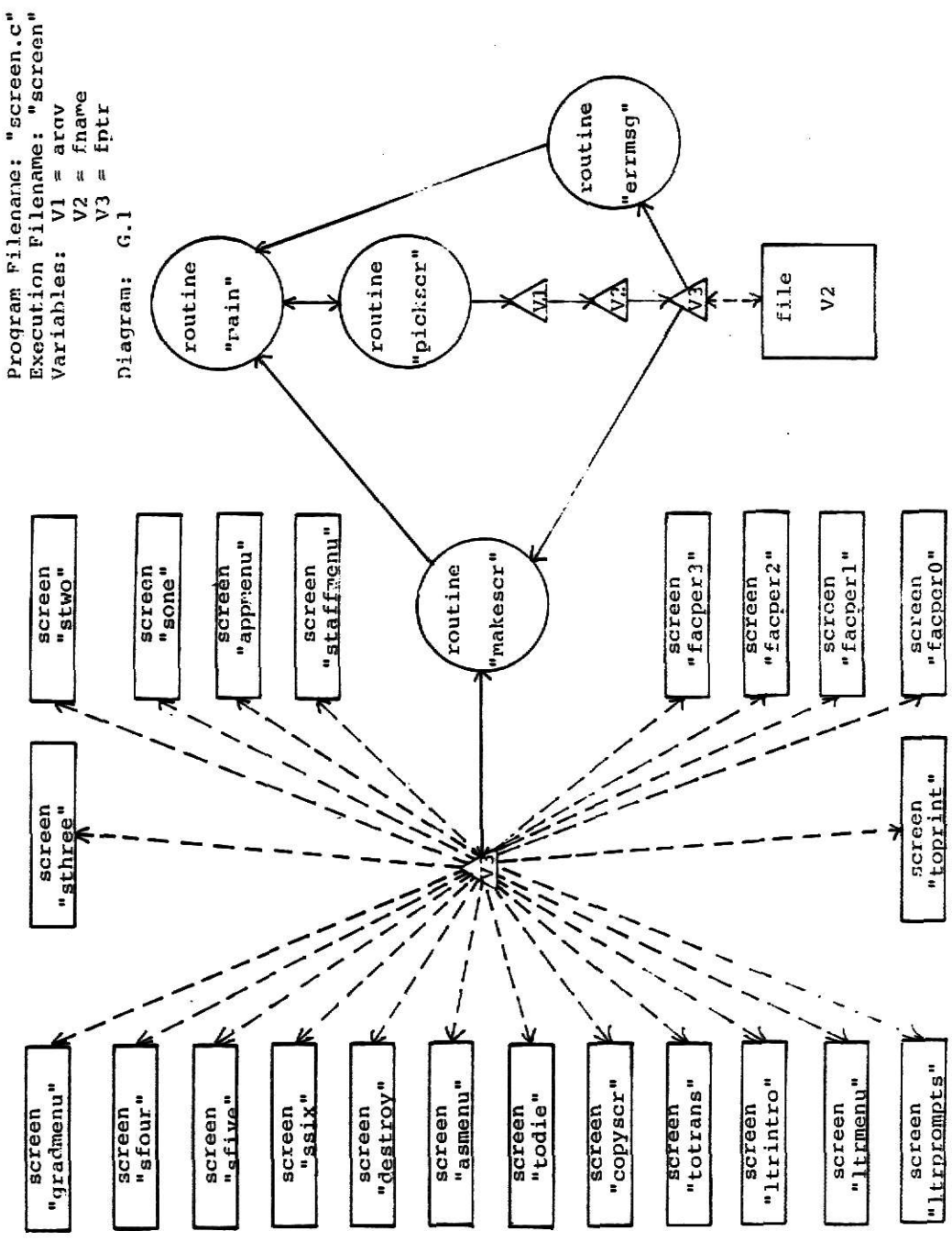
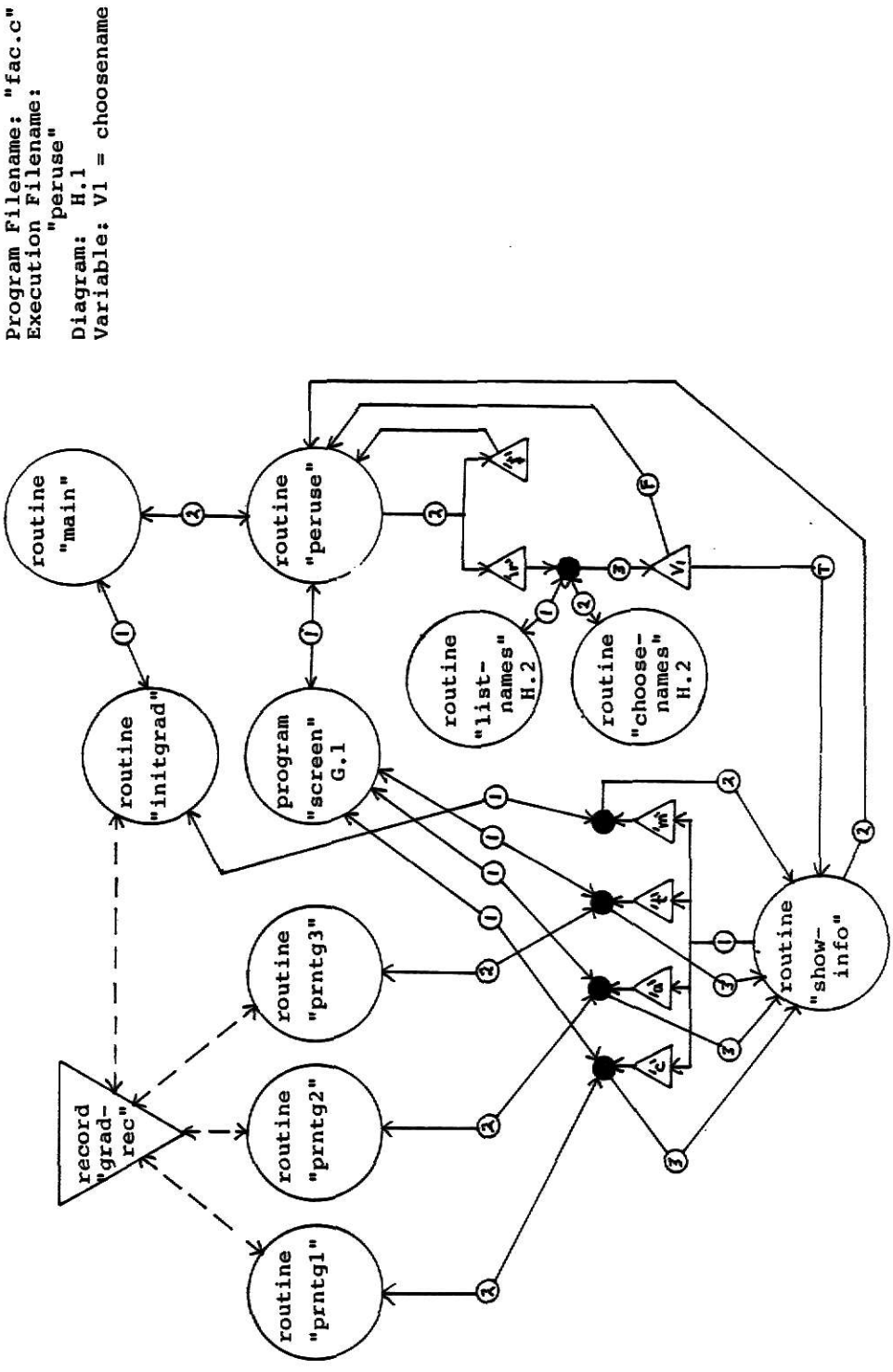
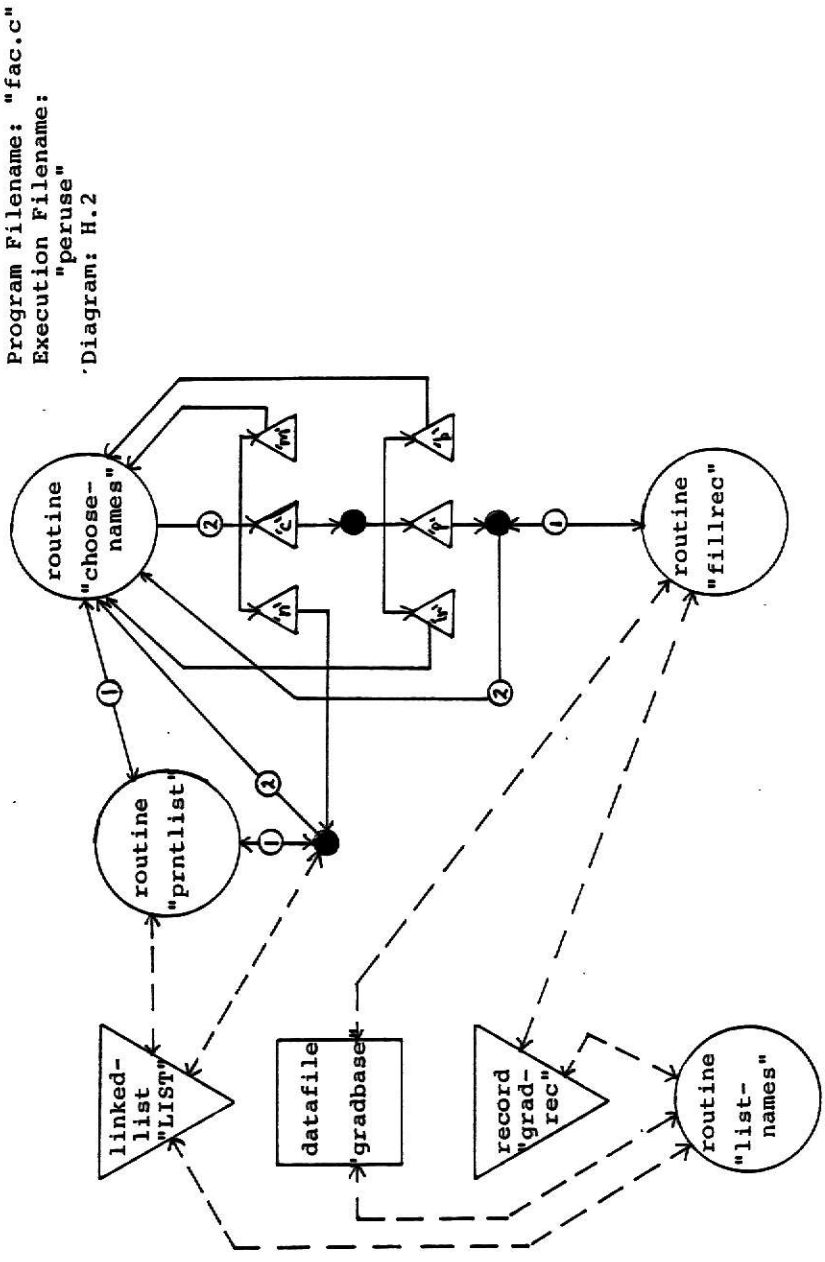

Program Filename:
"trans.c"
Execution Filename:
"transfer"
Variable:
V1 = samestr
Diagram: E.1
routine "main"
routine "transfer"
routine "to-transfer"
routine "init-structs"
program "screen" G.1
routine "copyrecs"
record "grad-rec"
record "temp-rec"
routine "infocopy"
routine "getname"
routine "findcopy"
routine "infofile"
array "names"
datafile "gradbase"
datafile "dbase"

Figure 3.8


Program Filename: "report.c"
Execution Filename:
"report"
Diagram: F.1
Variables: V1 = ch
V2 = c
routine "main"
routine "report"
program "screen" G.1
program "formltr"
routine "prntltr"
formletter
program "gradlist"
report "gsclist"
report "card"
program "card"
routine "copynum"
V1
V2
T
F
1
2
3


Program Filename: "screen.c"
Execution Filename: "screen"
Variables: V1 = argv
V2 = fname
V3 = fptr
Diagram: G.1
routine "main"
routine "pickscr"
routine "errmsg"
file V2
V1
V2
V3
routine "makescr"
screen "stwo"
screen "sone"
screen "appmenu"
screen "staffmenu"
screen "facper3"
screen "facper2"
screen "facper1"
screen "facper0"
screen "sthree"
screen "toprint"
screen "gradmenu"
screen "sfour"
screen "sfive"
screen "ssix"
screen "destroy"
screen "asmenu"
screen "todie"
screen "copyscr"
screen "totrans"
screen "ltrintro"
screen "ltrmenu"
screen "ltrprompts"


Program Filename: "fac.c"
Execution Filename:
"peruse"
Diagram: H.1
Variable: V1 = choosename
routine "main"
routine "peruse"
routine "initgrad"
program "screen" G.1
routine "list-names" H.2
routine "choose-names" H.2
routine "prntg3"
routine "prntg2"
routine "prntg1"
routine "show-info"
record "grad-rec"
V1
F
T


Program Filename: "fac.c"
Execution Filename:
"peruse"
Diagram: H.2
routine "choose-names"
routine "fillrec"
routine "prntlist"
linked-list "LIST"
datafile "gradbase"
record "grad-rec"
routine "list-names"

CHAPTER FOUR

CRITIQUE AND FUTURE ENHANCEMENTS OF THE "KOOL RECORDS SYSTEM"

4.1 OVERVIEW

The "KOOL RECORDS SYSTEM", as described in the previous chapters has been implemented and is currently running on the Perkin-Elmer 3220 minicomputer.

It has been the intent of the author throughout the implementation of the "KOOL RECORDS SYSTEM" to discover and correct potential problems before they arise. It is realized, however, that not all problems can be foreseen. This chapter will attempt to provide an objective critique of the "KOOL RECORDS SYSTEM" covering those areas where problems could exist. Possible future enhancements will also be discussed. It should be noted that these enhancements are derived only from foresight and not from hindsight. Because of this, an evaluation of the "KOOL RECORDS SYSTEM" after it has been in use for several months may provide more meaningful information as to the modifications needed to enhance the use of the system.

It should be mentioned that the current version of the "KOOL RECORDS SYSTEM" is constrained somewhat by the hardware available. The system was designed and implemented to run on terminals that are considered "dumb" as well as

terminals that are considered "intelligent". This allows the Computer Science Department to use the system without the difficulty of requiring a particular type of terminal. As the Computer Science Department continues to upgrade terminal hardware in the future, then perhaps the "KOOL RECORDS SYSTEM" can be enhanced by taking advantage of the various terminal features as they are made available. At the time of this writing, however, the "KOOL RECORDS SYSTEM" must be able to be accessed by the wide variety of terminals that are currently in use.

The discussion of the critique and future enhancements falls into three main areas, which also happen to be the "KOOL RECORDS SYSTEM" components. These areas are as follows: (1) the transaction processor; (2) the database systems; (3) the report generator. The remainder of this chapter will be divided into these three areas for this discussion.

4.2 THE TRANSACTION PROCESSOR

The transaction processor of the "KOOL RECORDS SYSTEM" as described previously, consists of a set of screens that provide an interface between the user and the database systems and report generator. The main problem with the existing implementation of the processor is the lack of ease with which the screens can be modified. For example, Figure 4.1 illustrates an algorithm which must be followed to add a field to an existing screen. The first four steps do not

Figure 4.1

Algorithm for Screen Field Addition

(1) add field to appropriate record structure

(2) locate and modify screen file

(3) modify the screen's input routines

(4) modify the screen's output routines

(5) reorganize database to provide for increased record size

appear difficult, unless there are already too many fields on this screen. If this is the case, the entire screen may need to be rearranged to allow room for the new field. This would require much modification of the screen's information input and output routines as most of the screen's field coordinates would need to be changed. In addition, once the screen has been modified, step five of the algorithm must occur for the information to be displayed at the correct locations on the screen. This is because all records exist in a Unix file in a contiguous manner. The extra space created by the addition of the screen field must also be included in this existing data file. This can only be accomplished by creating a new database file and transferring the contents of the old file to this new file. The new field space can then be added during this transfer. If a complete new screen is to be added, new routines for information input and output will be required in addition to reorganizing the database.

Security is also a major concern with the transaction processor. The current version has only the minimal security provided by the Unix operating system. Some additional security such as terminal isolation and identification may prevent an invalid user from modifying the screens or entering the database from a terminal that is not specified as being valid for use with the "KOOL RECORDS SYSTEM". In addition, the screen files may be allowed a "read-only" access to prevent tampering. This would allow only the

Computer Science Department's systems super-user access to the screens for any modification needed. Another possibility to alleviate the security problem might be to isolate the "KOOL RECORDS SYSTEM" on its own computer system and allowing access to only certain individuals such as the Department's faculty and staff. This may prevent invalid entry through the normal channels of the Unix file system.

4.3 THE DATABASE SYSTEMS

The databases are perhaps the weakest component of the "KOOL RECORDS SYSTEM". Originally designed as relational databases, the current implemented version consists of flat data files, one Unix file for each database. One can access information from any part of these files providing that the exact location of the desired information in the file is known. This exact location can be derived from the corresponding temporary record structure found in the declaration section of the corresponding program. This structure also is the exact size of each record existing in the file. Using these derived sizes, specific queries may be performed, but only by a programmer. It is hoped that a relational database software package will be made available in the future. If this occurs, the "KOOL RECORDS SYSTEM" could be greatly enhanced by the addition of an actual relational database, complete with a query language that could possibly be used by the system's users. Because of this possible addition, the existing database implementation

was developed as simply as possible to enable expedient transfer of information to a database created from such a database package.

Another problem that may arise in future involves the number of students or applicants that can exist at any given point in time. The current implementation incorporates a serial search method to locate the desired student's or applicant's record. Since usually only a small number of graduate students will be in the active file at any given time, a serial search of the graduate student database should not present any problems. This search method may prove to be too slow when used to locate an applicant's record, however, since the potential exists for several hundred applicants to exist in the applicant database at one time. Again, this problem may be solved with the introduction of a database package that incorporates a more efficient search method.

One last problem that should be discussed is that of security. Since each data file must be read from and written to, the possibility exists for illegal tampering with the information present in the file. Again, this problem may be prevented by isolating the "KOOL RECORDS SYSTEM" as previously described. Also, access to the files may be guarded by ownership laws, along with terminal identification and isolation, in addition to enforcing valid user identification.

4.4 THE REPORT GENERATOR

Once again, the problem of security exists with the report generator as with the entire "KOOL RECORDS SYSTEM". It is believed that this security problem may be partially alleviated with the use of the methods described in the previous two sections.

The existing version of the "KOOL RECORDS SYSTEM" report generator allows letters and listings to be printed via the transaction processor. This is done to allow a person's information to be pulled from the appropriate database and printed out with the letter or listing format. Difficulty may arise when a letter or list must be modified, since some programming may be required. In addition to knowledge of the C language, the programmer must also know the Unix system's nroff commands, as all existing letters are formatted using nroff. The necessity of requiring a programmer for these modifications removes the power from the "KOOL RECORDS SYSTEM" users. The addition of a commercially packaged report generator could greatly enhance the system's report generator by allowing the users to format their own reports and allowing the users to decide what information should be on these reports. This enhancement could be provided by choosing the right database package, one that includes report generation facilities. This would allow the users to have the power to change their office system as needed.

REFERENCES

1. Arnold, Kenneth, "Screen Updating and Cursor Movement Optimization : A Library Package", Department of Electrical Engineering and Computer Science, University of California, Berkeley.

2. Bourne, S.R., "The Unix System", Bell Telephone Laboratories, Inc., 1982.

3. Ellis, Clarence A. and Nutt, Gary L., "Computer Science and Office Information Systems", Xerox Palo Alto Research Center, Palo Alto, California, G-1979.

4. Hancok, Leo and Krieger, Morris, "The C Primer", McGraw-Hill, Inc., 1982.

5. Kernighan, Brian W. and Ritchie, Dennis M., "The C Programming Language", Bell Telephone Laboratories, Inc., 1978.

6. Terry, Michael S., "The Design of an Automated Office Information System for the Computer Science Department to Aid in Tracking Graduate Students", Department of Computer Science, Kansas State University, Manhattan, Kansas, 1983.

7. Thomas, Rebecca and Yates, Jean, "A User Guide to the Unix System", McGraw-Hill, Inc., 1982.

8. Warnier, Jean-Dominique, "The Logical Construction of Programs", 3rd Edition, H.E. Stenfert Kroese B.V., Leiden, Paris, 1974.

APPENDIX A

"KOOL RECORDS SYSTEM" Source Code

Appendix A-1

"screen.c"

```
/*****************************************************************************/
/*********************************> screen.c <*********************************/
/** This program receives an argument from a calling program and displays **/
/** a screen depending upon that argument. The screen has previously been **/
/** written onto a Unix file. If problems arise the appropriate error     **/
/** message is produced.                                                  **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>


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

        int     die();

        initscr();
        signal(SIGINT.die);
        crmode();
        picksor(argv);
        endwin();

} /* end of main */

/*****************************************************************************/
/** This program opens the appropriate screen file.                       **/

picksor(argv)
char *argv[];
{
        FILE    *fopen(), *fptr;
        char    *fname;

                switch (*argv[1])
                {
                 case 'k':
                        {
                              fname = "/usr/scott/screens/staffmenu";
                              break;
                        }
                 case 'a':
                        {
                              fname = "/usr/scott/screens/appmenu";
                              break;
                        }
                 case 'b':
                        {
                              fname = "/usr/scott/screens/sone";
                              break;
                        }
                 case 'c':
```

```
            {
                fname = "/usr/scott/screens/stwo";
                break;
            }
        case 'd':
            {
                fname = "/usr/scott/screens/sthree";
                break;
            }
        case 's':
            {
                fname = "/usr/scott/screens/studmenu";
                break;
            }
        case 'g':
            {
                fname = "/usr/scott/screens/sfour";
                break;
            }
        case 'h':
            {
                fname = "/usr/scott/screens/sfive";
                break;
            }
        case 'i':
            {
                fname = "/usr/scott/screens/ssix";
                break;
            }
        case 'r':
            {
                fname = "/usr/scott/screens/destroy";
                break;
            }
        case 'q':
            {
                fname = "/usr/scott/screens/asmenu";
                break;
            }
        case 't':
            {
                fname = "/usr/scott/screens/todie";
                break;
            }
        case 'x':
            {
                fname = "/usr/scott/screens/copyscr";
                break;
            }
        case 'y':
            {
                fname = "/usr/scott/screens/totrans";
                break;
            }
        case 'l':
            {
```

```
                                fname = "/usr/scott/screens/ltrintro";
                                break;
                        }
                case 'm':
                        {
                                fname = "/usr/scott/screens/ltrmenu";
                                break;
                        }
                case 'p':
                        {
                                fname = "/usr/scott/screens/toprint";
                                break;
                        }
                case '1':
                        {
                                fname = "/usr/scott/screens/facper0";
                                break;
                        }
                case '2':
                        {
                                fname = "/usr/scott/screens/facper1";
                                break;
                        }
                case '3':
                        {
                                fname = "/usr/scott/screens/facper2";
                                break;
                        }
                case '4':
                        {
                                fname = "/usr/scott/screens/facper3";
                                break;
                        }
                case '5':
                        {
                                fname = "/usr/scott/screens/ltrprompts";
                                break;
                        }
                default:
                        {
                                printf("SOMETHING IS MESSED UP SOMEWHERE");
                                exit(0);
                        }
                }
                if ((fptr = fopen(fname."r")) == NULL)
                     errmsg(argv);
                makescr(fptr);
} /* end of picksor */
/*****************************************************************************/
/** Errmsg displays an error message to the user if the screen cannot be    **/
/** displayed.                                                              **/

errmsg(argv)
char *argv[];
{
                 if (*argv[1] ==  'k')
```

```
                printf("KOOL RECORD SYSTEM MENU NOT FOUND");
            if (*argv[1] ==  'a')
                        printf("APPLICANT: MENU NOT FOUND");
            if (*argv[1] ==  'b')
                        printf("APPLICANT: INPUT 1 NOT FOUND");
            if (*argv[1] ==  'c')
                        printf("APPLICANT: INPUT 2 NOT FOUND");
            if (*argv[1] ==  'd')
                        printf("APPLICANT: INPUT 3 NOT FOUND");
            if (*argv[1] ==  's')
                        printf("GRADUATE: MENU NOT FOUND");
            if (*argv[1] ==  'g')
                        printf("GRADUATE: INPUT 1 NOT FOUND");
            if (*argv[1] ==  'h')
                        printf("GRADUATE: INPUT 2 NOT FOUND");
            if (*argv[1] ==  'i')
                        printf("GRADUATE: INPUT 3 NOT FOUND");
            if (*argv[1] ==  'q')
                        printf("ASSASSIN MENU NOT FOUND");
            if (*argv[1] ==  'r')
                        printf("DESTROY SCREEN NOT FOUND");
            if (*argv[1] ==  't')
                        printf("DESTROY NAMES NOT FOUND");
            if (*argv[1] == 'x')
                        printf("COPY SCREEN NOT FOUND");
            if (*argv[1] == 'y')
                        printf("NAME TRANSFER SCREEN NOT FOUND");
            if (*argv[1] == '1')
                        printf("PERUSAL MENU NOT FOUND");
            if (*argv[1] == '2')
                        printf("PERUSAL SCREEN 1  NOT FOUND");
            if (*argv[1] == '3')
                        printf("PERUSAL SCREEN 2  NOT FOUND");
            if (*argv[1] == '4')
                        printf("PERUSAL SCREEN 3  NOT FOUND");
            if (*argv[1] == '5')
                        printf("LETER PROMPT SCREEN NOT FOUND");
            if (*argv[1] == 'l')
                        printf("LETTER INTRO SCREEN NOT FOUND");
            if (*argv[1] == 'm')
                        printf("LETTER MENU SCREEN NOT FOUND");
            if (*argv[1] == 'p')
                        printf("NAME PRINT SCREEN NOT FOUND");

            printf("PLEASE SEE HARVARD TOWNSEND ABOUT YOUR PROBLEM");
            exit(0);
}

/*************************************************************************/
/** Makescr uses putc to print out the screen on the CRT (stdout) one char **/
/** at a time.                                                            **/

makescr(fptr)
FILE *fptr;
{    /* begin makescr */
```

```
        int ch;

        clear();
        refresh();
        while ((ch = getc(fptr)) != EOF)
                putc(ch,stdout);
        fclose(fptr);

} /* end of makescr */

/*****************************************************************************/
/** die allows the user to intervene externally into the internal affairs  **/
/** of the terminal. Perhaps the terminal should be called a Grenada.      **/

die()
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
} /* end of die */
/*******************************> end of screen.c <*****************************/
```

Appendix A-2

"super.c"

```
/*************************************************************************/
/***************************** "super.c " ********************************/
/*************************************************************************/
/**                                                                     **/
/** This program provides an entry point to the KOOL RECORDS SYSTEM.    **/
/** Super.c verifies that a user is valid by validating a password      **/
/** that is entered by a user.  The system programs are then provided   **/
/** to all valid users.                                                 **/
/*************************************************************************/

#include        <curses.h> /** required for the curses library package    **/
#include        <stdio.h>  /** standard library package for I/O functions **/
#include        <signal.h> /** this is needed so that the program can     **/
                           /** accept an external signal.                **/
#include        <ctype.h>

main()
{
        int     die();

        initscr();          /** initscr initializes the screen routines **/
        signal(SIGINT,die);
        crmode();
        if (verify() > 0)
         director();
        else               /** not a valid user **/
        {
         mvcur(0,0,10,23);
         printf("NO!  THAT IS NOT RIGHT!  GOODBYE!");
        }
        endwin();
}

/** verify checks the entered password for validity **/
verify()
{
 int  i,chance = 2;
 char *pswrd1 = "SCOTT\0"; /** these are the current valid passwords **/
 char *pswrd2 = "BETH\0";
 char *pswrd3 = "HARV\0";
 char usrwrd[7];
 char ch;

 for (i=0;i<7;++i)
  usrwrd[i] = ' ';
 chance = 2;
 clear();
 refresh();
 while (chance > 0)
 {
  mvcur(0,0,5,19);
  printf("PLEASE ENTER YOUR PASSWORD FOR THIS SYSTEM");
  mvcur(0,0,6,32);
  printf("HERE ===> ");
  i = 0;
  ch = getch();
```

```
  while ((ch != '\r') && (ch != '\n') && (i != 6))
  {
   if (islower(ch) != 0)
    ch = toupper(ch);
   usrwrd[i] = ch;
   ch = getch();
   ++i;
  }
  usrwrd[i] = '\0';
  if ((strcmp(pswrd1, usrwrd) != 0) && (strcmp(pswrd2, usrwrd) != 0)
     && (strcmp(pswrd3, usrwrd) != 0))
  {
     mvcur(0,0,1,30);
     printf("NO! THAT IS NOT RIGHT!");
     mvcur(0,0,2,23);
     printf("WE WILL GIVE YOU ONE MORE CHANCE!");
     --chance;
  }
  else return(chance);
 }
 return(chance);
}

/** director provides access to the KOOL Records System programs **/
director()
{
        int      done = 0;
        char     c;

        while (done == 0)
        {
                system("/usr/scott/xcode/screen k");
                mvcur(0,0,22,65);
                c = getch();
                switch (c)
                {
                case 'a':
                case 'A':
                        {
                                system("/usr/scott/xcode/applicant");
                                /** the applicant management program **/
                                break;
                        }
                case 's':
                case 'S':
                        {
                                system("/usr/scott/xcode/graduate");
                                /** the graduate management program **/
                                break;
                        }
                case 'l':
                case 'L':
                        {
                                system("/usr/scott/xcode/report");
                                /** the report generator **/
                                break;
```

```
                        }
                case '$':
                        {
                                done = 1;
                                break;
                        }
                default :
                        {
                                typo();
                                break;
                        }
                }
        }
}

/** typo simply informs the user that an entry error has been made **/
typo()
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

/** die accepts an external signal and kills the program **/
/** -this signal could be 'break' or 'del', etc.         **/
die()
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-3

"app.c"

```
/*****************************************************************************/
/*****************************> "app.c" <**************************************/
/*****************************************************************************/
/** This program allows the CS Dept staff to create or update applicant     **/
/** information. Also, by calling the program assassin, the user can        **/
/** delete an applicant's record.                                           **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>
#include        "/usr/scott/structs/appstrt.h" /** The applicant temporary **/
                                               /** record structure called **/
                                               /** "temprec".              **/
struct reject rec
{       /** temporary record forthe rejected applicant **/
        char    rejname[40];
        char    reason[30];
        char    rdate[6];
} rejapp;

int     length;  /** This variable is the length of **/
                 /** the name entered by the user.  **/
int     modify;  /** This variable determines if a  **/
                 /** record has been modified.      **/

/****************************> main <*****************************************/
main()
{   /*** begin of main (applicant) ***/
        int   die();

        initstruct();
        initscr();
        signal(SIGINT.die);
        crmode();
        applicant();
        endwin();

}   /*** end of main (applicant) ***/
/*****************************************************************************/

/***********************> initstruct <*****************************************/
initstruct()   /** initializes the temprec structure to blanks *****/
{   /*** begin initstruct ***/

        char *tr;

        for (tr= &temprec; tr < &temprec + 1;++tr)
                *tr = ' ';
}   /*** end initstruct ***/
/*****************************************************************************/

/***********************> initrejstr <****************************************/
initrejstr() /** initializes the rejapp structure to blanks **/
{   /*** begin irs ***/
```

```
        char *tr;

        for (tr = &rejapp;tr<&rejapp + 1;++tr)
                *tr = ' ';
}   /*** end of irs ***/
/*****************************************************************/

/*****************************> applicant <**********************/
applicant()
{   /*** begin applicant ***/

        int      type = 0.done = 0;
        char     c;
        FILE     *fopen(), *fpsize;

        while (done == 0)
        {
                if (type == 0)
                        system("/usr/scott/xcode/screen a");
                        /** displays applicant menu **/
                mvcur(0,0.21.53);
                c = getch();
                switch (c)
                {
                case 'n':
                case 'N':
                case 'u':
                case 'U':
                        {
                                if (name_in() == 1)
                                   /* if name is found or user wants */
                                   /* to start a new record          */
                                {
                                        modify = 0;
                                        showdate();
                                        setupscr();
                                        type = 0;
                                        break;
                                }
                                else break;
                        }
                case 'M':
                case 'm':
                        {
                                done = 1;
                                break;
                        }
                case 'D':
                case 'd':
                        {
            if ((fpsize = fopen("/usr/scott/datafiles/arecsize","w")) == NULL)
  /* The assassin needs the */    {
  /* size of the applicant  */     printf("CAN'T FIND THE SIZE\n");
  /* record. This is done   */     exit(0);
  /* by sending the size to */    }
  /* a file to be accessed  */  fprintf(fpsize."%d",sizeof(struct temp_rec));
```

```
  /* by the assasin program.*/  fclose(fpsize);
                                system("/usr/scott/xcode/assassin a");
                                 /* call the assassin program passing */
                                 /* the paramter 'a' for applicant    */
                                type = 0;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}
/*************************************************************************/

showdate()
{  /** This routine displays the last date that the record was modified **/
  char ch;

   clear();
   refresh();
   printfield(temprec.name.sizeof(temprec.name),8,25);
   mvcur(stdscr,0,10,19);
   printf("LAST DATE THAT THIS RECORD WAS ENTERED WAS:");
   printfield(temprec.lstdate.sizeof(temprec.lstdate),11,25);
   mvcur(stdscr,0,13,22);
   printf("* Please press return to continue. *");
   ch = getch();
}

enterdate()
{  /** This routine updates the date of record modification **/
   int  n read.fd;
   long offset = 0;

   system("date >/usr/scott/xcode/datefile");
   if ((fd = open("/usr/scott/xcode/datefile",0)) == NULL)
   {
      printf("datefile cannot be opened");
      exit(0);
   }
   lseek(fd,offset,0);
   n_read = read(fd,temprec.lstdate.sizeof(temprec.lstdate));
   close(fd);
 }

/************************> name in <************************************/
name in()             /* ========> user input of applicant's name ***/
{
        int present.found,leng = 0;
        char c;

        clear();
```

```
        refresh();
        mvcur(0,0.4,17);
        printf("PLEASE ENTER THE NAME OF THE DESIRED APPLICANT");
        mvcur(0,0.5,18);
        printf("IN THE FOLLOWING FORMAT: LAST. FIRST MIDDLE");
        mvcur(0,0,7.15);
        printf("===> ");
        length = getname(temprec.name. leng); /* retrieve name from user */
        found = readbase();
        if (found == 0)
        {   /* if name is not found in applicant database */
         present = rejbase();
         if (present == 0)
         {   /* if name is not found in rejected applicant database */
            mvcur(0,0.9,23);
            printf("NO RECORD IS FOUND FOR THIS PERSON.");
            mvcur(0,0.11,17);
            printf("AN APPLICANT RECORD WILL BE STARTED IF YOU WISH.");
            mvcur(0,0.14,16);
            printf("PLEASE ENTER A 'Y' IF YOU WANT TO START A RECORD");
            mvcur(0,0.15,21);
            printf("FOR THIS PERSON.  IF NOT. TOUCH RETURN");
            mvcur(0,0,16,37);
            printf("===> ");
            c = getch();
            if ((c != 'y') && (c != 'Y')) return(0);
                 /* if user does not want to start a record on entered name */
         }
         else if (present == 1)
         {        /* name has been rejected--display reason to user */
            printfield(rejapp.name.sizeof(rejapp.name),9,20);
            mvcur(0,0.11,20);
            printf("HAS BEEN REJECTED FOR THE FOLLOWING REASON:");
            printfield(rejapp.reason,sizeof(rejapp.reason),13,25);
            mvcur(0,0,15,24);
            printf("PLEASE PRESS RETURN FOR THE MENU");
            c = getch();
            return(0);
         }
        }
        return(1);
 } /* end of name_in */
/*****************************************************************************/

/***********************> rejbase <*****************************************/
rejbase()                              /* This routine returns a 1 if a name   */
{  /** begin rejbase **/               /* is found in the rejected applicant   */
   char   name[40];                    /* database. Otherwise a 0 is returned. */
   int    fd,samestr = 1.n_read;       /* The calling routine is name_in.      */
   long   offset = 0;                  /****************************************/

   if ((fd = open("/usr/scott/datafiles/reject", 0)) == -1)
   {
       printf("REJECTION FILE CANNOT BE OPENED");
       exit(0);
   }
```

```
   while (samestr != 0)
   {
          lseek(fd, offset, length);
          if ((n_read = read(fd,name.length)) == 0)
          {    /* if eof is reached */
             close(fd);
             return(0);
          }
          else if ((samestr = strncmp(name, temprec.name, length)) == 0)
          {         /* if name is found */
             lseek(fd, offset, 0);
             n_read = read(fd, &rejapp, sizeof(struct reject_rec));
             close(fd);
             return(1);
          }
          offset = offset + sizeof(struct reject rec);
   }
   close(fd);
   return(1);
}  /** end of rejbase **/
/*****************************************************************************/

/*********************************> setupscr <*****************************/
/* This program coordinates the three input screens for the applicants.  */
/* Controls which screen is displayed and input routine for that screen. */

setupscr()
{ /** begin setupscr **/
        int type = 0,done = 0,scrcode = 1;/* init. scrcode to one so that the */
        char     ch;                      /* 1st screen will be displayed     */

        while (done == 0)
        {
                if (type == 0)
                     putscr(scrcode);
                mvcur(0,0,22,73);
                ch = getch();
                switch (ch)
                {
                 case ' ':
                        {
                                if (scrcode == 3)
                                    scrcode = 1;
                                else ++scrcode;
                                type = 0;
                                break;
                        }
                case '-':
                        {
                                if (scrcode == 1)
                                    scrcode = 3;
                                else -- scrcode;
                                type = 0;
                                break;
                        }
                case 'm':
```

```
                case 'M':
                        {
                                if (modify > 0)
                                   enterdate();
                                inputtofile(); /* File the record in dbase */
                                initstruct(); /* Re-initialize record struct */
                                done = 1;
                                break;
                        }
                case 'c':
                case 'C':
                        {
                                if (scrcode == 1)
                                    inapone(); /* call input routine one */
                                else if (scrcode == 2)
                                         inaptwo(); /* call input routine two */
                                     else if (scrcode == 3)
                                              inapthree();
                                                /* call input routine three */
                                ++type;
                                break;
                        }
                case '\n':
                case '\r':
                        {
                                ++type;
                                break;
                        }
                default:
                        {
                                typo();
                                ++type;
                                break;
                        }
                }
        }
} /** end of setupscr **/
/*************************************************************************/

/*******************************> putscr <********************************/
/** This routine calls the program screen to display one of the three **/
/** input screens depending on the screen code sent to it.            **/
/** The corresponding routine to display the applicant's information  **/
/** is also called depending on the screen code.                      **/

putscr(code)
int  code;
{  /** begin of putscr **/
  switch(code)
  {
     case 1:
             {
                  system("/usr/scott/xcode/screen b");
                  printone();   /* display app. info on screen one */
                  break;
             }
```

```
     case 2:
             {
                  system("/usr/scott/xcode/screen c");
                  printtwo();   /* display app. info on screen two */
                  break;
             }
     case 3:
             {
                  system("/usr/scott/xcode/screen d");
                  printthree(); /* display app. info on screen three */
                  break;
             }
     default:
             {
                  printf("SYSTEM SCREENS ARE UNDERGOING DIFFICULTIES");
                  exit(0);
             }
    }
} /** end of putscr **/
/*************************************************************************/

/*****************************> inputtofile <*****************************/
/** This routine files the information in the temporary record structure **/
/** into the applicant database. dbase. If the applicant has been       **/
/** rejected, the name. reason for rejection. and date will be filed    **/
/** into the reject database.                                           **/

inputtofile()
{  /** begin of inputtofile **/
        int   n_read = 1,n_written,fd,fr,samestr = 1;
        char  name[40];
        long  offset = 0;

        if ((fd = open("/usr/scott/datafiles/dbase", 2)) == -1)
        {
                printf("APPLICANT DATABASE CANNOT BE OPENED\n");
                exit(0);
        }
        while ((samestr != 0) && (n_read != 0))
        {
              lseek(fd,offset,0);
              n_read = read(fd,&name,length);
              if ((samestr = strncmp(name,temprec.name,length)) == 0)
              {    /* name has been found in dbase */
                   lseek(fd,offset,0);
                   n_written = write(fd,&temprec,sizeof(struct temp_rec));
              }
              else offset = offset + sizeof(struct temp_rec);
         }
         offset = 0;
         while (samestr != 0)
         {     /* if name was not found in dbase. samstr will not be zero */
              lseek(fd,offset,0);
              n_read = read(fd,&name,length);
              if ((n_read == 0) || (name[0] == '+'))
              {   /* look for garbage flag, or place at end of the file */
```

```
                lseek(fd,offset.0);
                n_written = write(fd,&temprec,sizeof(struct temp_rec));
                samestr = 0; /* to terminate the while loop */
              }
              else offset = offset + sizeof(struct temp_rec);
          }
          close(fd);
          if (temprec.reject != ' ')
          {  /* if the applicant has been rejected */
             if ((fr = open("/usr/scott/datafiles/reject",2)) == -1)
             {
                 printf("REJECT DATABASE CANNOT BE OPENED");
                 exit(0);
             }
             strncpy(rejapp.name.temprec.name.sizeof(temprec.name));
             strncpy(rejapp.reason,temprec.rej reas,sizeof(temprec.rej reas));
             strncpy(rejapp.rdate.temprec.recdate.sizeof(temprec.recdate));
             n_read = samestr = 1;
             while ((samestr != 0) && (n_read != 0))
             {
                   lseek(fd,offset.0);
                   n_read = read(fd,&name.length);
                   if ((samestr = strncmp(name.rejapp.name.length)) == 0)
                   {   /* look for identical name */
                       lseek(fd,offset.0);
                       n_written = write(fd,&rejapp,sizeof(struct reject_rec));
                   }
                   else offset = offset + sizeof(struct reject_rec);
              }
              offset = 0;
              while (samestr != 0)
              {
                   lseek(fd,offset.0);
                   n_read = read(fd,&name.length);
                   if ((n_read == 0) || (name[0] == '+'))
                   {  /* look for garbage flag or place at eof */
                      lseek(fd,offset.0);
                      n_written = write(fd,&rejapp.sizeof(struct reject_rec));
                      samestr = 0;
                   }
                   else offset = offset + sizeof(struct reject_rec);
              }
              close(fr);
       }
}

/*******************************> readbase <*********************************/
/** This routine searches the database for the name entered by the user. **/
/** If name is found, the record is copied into the temporary record for **/
/** manipulation.                                                        **/

readbase()
{
        char  name[40];
        int   fd,samestr = 1.n_read;
        long  offset = 0;
```

```
        if ((fd = open("/usr/scott/datafiles/dbase",0)) == -1)
        {
                printf("FILE DBASE CANNOT BE OPENED\n");
                exit(0);
        }
        while (samestr != 0)
        {
                lseek(fd, offset, 0);
                n_read = read(fd, name, length);
                if (n_read == 0)  /* name not found in applicant dbase */
                {
                        close(fd);
                        return(0);
                }
                else if ((samestr = strncmp(name, temprec.name, length)) == 0)
                {         /* name found */
                        lseek(fd, offset, 0);
                        n_read = read(fd, &temprec, sizeof(struct temp_rec));
                }
                offset = offset + sizeof(struct temp_rec);
        }
        close(fd);
        return(1);
} /** end of readbase **/
/*************************************************************************/

/****************************> inapone <*******************************/
/** Allows the user to enter information on the first input screen. **/

inapone()
{
 char ch;

        fillup(temprec.name,0,35);
        fillup(temprec.address1,2,35);
        fillup(temprec.address2,3,35);
        fillup(temprec.address3,4,35);
        fillup(temprec.address4,5,35);
        fillup(temprec.ssn,7,30);
        mvcur(0,0,7,59);
        ch = getch();
        if ((ch != '\r') && (ch != '\n'))
             temprec.sex = ch;
        fillup(temprec.level,9,27);
        fillup(temprec.location,9,69);
        fillup(temprec.ksres,11,37);
        fillup(temprec.uscit,13,33);
        fillup(temprec.finassist,15,34);
        fillup(temprec.toefl,17,19);
        fillup(temprec.babs,17,40);
        fillup(temprec.procfee,19,38);
} /** end of inapone **/
/*************************************************************************/

/****************************> inaptwo <*******************************/
```

```
/** Allows user to enter info on screen two.                              **/

inaptwo()
{

        fillup(temprec.ksuformdate.2,30);
        fillup(temprec.csformdate.4,29);
        fillup(temprec.recomone.7.15);
        fillup(temprec.reconedate.7.56);
        fillup(temprec.recomtwo.8,15);
        fillup(temprec.rectwodate.8,56);
        fillup(temprec.recomthree,9,15);
        fillup(temprec.recthreedate.9,56);
        fillup(temprec.schoolone.12,22);
        fillup(temprec.schonedate.12,63);
        fillup(temprec.schooltwo.13,22);
        fillup(temprec.schtwodate.13.63);
        fillup(temprec.schoolthree.14,22);
        fillup(temprec.schthreedate.14,63);
        fillup(temprec.schoolfour.15,22);
        fillup(temprec.schfourdate.15,63);
        fillup(temprec.schoolfive.16,22);
        fillup(temprec.schfivedate.16,63);
        fillup(temprec.no_of_trans,18,38);
        fillup(temprec.all trans_rec,20,43);
} /** end of inaptwo **/
/*************************************************************************/

/*******************************> inapthree <******************************/
/** Allows user to enter info on input screen three.                      **/

inapthree()
{
        char  ch;

        fillup(temprec.mislett.4,45);
        fillup(temprec.tocomm.5,38);
        fillup(temprec.toxerox,6.34);
        fillup(temprec.tograd,7.40);
        fillup(temprec.statlett.8,39);
        fillup(temprec.gredate.10,25);
        fillup(temprec.lastcorrdate.10,64);
        mvcur(0,0,12,31);
        ch = getch();
        if ((ch != '\r') && (ch != '\n'))
                temprec.accept = ch;
        mvcur(0,0,12,50);
        ch = getch();
        if ((ch != '\r') && (ch != '\n'))
                temprec.reject = ch;
        mvcur(0,0,13,36);
        ch = getch();
        if ((ch != '\r') && (ch != '\n'))
                temprec.prov = ch;
        mvcur(0,0,13,51);
        ch = getch();
```

```
        if ((ch != '\r') && (ch != '\n'))
                temprec.spec = ch;
        mvcur(0,0,14,34);
        ch = getch();
        if ((ch != '\r') && (ch != '\n'))
                temprec.prob = ch;
        if (temprec.reject != ' ')
             fillup(temprec.rej_reas,15,45);
        fillup(temprec.recdate,17,47);
        fillup(temprec.startdate,19,39);
} /** end of inapthree **/
/*****************************************************************************/

/*********************************> printone <*********************************/
/** Displays applicant info on input screen one.                            **/

printone()
{
        printfield(temprec.name,sizeof(temprec.name),0,35);
        printfield(temprec.address1,sizeof(temprec.address1),2,35);
        printfield(temprec.address2,sizeof(temprec.address2),3,35);
        printfield(temprec.address3,sizeof(temprec.address3),4,35);
        printfield(temprec.address4,sizeof(temprec.address4),5,35);
        printfield(temprec.ssn,sizeof(temprec.ssn),7,30);
        mvcur(0,0,7,59);
        printf("%c",temprec.sex);
        printfield(temprec.level,sizeof(temprec.level),9,27);
        printfield(temprec.location,sizeof(temprec.location),9,69);
        printfield(temprec.ksres,sizeof(temprec.ksres),11,37);
        printfield(temprec.uscit,sizeof(temprec.uscit),13,33);
        printfield(temprec.finassist,sizeof(temprec.finassist),15,34);
        printfield(temprec.toefl,sizeof(temprec.toefl),17,19);
        printfield(temprec.babs,sizeof(temprec.babs),17,40);
        printfield(temprec.procfee,sizeof(temprec.procfee),19,38);
} /** end of printone **/
/*****************************************************************************/

/*********************************> printtwo <*********************************/
/** Displays info on input screen two.                                      **/

printtwo()
{
        printfield(temprec.name,sizeof(temprec.name),0,12);
        printfield(temprec.ksuformdate,sizeof(temprec.ksuformdate),2,30);
        printfield(temprec.csformdate,sizeof(temprec.csformdate),4,29);
        printfield(temprec.recomone,sizeof(temprec.recomone),7,15);
        printfield(temprec.reconedate,sizeof(temprec.reconedate),7,56);
        printfield(temprec.recomtwo,sizeof(temprec.recomtwo),8,15);
        printfield(temprec.rectwodate,sizeof(temprec.rectwodate),8,56);
        printfield(temprec.recomthree,sizeof(temprec.recomthree),9,15);
        printfield(temprec.recthreedate,sizeof(temprec.recthreedate),9,56);
        printfield(temprec.schoolone,sizeof(temprec.schoolone),12,22);
        printfield(temprec.schonedate,sizeof(temprec.schonedate),12,63);
        printfield(temprec.schooltwo,sizeof(temprec.schooltwo),13,22);
        printfield(temprec.schtwodate,sizeof(temprec.schtwodate),13,63);
        printfield(temprec.schoolthree,sizeof(temprec.schoolthree),14,22);
```

```
        printfield(temprec.schthreedate.sizeof(temprec.schthreedate),14,63);
        printfield(temprec.schoolfour.sizeof(temprec.schoolfour),15,22);
        printfield(temprec.schfourdate.sizeof(temprec.schfourdate),15,63);
        printfield(temprec.schoolfive.sizeof(temprec.schoolfive),16,22);
        printfield(temprec.schfivedate.sizeof(temprec.schfivedate),16,63);
        printfield(temprec.no_of trans,sizeof(temprec.no_of trans),18,38);
        printfield(temprec.all_trans_rec,sizeof(temprec.all_trans_rec),20,43);
} /** end of printtwo **/
/*****************************************************************************/

/*********************************> printthree <*********************************/
/** Displays info on input screen three.                                      **/

printthree()
{
        printfield(temprec.name.sizeof(temprec.name),2,12);
        printfield(temprec.mislett.sizeof(temprec.mislett),4,45);
        printfield(temprec.tocomm,sizeof(temprec.tocomm),5,38);
        printfield(temprec.toxerox,sizeof(temprec.toxerox),6,34);
        printfield(temprec.tograd,sizeof(temprec.tograd),7,40);
        printfield(temprec.statlett.sizeof(temprec.statlett),8,39);
        printfield(temprec.gredate.sizeof(temprec.gredate),10,25);
        printfield(temprec.lastcorrdate.sizeof(temprec.lastcorrdate),10,64);
        mvcur(0,0,12,31);
        printf("%c",temprec.accept);
        mvcur(0,0,12,50);
        printf("%c",temprec.reject);
        mvcur(0,0,13,36);
        printf("%c",temprec.prov);
        mvcur(0,0,13,51);
        printf("%c",temprec.spec);
        mvcur(0,0,14,34);
        printf("%c",temprec.prob);
        printfield(temprec.rej reas,sizeof(temprec.rej reas),15,45);
        printfield(temprec.recdate.sizeof(temprec.recdate),17,47);
        printfield(temprec.startdate.sizeof(temprec.startdate),19,39);
} /** end of printthree **/
/*****************************************************************************/

/*********************************> getname <*********************************/
/** Allows user to enter an applicant's name and converts this name to      **/
/** capital letters to maintain consistency.                                **/

getname(person,len)
char person[];
int  len;
{

        int done = 0,i = 0;
        char c,ch;

        len = 0;
        ch = getch();
        while (done == 0)
        {
                switch (ch)
```

```
{
case '\n':
case '\r':
case '\t':
        {       /* finished with input */
                done = 1;
                break;
        }
case '':
        {       /* backspace */
                --i;
                ch = getch();
                break;
        }
case '.':
case '.':
case '-':
        {       /* input of non-alpha characters */
                person[i] = ch;
                ++length;
                ++i;
                ch = getch();
                break;
        }
default:
        {       /* input of alpha characters */
                if (ch != ' ')
                {
                        if (islower(ch) != 0)
                        {
                                c = toupper(ch);
                                person[i] = c;
                        }
                        else person[i] = ch;
                        ++len;
                        ++i;
                        ch = getch();
                }
                else
                {
                        while ((ch = getch()) == ' ');
                        if (ch != '\t')
                        {
                                person[i] = ' ';
                                ++i;
                                if (islower(ch) != 0)
                                {
                                        c = toupper(ch);
                                        person[i] = c;
                                }
                                else person[i] = ch;
                                len = len + 2;
                                ++i;
                                ch = getch();
                        }
                }
```

```
                                break;
                        }
                }
        }
        return(len);
} /** end of getname **/
/*****************************************************************************/

/*******************************> printfield <*******************************/
/** Moves cursor to given screen coordinates and prints given information. **/

printfield(item.len,ycoord,xcoord)
char item[];
int len,ycoord,xcoord;
{
        int i = 0;

        mvcur(stdscr,0.ycoord,xcoord);
        while (i < len)
        {
                putc(item[i], stdout);
                ++i;
        }
} /** end of printfield **/
/*****************************************************************************/

/*******************************> fillup <*********************************/
/** Moves cursor to given screen coordinates and retrieves characters.   **/

fillup(item,ycoord,xcoord)
char item[];
int  ycoord,xcoord;
{
        char  ch;
        int   i = 0;

        mvcur(stdscr,0.ycoord,xcoord);
        ch = getch();
        while ((ch != '\r') && (ch != '\n'))
        {
                if (ch != ')
                {
                        item[i] = ch;
                        ++i;
                        ch = getch();
                }
                else if (ch == ')
                {
                        --i;
                        ch = getch();
                }
        ++modify;
        }

} /** end of fillup **/
/*****************************************************************************/
```

```
/*********************************> typo <************************************/
/** Moves cursor to bottom of screen and displays prompt for incorrect    **/
/** entry of commands.                                                    **/

typo()
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
} /** end of typo **/
/*****************************************************************************/

/*********************************> die <************************************/
/** Allows user to kill program by pressing the 'break' key.              **/

die()
{
        signal(SIGINT.SIG_IGN);
        endwin();
        exit(0);
} /** end of die **/
/*****************************************************************************/
/***************************> end of app.c <*********************************/
```

Appendix A-4

"grad.c"

```
/*************************************************************************/
/*******************************> grad.c <*********************************/
/** This program allows the CS Dept staff to create or update graduate **/
/** information. Also, by calling the program assassin, the user can   **/
/** delete a grad student's record. The program transfer, which is also**/
/** called from this program. transfers info from the applicant        **/
/** database to the graduate database.                                 **/
/*************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>
#include        "/usr/scott/structs/grstrt.h"
                    /** The graduate temporary structure called gradrec. **/

int length; /** Length of name entered by user. **/
int modify; /** To determine last date of record modification. **/

main()
{ /* begin main */

        int     die();

        initgrad();
        initscr();
        signal(SIGINT,die);
        crmode();
        graduate();
        endwin();
} /* end of main */

initgrad() /** Initializes temp record structure. **/
{
        char *gr;

        for (gr = &gradrec; gr < &gradrec + 1;++gr)
                *gr = ' ';
} /* end of initgrad */

graduate() /* Displays menu and coordinates program. */
{
        int      gr,done = 0,type = 0;
        char     c;
        FILE     *fopen(), *fpsize;

        while (done == 0)
        {
                if (type == 0)
                        system("/usr/scott/xcode/screen s ");
                                /* grad menu */
                mvcur(stdscr,0,22,53);
                c = getch();
                switch (c)
                {
                case 'u':
```

```
                case 'U':
                        {
                                if ((name in()) == 1)
                                {
                                        modify = 0;
                                        showdate();
                                        setupscr();
                                        type = 0;
                                        break;
                                }
                                else break;
                        }
                case 'm':
                case 'M':
                        {
                                done = 1;
                                break;
                        }
                case 'd':
                case 'D':
                        {
        if ((fpsize = fopen("/usr/scott/datafiles/grecsize","w")) == NULL)
                                {
                                        printf("CAN'T FIND THE GRAD SIZE");
                                        exit(0);
                                }
                                fprintf(fpsize, "%d", sizeof(struct grad_rec));
                                fclose(fpsize);
                                system("/usr/scott/xcode/assassin g");
                                type = 0;
                                break;
                        }
                case 't':
                case 'T':
                        {
                                system("/usr/scott/xcode/transfer");
                                type = 0;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
} /* end of graduate */

showdate() /** Displays the last date that the record was entered. **/
{
  char ch;

   clear();
   refresh();
   printfield(gradrec.name.sizeof(gradrec.name),8,30);
```

```
   mvcur(stdscr,0,10,19);
   printf("LAST DATE THAT THIS RECORD WAS ENTERED WAS:");
   printfield(gradrec.lastdt,sizeof(gradrec.lastdt),11,25);
   mvcur(stdscr,0,13,22);
   printf("<* Please Press Return To Continue *>");
   ch = getch();
}

enterdate() /** Stores current date if record was modified. **/
{
   int  n_read.fd;
   long offset = 0;

   system("date >/usr/scott/xcode/datefile");
   if ((fd = open("/usr/scott/xcode/datefile",0)) == NULL)
   {
      printf("datefile cannot be opened");
      exit(0);
   }
   lseek(fd,offset,0);
   n_read = read(fd,gradrec.lastdt.sizeof(gradrec.lastdt));
   close(fd);
 }

/*************************************************************************/
name_in()                  /* ========> user input of student's name */
{
        int  i,found,leng = 0;
        char c;

        clear();
        refresh();
        mvcur(0,0,4,19);
        printf("PLEASE ENTER THE NAME OF THE DESIRED STUDENT");
        mvcur(0,0,5,19);
        printf("IN THE FOLLOWNG FORMAT:  LAST. FIRST MIDDLE ");
        mvcur(0,0,7.15);
        printf("===> ");
        length = getname(gradrec.name. leng);
        found = readgrad();
        if (found == 0)
        {
                mvcur(0,0,9,23);
                printf("NO RECORD IS FOUND ON THIS STUDENT");
                mvcur(0,0,11,22);
                printf("A RECORD WILL BE STARTED IF YOU WISH");
                mvcur(0,0,14,16);
                printf("PLEASE ENTER A 'Y' IF YOU WANT TO START A RECORD");
                mvcur(0,0,15,21);
                printf("ON THIS STUDENT.  IF NOT. TOUCH RETURN");
                mvcur(0,0.16,37);
                printf("===> ");
                c = getch();
                if ((c != 'y') && (c != 'Y'))
                        return(0);
        }
```

```
        return(1);
} /* end of name in */
/*************************************************************************/

setupscr()  /** Coordinates which of the three screens will be displayed. **/
{
        int      scrcode = 4,type = 0.done = 0;
        char     c;

        while (done == 0)
        {
                if (type == 0)
                      putscr(scrcode);
                mvcur(stdscr,0.22,71);
                c = getch();
                switch(c)
                {
                case ' ':
                        {
                                if (scrcode == 6)
                                    scrcode = 4;
                                else ++scrcode;
                                type = 0;
                                break;
                        }
                case '-':
                        {
                                if (scrcode == 4)
                                    scrcode = 6;
                                else --scrcode;
                                type = 0;
                                break;
                        }
                case 'm':
                case 'M':
                        {
                                if (modify > 0)
                                    enterdate();
                                in_g_file();
                                initgrad();
                                done = 1;
                                break;
                        }
                case 'c':
                case 'C':
                        {
                                if (scrcode == 4)
                                      ingrone();
                                else if (scrcode == 5)
                                           ingrtwo();
                                     else if (scrcode == 6)
                                                ingrthree();
                                ++type;
                                break;
                        }
                case '\n':
```

```
                case '\r':
                        {
                                ++type;
                                break;
                        }
                default:
                        {
                                typo();
                                ++type;
                                break;
                        }
                }
        }

} /* end of gradone */

putscr(code) /** Calls screen program to display screen. **/
int code;
{
 switch(code)
 {
  case 4:
        {
            system("/usr/scott/xcode/screen g");
            prntg1();
            break;
        }
  case 5:
        {
            system("/usr/scott/xcode/screen h");
            prntg2();
            break;
        }
  case 6:
        {
            system("/usr/scott/xcode/screen i");
            prntg3();
            break;
        }
 default:
        {
            printf("GRAD SYSTEM SCREENS ARE UNDERGOING PROBLEMS");
            exit(0);
        }
 }
}

ingrone()  /** Allows user to input info on first input screen. **/
{
        int   y,jump,d_n_l = 0,c_l = 0,g_l = 0,p_l = 0;
        int   d_n_u = 4,c_u = 2,g_u = 1;
        int   onl = 0,ocl = 0,ogl = 0,ocrl = 0,odl = 0,opl = 0;
        int   onu = 8,ocu = 25,ogu = 1,ocru = 2,odu = 4;

        fillup(gradrec.name,0,12);
        for (y=4;y<14;++y)
```

```
        {
                jump = getinfo(gradrec.courseno. d_n_l. d_n_u,y.3);
                if (jump != 1)
                {
                        jump = getinfo(gradrec.cograde. g_l. g_u,y.13);
                        jump = getinfo(gradrec.corcredit. c_l. c_u,y.21);
                        jump = getinfo(gradrec.cordate. d_n_l. d_n_u,y.28);
                        mvcur(stdscr,0.y.36);
                        gradrec.corprog[p_l] = getch();
                        d_n_l = d_n_l + 4;
                        d_n_u = d_n_u + 4;
                        c_l = c_l + 2;
                        c_u = c_u + 2;
                        ++g_u;
                        ++g_l;
                        ++p_l;
                }
                else if (jump == 1)
                {
                        d_n_u = 40;
                        d_n_l = 44;
                        g_l = 10;
                        g_u = 11;
                        c_l = 20;
                        c_u = 22;
                        p_l = 10;
                        break;
                }
        } /* end for loop */
        for (y=4;y<14;++y)
        {
                jump = getinfo(gradrec.courseno. d_n_l. d_n_u,y.43);
                if (jump == 1)
                        break;
                else
                {
                        jump = getinfo(gradrec.cograde. g_l. g_u,y.53);
                        jump = getinfo(gradrec.corcredit. c_l. c_u,y.61);
                        jump = getinfo(gradrec.cordate. d_n_l. d_n_u,y.68);
                        mvcur(stdscr,0.y.76);
                        gradrec.corprog[p_l] = getch();
                        d_n_l = d_n_l + 4;
                        d_n_u = d_n_u + 4;
                        c_l = c_l + 2;
                        c_u = c_u + 2;
                        ++g_u;
                        ++g_l;
                        ++p_l;
                }
        } /* end of for loop */
        for (y=18;y<22;++y)
        {
                jump = getinfo(gradrec.ocorno.onl.onu,y.2);
                if (jump != 1)
                {
                jump = getinfo(gradrec.ocorname. ocl. ocu,y.13);
```

```
                jump = getinfo(gradrec.ocorgrade. ogl. ogu,y.42);
                jump = getinfo(gradrec.ocorcredit. ocrl. ocru,y.50);
                jump = getinfo(gradrec.ocordate. odl. odu,y.57);
                mvcur(stdscr,0.y,65);
                gradrec.ocorprog[opl] = getch();
                onl = onl + 8;
                onu = onu + 8;
                ocl = ocl + 25;
                ocu = ocu + 25;
                ++ogl;
                ++ogu;
                ocrl = ocrl + 2;
                ocru = ocru + 2;
                odu = odu + 4;
                odl = odl + 4;
                ++opl;
                }
                else break;
        }

}

ingrtwo()  /** Allows user to enter info on second input screen. **/
{
        int  al = 0.au = 25,jump,y.sdl = 0.sdu = 4,pl = 0.pu = 1;

        fillup(gradrec.adrs1.2,1);
        fillup(gradrec.adrs2,3.1);
        fillup(gradrec.adrs3.4,1);
        fillup(gradrec.adrs4,5,1);
        fillup(gradrec.usres,2,52);
        fillup(gradrec.ks_res.3.52);
        fillup(gradrec.gpa.5,44);
        fillup(gradrec.ss,2,69);
        mvcur(stdscr,0.3.77);
        gradrec.sx = getch();
        fillup(gradrec.phone.4,70);
        fillup(gradrec.graddate.7.23);
        fillup(gradrec.ba_bs,7,45);
        fillup(gradrec.prof adv.9,30);
        fillup(gradrec.com mem1,10,30);
        fillup(gradrec.com mem2,11.30);
        jump = getinfo(gradrec.proptitle. 0. 50,13,18);
        jump = getinfo(gradrec.proptitle. 50, 100.14,18);
        fillup(gradrec.ap_prop_date.15,18);
        fillup(gradrec.ms_opt.17.13);
        fillup(gradrec.toef.19,15);
        for (y=17;y<21;++y)
        {
                jump = getinfo(gradrec.status. sdl. sdu,y.52);
                jump = getinfo(gradrec.probat. pl. pu,y.63);
                jump = getinfo(gradrec.statdate. sdl. sdu,y.70);
                sdl = sdl + 4;
                sdu = sdu + 4;
                ++pl;
                ++pu;
```

```
        }
}

ingrthree() /** Allows user to enter info on third input screen. **/
{

        int   y,jump,x,ctl = 0,ctu = 4,qtl = 0,qtu = 3,exl = 0,exu = 11;

        for (y=3;y<20;++y)
        {
                        x = 4;
                        while (x < 11)
                        {
                        jump = getinfo(gradrec.crtaught.ctl,ctu,y,x);
                        ctl = ctl + 4;
                        ctu = ctu + 4;
                        x = x + 6;
                        }
                        if (jump == 1)
                                break;
        }
        x = 22;
        while (x < 38)
        {
                for (y=4;y<20;++y)
                {
                        jump = getinfo(gradrec.crteach. qtl. qtu,y,x);
                        if (jump == 1)
                        {
                                qtl = 45;
                                qtu = 48;
                                break;
                        }
                        qtl = qtl + 3;
                        qtu = qtu + 3;
                }
                x = x + 15;
        }
        for (y=4;y<20;++y)
        {
                jump = getinfo(gradrec.exp. exl. exu,y,49);
                if (jump == 1)
                        break;
                exl = exl + 11;
                exu = exu + 11;
        }
        fillup(gradrec.t effect.20,25);
        fillup(gradrec.ung_adv.20,71);
}

getinfo(item,left,right,ycoord,xcoord) /** Puts info into record field. **/
char item[];
int  left,right,ycoord,xcoord;
{
        char  ch;
```

```
        mvcur(stdscr,0,ycoord,xcoord);
        ch = getch();
        if (ch == '\t')
                return(1);
        while ((ch != '\n') && (ch != '\r') && (left < right))
        {
                if (ch != ')
                {
                        item[left] = ch;
                        ++left;
                        ch = getch();
                }
                else if (ch == ')
                {
                        --left;
                        ch = getch();
                }
                ++modify;
        }
        return(0);
}

in g file() /** Files record in grad database. **/
{
        int  fd,n_written.n_read = 1,samestr = 1;
        char name[40];
        long  offset = 0;

        if ((fd = open("/usr/scott/datafiles/gradbase", 2)) == -1)
        {
                printf("GRAD STUDENT DATABASE CANNOT BE OPENED\n");
                exit(0);
        }

        while ((samestr != 0) && (n_read != 0))
        {
                lseek(fd,offset,0);
                n read = read(fd,name.length);
                if ((samestr = strncmp(name.gradrec.name.length)) == 0)
                {   /* name has been found in gradbase */
                        lseek(fd,offset,0);
                        n_written = write(fd,&gradrec,sizeof(struct grad_rec));
                }
                else offset = offset + sizeof(struct grad_rec);
        }
        offset = 0;
        while (samestr != 0)
        {     /* if name was not found in gradbase. samestr will not be zero */
              lseek(fd,offset,0);
              n_read = read(fd,name.length);
              if ((n_read == 0) || (name[0] == '+'))
              {  /* look for garbage flag or eof */
                   lseek(fd,offset,0);
                   n written = write(fd,&gradrec,sizeof(struct grad_rec));
                   samestr = 0; /* to terminate loop */
              }
```

```
            else offset = offset + sizeof(struct grad_rec);
         }
        close(fd);
}

readgrad()  /** Reads record from database to temp structure. **/
{
        char  name[40];
        int  fd,samestr = 1.n_read;
        long  offset = 0;

        if ((fd = open("/usr/scott/datafiles/gradbase", 0)) == -1)
        {
                printf("FILE:  GRADBASE CANNOT BE OPENED\n");
                exit(0);
        }
        while (samestr != 0)
        {
                lseek(fd, offset. 0);
                if ((n_read = read(fd,name.length)) == 0)
                {
                        close(fd);
                        return(0);
                }
                else if ((samestr = strncmp(name. gradrec.name. length)) == 0)
                {
                        lseek(fd, offset, 0);
                        n_read = read(fd, &gradrec, sizeof(struct grad_rec));
                }
                offset = offset + sizeof(struct grad_rec);
        }
        close(fd);
        return(1);
}

prntg1()   /** Displays record info on first input screen. **/
{
     int y,dnl = 0.cl = 0.gl = 0.pl = 0.dnu = 4,cu = 2,gu = 1;
     int onl = 0.ogl = 0.ocrl = 0.opl = 0.ocl = 0.odl = 0;
     int onu = 8,ocu = 25,ogu = 1.ocru = 2.odu = 4.opu = 3;

        printinfo(gradrec.name.0,40,0,12);
        for (y=4;y<14;++y)
        {
                printinfo(gradrec.courseno.dnl.dnu,y.3);
                printinfo(gradrec.cograde.gl.gu,y.13);
                printinfo(gradrec.corcredit.cl.cu,y.21);
                printinfo(gradrec.cordate.dnl.dnu,y.28);
                mvcur(stdscr,0,y,36);
                printf("%c",gradrec.corprog[pl]);
                dnl = dnl + 4;
                dnu = dnu + 4;
                cl = cl + 2;
                cu = cu + 2;
                ++gu;
                ++gl;
```

```
                ++pl;
        }
        for (y=4;y<14;++y)
        {
                printinfo(gradrec.courseno.dnl.dnu,y.43);
                printinfo(gradrec.cograde,gl.gu,y.53);
                printinfo(gradrec.corcredit.cl.cu,y.61);
                printinfo(gradrec.cordate.dnl.dnu,y.68);
                mvcur(stdscr,0,y.76);
                printf("%c",gradrec.corprog[pl]);
                dnl = dnl + 4;
                dnu = dnu +4;
                cl = cl + 2;
                cu = cu + 2;
                ++gu;
                ++gl;
                ++pl;
        }
        for (y=18;y<22;++y)
        {
                printinfo(gradrec.ocorno, onl. onu,y.2);
                printinfo(gradrec.ocorname. ocl. ocu,y.13);
                printinfo(gradrec.ocorgrade. ogl. ogu,y.42);
                printinfo(gradrec.ocorcredit. ocrl. ocru,y.50);
                printinfo(gradrec.ocordate. odl. odu,y.57);
                mvcur(stdscr,0.y.65);
                printf("%c",gradrec.ocorprog[opl]);
                onl = onl + 8;
                onu = onu + 8;
                ocl = ocl + 25;
                ocu = ocu + 25;
                ++ogl;
                ++ogu;
                ocrl = ocrl + 2;
                ocru = ocru + 2;
                odu = odu + 4;
                odl = odl + 4;
                ++opl;
        }
}

prntg2()  /** Displays record info on second input screen. **/
{
        int y.al = 0.au = 25,sdl = 0.sdu = 4,pl = 0.pu = 1;

        printfield(gradrec.name.sizeof(gradrec.name),0.8);
        printfield(gradrec.adrs1,sizeof(gradrec.adrs1),2.1);
        printfield(gradrec.adrs2,sizeof(gradrec.adrs2),3.1);
        printfield(gradrec.adrs3,sizeof(gradrec.adrs3),4,1);
        printfield(gradrec.adrs4,sizeof(gradrec.adrs4),5.1);
        printfield(gradrec.usres,sizeof(gradrec.usres),2,52);
        printfield(gradrec.ks_res,sizeof(gradrec.ks_res),3.52);
        printfield(gradrec.gpa.sizeof(gradrec.gpa),5,44);
        printfield(gradrec.ss,sizeof(gradrec.ss),2,69);
        mvcur(0,0,3.77);
        printf("%c",gradrec.sx);
```

```
        printfield(gradrec.phone,sizeof(gradrec.phone),4,70);
        printfield(gradrec.graddate.sizeof(gradrec.graddate),7.23);
        printfield(gradrec.ba bs,sizeof(gradrec.ba bs),7.45);
        printfield(gradrec.prof adv.sizeof(gradrec.prof adv),9,30);
        printfield(gradrec.com mem1,sizeof(gradrec.com mem1),10,30);
        printfield(gradrec.com mem2,sizeof(gradrec.com mem2),11,30);
        printinfo(gradrec.proptitle,0,50.13,18);
        printinfo(gradrec.proptitle,50.100,14,18);
        printfield(gradrec.ap_prop_date.sizeof(gradrec.ap_prop_date),15,18);
        printfield(gradrec.ms_opt.sizeof(gradrec.ms_opt),17.13);
        printfield(gradrec.toef.sizeof(gradrec.toef),19,15);
        for (y=17;y<21;++y)
        {
                printinfo(gradrec.status,sdl.sdu,y.52);
                printinfo(gradrec.probat.pl.pu,y.63);
                printinfo(gradrec.statdate.sdl.sdu,y.70);
                sdl = sdl + 4;
                sdu = sdu + 4;
                ++pl;
                ++pu;
        }
}

prntg3() /** Displays info on third input screen. **/
{
        int   y,jump,x.ctl = 0.ctu = 4,qtl = 0.qtu = 3.exl = 0.exu = 11;

        printfield(gradrec.name. sizeof(gradrec.name),0,10);
        for (y=3;y<20;++y)
        {
                    x = 4;
                    while (x < 11)
                    {
                      printinfo(gradrec.crtaught.ctl.ctu,y.x);
                      ctl = ctl + 4;
                      ctu = ctu + 4;
                      x = x + 6;
                    }
        }
        x = 22;
        while (x < 38)
        {
                for (y=4;y<20;++y)
                {
                        printinfo(gradrec.crteach. qtl. qtu,y.x);
                        qtl = qtl + 3;
                        qtu = qtu + 3;
                }
                x = x + 15;
        }
        for (y=4;y<20;++y)
        {
                printinfo(gradrec.exp. exl. exu,y.49);
                exl = exl + 11;
                exu = exu + 11;
        }
```

```
        printfield(gradrec.t effect.sizeof(gradrec.t effect),20,25);
        printfield(gradrec.ung_adv.sizeof(gradrec.ung_adv),20,71);
}

printinfo(item.left.right.ycoord,xcoord) /** Prints part of struct array. **/
char item[];
int  left,right,ycoord,xcoord;
{
        mvcur(stdscr,0,ycoord,xcoord);
        while (left < right)
        {
                putc(item[left], stdout);
                ++left;
        }
}

getname(person.len) /** Accepts name and converts to caps. **/
char person[];
int  len;
{
        int done = 0,i = 0;
        char c,ch;

        len = 0;
        ch = getch();
        while (done == 0)
        {
                switch(ch)
                {
                case '\n':
                case '\r':
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case ':
                        {
                                --i;
                                ch = getch();
                                break;
                        }
                case '.':
                case ',':
                case '-':
                        {
                                person[i] = ch;
                                ++length;
                                ++i;
                                ch = getch();
                                break;
                        }
                default:
                        {
                                if (ch != ' ')
                                {
```

```
                                        if (islower(ch) != 0)
                                        {
                                                c = toupper(ch);
                                                person[i] = c;
                                        }
                                        else person[i] = ch;
                                        ++len;
                                        ++i;
                                        ch = getch();
                                }
                                else
                                {
                                        while ((ch = getch()) == ' ');
                                        if (ch != '\t')
                                        {
                                                person[i] = ' ';
                                                ++i;
                                                if (islower(ch) != 0)
                                                {
                                                        c = toupper(ch);
                                                        person[i] = c;
                                                }
                                                else person[i] = ch;
                                                len = len + 2;
                                                ++i;
                                                ch = getch();
                                        }
                                }
                                break;
                        }
                }
        }
        return(len);
}

printfield(item,len,ycoord,xcoord)  /** Prints out struct field. **/
char item[];
int len,ycoord,xcoord;
{
        int i = 0;

        mvcur(stdscr,0,ycoord,xcoord);
        while (i < len)
        {
                putc(item[i], stdout);
                ++i;
        }
}

fillup(item,ycoord,xcoord) /** Reads in info into structure field. **/
char item[];
int  ycoord,xcoord;
{
        char  ch;
        int   i = 0;
```

```
        mvcur(stdscr,0,ycoord,xcoord);
        ch = getch();
        while ((ch != '\n') && (ch != '\r'))
        {
                if (ch != ')
                {
                        item[i] = ch;
                        ++i;
                        ch = getch();
                }
                else if (ch == ')
                {
                        --i;
                        ch = getch();
                }
                ++modify;
        }

}

typo()  /** Error message. **/
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

die()  /** Allows external interrupt by user. **/
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-5

"assas.c"

```
/*****************************************************************************/
/*********************************> assas.c <**********************************/
/** This program. assas.c (short for assassin), allows the user to delete **/
/** a file on an applicant. a rejected applicant. or a graduate student   **/
/** depending on the argument passed to it. Rather than erasing the       **/
/** entire file. the program fills the name field with pluses (+) marking **/
/** the file for further over-write. This program displays various        **/
/** warnings to the user explaining what will happen if they continue.    **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>

char date[6]; /* global var. is used to delete rejected applicants */

main(argc, argv)
int argc;
char *argv[];
{
        int     die();

        initscr();
        signal(SIGINT.die);
        crmode();
        assassin(argv);
        endwin();
}

/*****************************************************************************/
/*********************************> assassin <*********************************/
/** This program generates menus and warnings to allow the user to make the**/
/** right choice.                                                          **/

assassin(argv)
char *argv[];
{
        int      done = 0;
        char     c;

        if (*argv[1] == 'g') /* if the graduate program called assassin */
             system("/usr/scott/xcode/screen r");
        else
        {
             appfeature(argv); /* if the applicant program called */
             done = 1;
        }
        while (done == 0)
        {
                mvcur(stdscr,0,9,58);
                c = getch();
                switch (c)
                {
                case 'M':
                case 'm':
```

```
                        {
                                done = 1;
                                break;
                        }
                case '\r':
                case '\n':
                        {
                                remove(argv);
                                done = 1;
                                break;
                        }
                default:
                        {
                                typo();
                                break;
                        }
                }
        }
} /** end of assassin **/

/*****************************************************************************/
/** Appfeature allows the user to delete applicants or rejects by name or   **/
/** rejects by date.                                                        **/

appfeature(argv)
 char *argv[];
{
 char  ch,plus[40],dbdate[40];
 int   i,fd,n_read,n_written,type = 0,namecnt = 0,done = 0,quit = 0;
 long  offset = 70; /* offset location starts at reject date */

while (done == 0)
{
 if (type == 0)
    system("/usr/scott/xcode/screen q");
          /* display applicant deletion menu */
 mvcur(stdscr,0,22,48);
 ch = getch();
 switch(ch)
 { /* begin switch */
   case 'd':
   case 'D':
           {  /* delete applicants or rejects by name */
              remove(argv);
              type = 0;
              break;
           }
   case 'm':
   case 'M':
           {
              exit(0);
           }
   case 'r':
   case 'R':  /* delete rejects before a particular date */
           {
              for (i=0;i<40;++i)
```

```
            plus[i] = '+';
    clear();
    refresh();
    mvcur(stdscr,0.4,18);
    printf("THE FUNCTION YOU HAVE CHOSEN WILL ALLOW YOU ");
    mvcur(0,0.6.19);
    printf("TO DELETE A GROUP OF APPLICANTS THAT WERE ");
    mvcur(0,0.8,23);
    printf("REJECTED PRIOR TO A CERTAIN DATE. ");
    mvcur(0,0.10,12);
    printf("PLEASE ENTER THAT DATE IN THE FOLLOWING ");
    printf("FORMAT: mmddyy. ");
    mvcur(0,0.12,21);
    printf("OR PRESS RETURN FOR THE PREVIOUS MENU.");
    mvcur(0,0.14,25);
    printf("PLEASE ENTER HERE ==> {      }");
    mvcur(0,0.14,48);
    ch = getch();
    if ((ch == '\n') || (ch == '\r'))
    {  /* return to menu */
        type = 0;
        break;
    }
    else
    {
     date[0] = ch;
     for (i=1;i<6;++i)
         date[i] = getch();
     if ((fd = open("/usr/scott/datafiles/reject", 2)) == -1)
     {
       printf("REJECT DATABASE CANNOT BE OPENED");
       exit(0);
     }
     while (quit == 0)
     {
      lseek(fd,offset,0);
      if ((n_read = read(fd, dbdate, sizeof(dbdate))) == 0)
      {
           close(fd);
           mvcur(0,0,20,17);
           printf("The number of names deleted ");
           printf("is equal to:  %d",namecnt);
           quit = 1;
      }
      else
      {
       if ((strncmp(dbdate,date,6)) <= 0)
       {
          lseek(fd,(offset-70),0);
          n_written = write(fd, plus, sizeof(plus));
          ++namecnt;
       }
      }
      offset = offset + 76; /* size of reject record */
     }
    }
```

```
            type = 0;
            break;
           }
     default:
           {
               typo();
               ++type;
               break;
           }
     }
  }
} /** end of appfeature **/

/*****************************************************************************/
/** Remove generates a screen to enter names upon and allows the user a     **/
/** choice of returning to the menu or continuing with the operation.       **/

remove(argv)
char *argv[];
{
        int      type = 0.done = 0;
        char     c;

        while (done == 0)
        {
                if (type == 0)
                   system("/usr/scott/xcode/screen t");
                        /* display name screen */
                mvcur(0,0.21,67);
                c = getch();
                switch (c)
                {
                case 'M':
                case 'm':
                        {
                                done = 1;
                                break;
                        }
                case 'c':
                case 'C':
                        {
                                rubout(argv);
                                ++type;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 0;
                                break;
                        }
                }
        }
} /* end of remove */

/*****************************************************************************/
```

```
/** Rubout allows the user to enter names on the screen format ten at a  **/
/** time. If an 'x' is then entered these names will be deleted.         **/
rubout(argv)
char *argv[];
{
        static char names[10][40]; /* 2d array to contain names */
        int  err = 0.i,j.y = 7.leng,done = 0;
        int  len[10]; /* lengths of names in above 2d array */
        int  found[10]; /* if each of the ten names is found */
        char ch;

        for (j=0;j<10;++j)
        {   /* init. names */
            for (i=0;i<40;++i)
                  names[j][i] = ' ';
            found[j] = 1;
        }
        while (done == 0)
        {
                for (j=0;j<10;++j)
                {   /* read name(s) into array */

                      len[j] = (getname(names,j.leng,y.16));
                      ++y;
                }
                mvcur(0,0.13.73);
                ch = getch();
                switch (ch)
                {
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case 'X':
                case 'x':
                        {
                                for (j=0;j<10;++j)
                                { /* attempt to delete each of the ten names */
                                      if (len[j] != 0)
                                            found[j] = x_out(names.j.len.argv);
                                }
                                y = 7;
                                for (i=0;i<10;++i)
                                {
                                        if (found[i] == 0)
                                        {  /* if name is not found */
                                                mvcur(0.0.y.0);
                                                printf("NOT FOUND ===>");
                                                err = 1;
                                        }
                                        ++y;
                                }
                                if (err == 1)
                                {
                                        mvcur(0.0.17.17);
```

```
                                        printf("IF NAME IS NOT FOUND, ");
                                        printf("PLEASE CHECK ");
                                        printf("TO SEE THAT ");
                                        mvcur(0,0,18,14);
                                        printf("THE SPELLING AND FORMAT ");
                                        printf("IS RIGHT.  ");
                                        printf("THEN PLEASE REENTER.");
                                }
                                done = 1;
                                break;
                        }
                case '\r':
                case '\n':
                        {
                                break;
                        }
                default:
                        {
                                typo();
                                break;
                        }
                }
        }
}

x out(stiff,j,len,argv)
char stiff[10][40],*argv[];
int  j,len[];
{
        int l,size,i,n_read, n_written, samestr = 1, fd,fr;
        long  offset = 0;
        char name[40],dead[40],person[40];
        FILE *fopen(), *fpsize;

        /* the size of the calling programs data structure has already */
        /* been determined and read into the files grecsize or appsize */
        if (*argv[1] == 'g')
        {  /* if gradaute is calling */
        if ((fpsize = fopen("/usr/scott/datafiles/grecsize","r")) == NULL)
                {
                        printf("REC SIZE CANNOT BE FOUND\n");
                        exit(0);
                }
                fscanf(fpsize,"%d",&size);
                fclose(fpsize);
                if ((fd = open("/usr/scott/datafiles/gradbase", 2)) == -1)
                {
                        printf("FILE GRADBASE CANNOT BE OPENED\n");
                        exit(0);
                }
        }
        if (*argv[1] == 'a')
        {  /* if applicant is calling */
        if ((fpsize = fopen("/usr/scott/datafiles/arecsize","r")) == NULL)
                {
                        printf("RECORD SIZE CANNOT BE FOUND\n");
```

```
                exit(0);
        }
        fscanf(fpsize,"%d",&size);
        fclose(fpsize);
        if ((fd = open("/usr/scott/datafiles/dbase", 2)) == -1)
        {
                printf("FILE DBASE CANNOT BE OPENED\n");
                exit(0);
        }
}

for (i=0;i<40;++i)
{
        name[i] = ' ';
        dead[i] = '+';
        person[i] = stiff[j][i];
}
l = len[j];
while (samestr != 0)
{
        lseek(fd, offset, 0);
        n_read = read(fd, name, l);
        if ((samestr = strncmp(name, person, l)) == 0)
        {  /* if name has been found in database */
                lseek(fd, offset, 0);
                n_written = write(fd, &dead, sizeof(dead));
                /* fill name field with pluses */
                close(fd);
                return(1);
        }
        else if ((n_read == 0) && (*argv[1] == 'a'))
        {     /* name not found in applicant database-look in reject */
              if ((fr = open("/usr/scott/datafiles/reject", 2)) == -1)
                {
                     printf("REJECT DATABASE CANNOT BE OPENED");
                     exit(0);
                }
                offset = 0;
             while (samestr != 0)
             {
               lseek(fr.offset.0);
               n_read = read(fr.name.l);
               if ((samestr = strncmp(name.person.l)) == 0)
               {   /* name is found in reject */
                   lseek(fr.offset.0);
                   n_written = write(fr.&dead,sizeof(dead));
                   close(fr);
                   return(1);
               }
               else if (n_read == 0)
                    samestr = 0;
               offset = offset + 76;/* size of structure reject_rec */
             } /* end of while */
        }
        else offset = offset + size;
}
```

```
        close(fr);
        return(0);
} /** end of x_out **/

/*****************************************************************************/
/** Getname accepts the name from the user and converts it to capital      **/
/** letters.                                                               **/

getname(person,j,len,ycoord,xcoord)
char person[10][40];
int  j,len,ycoord,xcoord;
{
        int done = 0,i = 0;
        char c,ch;

        len = 0;
        mvcur(stdscr,0,ycoord,xcoord);
        ch = getch();
        while (done == 0)
        {
                switch(ch)
                {
                case '\n':
                case '\r':
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case ':
                        {
                                --i;
                                ch = getch();
                                break;
                        }
                case '.':
                case '.':
                case '-':
                        {
                                person[j][i] = ch;
                                ++i;
                                ch = getch();
                                break;
                        }
                default:
                        {
                                if (ch != ' ')
                                {
                                        if (islower(ch) != 0)
                                        {
                                                c = toupper(ch);
                                                person[j][i] = c;
                                        }
                                        else person[j][i] = ch;
                                        ++len;
                                        ++i;
```

```
                                        ch = getch();
                                }
                                else
                                {
                                        while ((ch = getch()) == ' ');
                                        if (ch != '\t')
                                        {
                                                person[j][i] = ' ';
                                                ++i;
                                                if (islower(ch) != 0)
                                                {
                                                        c = toupper(ch);
                                                        person[j][i] = c;
                                                }
                                                else person[j][i] = ch;
                                                len = len + 2;
                                                ++i;
                                                ch = getch();
                                        }
                                }
                                break;
                        }
                }
        }
        return(len);
} /** end of getname **/

/*****************************************************************************/
/** Typo informs the user that a mistake has been made.                     **/

typo()
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
} /** end of typo **/

/*****************************************************************************/
/** Die allows the user to kill a program externally ie., by pressing the  **/
/** break key.                                                              **/

die()
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
} /** end of die **/
/*******************************> end of assas.c <****************************/
```

Appendix A-6

"trans.c"

```
/*************************************************************************/
/*****************************> trans.c <******************************/
/** This program allows transfer of certain information from the     **/
/** applicant database to the graduate database.                     **/
/*************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>
#include        "/usr/scott/structs/appstrt.h"
#include        "/usr/scott/structs/grstrt.h"

main()
{
        int     die();

        initstructs();
        initscr();
        signal(SIGINT,die);
        crmode();
        transfer();
        endwin();
}

initstructs() /** Initializes structures to blanks. **/
{
        char *gr,*tr;
        for (tr= &temprec; tr < &temprec + 1;++tr)
                *tr = ' ';
        for (gr = &gradrec; gr < &gradrec + 1;++gr)
                *gr = ' ';
}

transfer()  /** Allows user a chance to return to menu or to continue. **/
{
        char     c;
        int      type = 0.done = 0;

        while (done == 0)
        {
                if (type == 0)
                        system("/usr/scott/xcode/screen x");
                mvcur(stdscr,0,8,53);
                c = getch();
                switch (c)
                {
                case 'M' :
                case 'm' :
                        {
                                done = 1;
                                break;
                        }
                case '\n':
                case '\r':
                        {
```

```
                                totransfer();
                                done = 1;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}

totransfer() /** Displays screen to enter names. Allows user choice to **/
{            /** continue or return to menu. **/
        int type = 0,done = 0;
        char c;

        while (done == 0)
        {
                if (type == 0)
                        system("/usr/scott/xcode/screen y");
                mvcur(stdscr,0,21.68);
                c = getch();
                switch (c)
                {
                case 'm':
                case 'M':
                        {
                                done = 1;
                                break;
                        }
                case 'C':
                case 'c':
                        {
                                initstructs();
                                copyrecs();
                                type = 1;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}

copyrecs() /** Retrieves names off of screen and pulls records. **/
{
        static char names[10][40];
        int  len[10],err = 0.y = 7.i.j.found[10],leng,done = 0;
        char ch;
```

```
        for (j=0;j<10;++j)
        {
                for (i=0;i<40;++i)
                        names[j][i] = ' ';
                found[j] = 1;
        }
        while (done == 0)
        {
                for (j=0;j<10;++j)
                {
                        len[j] = (getname(names,j,leng,y,16));
                        ++y;
                }
                mvcur(stdscr,0,13,73);
                ch = getch();
                switch (ch)
                {
                case '\r':
                case '\n':
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case 'T':
                case 't':
                        {
                                for (j=0;j<10;++j)
                                {
                                       if (len[j] != 0)
                                        found[j] = findcopy(names,j,len);
                                }
                                y = 7;
                                for (i=0;i<10;++i)
                                {
                                        if (found[i] == 0)
                                        {
                                                mvcur(0,0,y,0);
                                                printf("NOT FOUND ===>");
                                                err = 1;
                                        }
                                        ++y;
                                }
                                if (err == 1)
                                {
                                        mvcur(0,0,17,17);
                                        printf("IF NAME IS NOT FOUND, ");
                                        printf("PLEASE CHECK ");
                                        printf("TO SEE THAT");
                                        mvcur(0,0,18,14);
                                        printf("THE SPELLING AND FORMAT ");
                                        printf("IS RIGHT.  ");
                                        printf("THEN PLEASE REENTER");
                                }
                                done = 1;
```

```
                                break;
                        }
                default:
                        {
                                typo();
                                break;
                        }
                }
        }
}

getname(person,j,len,ycoord,xcoord) /** Makes sure name is entered precisely.*/
char person[10][40];
int  len,ycoord,xcoord;
{

        int done = 0, i = 0;
        char c,ch;

        len = 0;
        mvcur(stdscr,0,ycoord,xcoord);
        ch = getch();
        while (done == 0)
        {
                switch (ch)
                {
                case '\n':
                case '\r':
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case ':
                        {
                                --i;
                                ch = getch();
                                break;
                        }
                case '.':
                case ',':
                case '-':
                        {
                                person[j][i] = ch;
                                ++i;
                                ch = getch();
                                break;
                        }
                default:
                        {
                                if (ch != ' ')
                                {
                                        if (islower(ch) != 0)
                                        {
                                                c = toupper(ch);
                                                person[j][i] = c;
```

```
                                        }
                                        else person[j][i] = ch;
                                        ++len;
                                        ++i;
                                        ch = getch();
                                }
                                else
                                {
                                        while ((ch = getch()) == ' ');
                                        if (ch != '\t')
                                        {
                                                person[j][i] = ' ';
                                                ++i;
                                                if (islower(ch) != 0)
                                                {
                                                        c = toupper(ch);
                                                        person[j][i] = c;
                                                }
                                                else person[j][i] = ch;
                                                len = len + 2;
                                                ++i;
                                                ch = getch();
                                        }
                                }
                                break;
                        }
                }
        }
        return(len);
}

findcopy(person,j,len)  /** Retrieves applicant info from database. **/
char person[10][40];
int  j,len[];
{
        int fd, i, n_read, samestr = 1;
        long  offset = 0;
        char name[40],trname[40];

        if ((fd = open("/usr/scott/datafiles/dbase", 0)) == -1)
        {
                printf("FILE DBASE CANNOT BE OPENED\n");
                exit(0);
        }
        for (i=0;i<40;++i)
        {
                name[i] = ' ';
                trname[i] = person[j][i];
        }
        while (samestr != 0)
        {
                lseek(fd, offset, 0);
                n_read = read(fd,name,len[j]);
                if ((samestr = strncmp(name,trname,len[j])) == 0)
                {
                        lseek(fd, offset, 0);
```

```
                        n_read = read(fd, &temprec, sizeof(struct temp_rec));
                        close(fd);
                        infocopy();
                        infofile(len[j]);
                        return(1);
                }
                else
                {
                        if (n_read != 0)
                                offset = offset + sizeof(struct temp_rec);
                        else return(0);
                }
        }
        close(fd);
        return(1);
}

infocopy() /** Copies pertinent info from one temp structure to other. **/
{
        strncpy(gradrec.name. temprec.name. sizeof(temprec.name));
        strncpy(gradrec.ss, temprec.ssn, sizeof(temprec.ssn));
        strncpy(gradrec.ks_res, temprec.ksres, sizeof(temprec.ksres));
        strncpy(gradrec.usres, temprec.uscit. sizeof(temprec.uscit));
        strncpy(gradrec.toef. temprec.toefl, sizeof(temprec.toefl));
        strncpy(gradrec.ba_bs, temprec.babs. sizeof(temprec.babs));
        strncpy(gradrec.sx, temprec.sex, sizeof(temprec.sex));
}

infofile(l) /** Files grad info in grad database. **/
int l;
{
        int  fd, n_written. n_read,i.copy = 0.allthru = 0;
        char name[40];
        long  offset = 0;

        if ((fd = open("/usr/scott/datafiles/gradbase", 2)) == -1)
        {
                printf("GRAD STUDENT DATABASE CANNOT BE OPENED\n");
                exit(0);
        }
        for (i=0;i<40;++i)
                name[i] = ' ';
        while (allthru != 1)
        {
                lseek(fd, offset. 0);
                if (((n_read = read(fd, name. 1)) == 0) && (copy == 0))
                {
                        copy = 1;
                        offset = 0;
                }
              . else if ((strncmp(name. gradrec.name. 1)) == 0)
                {
                        for (i=38;i>-1;--i)
                                gradrec.name[i+1] = gradrec.name[i];
                        gradrec.name[0] = '*';
                        copy = offset = 0;
```

```
                }
                else if (copy == 1)
                {
                 if ((n_read == 0) || (name[0] == '+'))
                  {
                   lseek(fd, offset, 0);
                   n_written = write(fd, &gradrec, sizeof(struct grad_rec));
                   allthru = 1;
                  }
                  else offset = offset + sizeof(struct grad_rec);
                }
                else {
                        offset = offset + sizeof(struct grad_rec);
                }
        }
        close(fd);
}

typo()  /** Error message. **/
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

die()  /** Allows external interrupt by user. **/
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-7

"fac.c"

```
/*****************************************************************************/
/*********************************> fac.c <************************************/
/** This program allows the faculty member to peruse any graduate           **/
/** record by choosing the name from a displayed alphabetical list.         **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>
#include        "/usr/scott/structs/grstrt.h"
                 /* grad student temporary data structure */

struct namelist
{
        char name[40];
        struct namelist *next;
};

typedef struct namelist LIST;  /** node of a linked list **/

int length,listcount,last;
LIST *listnames();

main()
{ /* begin main */

        int     die();

        initgrad();
        initscr();
        signal(SIGINT,die);
        crmode();
        peruse();
        endwin();
} /* end of main */

initgrad()
{
        char *gr;

        for (gr = &gradrec; gr < &gradrec + 1;++gr)
                *gr = ' ';
} /* end of initgrad */

peruse()  /** Displays information screen and setups linked list to display.**/
{
        LIST     *head;
        int      done = 0,type = 0;
        char     c;

        while (done == 0)
        {
                if (type == 0)
                        system("/usr/scott/xcode/screen 1 ");
                mvcur(stdscr,0,20,58);
```

```
                c = getch();
                switch (c)
                {
                case '$':
                        {
                                done = 1;
                                break;
                        }
                case '\n':
                case '\r':
                        {
                                head = listnames();
                                if ((choosename(head)) == 1)
                                {
                                        showdate();
                                        showinfo();
                                }
                                type = 0;
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}

showdate() /** Displays last date that students record was entered. **/
{
  char  ch;

   clear();
   refresh();
   printfield(gradrec.name.sizeof(gradrec.name),8,30);
   mvcur(stdscr,0,10,19);
   printf("LAST DATE THAT THIS RECORD WAS ENTERED WAS:");
   printfield(gradrec.lastdt.sizeof(gradrec.lastdt),11,25);
   mvcur(stdscr,0,20,22);
   printf("<* Please Press Return To Continue *>");
   ch = getch();
}

showinfo() /** Coordinates screen display and information display. **/
{
        int  done = 0;
        char ch = 'c';

        while (done == 0)
        {
                switch(ch)
                {
                case 'c':
                case 'C':
```

```
                        {
                                system("/usr/scott/xcode/screen 2");
                                prntg1();
                                mvcur(stdscr,0,22,77);
                                ch = getch();
                                break;
                        }
                case 'a':
                case 'A':
                        {
                                system("/usr/scott/xcode/screen 3");
                                prntg2();
                                mvcur(stdscr,0,22,77);
                                ch = getch();
                                break;
                        }
                case 't':
                case 'T':
                        {
                                system("/usr/scott/xcode/screen 4");
                                prntg3();
                                mvcur(stdscr,0,22,77);
                                ch = getch();
                                break;
                        }
                case 'm':
                case 'M':
                        {
                                initgrad();
                                done = 1;
                                break;
                        }
                default:
                        {
                                typo();
                                mvcur(stdscr,0,22,77);
                                ch = getch();
                                break;
                        }
                }
        }
}

LIST *listnames()  /** Builds alphabetized linked list for CRT display. **/
{
        LIST  *node,*nextnode,*prev,*tail;
        static LIST *head;
        int   i,fd,insert = 0,n_read;
        char  name[40];
        long  offset = 0;

        if (((head = malloc(sizeof(LIST))) != NULL) &&
            ((tail = malloc(sizeof(LIST))) != NULL))
        {
                for (i=0;i<40;++i)
                {
```

```
                        name[i] = head->name[i] = ' ';
                        tail->name[i] = 'z';
                }
                head->next = tail;
                tail->next = NULL;
                nextnode = head;
        }
        if ((fd = open("/usr/scott/datafiles/gradbase",0)) == -1)
        {
                printf("FILE GRADBASE CANNOT BE OPENED");
                exit(0);
        }
        lseek(fd,offset.0);
        while ((n_read = read(fd,name.sizeof(name))) != 0)
        {
                if ((name[0] != '+') && (name[0] != ' '))
                {
                    if ((node = malloc(sizeof(LIST))) != NULL)
                    {
                      for (i=0;i<40;++i)
                          node->name[i] = name[i];
                    }
                }
                else insert = 1;
                while (insert == 0)
                {
                if ((strncmp(node->name.nextnode->name.sizeof(node->name))) > 0)
                        {
                                prev = nextnode;
                                nextnode = nextnode->next;
                        }
                        else
                        {
                                node->next = prev->next;
                                prev->next = node;
                                insert = 1;
                        }
                } /* while insert */
                offset = offset + sizeof(struct grad_rec);
                nextnode = head;
                lseek(fd,offset.0);
                insert = 0;
        }
        close(fd);
        return(head);
}

choosename(head) /** Allows fac. member to choose which record to peruse **/
LIST *head;
{
        LIST *nextnode,*node;
        int y.scrent = 1.loc = 1.local = 1.done = 0;
        char c,ch;

        nextnode = node = head;
        if (node == head)
```

```
        node = head->next;
    nextnode = prntlist(node);
    while (done == 0)
    {
        mvcur(stdscr,0,20,77);
        ch = getch();
        switch(ch)
        {
        case 'c':
        case 'C':
            {
                y = 0;
                loc = local;
                while (y < 19)
                {
                    mvcur(stdscr,0,y,2);
                    c = getch();
                    switch(c)
                    {
                    case '\r':
                    case '\n':
                    case '\t':
                        {
                            if ((last == 1) &&
                             (y >= ((listcount*2)-2)))
                                y = 20;
                            else
                                {
                                  ++loc;
                                  y = y + 2;
                                }
                            break;
                        }
                    case 'p':
                    case 'P':
                        {
                          fillrec(head,loc);
                          return(1);
                          break;
                        }
                    case 'b':
                    case 'B':
                        {
                          y = 20;
                          loc = 1;
                          break;
                        }
                    default:
                        {
                          break;
                        }
                    }
                } /*while*/
                break;
            }
        case 'm':
```

```
                case 'M':
                        {
                                return(0);
                                break;
                        }
                case 'n':
                case 'N':
                        {
                                if (nextnode->next != NULL)
                                {
                                        node = nextnode;
                                        nextnode = prntlist(node);
                                        local = (((++scrent)*10) -9);
                                }
                                break;
                        }
                default:
                        {
                                typo();
                                break;
                        }
                } /*switch*/
        }
} /*choosename*/

prntlist(node) /** Displays the linked list. **/
LIST *node;
{
        int ycoord,i,y = 0;

        clear();
        refresh();
        listcount = last = 0;
        while ((y < 19) && (node->next != NULL))
        {
                mvcur(0,0,y,1);
                printf("{ }");
                mvcur(0,0,y,5);
                for (i=0;i<40;++i)
                        printf("%c",node->name[i]);
                node = node->next;
                y = y + 2;
                ++listcount;
        }
        if (node->next == NULL)
        {
              printf("\n");
              printf("* END OF LIST--PLEASE RETURN TO MENU *\n");
              printf("*      IF NAME IS NOT PRESENT.       *");
              last = 1;
        }
        for (ycoord=0;ycoord<22;++ycoord)
        {
             mvcur(0,0,ycoord,46);
             printf("*");
        }
```

```
        mvcur(0,0,0,48);
        printf("If desired name is on this");
        mvcur(0,0,2,48);
        printf("screen please enter a  'c'.");
        mvcur(0,0,4,48);
        printf("Then press return until the");
        mvcur(0,0,6,48);
        printf("cursor is next to the name and");
        mvcur(0,0,8,48);
        printf("enter a 'p' to pull this");
        mvcur(0,0,10,48);
        printf("student's record. If desired");
        mvcur(0,0,12,48);
        printf("name is not on this screen.");
        mvcur(0,0,14,48);
        printf("enter an 'n' for the next list");
        mvcur(0,0,16,48);
        printf("of names. If you wish to return");
        mvcur(0,0,18,48);
        printf("to the menu, enter an 'm'.");
        mvcur(0,0,20,48);
        printf("PLEASE ENTER CHOICE HERE ==>{ }");
        return(node);
}

fillrec(head,loc)  /** Fills the temp. structure with the chosen info. **/
LIST *head;
int loc;
{ /* fillrec begin */
        LIST *node;
        int fd,n_read,current = 1,done = 0;
        char name[40];
        long offset = 0;

        if ((fd = open("/usr/scott/datafiles/gradbase",0)) == -1)
        {
                printf("FILE GRADBASE CANNOT BE OPENED");
                exit(0);
        }
        node = head->next;
        while (current < loc)
        { /* begin while current */
                node = node->next;
                ++current;
        } /* end while current */
        lseek(fd,offset,0);
        while (((n_read = read(fd,name,sizeof(name))) != 0) && (done == 0))
        { /* begin while read */
                if ((strncmp(name,node->name, sizeof(name))) == 0)
                {
                 lseek(fd,offset,0);
                 n_read = read(fd,&gradrec,sizeof(struct grad_rec));
                 done = 1;
                }
                else
                {
```

```
                offset = offset + sizeof(struct grad_rec);
                lseek(fd,offset,0);
               }
        } /* end while read */
} /* end fillrec */

prntg1() /** Displays student info on screen one. **/
{
        int y,dnl = 0,cl = 0,gl = 0,pl = 0,dnu = 4,cu = 2,gu = 1;
        int onl = 0,ogl = 0,ocrl = 0,opl = 0,ocl = 0,odl = 0;
        int onu = 8,ocu = 25,ogu = 1,ocru = 2,odu = 4,opu = 3;

        printinfo(gradrec.name,0,40,0,12);
        for (y=4;y<14;++y)
        {
                printinfo(gradrec.courseno,dnl,dnu,y,3);
                printinfo(gradrec.cograde,gl,gu,y,13);
                printinfo(gradrec.corcredit,cl,cu,y,21);
                printinfo(gradrec.cordate,dnl,dnu,y,28);
                mvcur(stdscr,0,y,36);
                printf("%c",gradrec.corprog[pl]);
                dnl = dnl + 4;
                dnu = dnu + 4;
                cl = cl + 2;
                cu = cu + 2;
                ++gu;
                ++gl;
                ++pl;
        }
        for (y=4;y<14;++y)
        {
                printinfo(gradrec.courseno,dnl,dnu,y,43);
                printinfo(gradrec.cograde,gl,gu,y,53);
                printinfo(gradrec.corcredit,cl,cu,y,61);
                printinfo(gradrec.cordate,dnl,dnu,y,68);
                mvcur(stdscr,0,y,75);
                printf("%c",gradrec.corprog[pl]);
                dnl = dnl + 4;
                dnu = dnu +4;
                cl = cl + 2;
                cu = cu + 2;
                ++gu;
                ++gl;
                ++pl;
        }
        for (y=18;y<22;++y)
        {
                printinfo(gradrec.ocorno, onl, onu,y,2);
                printinfo(gradrec.ocorname, ocl, ocu,y,13);
                printinfo(gradrec.ocorgrade, ogl, ogu,y,42);
                printinfo(gradrec.ocorcredit, ocrl, ocru,y,50);
                printinfo(gradrec.ocordate, odl, odu,y,57);
                mvcur(stdscr,0,y,65);
                printf("%c",gradrec.ocorprog[opl]);
                onl = onl + 8;
                onu = onu + 8;
```

```
                ocl = ocl + 25;
                ocu = ocu + 25;
                ++ogl;
                ++ogu;
                ocrl = ocrl + 2;
                ocru = ocru + 2;
                odu = odu + 4;
                odl = odl + 4;
                ++opl;
        }
}

prntg2() /** Displays student info on screen two. **/
{
        int y,sdl = 0.sdu = 4,pl = 0.pu = 1;

        printfield(gradrec.name.sizeof(gradrec.name),0,8);
        printfield(gradrec.adrs1.sizeof(gradrec.adrs1),2,1);
        printfield(gradrec.adrs2.sizeof(gradrec.adrs2),3.1);
        printfield(gradrec.adrs3.sizeof(gradrec.adrs3),4.1);
        printfield(gradrec.adrs4,sizeof(gradrec.adrs4),5.1);
        printfield(gradrec.usres,sizeof(gradrec.usres),2,52);
        printfield(gradrec.ks_res,sizeof(gradrec.ks_res),3.52);
        printfield(gradrec.gpa.sizeof(gradrec.gpa),5.44);
        printfield(gradrec.ss,sizeof(gradrec.ss),2,69);
        mvcur(stdscr,0.3.77);
        printf("%c",gradrec.sx);
        printfield(gradrec.phone,sizeof(gradrec.phone),4,70);
        printfield(gradrec.graddate.sizeof(gradrec.graddate),7.23);
        printfield(gradrec.ba bs,sizeof(gradrec.ba bs),7.45);
        printfield(gradrec.prof_adv.sizeof(gradrec.prof adv),9,30);
        printfield(gradrec.com mem1,sizeof(gradrec.com mem1),10,30);
        printfield(gradrec.com mem2,sizeof(gradrec.com mem2),11,30);
        printinfo(gradrec.proptitle,0,50.13.18);
        printinfo(gradrec.proptitle.50,100,14,18);
        printfield(gradrec.ap_prop_date.sizeof(gradrec.ap_prop_date),15,18);
        printfield(gradrec.ms_opt.sizeof(gradrec.ms_opt),17.13);
        printfield(gradrec.toef.sizeof(gradrec.toef),19,15);
        for (y=17;y<21;++y)
        {
                printinfo(gradrec.status,sdl.sdu,y.52);
                printinfo(gradrec.probat.pl.pu,y.63);
                printinfo(gradrec.statdate.sdl.sdu,y.70);
                sdl = sdl + 4;
                sdu = sdu + 4;
                ++pl;
                ++pu;
        }
}

prntg3()  /** Displays student info on screen three. **/
{
        int   y,jump,x,ctl = 0.ctu = 4,qtl = 0.qtu = 3.exl = 0.exu = 11;
        int   dnl = 0. dnu = 4, gl = 0. gu = 1;

        printfield(gradrec.name. sizeof(gradrec.name),0,10);
```

```
        for (y=3;y<20;++y)
        {
                printinfo(gradrec.courseno.dnl.dnu,y.0);
                dnl = dnl + 4;
                dnu = dnu + 4;
        }
        for (y=3;y<20;++y)
        {
                printinfo(gradrec.cograde.gl.gu,y.5);
                ++gl;
                ++gu;
        }
        dnl = 0;
        dnu = 4;
        for (y=3;y<20;++y)
        {
                printinfo(gradrec.cordate.dnl.dnu,y.8);
                dnl = dnl + 4;
                dnu = dnu + 4;
        }
        for (y=3;y<20;++y)
        {
                x = 16;
                while ( x < 23)
                {
                        printinfo(gradrec.crtaught.ctl.ctu,y.x);
                        ctl = ctl + 4;
                        ctu = ctu + 4;
                        x = x + 6;
                }
        }
        x = 35;
        while (x < 51)
        {
                for (y=4;y<20;++y)
                {
                        printinfo(gradrec.crteach. qtl. qtu,y.x);
                        qtl = qtl + 3;
                        qtu = qtu + 3;
                }
                x = x + 15;
        }
        for (y=4;y<20;++y)
        {
                printinfo(gradrec.exp, exl. exu,y.61);
                exl = exl + 11;
                exu = exu + 11;
        }
        printfield(gradrec.t effect.sizeof(gradrec.t effect),20,25);
        printfield(gradrec.ung_adv.sizeof(gradrec.ung_adv),20,71);
}

printinfo(item,left,right.ycoord,xcoord) /** Prints out array segments **/
char item[];                             /** to terminal. **/
int  left,right,ycoord,xcoord;
{
```

```
        mvcur(stdscr,0,ycoord,xcoord);
        while (left < right)
        {
                putc(item[left], stdout);
                ++left;
        }
}

printfield(item.len,ycoord,xcoord)      /** Prints out structure field **/
char item[];                            /** to terminal. **/
int len,ycoord,xcoord;
{
        int i = 0;

        mvcur(stdscr,0,ycoord,xcoord);
        while (i < len)
        {
                putc(item[i], stdout);
                ++i;
        }
}

typo()  /** Error message. **/
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

die()  /** External interference to interrupt execution. **/
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-8

"report.c"

```
/*****************************************************************************/
/*****************************> report.c <*************************************/
/** Allows user to print out various letters and reports.                   **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>

main()
{
        int     die();
        char    ch;

        initscr();
        signal(SIGINT.die);
        crmode();
        system("/usr/scott/xcode/screen 1"); /** Information screen. **/
        mvcur(stdscr,0,20.68);
        ch = getch();
        if ((ch == '\r') || (ch == '\n'))
            report();
        endwin();
}


report() /* Displays menu of letter options to user and calls program formltr*/
{
        char     c;
        int      cpyquan = 0.i.type = 0.done = 0;

        while (done == 0)
        {
                if (type == 0)
                    system("/usr/scott/xcode/screen m");
                mvcur(stdscr,0,22.71);
                c = getch();
                switch(c)
                {
                case 'a' :
                case 'A' :
                        {
                                system("/usr/scott/xcode/formltr a");
                                type = 0;
                                break;
                        }
                case 'b' :
                case 'B' :
                        {
                                system("/usr/scott/xcode/formltr b");
                                type = 0;
                                break;
                        }
                case 'c' :
                case 'C' :
```

```
                {
                        system("/usr/scott/xcode/formltr c");
                        prntltr();
                        type = 0;
                        break;
                }
        case 'd' :
        case 'D' :
                {
                        system("/usr/scott/xcode/formltr d");
                        prntltr();
                        type = 0;
                        break;
                }
        case 'e' :
        case 'E' :
                {
                        system("/usr/scott/xcode/formltr e");
                        type = 0;
                        break;
                }
        case 'f' :
        case 'F' :
                {
                        system("/usr/scott/xcode/formltr f");
                        type = 0;
                        break;
                }
        case 'g' :
        case 'G' :
                {
                        system("/usr/scott/xcode/formltr g");
                        type = 0;
                        break;
                }
        case 'h' :
        case 'H' :
                {
                        system("/usr/scott/xcode/formltr h");
                        type = 0;
                        break;
                }
        case 'i':
        case 'I':
                {
                        system("/usr/scott/xcode/gradlist >gscfile");
                        cpyquan = cpynum();
                        for (i=0;i < cpyquan;++i)
                                system("pr gscfile|lpr");
                        type = 0;
                        break;
                }
        case 'j':
        case 'J':
                {
                        system("/usr/scott/xcode/card >crdfile");
```

```
                                cpyquan = cpynum();
                                for (i=0;i < cpyquan;++i)
                                      system("pr crdfile|lpr");
                                type = 0;
                                break;
                        }
                case 'm':
                case 'M':
                        {
                                done = 1;
                                break;
                        }
                default :
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}

cpynum() /** Allows user to choose number of report copies. **/
{
  char string[2],*number;

  clear();
  refresh();
  mvcur(stdscr,0,10,23);
  printf("PLEASE ENTER THE NUMBER OF COPIES ");
  mvcur(stdscr,0,11,29);
  printf("DESIRED HERE ===> {  }");
  mvcur(stdscr,0,11,48);
  number = atoi(gets(string));
  return(number);
}

prntltr() /** Allows choice of printer and begins printing. **/
{
 int done = 0;
 char ch;

  clear();
  refresh();
  mvcur(stdscr,0,5,11);
  printf("PLEASE CHOOSE THE PRINTER FOR THE LETTERS TO BE PRINTED ON");
  mvcur(stdscr,0,8,9);
  printf("PLEASE ENTER AN 's' FOR THE SPINWRITER, OR AN 'e' FOR THE EPSON");
  mvcur(stdscr,0,9,33);
  printf("HERE ===> { }");
 while (done == 0)
 {
  mvcur(stdscr,0,9,44);
  ch = getch();
  switch(ch)
  {
```

```
    case 's': { system("cat ltrfile|lpr"); done = 1; break; }
    case 'e': { system("cat ltrfile|eps1");done = 1;  break;}
    case '1': { system("cat ltrfile|eps2");done = 1;break;}/*for personal use */
    case '2': { system("cat ltrfile|dec");done = 1;break;}/* for personal use */
    default:  { typo(); break; }
  }
 }
}

typo() /** Error message. **/
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

die() /** External interrupt for user. **/
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-9

"crdlst.c"

```
/*************************************************************************/
/*****************************> crdlst.c <*********************************/
/** This program enables a user to print an alphabetical listing of the **/
/** applicant database.                                                 **/
/*************************************************************************/

#include    <curses.h>
#include    <stdio.h>
#include    <signal.h>
#include    <ctype.h>
#include    "/usr/scott/structs/appstrt.h"

struct apprec   /** Information to be contained in each node of linked list */
{
  char  nam[40];
  char  ss[9];
  char  ad1[25];
  char  ad2[25];
  char  ad3[25];
  char  ad4[25];
  char  status[11];
  char  rec[11];
  char  loc[4];
  char  lev[4];
  char  stdate[5];
  struct apprec *next;
};

typedef  struct apprec ALIST;

main()
{   /*** begin of main of cardlist ***/
  int die();

  initscr();
  signal(SIGINT.die);
  crmode();
  filelst();
  endwin();
 }

filelst()   /** Calls routines to make alpha list and formats report. **/
{  /** begin of filelst **/

  ALIST *head,*node;
  long  offset = 4;
  int   i;

 head = makelst();
 node = head->next;
 prntheading();
 while (node->next != NULL)
 {
   prntitem(node->nam,40);
   blank(3);
   prntitem(node->ss,9);
```

```
   blank(3);
   prntitem(node->ad1,25);
   blank(3);
   prntitem(node->loc,4);
   blank(3);
   prntitem(node->lev,4);
   blank(2);
   prntitem(node->stdate,5);
   blank(2);
   prntitem(node->rec,11);
   blank(2);
   prntitem(node->status,11);
   printf("\n");
   blank(55);
   prntitem(node->ad2,25);
   printf("\n");
   blank(55);
   prntitem(node->ad3,25);
   printf("\n");
   blank(55);
   prntitem(node->ad4,25);
   printf("\n");
   printf("\n");
   node = node->next;
 }  /** end while **/
}  /** end of filelst  **/

blank(num)  /** This routine adds blanks for formatting. **/
  int num;
{
  int i;
 for (i=0;i<num;++i)
     putchar(' ');
}

prntheading()
{   /** Routine to print out report heading. **/

  static char title[] = "COMPUTER SCIENCE DEPT. APPLICANT LISTING";
  static char key1[] = "RECOMMEND KEY ==>( ac = accept. pv = provisional. ";
  static char key2[] = "pb = probation. sp = special )";
  static char head1[] = "NAME";
  static char head2[] = "SSN";
  static char head3[] = "ADDRESS";
  static char head4[] = "LOC";
  static char head5[] = "LEVEL";
  static char head6[] = "START";
  static char head7[] = "RECOMMEND.";
  static char head8[] = "STATUS";
  int i;

 printf("%86s\n",title);
 printf("\n");
 printf("%76s",key1);
 printf("%s\n",key2);
 printf("\n");
```

```
 printf("%12s",head1);
 printf("%38s",head2);
 printf("%19s",head3);
 printf("%18s",head4);
 printf("%7s",head5);
 printf("%7s",head6);
 printf("%12s",head7);
 printf("%12s\n",head8);
 for (i=0;i<132;++i)
     putchar('=');
 printf("\n");
 printf("\n");
}

prntitem(item.len) /** Prints out an item of a certain length. **/
  char item[];
  int  len;
{
  int i;

  for (i=0;i<len;++i)
       putchar(item[i]);
}

makelst()  /** Constructs alphabetized linked list. **/
{  /** begin of makelst  **/

   ALIST  *node,*head,*nextnode,*prev,*tail;
   int  fd,i,n_rd,insert = 0;
   char  *tr;
   long offset = 0;

 if (((head = malloc(sizeof(ALIST))) != NULL) &&
     ((tail = malloc(sizeof(ALIST))) != NULL))
 {
    for (i=0;i<40;++i)
    {
        head->nam[i] = ' ';
        tail->nam[i] = 'Z';
    }
    head->next = tail;
    tail->next = NULL;
    nextnode = head;
 }
 if ((fd = open("/usr/scott/datafiles/dbase",0)) == -1)
 {
    printf("dbase FILE CANNOT BE OPENED");
    exit(0);
 }
 lseek(fd,offset,0);
 while ((n_rd = read(fd,&temprec,sizeof(struct temp_rec))) != 0)
 {
  insert = 0;
  if ((temprec.name[0] != '+') && (temprec.name[0] != ' '))
  {
   if ((node = malloc(sizeof(ALIST))) != NULL)
```

```
   {
       node = initnode(node);
       node = reccopy(node);
   }
  }
  else insert = 1;
   while (insert == 0)
   {
    if ((strncmp(node->nam,nextnode->nam,sizeof(node->nam))) > 0)
    {
       prev = nextnode;
       nextnode = nextnode->next;
    }
    else
    {
       node->next = prev->next;
       prev->next = node;
       insert = 1;
    }
   }  /** end of while insert **/
  nextnode = head;
  offset = offset + sizeof(struct temp_rec);
  lseek(fd,offset,0);
 } /** end of while read **/
 close(fd);
 return(head);
} /** end of makelst  **/

initnode(node) /** Initializes each node by calling initfield. **/
 ALIST *node;
{
  initfield(node->nam,40);
  initfield(node->ss,9);
  initfield(node->ad1,25);
  initfield(node->ad2,25);
  initfield(node->ad3,25);
  initfield(node->ad4,25);
  initfield(node->status,11);
  initfield(node->rec,11);
  initfield(node->loc,4);
  initfield(node->lev,4);
  initfield(node->stdate,5);
  return(node);
}

initfield(item,len) /** Initializes any item passed to it to blanks. **/
 char item[];
 int  len;
{
  int i;

  for (i=0;i<len;++i)
      item[i] = ' ';
}

reccopy(node)           /** Copies info out of the applicant record **/
```

```
  ALIST *node;          /** into the corresponding list node. **/
{  /** begin of reccopy  **/
   int i,j;

  strncpy(node->nam.temprec.name.sizeof(temprec.name));
  strncpy(node->ss,temprec.ssn,sizeof(temprec.ssn));
  strncpy(node->ad1.temprec.address1.sizeof(temprec.address1));
  strncpy(node->ad2,temprec.address2,sizeof(temprec.address2));
  strncpy(node->ad3.temprec.address3.sizeof(temprec.address3));
  strncpy(node->ad4,temprec.address4,sizeof(temprec.address4));
  strncpy(node->loc,temprec.location,sizeof(temprec.location));
  strncpy(node->lev,temprec.level.sizeof(temprec.level));
  strncpy(node->stdate,temprec.startdate.sizeof(temprec.startdate));
  if (temprec.accept != ' ')
  {
     node->rec[0] = 'a';
     node->rec[1] = 'c';
     node->rec[2] = '-';
     if (temprec.prov != ' ')
     {
        node->rec[3] = 'p';
        node->rec[4] = 'v';
        node->rec[5] = '-';
     }
     if (temprec.spec != ' ')
     {
        node->rec[6] = 's';
        node->rec[7] = 'p';
        node->rec[8] = '-';
     }
     if (temprec.prob != ' ')
     {
        node->rec[9] = 'p';
        node->rec[10] = 'b';
     }
  }
  else if (temprec.reject != ' ')
       {
         node->rec[0] = 'r'; node->rec[1] = 'e'; node->rec[2] = 'j';
         node->rec[3] = 'e'; node->rec[4] = 'c'; node->rec[5] = 't';
       }
  node->status[4] = '-';
  if (temprec.mislett[0] != ' ')
  {
     node->status[0] = 'm'; node->status[1] = 'i';
     node->status[2] = node->status[3] = 's';
     j = 5;
     for (i=0;i<6;++i)
     {
         node->status[j] = temprec.mislett[i];
         ++j;
     }
  }
  if (temprec.tocomm[0] != ' ')
  {
     node->status[0] = 'c';node->status[1] = 'o';
```

```
      node->status[2] = node->status[3] = 'm';
      j = 5;
      for (i=0;i<6;++i)
      {
        node->status[j] = temprec.tocomm[i];
        ++j;
      }
   }
   if (temprec.toxerox[0] != ' ')
   {
      node->status[0] = 'x';node->status[1] = 'e';
      node->status[2] = 'r';node->status[3] = 'o';
      j = 5;
      for (i=0;i<6;++i)
      {
        node->status[j] = temprec.toxerox[i];
        ++j;
      }
   }
   if (temprec.tograd[0] != ' ')
   {
      node->status[0] = 'g';node->status[1] = 'r';
      node->status[2] = 'a';node->status[3] = 'd';
      j = 5;
      for (i=0;i<6;++i)
      {
        node->status[j] = temprec.tograd[i];
        ++j;
      }
   }
   if (temprec.statlett[0] != ' ')
   {
      node->status[0] = 's';node->status[2] = 'a';
      node->status[1] = node->status[3] = 't';
      j = 5;
      for (i=0;i<6;++i)
      {
        node->status[j] = temprec.statlett[i];
        ++j;
      }
   }
   return(node);
 }  /**  end of reccopy  **/

die()
{
   signal(SIGINT,SIG_IGN);
   endwin();
   exit(0);
}
```

Appendix A-10

"gsclst.c"

```
/*************************************************************************/
/*****************************> gsclst.c <*********************************/
/** Allows user to print an alphabetized list of all grad students for */
/** the Graduate Studies Committee.                                    */
/*************************************************************************/

#include    <stdio.h>
#include    <signal.h>
#include    "/usr/scott/structs/grstrt.h"

struct grrec /** Temp structure for report info. **/
{
  char  nam[40];    /* student's name                       */
  char  ga[4];      /* student's gpa                        */
  char  pradv[30];  /* student's advisor or major advisor   */
  char  pos;        /* prog. of study filed or not          */
  char  frstprg[4]; /* first course taken on prog of study */
  char  lstcrs[4];  /* last course taken on prog of study   */
  char  stat[3];    /* student finished,English,need pr */
  struct grrec *next;
};

typedef  struct grrec GLIST;

main()
{   /*** begin of main of cardlist ***/
  int die();

  signal(SIGINT.die);
  filelst();
 }

filelst()  /** Calls routines to construct linked list and formats report. **/
{  /** begin of filelst **/

  GLIST *head,*node;

 head = makelst();
 node = head->next;
 prntheading();
 while (node->next != NULL)
 {
   prntitem(node->nam.35);
   blank(4);
   prntitem(node->ga,4);
   blank(4);
   prntitem(node->pradv,21);
   blank(5);
   printf("%c",node->pos);
   blank(10);
   prntitem(node->frstprg,4);
   blank(12);
   prntitem(node->lstcrs,4);
   blank(12);
   prntitem(node->stat.3);
   printf("\n");
```

```
   node = node->next;
 }  /** end while **/
}  /** end of filelst  **/

blank(num) /** Prints out blank for formatting. **/
  int num;
{
  int i;
 for (i=0;i<num;++i)
     putchar(' ');
}

prntheading() /** Prints out report heading. **/
{
  static char title[] = "COMPUTER SCIENCE DEPT. GRADUATE STUDIES LISTING";
  static char key1[] = "STATUS KEY: * = finished; ~ = need proposal; ";
  static char key2[] = "E = English deficiency;";
  static char head1[] = "NAME";
  static char head2[] = "GPA";
  static char head3[] = "ADVISOR/MAJOR ADVISOR";
  static char head4[] = "PRGM OF";
  static char head5[] = "DATE FIRST PRGM";
  static char head6[] = "COURSE WAS TAKEN";
  static char head7[] = "DATE LAST PRGM";
  static char head8[] = "STATUS";
  static char head9[] = "STUDY";
  static char head10[] = "PROGRESS";
  int i;
 printf("%89s\n",title);
 printf("%87s",key1);
 printf("%s\n",key2);
 printf("\n");
 printf("\n");
 printf("%12s",head1);
 printf("%30s",head2);
 printf("%26s",head3);
 printf("%9s",head4);
 printf("%17s",head5);
 printf("%16s",head7);
 printf("%10s",head8);
 printf("%11s\n",head10);
 printf("%76s",head9);
 printf("%18s",head6);
 printf("%18s\n",head6);
 for (i=0;i<132;++i)
     putchar('*');
 printf("\n");
 printf("\n");
}

prntitem(item,len) /** Prints out item sent to it. **/
  char item[];
  int  len;
{
  int i;
```

```
  for (i=0;i<len;++i)
       putchar(item[i]);
}

makelst() /** Constructs linked list using GLIST nodes. **/
{  /** begin of makelst  **/

   GLIST  *node,*head,*nextnode.*prev,*tail;
   int  fd,i,n_rd,insert = 0;
   long offset = 0;

 if (((head = malloc(sizeof(GLIST))) != NULL) &&
     ((tail = malloc(sizeof(GLIST))) != NULL))
 {
    for (i=0;i<40;++i)
    {
        head->nam[i] = ' ';
        tail->nam[i] = 'Z';
    }
    head->next = tail;
    tail->next = NULL;
    nextnode = head;
 }
 if ((fd = open("/usr/scott/datafiles/gradbase",0)) == -1)
 {
    printf("gradbase FILE CANNOT BE OPENED");
    exit(0);
 }
 lseek(fd,offset,0);
 while ((n_rd = read(fd,&gradrec,sizeof(struct grad_rec))) != 0)
 {
  insert = 0;
  if ((gradrec.name[0] != '+') && (gradrec.name[0] != ' '))
  {
   if ((node = malloc(sizeof(GLIST))) != NULL)
   {
       node = initnode(node);
       node = reccopy(node);
   }
  }
  else insert = 1;
   while (insert == 0)
   {
    if ((strncmp(node->nam.nextnode->nam.sizeof(node->nam))) > 0)
    {
       prev = nextnode;
       nextnode = nextnode->next;
    }
    else
    {
       node->next = prev->next;
       prev->next = node;
       insert = 1;
    }
   }  /** end of while insert **/
  nextnode = head;
```

```
  offset = offset + sizeof(struct grad_rec);
  lseek(fd,offset,0);
 } /** end of while read **/
 close(fd);
 return(head);
} /** end of makelst  **/

initnode(node) /** Initializes each node as it is allocated. **/
 GLIST *node;
{
  initfield(node->nam,40);
  initfield(node->ga,4);
  initfield(node->pradv,30);
  node->pos = ' ';
  initfield(node->frstprg,4);
  initfield(node->lstcrs,4);
  initfield(node->stat,8);
  return(node);
}

initfield(item,len) /** Inits. field passed to it to blanks. **/
 char item[];
 int  len;
{
   int i;

  for (i=0;i<len;++i)
      item[i] = ' ';
}

reccopy(node) /** Copies each student's record info to list node. **/
  GLIST *node;
{  /** begin of reccopy  **/
  int i,j,prop = 0,k = 0,prgcnt = 0;
  char tdate[3],ldate[4],cdate[4],fdate[4];

  initfield(cdate,4);
  initfield(ldate,4);
  tdate[0] = tdate[2] = '5';
  tdate[1] = '7';
  ldate[2] = ldate[3] = '0';
  fdate[2] = fdate[3] = '9';
  fdate[0] = fdate[1] = 'Z';
  strncpy(node->nam,gradrec.name,sizeof(gradrec.name));
  strncpy(node->ga,gradrec.gpa,sizeof(gradrec.gpa));
  strncpy(node->pradv,gradrec.prof_adv,sizeof(gradrec.prof adv));
  for (i=0;i<20;++i)
  {
       if ((gradrec.corprog[i] = 'y') || (gradrec.corprog[i] = 'Y'))
       {
            ++prgcnt;
            for (j=0;j<4;++j)
            {
                 cdate[j] = gradrec.cordate[k];
                 ++k;
            }
```

```
            if (cdate[2] == ldate[2])
            {
               if (cdate[3] > ldate [3])
                  strncpy(ldate.cdate.4);
               else if (cdate[3] == ldate[3])
                    {
                         if ((cdate[0] == 'f') || (cdate[0] == 'F') ||
                             (cdate[1] == 'f') || (cdate[1] == 'F'))
                               strncpy(ldate.cdate.4);
                         else if ((strncmp(ldate.cdate.4)) >= 0)
                                   strncpy(ldate,cdate.4);
                    }
            }
            else if (cdate[2] > ldate[2])
                     strncpy(ldate,cdate.4);
            if ((cdate[0] != ' ') || (cdate[1] != ' '))
            {
            if (cdate[2] == fdate[2])
            {
               if (cdate[3] < fdate[3])
               {
                   printf("%c",cdate[3]);
                   strncpy(fdate.cdate.4);
               }
               else if (cdate[3] == fdate[3])
                    {
                         printf("%c",cdate[3]);
                         if ((cdate[0] != 'f') && (cdate[0] != 'F') ||
                             (cdate[1] != 'f') && (cdate[1] != 'F'))
                         {
                              if (cdate[1] <= fdate[1])
                                   strncpy(fdate.cdate.4);
                         }
                    }
             }
             else if (cdate[2] < fdate[2])
                      strncpy(fdate.cdate.4);
             }
        }
    }
    if (prgcnt > 0)
       node->pos = 'y';
    strncpy(node->lstcrs,ldate.4);
    strncpy(node->frstprg,fdate.4);
    for (i=0;i<6;++i)
    {
        if (gradrec.ap_prop_date[i] != ' ')
           ++prop;
    }
    if (prop > 0)
       node->stat[0] = '~';
    if (gradrec.toef[1] != ' ')
    {
         if ((strncmp(tdate.gradrec.toef.3)) > 0)
              node->stat[1] = 'E';
    }
```

```
  if ((strncmp(ldate.gradrec.graddate.4)) == 0)
     node->stat[2] = '*';
  return(node);
 } /** end of reccopy **/

die() /** Allows external interrupt by user. **/
{
   signal(SIGINT.SIG_IGN);
   exit(0);
}
```

Appendix A-11

"lttr.c"

```
/*****************************************************************************/
/*********************************> lttr.c <*********************************/
/** Program to pull information and execute nroff letter format. Output **/
/** goes to a file called ltrfile that will be printed out by report.c. **/
/*****************************************************************************/

#include        <curses.h>
#include        <stdio.h>
#include        <signal.h>
#include        <ctype.h>
#include        "/usr/scott/structs/appstrt.h"
#include        "/usr/scott/structs/grstrt.h"

struct ap_dates /** This structure contains information that will be **/
{               /** by some of the letters.                         **/
  char ltrdate[20];
  char semest[6];
  char enrdates[20];
  char clssdate[20];
  char sal[5];
  char wrkstrt[20];
  char wrkend[20];
} apdates;

struct ap_info /** This structure contains info neede from the databases. **/
{
  char nm[40];
  char s;
  char title[4];
  char ads1[25];
  char ads2[25];
  char ads3[25];
  char ads4[25];
  char recmark[5];
  char rec1[30];
  char rec2[30];
  char rec3[30];
  char schmark[5];
  char sch1[30];
  char sch2[30];
  char sch3[30];
  char sch4[30];
  char sch5[30];
  char toefmark[5];
} appinfo;

main(argc,argv)
 int argc;
 char *argv[];
{
        int     die();

        initscr();
        signal(SIGINT,die);
        crmode();
        pickltr(argv);
```

```
        endwin();
}

pickltr(argv) /** Coordinates sequence of events for each letter. **/
 char *argv[];
{
  int fd;

   if ((fd = creat("/usr/scott/xcode/ltrfile",0755)) == -1)
   {
       printf("THE MASTER LETTER FILE CANNOT BE CREATED AT THIS TIME");
       exit(0);
   }
   close(fd);
   initdates();
   switch (*argv[1])
   {
    case 'b':   /** acceptance letter for applicant. accept.nr **/
    case 'c':   /** acceptance letter - no support. admitnon.nr **/
    case 'd':   /** instate gta letter, instta.nr **/
    case 'e':   /** out-of-state gta letter. **/
    case 'g':   /** grad eval. letter-good **/
    case 'h':   /** grad eval. letter-fair **/
            {
                initdates();
                promptusr(argv);
                printem(argv);
                break;
            }
    case 'a':   /** missing letter for applicant. missing.nr **/
    case 'f':  /** rejection letter for applicant. reject.nr **/
            {
                promptusr(argv);
                recsrch(argv);
                break;
            }
   }
}

initdates() /** Initializes structure of dat info. to blanks. **/
{
  char *ptr;

  for (ptr = &apdates;ptr<&apdates + 1;++ptr)
          *ptr = ' ';
}

promptusr(argv) /** Prompts user to enter info needed within desired letter **/
 char *argv[];
{
 int  done = 0;
 char ch;

 while (done == 0)
 {
  switch(*argv[1])
```

```
 {
  case 'a':
  case 'f':
  case 'g':
  case 'h':
          {
              clear();
              refresh();
              mvcur(stdscr,0,10,13);
              printf("PLEASE ENTER DATE OF LETTER =>{                    }");
              mvcur(stdscr,0,11,15);
              printf("(ie., January 1. 1984)");
              getprmpt(apdates.ltrdate.10,44);
              done = 1;
              break;
          }
  case 'b':
  case 'c':
          {
              getdates();
              mvcur(stdscr,0,15,41); printf("NOT REQUIRED");
              mvcur(stdscr,0,17.46); printf("NOT REQUIRED");
              mvcur(stdscr,0.19,46); printf("NOT REQUIRED");
              break;
          }
  case 'd':
  case 'e':
          {
              getdates();
              getprmpt(apdates.sal.15,41);
              getprmpt(apdates.wrkstrt.17,46);
              getprmpt(apdates.wrkend,19,46);
              break;
          }
  default:
          {
              printf("REPORT GENERATOR HAS PROBLEMS\n");
              printf("PLEASE SEE THE SYSTEMS ADMINISTRATOR");
              done = 1;
              break;
          }
 }
 if (done != 1)
 {
     mvcur(stdscr,0,21,14);
     printf("HAS EVERYTHING BEEN ENTERED CORRECTLY ( Y/N ) ? ==>{ }");
     mvcur(stdscr,0,21.66);
     ch = getch();
     if ((ch == 'y') || (ch == 'Y'))
          done = 1;
 }
}
 mvcur(stdscr,0,22.26);
 printf("THANK YOU FOR YOUR ASSISTANCE\n");
 for (done = 75;done>0;--done)
 {
```

```
        mvcur(stdscr,0,0,done);
        printf("<==");
    }
    for (done = 0;done<24;++done)
    {
        mvcur(stdscr,0,done,0);
        printf("||");
    }
    for (done=0;done<76;++done)
    {
        mvcur(stdscr,0,23,done);
        printf("==>");
    }
}

getdates() /** Displays screen to enter dates on. **/
{
    system("/usr/scott/xcode/screen 5");
    getprmpt(apdates.ltrdate,6,44);
    getprmpt(apdates.semest,8,55);
    getprmpt(apdates.enrdates,10,46);
    getprmpt(apdates.clssdate,13,52);
}

getprmpt(item,ycoord,xcoord) /** Allows entry of date info. **/
 char item[];
 int  ycoord,xcoord;
{
  char  ch;
  int   i = 0;

  mvcur(stdscr,0,ycoord,xcoord);
  ch = getch();
  while ((ch != '\n') && (ch != '\r'))
  {
        if (ch != ')
        {
            item[i] = ch;
            ++i;
            ch = getch();
        }
        else if (ch == ')
        {
            --i;
            ch = getch();
        }
   }
}

printem(argv) /** Displays screen to enter names for letters. **/
 char *argv[];
{
        int     type = 0,done = 0;
        char    c;

        while (done == 0)
```

```
        {
                if (type == 0)
                    system("/usr/scott/xcode/screen p");
                mvcur(stdscr,0,21,71);
                c = getch();
                switch (c)
                {
                case 'm':
                case 'M':
                        {
                                done = 1;
                                break;
                        }
                case '\r':
                case '\n':
                        {
                                type = 0;
                                break;
                        }
                case 'c':
                case 'C':
                        {
                                type = 1;
                                prntltr(argv);
                                break;
                        }
                default:
                        {
                                typo();
                                type = 1;
                                break;
                        }
                }
        }
}

initstructs() /** Initializes structures to blanks. **/
{
        char *ap,*gr,*tr;
        for (tr= &temprec; tr < &temprec + 1;++tr)
                *tr = ' ';
        for (gr = &gradrec; gr < &gradrec + 1;++gr)
                *gr = ' ';
        for (ap = &appinfo;ap < &appinfo + 1;++ap)
                *ap = ' ';
}

prntltr(argv) /** Accepts entry of names. **/
 char *argv[];
{
        static char names[10][40];
        int  len[10],err = 0,y = 5,i,j,found[10],leng,done = 0;
        char ch;

        for (j=0;j<10;++j)
        {
```

```
        for (i=0;i<40;++i)
                names[j][i] = ' ';
        found[j] = 1;
}
while (done == 0)
{
        for (j=0;j<10;++j)
        {
                len[j] = (getname(names,j,leng,y,16));
                ++y;
        }
        mvcur(stdscr,0,10,74);
        ch = getch();
        switch (ch)
        {
        case '\r':
        case '\n':
        case '\t':
                {
                        done = 1;
                        break;
                }
        case 'P':
        case 'p':
                {
                        for (j=0;j<10;++j)
                        {
                               initstructs();
                               if (len[j] != 0)
                               {
                                found[j] = findcopy(names,j,len,argv);
                                if (found[j] != 0)
                                  mklttr(argv);
                               }
                        }
                        y = 5;
                        for (i=0;i<10;++i)
                        {
                                if (found[i] == 0)
                                {
                                        mvcur(0,0,y,0);
                                        printf("NOT FOUND ===>");
                                        err = 1;
                                }
                                ++y;
                        }
                        if (err == 1)
                        {
                                mvcur(0,0,17,17);
                                printf("IF NAME IS NOT FOUND, ");
                                printf("PLEASE CHECK ");
                                printf("TO SEE THAT");
                                mvcur(0,0,18,14);
                                printf("THE SPELLING AND FORMAT ");
                                printf("IS RIGHT.  ");
                                printf("THEN PLEASE REENTER");
```

```
                                }
                                done = 1;
                                break;
                        }
                default:
                        {
                                typo();
                                break;
                        }
                }
        }
}

getname(person,j,len,ycoord,xcoord) /** Ensures proper entry of names. **/
char person[10][40];
int  len,ycoord,xcoord;
{

        int done = 0, i = 0;
        char c,ch;

        len = 0;
        mvcur(stdscr,0,ycoord,xcoord);
        ch = getch();
        while (done == 0)
        {
                switch (ch)
                {
                case '\n':
                case '\r':
                case '\t':
                        {
                                done = 1;
                                break;
                        }
                case '':
                        {
                                --i;
                                ch = getch();
                                break;
                        }
                case '.':
                case ',':
                case '-':
                        {
                                person[j][i] = ch;
                                ++i;
                                ch = getch();
                                break;
                        }
                default:
                        {
                                if (ch != ' ')
                                {
                                        if (islower(ch) != 0)
                                        {
```

```
                                        c = toupper(ch);
                                        person[j][i] = c;
                                }
                                else person[j][i] = ch;
                                ++len;
                                ++i;
                                ch = getch();
                        }
                        else
                        {
                                while ((ch = getch()) == ' ');
                                if (ch != '\t')
                                {
                                        person[j][i] = ' ';
                                        ++i;
                                        if (islower(ch) != 0)
                                        {
                                                c = toupper(ch);
                                                person[j][i] = c;
                                        }
                                        else person[j][i] = ch;
                                        len = len + 2;
                                        ++i;
                                        ch = getch();
                                }
                        }
                        break;
                }
        }
    }
    return(len);
}

findcopy(person,j,len,argv) /* Finds database info for each name passed to it.*/
char person[10][40],*argv[];
int  j,len[];
{
    int fd, i, n_read, samestr = 1;
    long  offset = 0;
    char name[40],trname[40];

    switch(*argv[1])
    {
     case 'a':
     case 'b':
     case 'c':
     case 'd':
     case 'e':
            {
             if ((fd = open("/usr/scott/datafiles/dbase", 0)) == -1)
             {
                    printf("FILE DBASE CANNOT BE OPENED\n");
                    exit(0);
             }
             break;
            }
```

```
        case 'g':
        case 'h':
                {
                 if ((fd = open("/usr/scott/datafiles/gradbase", 0)) == -1)
                 {
                        printf("FILE GRADBASE CANNOT BE OPENED\n");
                        exit(0);
                 }
                 break;
                }
        default:
               {
                  printf("PLEASE SEE THE SYSTEMS ADMINISTRATOR");
                  break;
               }
       }
       for (i=0;i<40;++i)
       {
               name[i] = ' ';
               trname[i] = person[j][i];
       }
       while (samestr != 0)
       {
               lseek(fd, offset, 0);
               n_read = read(fd,name.len[j]);
               if ((samestr = strncmp(name.trname.len[j])) == 0)
               {
                       lseek(fd, offset, 0);
                       if ((*argv[1] == 'g') || (*argv[1] == 'h'))
                         n_read = read(fd, &gradrec, sizeof(struct grad_rec));
                       else
                         n_read = read(fd, &temprec, sizeof(struct temp_rec));
                       close(fd);
                       return(1);
               }
               else
               {
                       if (n_read != 0)
                       {
                           if ((*argv[1] == 'g') || (*argv[1] == 'h'))
                               offset = offset + sizeof(struct grad_rec);
                           else
                               offset = offset + sizeof(struct temp_rec);
                       }
                       else return(0);
               }
       }
       close(fd);
}

recsrch(argv)  /** This routine is used to search entire database for **/
 char *argv[]; /** missing letter or rejection letter to autoprint. **/
{
       int cnt = 0,fd,n_read;
       long  offset = 0;
```

```
        if ((fd = open("/usr/scott/datafiles/dbase", 0)) == -1)
        {
          printf("FILE DBASE CANNOT BE OPENED\n");
          exit(0);
        }
        lseek(fd,offset,0);
        while ((n_read = read(fd,&temprec,sizeof(struct temp_rec))) != 0)
        {
              if ((temprec.name[0] != '+') && (temprec.name[0] != ' '))
              {
                   if ((chckrec(argv)) >= 1)
                   {
                      ++cnt;
                      mklttr(argv);
                   }
              }
              initstructs();
        }
        close(fd);
}

chckrec(argv) /** Checks each database record for reject or miss. info. **/
 char *argv[];
{
  int  i,rvalue = 0,value,r1value = 0,r2value = 0,r3value = 0,svalue = 0;
  int  tvalue = 0,s1value = 0,s2value = 0,s3value = 0,s4value = 0,s5value = 0;
  int  rjval = 0;

  if (*argv[1] == 'f')
  {
   if (temprec.reject != ' ')
       ++rjval;
   value = rjval;
  }
  if (*argv[1] == 'a')
  {
      for (i=0;i<30;++i)
      {
          appinfo.rec1[i] = appinfo.rec2[i] = appinfo.rec3[i] = '_';
          appinfo.sch1[i] = appinfo.sch2[i] = appinfo.sch3[i] = '_';
          appinfo.sch4[i] = appinfo.sch5[i] = '_';
      }
      for (i=0;i<5;++i)
          appinfo.recmark[i] = appinfo.schmark[i] = appinfo.toefmark[i] = '_';
      r1value = chckinfo(temprec.recomone,temprec.reconedate,appinfo.rec1);
           /** first chck for presence of rec name. then recdate. **/
      r2value = chckinfo(temprec.recomtwo,temprec.rectwodate,appinfo.rec2);
      r3value = chckinfo(temprec.recomthree,temprec.recthreedate,appinfo.rec3);
      rvalue = r1value + r2value + r3value;
      if (rvalue > 0)
       appinfo.recmark[2] = 'X';
      s1value = chckinfo(temprec.schoolone,temprec.schonedate,appinfo.sch1);
      s2value = chckinfo(temprec.schooltwo,temprec.schtwodate,appinfo.sch2);
      s3value = chckinfo(temprec.schoolthree,temprec.schthreedate,appinfo.sch3);
      s4value = chckinfo(temprec.schoolfour,temprec.schfourdate,appinfo.sch4);
      s5value = chckinfo(temprec.schoolfive,temprec.schfivedate,appinfo.sch5);
```

```
      svalue = s1value + s2value + s3value + s4value + s5value;
      if (svalue > 0)
       appinfo.schmark[2] = 'X';
      if ((temprec.uscit[0] == 'n') || (temprec.uscit[0] == 'N') ||
          (temprec.uscit[1] == 'n') || (temprec.uscit[1] == 'N') ||
          (temprec.uscit[2] == 'n') || (temprec.uscit[2] == 'N'))
      {
              if ((temprec.toefl[0] == ' ') || (temprec.toefl[1] == ' '))
              {
                  ++tvalue;
                  appinfo.toefmark[2] = 'X';
              }
      }
      value = tvalue + rvalue + svalue;
 }
 return(value);
}

chckinfo(item1,item2,item3)  /** Called by chckrec to check for presence **/
 char item1[],item2[],item3[]; /** of information in field. **/
{
  int value = 0;
      if ((hasinfo(item1,sizeof(item1))) > 0)
      {
         if ((hasinfo(item2,sizeof(item2))) <= 1)
         {
              strncpy(item3.item1,sizeof(item1));
              ++value;
         }
      }
      return(value);
 }

hasinfo(item.size) /** Determines if field contains all blanks or not.**/
 char item[];
 int  size;
{
  int i,value = 0;
 for (i=0;i<size;++i)
 {
     if (item[i] != ' ')
         ++value;
 }
 return(value);
}

mklttr(argv)  /** Setups letters and runs through nroff. **/
 char *argv[];
{
  int i=3,j=4;

   exchng(argv);
   modname();
   while (i >= 0)
   {
        apdates.sal[j] = apdates.sal[i];
```

```
        --j;
        --i;
    }
    apdates.sal[0] = '$';
    appinfo.title[0] = 'M'; appinfo.title[2] = '.';
    if ((appinfo.s == 'm') || (appinfo.s == 'M'))
        appinfo.title[1] = 'r';
    else if ((appinfo.s == 'f') || (appinfo.s == 'F'))
             appinfo.title[1] = 's';
        else
        {
             appinfo.title[1] = '.';
             appinfo.title[2] = ' ';
        }
    mkfiles(argv);
    switch(*argv[1])
    {
     case 'a':{
      system("nroff /usr/scott/letters/missing.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'b':{
      system("nroff /usr/scott/letters/accept.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'c':{
      system("nroff /usr/scott/letters/admitnon.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'd':{
      system("nroff /usr/scott/letters/instta.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'e':{
      system("nroff /usr/scott/letters/outstta.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'f':{
      system("nroff /usr/scott/letters/reject.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'g':{
      system("nroff /usr/scott/letters/masterok.nr >>/usr/scott/xcode/ltrfile");
      break; }
     case 'h':{
      system("nroff /usr/scott/letters/status.nr >>/usr/scott/xcode/ltrfile");
      break; }
     default: {
       printf("PLEASE SEE THE SYSTEM SUPERVISOR. NROFF DOES NOT WORK");
       break; }
    }
}

exchng(argv) /** Exchanges info from temp data structures to lttr structs **/
 char *argv[];
{
  char *ap;

  if ((*argv[1] == 'g') || (*argv[1] == 'h'))
  {
     strncpy(appinfo.nm,gradrec.name.sizeof(gradrec.name));
     strncpy(appinfo.ads1,gradrec.adrs1,sizeof(gradrec.adrs1));
```

```
     strncpy(appinfo.ads2,gradrec.adrs2,sizeof(gradrec.adrs2));
     strncpy(appinfo.ads3,gradrec.adrs3,sizeof(gradrec.adrs3));
     strncpy(appinfo.ads4,gradrec.adrs4,sizeof(gradrec.adrs4));
     appinfo.s = gradrec.sx;
  }
  else
  {
     strncpy(appinfo.nm,temprec.name,sizeof(temprec.name));
     strncpy(appinfo.ads1,temprec.address1,sizeof(temprec.address1));
     strncpy(appinfo.ads2,temprec.address2,sizeof(temprec.address2));
     strncpy(appinfo.ads3,temprec.address3,sizeof(temprec.address3));
     strncpy(appinfo.ads4,temprec.address4,sizeof(temprec.address4));
     appinfo.s = temprec.sex;
  }
}

modname() /** Modifies name by deleting first and middle and coverting **/
{         /** last name to lower case. **/
 int  i;
 char name[40];

 for (i=0;i<40;++i)
      name[i] = ' ';
 i = 0;
 while ((appinfo.nm[i] == ' ') && (i != 40))
       ++i;
 name[i] = appinfo.nm[i];
 ++i;
 while ((appinfo.nm[i] != ',') && (i < 40))
 {
       if ((isupper(appinfo.nm[i])) != 0)
             name[i] = (tolower(appinfo.nm[i]));
       ++i;
 }
 strncpy(appinfo.nm,name.sizeof(name));
}

mkfiles(argv)  /** Writes info to files to be used by nroff letters. **/
 char *argv[];
{
   static char df[] = "/usr/scott/lttrinfo/datefile";
   static char af1[] = "/usr/scott/lttrinfo/ad1file";
   static char af2[] = "/usr/scott/lttrinfo/ad2file";
   static char af3[] = "/usr/scott/lttrinfo/ad3file";
   static char af4[] = "/usr/scott/lttrinfo/ad4file";
   static char tf[] = "/usr/scott/lttrinfo/titlefile";
   static char nf[] = "/usr/scott/lttrinfo/namefile";
   static char sf[] = "/usr/scott/lttrinfo/semester";
   static char ef[] = "/usr/scott/lttrinfo/enrdates";
   static char cf[] = "/usr/scott/lttrinfo/classdate";
   static char sa[] = "/usr/scott/lttrinfo/salary";
   static char ws[] = "/usr/scott/lttrinfo/workstrt";
   static char we[] = "/usr/scott/lttrinfo/wrkend";
   static char r1[] = "/usr/scott/lttrinfo/rec1";
   static char r2[] = "/usr/scott/lttrinfo/rec2";
   static char r3[] = "/usr/scott/lttrinfo/rec3";
```

```
    static char s1[] = "/usr/scott/lttrinfo/sch1";
    static char s2[] = "/usr/scott/lttrinfo/sch2";
    static char s3[] = "/usr/scott/lttrinfo/sch3";
    static char s4[] = "/usr/scott/lttrinfo/sch4";
    static char s5[] = "/usr/scott/lttrinfo/sch5";
    static char rmk[] = "/usr/scott/lttrinfo/recmark";
    static char smk[] = "/usr/scott/lttrinfo/schmark";
    static char tk[] = "/usr/scott/lttrinfo/toefmark";

    writfile(apdates.ltrdate.df,sizeof(apdates.ltrdate));
    writfile(appinfo.ads1,af1,sizeof(appinfo.ads1));
    writfile(appinfo.ads2,af2,sizeof(appinfo.ads2));
    writfile(appinfo.ads3,af3,sizeof(appinfo.ads3));
    writfile(appinfo.ads4,af4,sizeof(appinfo.ads4));
    writfile(appinfo.title,tf.sizeof(appinfo.title));
    writfile(appinfo.nm,nf,sizeof(appinfo.nm));
    switch (*argv[1])
    {
     case 'a':{
              writfile(appinfo.rec1.r1.sizeof(appinfo.rec1));
              writfile(appinfo.rec2,r2,sizeof(appinfo.rec2));
              writfile(appinfo.rec3.r3.sizeof(appinfo.rec3));
              writfile(appinfo.sch1.s1,sizeof(appinfo.sch1));
              writfile(appinfo.sch2.s2,sizeof(appinfo.sch2));
              writfile(appinfo.sch3.s3.sizeof(appinfo.sch3));
              writfile(appinfo.sch4,s4,sizeof(appinfo.sch4));
              writfile(appinfo.sch5,s5,sizeof(appinfo.sch5));
              writfile(appinfo.recmark,rmk,sizeof(appinfo.recmark));
              writfile(appinfo.schmark,smk,sizeof(appinfo.schmark));
              writfile(appinfo.toefmark,tk,sizeof(appinfo.toefmark));
              break;
             }
     case 'b':
     case 'c':
     case 'd':
     case 'e':{
              writfile(apdates.semest.sf.sizeof(apdates.semest));
              writfile(apdates.enrdates.ef.sizeof(apdates.enrdates));
              writfile(apdates.clssdate.cf.sizeof(apdates.clssdate));
              writfile(apdates.sal.sa.sizeof(apdates.sal));
              writfile(apdates.wrkstrt.ws.sizeof(apdates.wrkstrt));
              writfile(apdates.wrkend,we.sizeof(apdates.wrkend));
              break;
              }
     default: { break; }
     }
}

writfile(item.fname.size) /** Creates lttrinfo file and writes info to it.*/
  char item[];
  char fname[];
  int  size;
{
   int i,fd,n_written;
   long offset = 0;
```

```
   if ((fd = creat(fname,0755)) == -1)
   {
        printf("CANNOT CREATE LETTER FILES AT THIS TIME");
        exit(0);
   }
   n_written = write(fd,item.size);
   n_written = write(fd,"\n",1);
   close(fd);
}

typo() /** Error message **/
{
        mvcur(0,0,23,10);
        printf("SORRY, THAT'S NOT QUITE RIGHT. DO TRY AGAIN!");
}

die() /** Allows external interrupt by user. **/
{
        signal(SIGINT,SIG_IGN);
        endwin();
        exit(0);
}
```

Appendix A-12

"temprec"

```
struct temp rec
{
        char    name[40];
        char    lstdate[28];
        char    address1[25];
        char    address2[25];
        char    address3[25];
        char    address4[25];
        char    ssn[9];
        char    sex;
        char    level[3];
        char    location[4];
        char    ksres[3];
        char    uscit[3];
        char    finassist[3];
        char    toefl[4];
        char    babs[30];
        char    procfee[3];
        char    ksuformdate[6];
        char    csformdate[6];
        char    recomone[30];
        char    reconedate[6];
        char    recomtwo[30];
        char    rectwodate[6];
        char    recomthree[30];
        char    recthreedate[6];
        char    schoolone[30];
        char    schonedate[6];
        char    schooltwo[30];
        char    schtwodate[6];
        char    schoolthree[30];
        char    schthreedate[6];
        char    schoolfour[30];
        char    schfourdate[6];
        char    schoolfive[30];
        char    schfivedate[6];
        char    no_of_trans[2];
        char    all trans_rec[3];
        char    mislett[6];
        char    tocomm[6];
        char    toxerox[6];
        char    tograd[6];
        char    statlett[6];
        char    gredate[6];
        char    lastcorrdate[6];
        char    accept;
        char    reject;
        char    prov;
        char    spec;
        char    prob;
        char    rej reas[30];
        char    recdate[6];
        char    startdate[5];

}
temprec;
```

Appendix A-13

"gradrec"

```
struct grad_rec
{
        char    name[40];
        char    lastdt[30];
        char    courseno[80];
        char    cograde[20];
        char    corcredit[40];
        char    cordate[80];
        char    corprog[20];
        char    ocorno[32];
        char    ocorname[100];
        char    ocorgrade[4];
        char    ocorcredit[8];
        char    ocordate[16];
        char    ocorprog[4];
        char    adrs1[25];
        char    adrs2[25];
        char    adrs3[25];
        char    adrs4[25];
        char    usres[3];
        char    ks_res[3];
        char    gpa[4];
        char    ss[9];
        char    sx;
        char    phone[8];
        char    graddate[4];
        char    ba_bs[30];
        char    prof_adv[30];
        char    com_mem1[30];
        char    com_mem2[30];
        char    proptitle[100];
        char    ap_prop_date[6];
        char    status[16];
        char    probat[4];
        char    statdate[16];
        char    ms_opt[4];
        char    toef[4];
        char    crtaught[136];
        char    crteach[96];
        char    exp[176];
        char    t_effect[4];
        char    ung_adv[3];
}
gradrec;
```

APPENDIX B

Transaction Processor Screens

Oct 5 18:16 1983 staffmenu Page 1

****** WELCOME TO THE FRIENDLY "KOOL RECORD SYSTEM" ******

This system allows the staff to keep and maintain records on HS applicants and students. A variety of reports and letters can be printed, also. So, please choose what you want to do by entering the desired option letter.

YOUR CHOICES ARE:

> KOOL RECORDS FOR HS APPLICANTS (a)

> KOOL RECORDS FOR HS STUDENTS (s)

> KOOL RECORDS FOR LETTER WRITING (l)

> TAKE A BREAK ($)

PLEASE ENTER YOUR CHOICE HERE ==> { }

Oct 5 18:16 1983 appmenu Page 1

****** WELCOME TO KOOL RECORDS FOR MS APPLICANTS ******

This feature allows you to track an MS applicant's progress through the applicant system. You can add a new applicant record to the MS applicant file drawer and you can also update an applicant's current record. You can also have the option to delete an applicant's file. Just choose an option and follow the directions from there.

HERE ARE YOUR OPTIONS:

> START NEW APPLICANT'S FILE (n)

> UPDATE APPLICANT'S RECORD (u)

> DESTROY APPLICANT'S RECORD (d)

> RETURN TO MAIN MENU (m)

PLEASE ENTER YOUR CHOICE HERE ==> [ ]

```
Oct  5 18:26 1983  sone Page 1

    NAME ( LAST, FIRST MIDDLE ) > {                                    }

    PERMANENT ADDRESS,          > {                    }
                                  {                    }
                                  {                    }
                                  {                    }

    SOCIAL SECURITY NUMBER > {         }    SEX ( M F ) > { }

    LEVEL ( MS PHD SP ) > {   }   LOCATION ( KSU KC LEAV NCR WE ) > {    }

    RESIDENT OF KANSAS ( YES NO ) > {   }

    CITIZEN OF USA ( YES NO ) > {   }

    NEED ASSISTANCE ( YES NO ) > {   }

    TOEFL SCORE > {    }  BA/BS AREA > {                          }

    PROCESSING FEE PAID ( YES NO ) > {   }

CURRENT SCREEN ( C ), MENU ( M ), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }
```

Oct 5 18:28 1983 stwo Page 1

NAME > { }

RECEIVED KSU FORM DATE > { }

RECEIVED CS FORM DATE > { }

RECOMMENDATIONS
NAME > { } DATE > { }
NAME > { } DATE > { }
NAME > { } DATE > { }

TRANSCRIPTS (TWO COPIES)
SCHOOL NAME > { } DATE > { }
SCHOOL NAME > { } DATE > { }
SCHOOL NAME > { } DATE > { }
SCHOOL NAME > { } DATE > { }
SCHOOL NAME > { } DATE > { }

NUMBER OF TRANSCRIPTS EXPECTED > { }

ALL TRANSCRIPTS RECEIVED (YES NO) > { }

CURRENT SCREEN (C), MENU (M), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }

Oct 5 16:28 1983 sthree Page 1

```
    NAME > {                                          }

    STATUS ( DATE ) > MISSING LETTER SENT > {       }
                      TO COMMITTEE > {      }
                      TO XEROX > {      }
                      TO GRAD SCHOOL > {      }
                      STATUS LETTER > {      }

    DATE GRE RECEIVED > {      } DATE OF LAST CORRESPONDANCE > {      }

    RECOMMENDATION : ACCEPT > { }       REJECT > { }
                     PROVISIONAL > { }  SPECIAL > { }
                     PROBATION > { }
                     REASON FOR REJECTION > {                          }

    DATE RECOMMENDATION RECEIVED ( mmddyy ) > {      }

    ENTRY DATE ( F-YR SP-YR SU-YR ) > {     }


CURRENT SCREEN ( C ), MENU ( M ), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }
```

Oct 5 16:16 1983 studmenu Page 1

****** WELCOME TO KOOL RECORDS FOR MS STUDENTS ******

This feature allows you to transfer students from the applicant file drawer to the student file drawer. You can also add to a student's record or delete a student's record. Just choose an option and then follow the directions from there.

HERE ARE YOUR OPTIONS:

> KOOL RECORD TRANSFER (t)

> UPDATE STUDENT'S RECORD (u)

> DESTROY STUDENT RECORD (d)

> RETURN TO MAIN MENU (m)

PLEASE ENTER YOUR CHOICE HERE ==> { }

```
Oct  6 09:09 1983  sfour Page 1

   NAME > {                                         }

  COURSE   GRADE   CREDIT   DATE   PROG * COURSE   GRADE   CREDIT   DATE   PROG
                                        *
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
  {     }  { }     {  }    {     } { } * {     }   { }     {  }    {     } { }
################################################################################
  OTHER COURSES :
   NUMBER          COURSE NAME            GRADE   CREDIT   DATE    PROG
 {        } {                          }  { }     {  }    {     }  { }
 {        } {                          }  { }     {  }    {     }  { }
 {        } {                          }  { }     {  }    {     }  { }
 {        } {                          }  { }     {  }    {     }  { }
CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ===> { }
```

```
Oct  5 16:26 1983  sfive Page 1


NAME > {                                                    }
       # ADDRESS #                  # CITIZENSHIP #                  # PERSONAL #
{                          }  -OF USA (YES NO)   > {   }      SSN > {          }
{                          }  -OF KANSAS (YES NO) > {   }     SEX ( M F ) > { }
{                          }                                  PHONE >{         }
{                          }  CURRENT GPA > {    }

EXPECTED GRAD. DATE > {     }   BA/BS AREA > {                          }

MAJOR PROFESSOR OR ADVISOR > {                          }
          COMMITTEE MEMBER > {                          }
          COMMITTEE MEMBER > {                          }

PROPOSAL TITLE > {                                                  }
                 {                                                  }
 DATE APPROVED > {       }
                                  STATUS HISTORY  >  STATUS  PROBATION  DATE
MS OPTION > {     }                                   {    }    { }    {    }
                                                      {    }    { }    {    }
TOEFL SCORE > {     }                                 {    }    { }    {    }
                                                      {    }    { }    {    }

CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ====>[ ]
```

```
Oct  5 16:28 1983  ssix Page 1

  NAME > {                                                    }

  COURSES TAUGHT  COURSES QUALIFIED TO TEACH     EXPERIENCE
   {    }{    }  FACULTY VIEW    STUDENT VIEW
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
   {    }{    }      {   }          {   }       {           }
TEACHER EFFECTIVENESS > {    }     UNDERGRADUATE ADVISING (YES NO ) > {   }

CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ===> { }
```

****

THE OPTION YOU HAVE CHOSEN ALLOWS YOU TO DESTROY A STUDENT'S OR APPLICANT'S RECORD. PLEASE RECONSIDER YOUR CHOICE. IF YOU WISH TO RETURN BACK TO THE SUB MENU PLEASE ENTER AN ' m ' HERE ==> { }. IF NOT, THEN TOUCH RETURN AND PROCEED AT YOUR OWN RISK!

****

Oct 5 18:16 1983 asmenu Page 1

****

THIS FEATURE WILL DESTROY THE FILES OF AN APPLICANT OR A REJECTED APPLICANT. PLEASE MAKE SURE THAT YOU KNOW WHAT YOU ARE DOING. LISTED BELOW ARE THREE OPTIONS THAT CAN BE CHOSEN. THE FIRST OPTION ALLOWS A SPECIFIC APPLICANT OR REJECTED TO BE DELETED. THE SECOND OPTION ALLOWS A GROUP OF REJECTED APPLICANTS TO BE DELETED PRIOR TO A SPECIFIC DATE. THE THIRD ALLOWS A RETURN TO THE MENU.

****

** DELETE SPECIFIC APPLICANTS OR REJECTED APPLICANTS (D)

** DELETE REJECTED APPLICANTS PRIOR TO A GIVEN DATE (R)

** RETURN TO THE MENU (M)

** PLEASE ENTER OPTION HERE ===> ( )

PLEASE ENTER THE NAMES OF THE PEOPLE WHOSE RECORDS YOU WANT TO DESTROY

NAMES (LAST, FIRST MIDDLE)

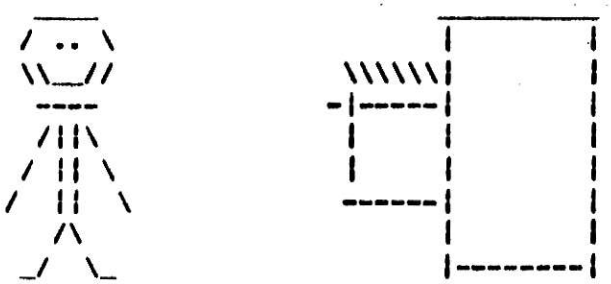
{ } Please Place an 'x' Here to Destroy the Records ==> { }

PLEASE ENTER 'c' FOR CURRENT SCREEN OR 'm' FOR MENU ==> { }

Oct 5 16:26 1983 copyscr Page 1

This feature allows you to transfer files from the applicant file drawer over to the graduate student file drawer. Simply touch return and enter the names of the students who you want to transfer. If you've decided to return to the menu enter an 'm' here ==> { } otherwise touch return and we will get started.

****



Oct 5 18:29 1983 totrans Page 1

PLEASE ENTER THE NAMES OF THE PEOPLE WHOSE RECORDS YOU WANT TO TRANSFER

NAMES (FORMAT: LAST, FIRST MIDDLE)

{ } Please Place a 't' Here to Transfer the Records ==> { }

PLEASE ENTER 'c' FOR CURRENT SCREEN OR 'm' FOR MENU =====> { }

****** WELCOME TO KOOL RECORDS FOR FACULTY PERUSAL ******

Hello! This is the faculty perusal facility. This function allows you, the faculty member, to look at a student's course information, teaching information, or other specific academic information. Simply touch return and an alphabetized name list will appear on the screen. If the name that you desire is not on this list then enter an 'n' for the next portion of the list to appear. When the name is found, follow the directions on the right side of the screen to "pull" this student's file.

The following are options that can be taken now:

* touch return for the perusal process to begin
* enter a '$' to exit the program *===> { }

```
Oct  5 16:26 1983  facper1 Page 1

    NAME > {                                        }

  COURSE   GRADE   CREDIT   DATE   PROG * COURSE   GRADE   CREDIT   DATE   PROG
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *
**************************************************************************************
  OTHER COURSES :
   NUMBER          COURSE NAME              GRADE   CREDIT   DATE   PROG
 {         } {                          } { }     { }    {    } {   }
 {         } {                          } { }     { }    {    } {   }
 {         } {                          } { }     { }    {    } {   }
 {         } {                          } { }     { }    {    } {   }
ENTER 'M' FOR MENU, 'A' FOR OTHER ACADEMIC INFO, OR 'T' FOR TEACHING INFO =>{ }
```

```
Oct  5 18:26 1983  fecper2 Page 1


NAME > {                                          }
       * ADDRESS *                 * CITIZENSHIP *                  * PERSONAL *
{                         }  -OF USA (YES NO)   > {   }      SSN > {          }
{                         }  -OF KANSAS (YES NO) > {   }     SEX ( M F ) > { }
{                         }                                  PHONE >{          }
{                         }  CURRENT GPA > {    }

EXPECTED GRAD. DATE > {    }   BA/BS AREA > {                              }

MAJOR PROFESSOR OR ADVISOR > {                          }
          COMMITTEE MEMBER > {                          }
          COMMITTEE MEMBER > {                          }

PROPOSAL TITLE > {                                                  }
                 {                                                  }
 DATE APPROVED > {       }
                                  STATUS HISTORY  >  STATUS  PROBATION  DATE
MS OPTION > {    }                                  {    }    { }    {    }
                                                    {    }    { }    {    }
TOEFL SCORE > {    }                                {    }    { }    {    }
                                                    {    }    { }    {    }

ENTER A 'C' FOR COURSE INFO, 'T' FOR TEACHING INFO, OR 'M' FOR MENU  =====> { }
```

Oct 5 18:26 1983 facper3 Page 1

NAME > { }

COURSES TAKEN COURSES TAUGHT COURSES QUALIFIED TO TEACH EXPERIENCE
                               FACULTY VIEW STUDENT VIEW

TEACHER EFFECTIVENESS > { } UNDERGRADUATE ADVISING (YES NO) > { }

ENTER 'M' FOR MENU, 'C' FOR COURSE INFO, OR 'A' FOR OTHER ACADEMIC INFO ===>{ }

****** WELCOME TO THE KOOL RECORDS SYSTEM REPORT GENERATOR ******

This feature allows you to print a letter to be sent to the student(s) or applicant(s) of your choice. Just choose the letter that you want to print from the menu on the next screen and follow the prompts from there. You will be required to enter information depending on which letter is to be printed such as enrollment dates, date of the letter, etc. The prompts should be self-explanatory.

The report generator will also allow you to print out a complete applicant database listing and a graduate student listing for the Graduate Studies Committee. So, please press the return key for the report menu or enter an 'm' to return to the KOOL Records System main menu *====> { }.

```
Oct 16 10:40 1983  ltrmenu Page 1

     HERE ARE YOUR CHOICES:

      > MISSING LETTER FOR APPLICANT ( a )

      > ACCEPTANCE LETTER FOR APPLICANT ( b )

      > ACCEPTANCE LETTER FOR APPLICANT (no support) ( c )

      > IN-STATE GTA LETTER ( d )

      > OUT-OF-STATE GTA LETTER ( e )

      > REJECTION LETTER FOR APPLICANT ( f )

      > YEARLY EVALUATION LETTER ( the good one )  ( g )

      > YEARLY EVALUATION LETTER ( the fair one )  ( h )

      > GSC EVALUATION LIST OF MS STUDENTS ( i )    {autoprint}

      > COPY OF APPLICANT CARD FOR OFFICE ( j )     {autoprint}

> RETURN TO THE KOOL RECORDS SYSTEM MAIN MENU ( m )     *==========> { }
```

The following information is required fo the letter(s) that will be sent out. Please enter the information just as you wish it to appear in the letter(s).

PLEASE ENTER DATE OF LETTER =>{ }

PLEASE ENTER SEMESTER (fall or spring) =>{ }

PLEASE ENTER ENROLLMENT DATES =>{ }
(ie., January 10-11, 1983)

PLEASE ENTER THE DATE THAT CLASSES BEGIN =>{ }

PLEASE ENTER GTA SALARY =>{$ }

PLEASE ENTER WORK START DATE => { }

PLEASE ENTER WORK ENDING DATE =>{ }

PLEASE ENTER THE NAMES OF THE PERSONS WHO YOU WOULD LIKE TO SEND THIS LETTER TO.

NAMES (LAST,FIRST,MIDDLE)

{ }
{ }
{ }
{ }
{ } PLACE A 'p' HERE
{ } TO PRINT ==> { }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }
{ }

CURRENT SCREEN (c), MENU (m), OR TOUCH RETURN FOR NEXT SCREEN ==> { }

APPENDIX C

"KOOL RECORDS SYSTEM" Letters

Oct 26 20:33 1983 missing.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
Your application for Graduate School in the Computer Science Department
is being delayed because we have not yet received all of your credentials.
Our Graduate Committee cannot review your application until all of your
credentials are received.
.sp
We show that we still have
.ul 1
not
received:
.sp
.in 5
__
.ul 1
.so /usr/scott/lttrinfo/recmark
__ 2 copies of an official transcript from:
.sp
.in 5
.ul 3
.so /usr/scott/lttrinfo/rec1
.so /usr/scott/lttrinfo/rec2
.so /usr/scott/lttrinfo/rec3
.sp
.in
_____ TOEFL score
_____ Financial statement
_____ Computer Science form
__
.ul 1
.so /usr/scott/lttrinfo/schmark
__ Letters of recommendation from:
.sp
```

```
.in 6
.so /usr/scott/lttrinfo/sch1
.so /usr/scott/lttrinfo/sch2
.so /usr/scott/lttrinfo/sch3
.so /usr/scott/lttrinfo/sch4
.so /usr/scott/lttrinfo/sch5
.sp
.in
_____ Statement of academic objectives
_____ Health form
.in
.sp
If you still wish your application to be considered for Graduate
School, please have these credentials sent as soon as possible.
If not, please sign the line below and mail it back to us.
.sp
I no longer wish to be considered for Graduate School.
.sp
.in 35
_____
.ti 16
Signature
.in
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger, Chairperson
Graduate Studies Committee
```

Appendix C-2

Oct 20 20:09 1983 accept.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
Welcome to K-State.  The Department Graduate Committee has recommended
to the Graduate School that your application be accepted.  You will
receive official notice from the Graduate School.
.sp
Enrollment for
.so /usr/scott/lttrinfo/semester
will be held
.so /usr/scott/lttrinfo/enrdates
and classes begin
.so /usr/scott/lttrinfo/classdate
\&.  There will be general advising sessions for new
graduate students scheduled during the enrollment period.
.sp
We are looking forward to having you in the Department.
.sp
Sincerely,
.sp3
.nf
Elizabeth A. Unger, Chairperson
Graduate Studies Committee
.sp
EAU:jsa
```

Oct 20 20:10 1983 admitnon.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
Welcome to K-State.  The Department Graduate Committee has recommended to
the Graduate School that your application be accepted.  You will
receive official notice from the Graduate School.
.sp
Enrollment for
.so /usr/scott/lttrinfo/semester
will be held
.so /usr/scott/lttrinfo/enrdates
and classes begin
.so /usr/scott/lttrinfo/classdate
\&. There will be general advising sessions for new graduate
students scheduled during the enrollment period.
.sp
We are not able to offer you support at this time.  However, it is
likely that you will be able to find support after having gained
experience in the Graduate Program.
.sp
We are looking forward to having you in the Department.
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger
Grad. Studies Committee
.sp
EAU:jsa
```

Appendix C-4

Oct 20 20:10 1983 instta.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.ls 1
.pa 11
.po 2
.wh -5 fo
.wh 0 hd
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.br
.sp
Welcome to K-State.  The Department Graduate Committee has recommended
to the Graduate School that your application be accepted.  You will
receive official notice from the Graduate School.
.sp
Enrollment for
.so /usr/scott/lttrinfo/semester
will be held
.so /usr/scott/lttrinfo/enrdates
and classes begin
.so /usr/scott/lttrinfo/classdate
\&.  There will be general advising sessions for new
graduate students scheduled during the enrollment period.
.sp
The Department is pleased to offer you a Graduate Teaching
Assistantship.  Your salary will be
.so /usr/scott/lttrinfo/salary
per month for four and a half months (
.so /usr/scott/lttrinfo/workstrt
thru
.so /usr/scott/lttrinfo/wrkend
) paid at the end of each month.  Salaries are reviewed annually.
Your duties will be:  to teach two programming language
classes (each class will be 2 hrs. lecture, 3 hrs. lab per week).
General organization, assignments, and
common examinations for the language classes are prepared by the course
director.
.sp
As a graduate assistant, you will be expected to:  carry nominally nine
(9) hours of credit courses each semester; to make satisfactory progress
```

towards your degree; and to attend Department seminars.
Later, there may be opportunity to teach other courses or to move
to a research position.
.sp
If you accept the offer, please sign the attached form and return it to
me as soon as possible, but no later than November 30, 1982. If you have
any question, please call. We are looking forward to having you in
the Department.
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger
Grad. Student Committee
.sp
EAU:jsa

Oct 21 10:27 1983 outstta.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.ls 1
.pa 11
.po 2
.wh -5 fo
.wh 0 hd
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.br
.sp
Welcome to K-State.  The Department Graduate Committee has recommended
to the Graduate School that your application be accepted.  You will
receive official notice from the Graduate School.
.sp
Enrollment for
.so /usr/scott/lttrinfo/semester
will be held
.so /usr/scott/lttrinfo/enrdates
and classes begin
.so /usr/scott/lttrinfo/classdate
\&.  There will be general advising sessions for new
graduate students scheduled during the enrollment period.
.sp
The Department is pleased to offer you a Graduate Teaching
Assistantship.  Your salary will be
.so /usr/scott/lttrinfo/salary
per month for four and a half months (
.so /usr/scott/lttrinfo/workstrt
thru
.so /usr/scott/lttrinfo/wrkend
) paid at the end of each month.  In addition, you will receive staff
privileges, including waiver of out-of-state tuition fees.
Salaries are reviewed annually.
Your duties will be:  to teach two programming language
classes (each class will be 2 hrs. lecture, 3 hrs. lab per week).
General organization, assignments, and
common examinations for the language classes are prepared by the course
director.
.sp
```

As a graduate assistant, you will be expected to: carry nominally nine (9) hours of credit courses each semester; to make satisfactory progress towards your degree; and to attend Department seminars.
Later, there may be opportunity to teach other courses or to move to a research position.
.sp
If you accept the offer, please sign the attached form and return it to me as soon as possible, but no later than November 30, 1982. If you have any question, please call. We are looking forward to having you in the Department.
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger
Grad. Student Committee
.sp
EAU:jsa

Oct 20 20:11 1983 reject.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
We have received your inquiry about graduate study in Computer Science.
.sp
Among our requirements for graduate work in Computer Science are that we
require a TOEFL of 575 or better, a B.S. in Computer Science or
equivalent experience, and at least a B+ grade average (80%-85%).
Furthermore, the applicant must demonstrate support of at least $9,000
per year.  After viewing your inquiry, we find that you have not met
these requirements.
.sp
Because of limitations of size and resources of our Department, we
cannot admit you.
.sp
Please accept our best wishes for success in your pursuit of graduate
studies.
.sp
Sincerely,
.sp 3
.nf
Elizabeth A. Unger, Chairperson
Graduate Studies Committee
.sp
EAU:jsa
```

Appendix C-7

Oct 20 20:11 1983 masterok.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp 2
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
Congratulations!  The Graduate Studies Committee, which includes
Drs. Wallentine and Hankley, has reviewed your progress as a
Master's Degree student and finds you are making good progress
toward that degree.  For a definition of normal progress see the
'Guidelines for Master's Degree in the Department of Computer
Science'.
.sp
The faculty has recommitted itself to excellence in teaching and
research.  Your record indicates you are capable of achieving
excellence in your work.  Thus we urge you to strive for excellence
in all of your academic pursuits especially in your Master's
research project.
.sp
Sincerely,
.sp3
.nf
E. A. Unger, Chairperson
Graduate Studies Committee
.sp
cc:  File
.in 5
Dr. Virgil E. Wallentine
```

Appendix C-8

Oct 22 08:57 1983 status.nr Page 1

```
.de hd
'sp 7
..
.de fo
'bp
..
.ll 72
.lt 72
.po 2
.wh 0 hd
.wh -5 fo
.nh
.nf
.so /usr/scott/lttrinfo/datefile
.sp 4
.so /usr/scott/lttrinfo/ad1file
.so /usr/scott/lttrinfo/ad2file
.so /usr/scott/lttrinfo/ad3file
.so /usr/scott/lttrinfo/ad4file
.sp
.fi
Dear
.so /usr/scott/lttrinfo/titlefile
.so /usr/scott/lttrinfo/namefile
.sp
The Graduate Studies Committee, consisting of Drs. Wallentine, Unger
and Hankley, has reviewed your progress as a Master's Degree student.
You need to accomplish the following to bring your performance to a
level commensurate with normal progress toward the degree:
.sp
.ti 5
_____ Need to file a Student Information form
.sp
.ti 5
_____ Need to select an advisor
.sp
.ti 5
_____ Need to select a major advisor
.sp
.ti 5
_____ Need to file a Program of Study
.sp
.ti 5
_____ Need to propose a research project
.sp
For a definition of normal progress see the "Guidelines for Master's
Degree in the Department of Computer Science".
.sp
The faculty has re-committed itself to excellence in teaching and
research.  Your record indicates that you are capable of achieving
excellence.  We urge you to continue striving for excellence in all
your academic pursuits, especially in your Master's research project.
.sp
Sincerely,
.sp 3
E. A. Unger, Chairperson
```

Graduate Studies Committee
.sp
EAU:jsa
.sp
cc: file

APPENDIX D

"KOOL RECORDS SYSTEM" Users Manual

"KOOL RECORDS SYSTEM" USERS GUIDE TO THE UNIVERSE"

by

J. S. Anderson

Department of Computer Science

Kansas State University

1983

TABLE OF CONTENTS

1.1 OVERVIEW

The "KOOL RECORDS SYSTEM" is a menu-driven system that was designed and implemented for use by the faculty and staff of Kansas State University's Department of Computer Science. This system was intended to allow monitoring of the Department's graduate applicant's and graduate student's progress through the Department's graduate program. The term "menu-driven" simply means that navigation is allowed through the system by presentation of a series of menus to the user, and allowing the user to choose which function he/she desires to perform. These menus provide most of the directions required to use the "KOOL RECORDS SYSTEM" correctly.

For the Department's staff, the "KOOL RECORDS SYSTEM" allows a single user to enter, delete, and peruse information on a graduate student or applicant. The system will "file" the information entered on each student/applicant into the appropriate database. The system will also allow retrieval of this information upon presentation of the student's/applicant's name to the system.

For the Department's faculty, the "KOOL RECORDS SYSTEM" will allow any faculty member to peruse a student's record. Some of the information that can be viewed by the faculty includes course information, teaching information, and pertinent academic information.

The "KOOL RECORDS SYSTEM" is relatively easy to use and has been deemed "user-friendly". There are, however, a few system idiosyncrasies that one should know before attempting to use the system. These "character-traits" will be covered in the next section.

Later sections will demonstrate sample scenarios as a user navigates through the various programs and functions of the "KOOL RECORDS SYSTEM".

1.2 SPECIAL CONSIDERATIONS

1.2.1 INTRODUCTION

The special considerations that are discussed here are few in number, but should be strictly followed in order for the "KOOL RECORDS SYSTEM" to be used properly and to ensure that the system's programmer and designer are not ridiculed and subsequently tracked down, tarred, and feathered. (Perhaps this is the time to discuss a disclaimer).

1.2.2 WHOSE NAME?

The name is the "key to the universe" as far as the "KOOL RECORDS SYSTEM" is concerned. If it is not entered in the correct format, either the user may find that he/she is viewing a record that he/she didn't want, or the system will state that it does not know anyone by that name. To ensure that either of these cases do not occur, the correct format

that should be explicitly followed is as follows: LASTNAME, FIRSTNAME MIDDLENAME. Yes, those are blanks between the comma and the first name and between the first name and middle name, with one blank only at each location (since this manual is being typed on the script program, more than one blank may appear). The name can be typed in either lowercase or uppercase letters, but the system will convert it to uppercase anyway. Also, it might be helpful to know that if the first or middle names are not known, the system will attempt to locate the desired record using just the last name. The system will compare only the letters entered with its list of names. If it finds a match, a record will be retrieved for perusal or modification. If the name does not match the system's list, the system will state that it does not have a record for that name. As always, the user should take great care to ensure that he/she has retrieved the correct record.

If the user is lucky enough to be a faculty member, he/she does not have to enter the name at all. A name list will be presented and the system will allow the faculty member to choose the desired name from this list. If that isn't user-friendly, what is?

1.2.2.1 NAME DUPLICITY

The decision about what to do if two or more different students or applicants have exactly the same name was most difficult to reach. It was assured by the Department's

staff that this problem rarely occurs, but the problem must still be addressed. The most critical point of this problem is to ensure that the two records do not get "mixed-up".

As an example, lets say we have a record on "Anderson, John Scott" in our applicant database. (It is realized that this name is not very common, but it will suffice for our example). Now, if we want to start a new record on a different "Anderson, John Scott" we have a problem. The solution is to slightly modify the second name so that it is not exactly the same as the first. To do this the user can simply type an asterisk as the first letter of the last name. This name would now look like this: "*Anderson, John Scott". If a third "Anderson, John Scott" is to be entered, it could take the following form: "**Anderson, John Scott", and etc. This procedure should keep each person's record separate. The user must remember, however, that the system will only recognize "*Anderson, John Scott" to be the valid owner of the second record entered, and not "Anderson, John Scott" or "**Anderson, John Scott". To aid the staff with this problem, a hard copy of all names should be printed from the databases at least once a week if not more often.

1.2.3 USER COMMANDS

The only thing that is required of the user here is to enter the correct single-letter command, and once again, the system will accept either uppercase or lowercase letters as valid input. If the correct command is not entered, but is

still recognized by the system, the user may find that he/she has entered an undesired area. If this occurs, the user can very simply navigate back to where he/she was. If the command is unrecognizable to the system, a message will appear stating that a mistake has been. The system will refrain from calling the user an idiot, as this is not very user-friendly.

1.2.4 LOSS OF INFORMATION

Each time that information is entered on a student or applicant, the input is not filed into the database until a menu is encountered. If the Department's computer should crash while information is being entered, that information will be lost. The original record will still be intact, but the user will have to reenter the information that was being entered at the time of the incident.

If a record is accidentally deleted using the system's delete function, the information may still be present in the database, if new records have not been added since the deletion procedure took place. If this should occur, please consult the database administrator for the appropriate action to take.

1.2.5 PASSWORDS

The Department's staff users will require a password to enter the "KOOL RECORDS SYSTEM". This password should be given to the system's programmer or the database

administrator so that access to the system can be ensured. Entry to the "KOOL RECORDS SYSTEM" can only be achieved in the following manner:

STEP 1.: Login to Unix as is normally done.

STEP 2.: Logon to the "KOOL RECORDS SYSTEM" by typing the word "kool".

STEP 3.: Enter the assigned password.

If the wrong password is typed or the correct password is mistyped, the system will allow the user a second chance. Exit from the system will occur if the second chance proves unsatisfactory. The faculty can simply type the same password as is used for Unix in step 3 above after typing the program name to initiate the perusal function.

2.1 THE APPLICANT MANAGEMENT PROGRAM

2.1.1 INTRODUCTION

This program of the "KOOL RECORDS SYSTEM" allows the Computer Science Department's staff to enter, modify, or peruse applicant information. A sample scenario of this program, including illustration of the actual screens, follows.

2.1.2 SAMPLE SCENARIO

After successful "logon" to Unix and the "KOOL RECORDS SYSTEM", the menu shown in Figure 2.1 will appear. To enter the applicant management program enter an "a" at this point. Once this is done, the "submenu" shown in Figure 2.2 will appear. To begin a new record enter an "n". The system will then ask you to enter the applicant's name. After the name has been entered one of two things can occur. The system could tell you that no record exists for this person. If this is the case, you will be given a choice of either continuing (by entering a "y" for yes) or going back to the applicant menu (by entering an "m" for menu). If , however, a record already exists for this name, make sure that you don't have a "name duplicity" problem as previously discussed (section 1.2.2.1). Also, if the applicant has been previously rejected, the name may appear along with a reason for the rejection. If this is the case, the return key must then be pressed to return to the applicant menu. If

Oct 5 18:16 1983 staffmenu Page 1

****** WELCOME TO THE FRIENDLY "KOOL RECORD SYSTEM" ******

This system allows the staff to keep and maintain records on HS applicants and students. A variety of reports and letters can be printed, also. So, please choose what you want to do by entering the desired option letter.

YOUR CHOICES ARE:

> KOOL RECORDS FOR HS APPLICANTS (a)

> KOOL RECORDS FOR HS STUDENTS (s)

> KOOL RECORDS FOR LETTER WRITING (l)

> TAKE A BREAK ($)

PLEASE ENTER YOUR CHOICE HERE ==> ( )

Figure 2.1

Oct 5 18:16 1983 appmenu Page 1

****** WELCOME TO KOOL RECORDS FOR HS APPLICANTS ******

This feature allows you to track an HS applicant's progress through the applicant system. You can add a new applicant record to the HS applicant file drawer and you can also update an applicant's current record. You can also have the option to delete an applicant's file. Just choose an option and follow the directions from there.

HERE ARE YOUR OPTIONS:

> START NEW APPLICANT'S FILE (n)

> UPDATE APPLICANT'S RECORD (u)

> DESTROY APPLICANT'S RECORD (d)

> RETURN TO MAIN MENU (m)

PLEASE ENTER YOUR CHOICE HERE ==> { }

Figure 2.2

an applicant record exists for the name entered, and you wish to continue anyway, or if you have entered a "y", the screen shown in Figure 2.3 will appear next. The cursor will appear in the lower right-hand corner of the screen prompting you to make a choice. At this point, you can enter an "m", which will allow you to return to the applicant menu, or you can enter a "c" which will allow you to input or change information on the current screen. (Also, you may notice that the name that was previously entered has magically appeared in the name field at the top of the screen. This is to prove that you are processing that person's record). If a "c" is entered here, the cursor will "jump" to the first field of this screen, which happens to be the address field. The applicant's address can now be entered. When the end of the address field has been reached, or you don't happen to have that information from the applicant yet, simply press the return key. This action will cause the cursor to jump to the next field allowing this field to be "filled" as if you were "filling out" a paper form. So, each time that the return key is pressed will cause the cursor to go to the next consecutive field. If you find that you've made a typing mistake (which will probably never happen), and you're still in the same field, simply backspace to the incorrect letter and make the correction. You will have to retype the rest of the field, however. If the mistake is not discovered until after you are out of the field, you must travel to the bottom of the screen and enter

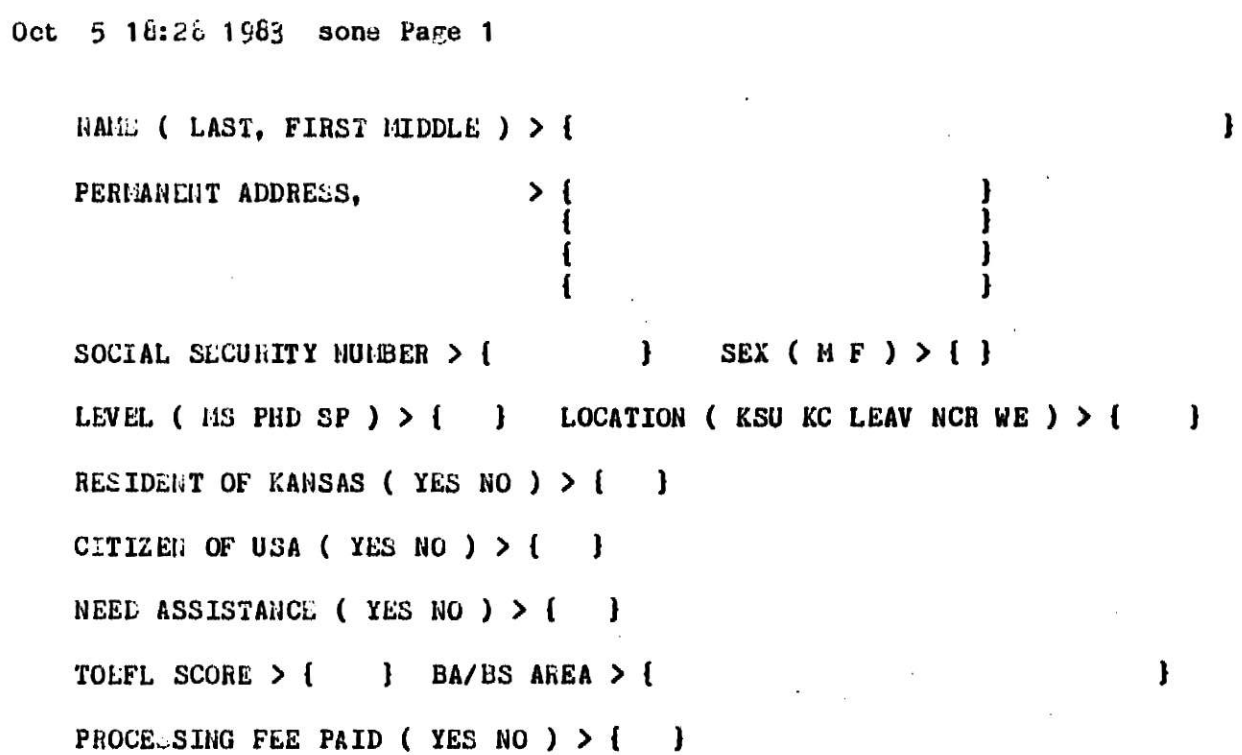

Oct 5 18:26 1983 sone Page 1
NAME (LAST, FIRST MIDDLE) > {
PERMANENT ADDRESS, > {
SOCIAL SECURITY NUMBER > { } SEX (M F) > { }
LEVEL (MS PHD SP) > { } LOCATION (KSU KC LEAV NCR WE) > { }
RESIDENT OF KANSAS (YES NO) > { }
CITIZEN OF USA (YES NO) > { }
NEED ASSISTANCE (YES NO) > { }
TOEFL SCORE > { } BA/BS AREA > {
PROCESSING FEE PAID (YES NO) > { }
CURRENT SCREEN (C), MENU (M), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }

Figure 2.3

a "c", and then navigate from the first field down to the field that contains the mistake.

When the information has been entered on the first screen and the cursor is back in the lower right-hand corner, an "m" can be entered for the menu or the space bar can be pressed for the next screen. If the space bar is pressed, the screen shown in Figure 2.4 will appear,and once aqain the applicant's name will magically appear in the name field at the top of the screen. As with the previous screen, a "c" can be entered to allow input on this screen, or an "m" can be entered, which will return you to the applicant menu, or the space bar can be pressed for the third and final applicant input screen. If a "c" is entered, input can be done as previously described. If an "m" is entered, any information that has been entered in this session will be permanently filed in the database. If the space bar is pressed, the screen shown in Figure 2.5 will next appear alonq with the applicant's name. The same choices are provided as on the previous two input screens. This screen contains, however, the power to reject an applicant. This can be done by enterinq an 'x' in the "REJECT" field. When this is done the cursor will move to the "REASON" field where a rejection reason must be entered.

Since this is the last input screen of the applicant management program, pressinq the space bar will qive the first input screen as shown in Figure 2.3. At this point it

```
Oct  5 16:28 1983  stwo Page 1

    NAME > {                                        }

    RECEIVED KSU FORM DATE > {       }

    RECEIVED CS FORM DATE > {       }

    RECOMMENDATIONS
       NAME > {                                  }  DATE > {       }
       NAME > {                                  }  DATE > {       }
       NAME > {                                  }  DATE > {       }

    TRANSCRIPTS ( TWO COPIES )
       SCHOOL NAME > {                                  }  DATE > {       }
       SCHOOL NAME > {                                  }  DATE > {       }
       SCHOOL NAME > {                                  }  DATE > {       }
       SCHOOL NAME > {                                  }  DATE > {       }
       SCHOOL NAME > {                                  }  DATE > {       }

    NUMBER OF TRANSCRIPTS EXPECTED > {  }

    ALL TRANSCRIPTS RECEIVED ( YES NO ) > {   }

CURRENT SCREEN ( C ), MENU ( M ), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }
```

Figure 2.4

```
NAME > {                                          }

STATUS ( DATE ) > MISSING LETTER SENT > {       }
                  TO COMMITTEE > {      }
                  TO XEROX > {      }
                  TO GRAD SCHOOL > {      }
                  STATUS LETTER > {      }

DATE GRE RECEIVED > {      } DATE OF LAST CORRESPONDANCE > {      }

RECOMMENDATION : ACCEPT > { }       REJECT > { }
                 PROVISIONAL > { }  SPECIAL > { }
                 PROBATION > { }
                 REASON FOR REJECTION > {                          }

DATE RECOMMENDATION RECEIVED ( mmddyy ) > {      }

ENTRY DATE ( F-YR SP-YR SU-YR ) > {     }

CURRENT SCREEN ( C ), MENU ( M ), OR PRESS SPACE BAR FOR NEXT SCREEN ==>{ }
```

Figure 2.5

may be noticed that the information entered previously has been printed out on the screen. This will prove that the information that has been entered has actually been saved. It is recommended, at this point, to peruse all information on the three screens to ensure that everything is correct. After this is done an "m" response will allow a return to the applicant menu. Once this is done, the input process can begin again by entering an "n". Also, a "u" can be entered if it is desired to update information on an already existent student. (Actually there is no difference between the "n" or the "u" response since both options will force the database to be searched for the name that was entered). If you happen to be looking at the applicant menu at this point, you may have noticed that there is an option that will allow deletion of an applicant record. Please do not take this option at this time. The deletion procedure will be discussed in section 4.1. The option to take here, then, is an "m" which will allow a return to the "KOOL RECORDS SYSTEM" main menu, as shown in Figure 2.1.

3.1 THE GRADUATE MANAGEMENT PROGRAM

3.1.1 INTRODUCTION

The graduate management program of the "KOOL RECORDS SYSTEM" allows a single user to enter, modify, delete, or peruse information on a Computer Science Department's masters student. This program works in much the same way as the applicant management program. There are a few differences, however, but this is due primarily to the format of the program's input screens. A sample scenario follows, along with illustrations of the program's screens.

3.1.2 SAMPLE SCENARIO

After a "logon" to the "KOOL RECORDS SYSTEM" or a return from the applicant management program, the main menu as shown in Figure 2.1 will appear. If the user has an extreme desire to take the graduate management program "out for a spin" then a user response of "s" is required here to initiate the appropriate program. When the "s" command has been entered, the menu, as shown in Figure 3.1 will appear. As with all menus, there are several choices to decide among. For example, an "m" can be entered which will allow a return to the main menu, or a "d" can be entered to allow files to be utterly destroyed (lets not do this just yet), or a "u" can be entered to update existing students' records. The last choice available is a "t" which will allow

Oct 5 16:16 1983 studmenu Page 1

****** WELCOME TO KOOL RECORDS FOR MS STUDENTS ******

This feature allows you to transfer students from the applicant file drawer to the student file drawer. You can also add to a student's record or delete a student's record. Just choose an option and then follow the directions from there.

HERE ARE YOUR OPTIONS:

> KOOL RECORD TRANSFER (t)

> UPDATE STUDENT'S RECORD (u)

> DESTROY STUDENT RECORD (d)

> RETURN TO MAIN MENU (m)

PLEASE ENTER YOUR CHOICE HERE ==> { }

Figure 3.1

information from the applicant database to be transferred to the graduate student database. This is to be done after a student has been accepted for graduate study. (Actual transfer may not occur until the student has "shown up", however). Since it is always difficult to make decisions, one will be made for you. So, enter an "s", NOW.

When an "s" is entered the program will ask you to enter a student's name so that his/her record may be pulled. After a name has been entered, the program in all of its infinite wisdom, may tell you that it doesn't have a record on this student. It will then ask you if you want a record started (by entering a "y") or if the whole thing is to be forgotten (by pressing return). Again a decision is required of the user. First, there may be a name duplicity problem, which must be checked out, or the student's record just does not exist. If a "y" is entered at this point, or if a record does exist, the screen as shown in Figure 3.2 will appear, and again, the student's name will magically appear at the top. As in the applicant program, the same choices are given at the bottom of the screen. Either an "m" can be entered to return to the menu, or the space bar can be pressed for the next screen, or a "c" can be entered to allow input of information on this screen. If a "c" is entered, the cursor will move to the top left-hand corner of the column labelled "COURSE". This screen, as you may have surmised, allows a student's course information to be entered, so, the course number is to be entered under the "COURSE" column. For

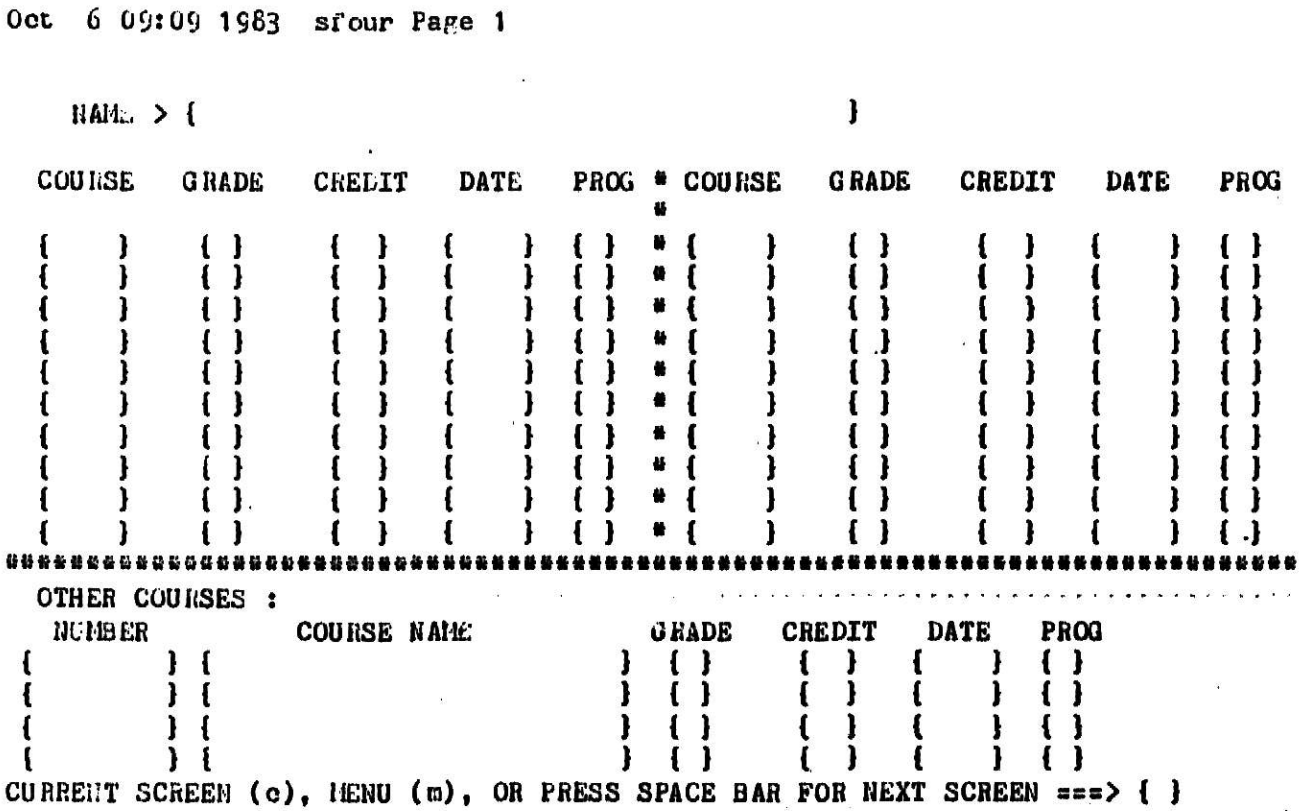

Oct 6 09:09 1983 sfour Page 1
NAME > { }
COURSE GRADE CREDIT DATE PROG
COURSE GRADE CREDIT DATE PROG
OTHER COURSES :
NUMBER COURSE NAME GRADE CREDIT DATE PROG
CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ===> { }

Figure 3.2

example, 700 or 720 can be entered for translator design, or operating systems, respectively. Also, there are four spaces available in this field and there are only three numbers in the course number. This isn't a mistake. The extra space is for code letters. For example, an "E" can be entered to designate that equivalent credit is to be given for a course that was taken at another institution. This could appear as "700E" or "E700". Also, a "D" can be entered to show that the course is a deficiency course which the student must take before regular enrollment status is allowed. After, a course number has been entered the return key can be pressed to move the cursor to the next field which is labelled "GRADE". This is where the course grade is to be entered. Pressing return again will move the cursor to the "CREDIT" column, then the "DATE" column, and finally the "PROG" column (three "returns" in all). The "CREDIT" column is to contain the course credit (ie., 03 for three hours credit). The "DATE" column is to contain the date that the course was taken (ie., SP81, SU81, or F81). The last column labelled "PROG" is to contain the information stating that the course is either on the program of study, or is not on the program of study (ie., "Y" or "N"). This screen also has columns available to contain information on other courses taken outside of the Computer Science Department. These columns are under the heading "OTHER COURSES". These columns allow the entire course number to be entered along with the course name. The name may have to be abbreviated to

"fit" within the brackets, however. Once again, pressing return will move the cursor to the next consecutive field. An added feature is provided for this screen. If the cursor is at the top of the left-hand "COURSE" column, pressing the tab key will move the cursor to the top of the right-hand "COURSE" column. Pressing the tab again will move the cursor to the "NUMBER" column under "OTHER COURSES". If the tab key is pressed again, the cursor will move to the bottom of the screen to allow a command to be entered.

Pressing the space bar at this point will cause the next screen to appear. This screen is shown in Figure 3.3. As before, choices can be made to either return to the menu or to navigate to the third and last input screen. Also, a "c" can be entered to allow input on this screen. If input is desired, a "c" response will move the cursor to the "PHONE" field. Pressing the return key will move the cursor to the next consecutive field as before. If input is not desired, the space bar can be pressed to allow the third input screen to appear. This screen can be seen in Figure 3.4.

The last input screen allows the user to enter "teaching" information. Information concerning working experience can also be entered. To input information at this point, simply enter a "c". This will move the cursor to the top left-hand column under the "COURSES TAUGHT" heading. This column is to contain the course numbers of classes that were taught by the student. The column to the immediate right is to contain

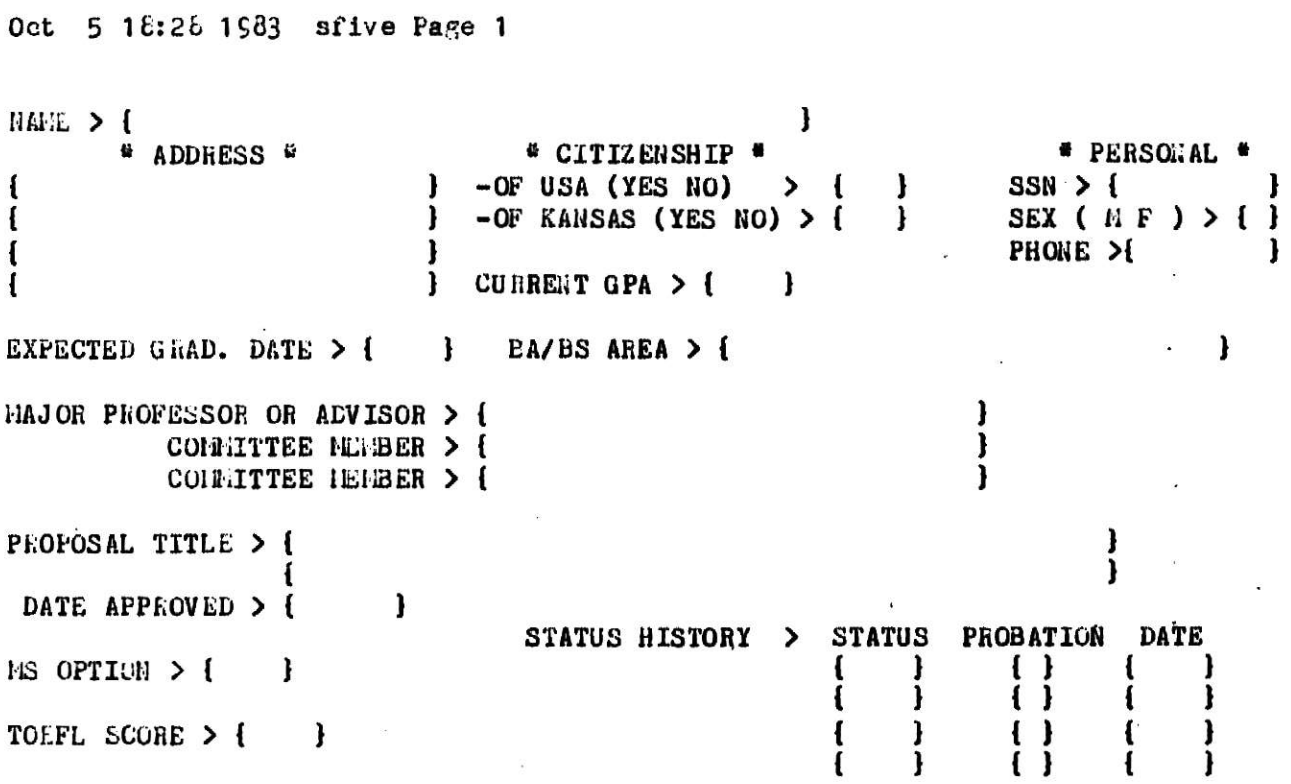

Oct 5 18:28 1983 sfive Page 1
NAME > {
ADDRESS
CITIZENSHIP
-OF USA (YES NO) > { }
-OF KANSAS (YES NO) > { }
CURRENT GPA > { }
PERSONAL
SSN > { }
SEX (M F) > { }
PHONE >{ }
EXPECTED GRAD. DATE > { } BA/BS AREA > { }
MAJOR PROFESSOR OR ADVISOR > {
COMMITTEE MEMBER > {
COMMITTEE MEMBER > {
PROPOSAL TITLE > {
DATE APPROVED > { }
STATUS HISTORY > STATUS PROBATION DATE
MS OPTION > { }
TOEFL SCORE > { }
CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ====>{ }

Figure 3.3

```
Oct  5 16:28 1983  ssix Page 1

  NAME > {                                                    }

  COURSES TAUGHT   COURSES QUALIFIED TO TEACH      EXPERIENCE
                    FACULTY VIEW     STUDENT VIEW
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
   {    }{    }      {   }            {   }        {          }
TEACHER EFFECTIVENESS > {    }     UNDERGRADUATE ADVISING (YES NO ) > {   }

CURRENT SCREEN (c), MENU (m), OR PRESS SPACE BAR FOR NEXT SCREEN ===> { }
```

Figure 3.4

the date that the course was taught. Pressing return will move the course to this column field from the number column field. A sample entry could look like this:

(207) (SP83)

(300) (SU83)

The next two columns under the "COURSES QUALIFIED TO TEACH" heading, allows information to be entered stating what courses the student thinks he/her can teach and what courses the faculty thinks the student can teach. Consecutive returns here will move the cursor down the "FACULTY VIEW" column. When the bottom of this column is reached, the cursor will jump to the top of the "STUDENT VIEW" column. Once again, consecutive returns will move the cursor to the bottom of the column and then to the top of the "EXPERIENCE" column. The "EXPERIENCE" column can be thought of as a "free-form" column. In other words, the user can enter information until the end of the field is reached. At this point, pressing return will move the cursor down to the next field in the same column, etc. A sample entry might look like this:

(2 yrs Cobol)
(3yrs Pascal)
(programming)
(for NDX)

When the bottom of the "EXPERIENCE" column is reached, pressing return will move the cursor to the "TEACHER EFFECTIVENESS" field, etc. When the user command field is

reached, an "m" can be entered to allow a return to the menu, or the space bar can be pressed to allow a return to the first input screen, as in the applicant management program. The same "tab" feature is also present for this screen. Pressing the tab key will move the cursor to the top of each column and then down to the "TEACHER EFFECTIVENESS" field. Once this field is reached the return key may be pressed to navigate to the bottom of the screen so that a command can be entered. Once again, it would be wise to recheck the entered information on each of the three screens to ensure that it was entered correctly.

The function that should be discussed next is the transfer function. As previously described, this function allows the user to transfer information from the applicant database to the graduate database. The program to perform this function is initiated by entering a "t" at the graduate management program menu. When this is done, the screen shown in Figure 3.5 will appear. At this point there are two choices to decide upon. An "m" can be entered to return to the menu, or the space bar can be pressed to allow the screen in Figure 3.6 to appear. This screen allows the user to input the names of the applicants so that their files can be "pulled" from the applicant database and refiled into the graduate database. To enter a name or names, simply enter a "c" at the bottom right-hand corner of this screen. This will move the cursor to the top name field. At this point the name can be entered. It should be noticed that a

Oct 5 16:26 1983 copyscr Page 1

This feature allows you to transfer files from the applicant file drawer over to the graduate student file drawer. Simply touch return and enter the names of the students who you want to transfer. If you've decided to return to the menu enter an 'm' here ==> { } otherwise touch return and we will get started.

****

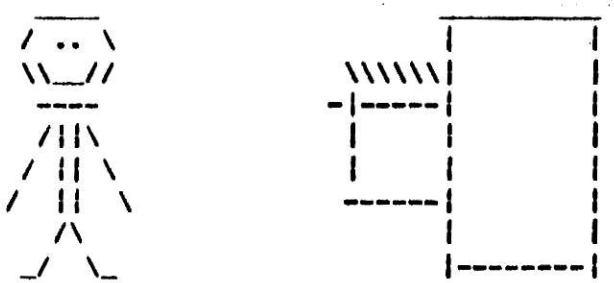


Figure 3.5

Oct 5 16:29 1983 totrans Page 1

PLEASE ENTER THE NAMES OF THE PEOPLE WHOSE RECORDS YOU WANT TO TRANSFER

NAMES (FORMAT: LAST, FIRST MIDDLE)

{ } Please Place a 't' Here to Transfer the Records ==> { }

PLEASE ENTER 'c' FOR CURRENT SCREEN OR 'm' FOR MENU =====> { }

Figure 3.6

specific format is to be followed when entering this name as has been previously discussed in section 1.2.2 of chapter 1. After the name has been properly entered the return key can be pressed to move the cursor to the next name field. A second name can be entered here. In fact, several more names can be entered by simply pressing the return key to move to the next name field. Also, it should be noted that the user can simply navigate to the last name field by repeatedly pressing the return key or if the terminal allows the return to repeat automatically, the user can simply hold down the return key to accomplish the same thing. When the last name field is reached, pressing the return key will move the cursor to the right-hand side of the name fields. At this point a "t" can be entered. This action will automatically transfer all of the records found under the entered names to the graduate database. It should be noted, however, that these records will not be automatically deleted from the applicant database, but merely copied from one to the other. If for some reason it is decided that the entered names are not to be transferred, the return key may be pressed instead of entering a "t". This allows the user to ignore the names that have just been entered. This action also moves the cursor to the bottom right-hand corner of the screen to await a user command. If a "t" was entered to initiate the information transfer, a message may appear stating that the program could not find a file under any or all of the names entered. If this occurs, the invalid names will be pointed

out so that corrections can be made. The correct names will have already been transferred, however, so they do not have to be reentered. If you have more names to enter than the screen has fields, the space bar may be pressed to generate another name screen after the names on the first screen have been transferred. The same procedure can be followed here until all of the user's transfer work is completed for this session. As always, when finished, an "m" may be entered to return to the graduate management program's menu.

4.1 HOW TO DELETE RECORDS (OR HOW NOT TO DELETE RECORDS)

This section of the "KOOL RECORDS SYSTEM" guide to the universe discusses the deletion programs, as has been previously avoided. This program functions in exactly the same way for either the applicant management program or the graduate management program, so it will only need to be mentioned once.

To enter the realm of the deletion program, the user may simply enter a "d" from either the graduate menu or the applicant menu. It is the responsiblity of the deletion program to know where the user is coming from and hence, which database to delete from. Once a "d" command has been entered the screen shown in Figure 4.1 will appear if deletion from the graduate database is to occur. Otherwise, the screen in Figure 4.2 will be displayed. This latter screen allows either applicants or rejected applicants to be deleted from their respective databases. It is realized that the wording of these screens comes across as being rather harsh, but it could save lives someday (most likely the programs' designer and programmer). These screens will give the user one last chance to return to the menu by entering an 'm'. Otherwise, the return key may be pressed if the screen in Figure 4.1 has been displayed, or a choice is given for applicants or rejected applicants from the screen in Figure 4.2. If it is decided to continue with the

Oct 5 16:49 1983 destroy Page 1

```
**************************************************************************
```

THE OPTION YOU HAVE CHOSEN ALLOWS YOU TO DESTROY A STUDENT'S OR APPLICANT'S RECORD. PLEASE RECONSIDER YOUR CHOICE. IF YOU WISH TO RETURN BACK TO THE SUB MENU PLEASE ENTER AN ' m ' HERE ==> { }. IF NOT, THEN TOUCH RETURN AND PROCEED AT YOUR OWN RISK!

```
**************************************************************************
```

Figure 4.1

Oct 5 18:16 1983 asmenu Page 1

```
**************************************************************************

      THIS FEATURE WILL DESTROY THE FILES OF AN APPLICANT OR A REJECTED
      APPLICANT.  PLEASE MAKE SURE THAT YOU KNOW WHAT YOU ARE DOING.
      LISTED BELOW ARE THREE OPTIONS THAT CAN BE CHOSEN.  THE FIRST OPTION
      ALLOWS A SPECIFIC APPLICANT OR REJECTED TO BE DELETED.  THE SECOND
      OPTION ALLOWS A GROUP OF REJECTED APPLICANTS TO BE DELETED PRIOR
      TO A SPECIFIC DATE.  THE THIRD ALLOWS A RETURN TO THE MENU.
**************************************************************************

         ** DELETE SPECIFIC APPLICANTS OR REJECTED APPLICANTS ( D )
         ** DELETE REJECTED APPLICANTS PRIOR TO A GIVEN DATE  ( R )
         ** RETURN TO THE MENU ( M )
             ** PLEASE ENTER OPTION HERE ===> { }
```

Figure 4.2

deletion process, the screen shown in Figure 4.3 will be displayed. This screen is just like the transfer screen discussed above. So, to enter names, a "c" may be entered, or the user may return to the menu by entering an "m" (o.k. so there is one more chance to quit before something terrible happens). As with the transfer function, the cursor will move to the top name field after a "c" has been entered. The names are also entered in the same fashion as in the transfer program. And, once again pressing return will allow navigation to all of the name fields and to the field at the right side of the screen. When the cursor is moved to this location an "x" may be entered to delete files on the names just entered or the return key may be pressed to bypass the actual deletion procedure. Also, as before if any of the names are not familiar to the system, a message will appear stating this and the incorrect names will be pointed out. In addition, the correct names at this point will have been deleted.

It should be mentioned here that when an applicant has been rejected, the applicant's record will not be deleted from the applicant database automatically. It must be done through the deletion program.

Oct 5 16:29 1983 todie Page 1

PLEASE ENTER THE NAMES OF THE PEOPLE WHOSE RECORDS YOU WANT TO DESTROY

NAMES (LAST, FIRST MIDDLE)

{ } Please Place an 'x' Here to Destroy the Records ==> { }

PLEASE ENTER 'c' FOR CURRENT SCREEN OR 'm' FOR MENU ==> { }

Figure 4.3

5.1 THE FACULTY PERUSAL FACILITY

This feature of the "KOOL RECORDS SYSTEM" is perhaps one of the most important in that it allows faculty members to peruse a student's record from their office terminals (or from their home terminals, if desired). This feature should greatly enhance the advisor's role that all faculty members have to portray.

The perusal facility is initiated by typing the word peruse. Once this has been done, a password will be required to gain access to the system's program. If access is granted the screen in Figure 5.1 will appear. This screen plays two roles in that it acts as a menu and it gives a few instructions on how the perusal facility operates. The options provide the user with a way to leave the system (by entering a "$") and a way to proceed with the perusal program (by pressing the return key). If the return key is pressed an alphabetized list of names will appear on the screen. Depending on how many students are currently in the database, the number of names may range from zero to ten. After this list appears, the user will be given the option of choosing one of these names, returning to the menu, or generating the next portion of the name list on a new screen (this will only occur if there are more than ten students in the graduate database). Once the desired name is located, the user should enter a "c" to move the cursor to

Oct 5 18:26 1983 facper0 Page 1

****** WELCOME TO KOOL RECORDS FOR FACULTY PERUSAL ******

Hello! This is the faculty perusal facility. This function allows you, the faculty member, to look at a student's course information, teaching information, or other specific academic information. Simply touch return and an alphabetized name list will appear on the screen. If the name that you desire is not on this list then enter an 'n' for the next portion of the list to appear. When the name is found, follow the directions on the right side of the screen to "pull" this student's file.

The following are options that can be taken now:

* touch return for the perusal process to begin

* enter a '$' to exit the program #===> { }

Figure 5.1

the top of the name list. At this point, the cursor needs to be moved next to the desired name by repeatedly pressing the return key. When the cursor is next to the desired name, a "p" can be entered to "pull" that student's record. It should be noted that if the cursor has inadvertently been moved to the name list, the user can simply enter a "b" to move the cursor back to the command input location. Also, the cursor can be moved back to this location by repeatedly pressing the return key to the last name on the current list section. A return key pressed here will move the cursor to the command input location.

Once the student's file has been pulled, the screen shown in Figure 5.2 will appear. This screen closely resembles the first input screen of the graduate management program. The only difference is that input is not allowed and the commands at the bottom of the screen are different. From this course information screen, three options are available. An "m" may be entered to return to the menu, an "a" may be entered to peruse the student's academic information (shown in Figure 5.3), or a "t" can be entered to view the student's teaching information (shown in Figure 5.4). So, whatever screen the user is on the only commands that need to be remembered are "c", "a", "t", or "m", which happen to be the first letters of the information type desired (with the exception that "m" designates menu).

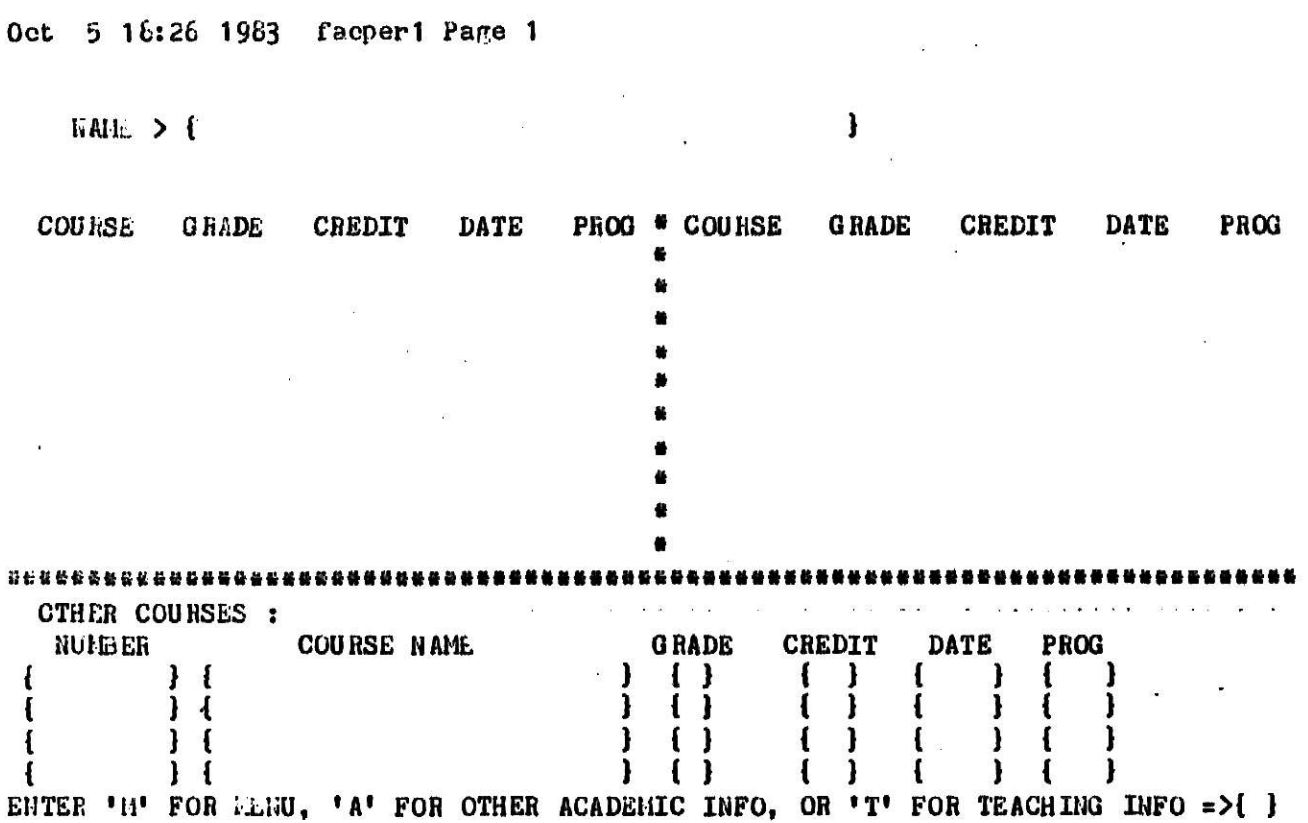

Oct 5 16:26 1983 facper1 Page 1
NAME > {
}
COURSE GRADE CREDIT DATE PROG # COURSE GRADE CREDIT DATE PROG
OTHER COURSES :
NUMBER COURSE NAME GRADE CREDIT DATE PROG
ENTER 'H' FOR MENU, 'A' FOR OTHER ACADEMIC INFO, OR 'T' FOR TEACHING INFO =>{ }

Figure 5.2

```
Oct  5 10:26 1983  fscper2 Page 1


NAME > {                                          }
       * ADDRESS *              * CITIZENSHIP *                 * PERSONAL *
{                        }  -OF USA (YES NO)   > {   }      SSN > {          }
{                        }  -OF KANSAS (YES NO) > {   }      SEX ( M F ) > { }
{                        }                                   PHONE >{         }
{                        }  CURRENT GPA > {    }

EXPECTED GRAD. DATE > {    }   BA/BS AREA > {                          }

MAJOR PROFESSOR OR ADVISOR > {                          }
          COMMITTEE MEMBER > {                          }
          COMMITTEE MEMBER > {                          }

PROPOSAL TITLE > {                                                 }
                 {                                                 }
 DATE APPROVED > {      }
                                  STATUS HISTORY  >  STATUS  PROBATION  DATE
MS OPTION > {    }                                  {    }    { }    {    }
                                                    {    }    { }    {    }
TOEFL SCORE > {    }                                {    }    { }    {    }
                                                    {    }    { }    {    }

ENTER A 'C' FOR COURSE INFO, 'T' FOR TEACHING INFO, OR 'M' FOR MENU  =====> { }
```

Figure 5.3

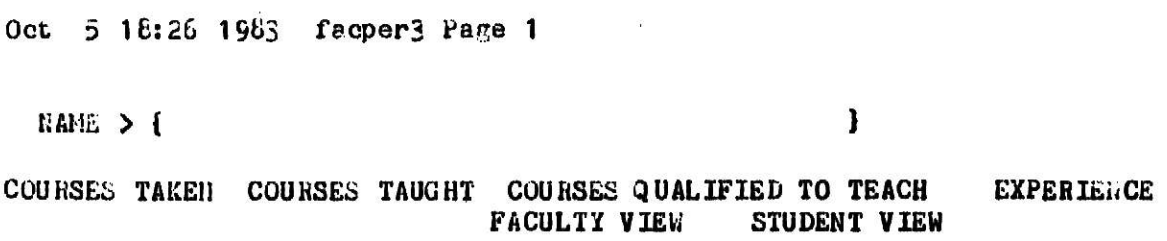

Oct 5 18:26 1983 facper3 Page 1
NAME > {
}
COURSES TAKEN
COURSES TAUGHT
COURSES QUALIFIED TO TEACH
FACULTY VIEW
STUDENT VIEW
EXPERIENCE
TEACHER EFFECTIVENESS > { }
UNDERGRADUATE ADVISING (YES NO) > { }
ENTER 'M' FOR MENU, 'C' FOR COURSE INFO, OR 'A' FOR OTHER ACADEMIC INFO ===>{ }

Figure 5.4

6.1 THE REPORT GENERATOR

The "KOOL RECORDS SYSTEM" report generator enables a user to print out a variety of listings and letters. To enter the report generator program, the user must enter an '1' from the main menu of the "KOOL RECORDS SYSTEM". When this is done, the screen shown in Figure 6.1 is displayed. This screen gives a brief introduction to the report generator and allows the user to continue or return to the main menu. If the user should decide to continue the letter-listing menu shown in Figure 6.2 is displayed. If the user desires to print out one of the listings, the corresponding command should be entered. At this point, the system will prompt the user to enter the current date. For example, the correct date format would be: January 30, 1983. This date will be printed exactly as entered by the user. After the date has been appropriately entered, the system will display a message asking the user if the printing is to occur. If so, the return key must be pressed. If the user would like to return to the letter menu then an '1' may be entered. This provides the user with an escape route if a mistake has been made. If the user wishes to continue with the print function, pressing the return key will achieve this. At this point, the user may walk, but not run, to the nearest high-speed line printer and retrieve the desired listing. (This printer will most likely be the 3220 line printer located in Room 117 of Fairchild Hall).

Oct 16 10:39 1983 ltrintro Page 1

****** WELCOME TO THE KOOL RECORDS SYSTEM REPORT GENERATOR ******

This feature allows you to print a letter to be sent to the student(s) or applicant(s) of your choice. Just choose the letter that you want to print from the menu on the next screen and follow the prompts from there. You will be required to enter information depending on which letter is to be printed such as enrollment dates, date of the letter, etc. The prompts should be self-explanatory.

The report generator will also allow you to print out a complete applicant database listing and a graduate student listing for the Graduate Studies Committee. So, please press the return key for the report menu or enter an 'm' to return to the KOOL Records System main menu *====> { }.

Figure 6.1

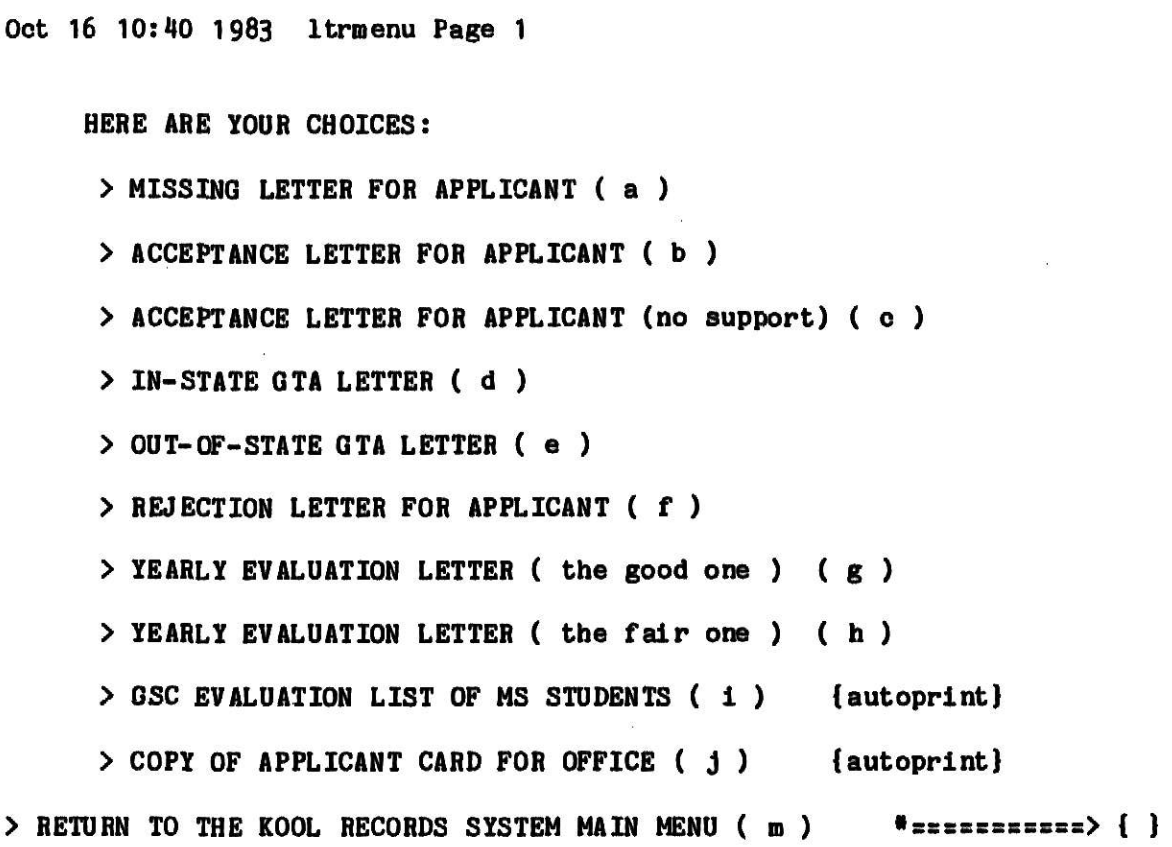

Oct 16 10:40 1983 ltrmenu Page 1
HERE ARE YOUR CHOICES:
> MISSING LETTER FOR APPLICANT (a)
> ACCEPTANCE LETTER FOR APPLICANT (b)
> ACCEPTANCE LETTER FOR APPLICANT (no support) (c)
> IN-STATE GTA LETTER (d)
> OUT-OF-STATE GTA LETTER (e)
> REJECTION LETTER FOR APPLICANT (f)
> YEARLY EVALUATION LETTER (the good one) (g)
> YEARLY EVALUATION LETTER (the fair one) (h)
> GSC EVALUATION LIST OF MS STUDENTS (i) {autoprint}
> COPY OF APPLICANT CARD FOR OFFICE (j) {autoprint}
> RETURN TO THE KOOL RECORDS SYSTEM MAIN MENU (m) *==========> { }

Figure 6.2

If the user wishes to send out letters to students or applicants, the desired letter option may be entered at the displayed letter menu. Again, the system will respond with a prompt asking for the current date. In addition, the system may require more information from the user at this point.This information required depends on the letter to be printed. For example, the acceptance letter that is to be sent to a potential masters student requires the enrollment dates and the date that classes begin for the semester that the student is to start. The system will somehow know this, and ask the user for this information.

Once all of the letter information has been obtained, the screen shown in Figure 6.3 will be displayed. The user must then enter a 'c' to allow entry of the names to send the letters to. When the names have all been entered, a 'p' must be entered at the location specified on the screen. The system will then allow the user a choice of what printer to print these letters on. After the choice is made, the program will then pull all of the entered names along with their addresses and print all of the pertinent information out in the correct letter format. At this point, the letters may be retrieved at the chosen printer.

Oct 5 18:29 1983 toprint Page 1

PLEASE ENTER THE NAMES OF THE PERSONS WHO YOU WOULD LIKE TO SEND THIS LETTER TO.

NAMES (LAST,FIRST,MIDDLE)

{ }

PLACE A 'p' HERE TO PRINT ==> { }

CURRENT SCREEN (c), MENU (m), OR TOUCH RETURN FOR NEXT SCREEN ==> { }

Figure 6.3

IMPLEMENTATION OF AN OFFICE INFORMATION SYSTEM FOR THE DEPARTMENT OF COMPUTER SCIENCE

by

JOHN SCOTT ANDERSON

B.S., Kansas State University, 1979

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1983

ABSTRACT

This report details the implementation of an automated office information system for the Department of Computer Science. This system is intended to aid the Department's faculty and staff in monitoring the progress of graduate students and graduate applicants.

This office information system consists of a menu-driven transaction processor interfacing with a database and report generator. The system is written in the C programming language and incorporates Unix operating system library software.

This report describes the program structure of the system along with the various data structures used by the system. Also described is the logical design of the system. This report introduces a new software diagrammatic method to aid this description.

Also included in this report is a critique of the system along with a discussion of possible future enhancements.