

/BIBLIOGRAPHIC SYSTEM FOR MICROCOMPUTER ENVIRONMENTS/

By

207

Wei Lee

..

B.S., National Chung Hsiung University, Taiwan, R.O.C., 1980

A MASTER'S REPORT

Submitted in partial fulfillment of the

requirements for the degree

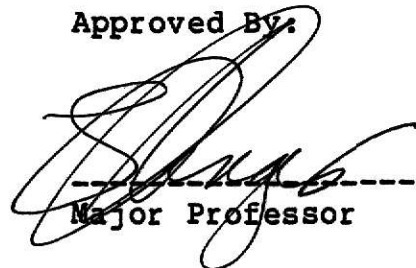
MASTER OF SCIENCE

Department of Computer Science

**Kansas State University
Manhattan, Kansas**

1986

Approved By:

A handwritten signature in black ink, appearing to be "D. H. H.", is written over a horizontal dashed line.

Major Professor

LD
2668
.24
1986
L436
c. 2

ACKNOWLEDGEMENT

A11202 663724

I would like to acknowledge the help and encouragement of Dr. Elizabeth Unger. my major professor. She is a conscientious educator and a dedicated computer scientist.

Thanks are also due to Dr. Paul Fisher and Dr. Rich McBride for acting as my committee members.

After all. I give my heartfelt appreciation to my parents for their love and support.

TABLE OF CONTENTS

	PAGE
CHAPTER 1. INTRODUCAION	1
1.1 OVERVIEW	1
1.2 DISCUSSION OF THE AVAILABLE TOOLS FOR MANAGING REFERENCES	1
1.2.1 KCS	2
1.2.2 DATA BASE MANAGEMENT SYSTEMS	2
1.2.3 FORMATTING PROGRAMS	3
1.2.4 WORD PROCESSORS	3
1.3 THE FEATURES OF REFWELL	4
1.4 FUNCTIONS OF THE REFWELL SYSTEM	5
1.5 HARDWARE/SOFTWARE CONSIDERATION	5
1.5.1 THE HARDWARE	5
1.5.2 THE SUPPORT SOFTWARE	6
1.6 REPORT ORGANIZATION	7
CHAPTER 2. SYSTEM DESIGN	8
2.1 OVERVIEW	8
2.2 DESIGN DECISIONS	8
2.3 FILE STRUCTURES AND THE DESIGN JUSTIFICATION	10
2.4 FUNCTIONS	16
2.4.0 MAIN	16
2.4.1 CREATE	17
2.4.2 APPEND	20
2.4.3 CHANGE	21
2.4.4 RETRIEVE	22
2.4.5 DELETE	25
2.4.6 EXTRACT	25

	PAGE
2.4.7 JOIN	25
2.4.8 FORMAT	26
CHAPTER 3. SYSTEM IMPLEMENTATION	30
3.1 OVERVIEW	30
3.2 SOFTWARE AND HARDWARE USED	30
3.2.1 THE SUPPORT SOFTWARE	30
3.2.2 THE HARDWARE	34
3.3 PROBLEMS IN IMPLEMENTATION	34
3.4 JUSTIFICATION	37
CHAPTER 4. CONCLUSION	40
4.1 SUMMARY	40
4.2 POSSIBLE FUTURE EHHANCEMENTS	40
BIBLIOGRAPHIC	44
APPENDIX:	
A. USER'S MANUAL	45
1. INTRODUCTION	45
2. GETTING STARTED WITH REFWELL	45
2.1 WHAT THE USER NEEDS TO USE REFWELL	45
2.2 THE STEPS TO USE REFWELL	46
3. FUNCTION MODES	46
3.1 MAIN	46
3.2 CREATE MODE	47
3.3 APPEND MODE	48
3.4 CHANGE MODE	48
3.5 RETRIEVE MODE	49

	PAGE
3.6 FORMAT MODE	51
3.7 JOIN MODE	52
4. MISCELLANIES	54
B. PROGRAM LISTING	56

LIST OF FIGURES

	PAGE
FIGURE 1. FORMAT OF THE LEDGER FILE	11
FIGURE 2. FORMAT OF THE AUTHOR DICTIONARY	12
FIGURE 3. FORMAT OF THE KEYWORD DICTIONARY	12
FIGURE 4. JOURNAL STRUCTURE	13
FIGURE 5. BOOK STRUCTURE	14
FIGURE 6. ACCESS DIAGRAM	16
FIGURE 7. WARNIER-ORR DIAGRAM FOR FUNCTION MAIN	17
FIGURE 8. JOURNAL SCREEN ENTRY FORMAT	19
FIGURE 9. BOOK SCREEN ENTRY FORMAT	19
FIGURE 10. WARNIER-ORR DIAGRAM FOR FUNCTION CREATE	20
FIGURE 11. WARNIER-ORR DIAGRAM FOR FUNCTION APPEND	21
FIGURE 12. WARNIER-ORR DIAGRAM FOR FUNCTION CHANGE	22
Figure 13. CRITERIA SCREEN ENTRY FORMAT	23
FIGURE 14. WARNIER-ORR DIAGRAM FOR FUNCTION RETRIEVE	25
FIGURE 15. WARNIER-ORR DIAGRAM FOR FUNCTION JOIN	26
FIGURE 16. SAMPLE LISTING LAYOUTS	27
FIGURE 17. WARNIER-ORR DIAGRAM FOR FUNCTION FORMAT	29
FIGURE 18. MAIN MENU	47
FIGURE 19. SCREEN ENTRY FORMAT SELECTION OR EXIT	48
FIGURE 20. MENU FOR EDITING THE REFERENCES	49
FIGURE 21. MENU FOR VIEWING, EXTRACTING AND DELETING SELECTED REFERENCES	50
FIGURE 22. HEADING OF THE QUICK VIEW LISTING	50
FIGURE 23. MENU FOR SELECTING THE DESTINATION FOR THE OUTPUT LISTING	51
FIGURE 24. MENU FOR SELECTING THE FORMAT	52
FIGURE 25. MENU FOR SELECTING THE ORDER OF THE LISTING	52

FIGURE 26. MESSAGES FOR VERIFYING THE FILE NAMES FOR JOIN 53

LIST OF TABLES

PAGE

TABLE 1. LIST OF FILES	33
------------------------------	----

CHAPTER 1. INTRODUCTION

1.1 OVERVIEW

Researchers and teachers have a need to catalog and retrieve selective sets of bibliographic references in order to form reading list to students or bibliographies for papers. The bibliographies may have to be formatted in various styles to accomodate the publications to which paper is to be directed. Many researchers/teachers now have access to personal computers yet there is no known available software tool which is specifically designed to accomplish all the functions associated with keeping a bibliographic system, eg., storage, selective retrieval, and formatting.

The report describes the design and implementation of a user friendly bibliographic system for microcomputer systems. The system named REFWELL is based upon a relational data base system and is the first version of what could be a valuable tool for busy educational professionals trying to organize the large and varied amount of bibliographic information. The REFWELL bibliographic system was designed and implemented specifically for the faculty in the Kansas State University, Computer Science Department. Later enhancements could add capabililty needed by other desciplines.

1.2 DISCUSSION OF THE AVAILABLE TOOLS FOR MANAGING REFERENCES

There are several available tools to manage bibliographic references. These tools are discussed below.

1.2.1 KGS

KGS is an interactive bibliographic reference system. It is currently implemented using the Perkin-Elmer 8/32 Minicomputer running under the UNIX operating system in the Kansas State University, Computer Science Department[9].

KGS provides the functions to create, edit, and retrieve the references. KGS maintains one master file containing all the information, an author and a keyword dictionary, and two index files for the two dictionaries.

KGS supports only very simple line-editing activities, also, it is restrictive in its editing mode in which a user may only retrieve a reference for editing by submitting the appropriate record number. KGS provides only a hardcopy listing of a selection of references in KGS report format.

1.2.2 DATA BASE MANAGEMENT SYSTEMS

Commercial data base management systems on a personal computer such as PSF:file and dBase III, provide the functions, for example, create, update, and retrieve which can be used to manage the references.

To use these softwares as tools to manage bibliographic references, the user needs to have good knowledge of the systems. Learning and using a commercial software efficiently not only takes time, but also may not give the user the expected result because of the software's limitations. For example, it is

difficult to obtain a formatted bibliographic listing using the report function in dBase III without the help of a word processor. This means that the user may need to use more than one software system to obtain the required functions and receive the desired result.

1.2.3 FORMATTING PROGRAMS

The troff and nroff formatted programs on the UNIX system[8]. Input to nroff is a file of text created using an editor such as vi. Output from nroff is paginated, formatted text that can be sent to a terminal, printer, or plain file. Commands imbedded in the input file determine what the output text looks like[13].

Nroff provides several predefined packages of macros. Using the existing macro packages can make a job easier. Because nroff has no data base functions and the dot commands must be imbedded in the file, nroff and troff are only suitable for preparing a hardcopy listing. In addition, the user must be familiar with the use of the commands and macros to obtain the desired formats. There are various bibliographic formats are used in different publications, it is inefficient to use nroff and troff to manage large amounts of bibliographic references to meet the various formats.

1.2.4 WORD PROCESSORS

It is possible for the user to handle the references by typing in all the references using a word processor such as xedit

on CMS, vi on VAX, and Wordstar on a personal computer. This process is a time-consuming job.

The files containing the references may not be portable from one editor to another and from one machine to another. For example, a document file created by Wordstar can not be transferred to VAX correctly because there are control characters in the file which can not be recognized by VAX. The incompatibilities often restrict the user to a specific machine and editor.

Word processors may perform file read and write, but do not have a data base management system functions. The references in a file can not be sorted or retrieved by a given criteria. To convert one reference format to another using word processor is almost the same as retyping the references.

1.3 THE FEATURES OF REFWELL

An ideal bibliographic reference system should provide the data base management system functions, the formatting facilities, and a user-friendly environment. The user can create, update, retrieve and format the references easily. The formatted listing should be portable from one machine to another such that the listing can be shared by many people.

REFWELL is designed and developed to meet these purposes. REFWELL is a menu-driven system with full-screen data entry and on-line help messages facilities. REFWELL can format the reference listing into two defined formats. With REFWELL, the

faculty members can handle their references more easily than before not only for research and teaching but also for creating the bibliographs/referencrd required for publishing a paper.

1.4 FUNCTIONS OF THE REFWELL SYSTEM

One of the most unique and powerful features of REFWELL is automatic formatting of the listing of references into one of the two predefined formats. The formats are selected for this prototype implementation are the ones used in the ACM Transactions on Office Information Systems and ACM Computing Surveys (see CHAPTER 2). The listing can be in either ascending or descending order sorted by author name.

The REFWELL program provides six major functions for maintaining and inspecting the data base of references. REFWELL allows the user to create a data file, append additional records to a data file, or update the content of a data file. Full-screen editing makes data entry and updating very easy. References can be retrieved from a data file through criteria such as publication date, author name, or both. The searched references can be either extracted to a new data file or deleted from the data file. Two data files can be joined into one new data file.

1.5 HARDWARE/SOFTWARE CONSIDERATION

1.5.1 THE HARDWARE

REFWELL was designed and implemented on a personal computer, because there was no similar commercial software systems

available on the market at that time, as well as the following considerations:

1. the faculty in the Computer Science Department can easily access a Zenith 150 or IBM compatible personal computer.
2. information may be stored in floppy diskette which is easy to carry and ensures the security and privacy.
3. the formatted reference listing can be printed through a personal computer on any kind of printer.
4. the formatted listing is portable from personal computer to mainframe using a transfer system such as KERMIT. Through the electronic mail system on the VAX system, it is easy to exchange reference information with other faculty members.

1.5.2 THE SUPPORT SOFTWARE

The REFWELL system was implemented in the C language. The C language provides support for modular programming, which speeds program development and simplifies maintenance, since each module can be designed and tested independently[12]. C also supports many primitive functions for character string manipulation and dynamic memory allocation, which make the C language a very good high level programming language to be used in implementing a text processing system.

1.6 REPORT ORGANIZATION

Chapter two provides a view of the logical system design. A description of the implementation details is included in chapter three. Chapter four contains the conclusion, and discussion of possible future enhancements.

CHAPTER 2. SYSTEM DESIGN

2.1 OVERVIEW

This chapter will describe the logical design of the REFWELL bibliographic system. While various system design methods were considered, the Warnier-Orr design methodology was chosen as an ideal method for developing logically correct program design structure from the primitive data structure to be used.

Higgins[6] stated that the various forms of the Warnier-Orr diagrams are useful in many stages of a software development project, from requirement definition all the way through the operation and use of the developed project. Warnier-Orr diagrams can enhance the understanding of computer programs actions and the data relationships, as well as increase programmer and analyst productivity.

Warnier-Orr diagrams will be used throughout this chapter for describing the structure during the design phase of the REFWELL bibliographic system.

2.2 DESIGN DECISIONS

A data base management system should have the following capabilities : create a data base, append records, update data item within a data base, search a data base using given criteria, extract records, delete records and produce a report from a data base. So the decision as to what functions the REFWELL program should perform is obvious. Some of the functions will have to be implemented with slight limitations. We assume that once a data

base file is created, it is not likely to be deleted. So the DELETE function is designed imbedded in the RETRIEVE function instead of as an independent function. Function EXTRACT is also designed imbedded in the RETRIEVE function, because the extraction will be needed only when the desired references are selected by searching. Other than these two exceptions, the major functions work independently of one another.

Consideration is given to the design of elegant application software with respect to ease-of-use. To achieve this goal, the following characteristics were considered:

1. MENU-DRIVEN

Each menu contains some function selections, the user can select the desired function with one keystroke. The menus are organized hierarchically. After finishing the execution of any function, the system will return to the main menu for the next selection. Any invalid selection will invoke the system's default selection which could be to return to the main menu or to execute a function with no effect.

2. ON-LINE HELP

On-line help messages are short explanatory. They describe how the functions work, what the data formats are, what the defined function keys are, and provide warning messages for each of the functions on the associated menu. After displaying the help messages, the previous menu will be redisplayed.

3. FULL-SCREEN DATA ENTRY

In a full-screen data entry system the information presented in a visual format are related. The display of information, visible cursor editing position, and immediate display of results of the user actions are the features of a full-screen data entry system.

Data can be entered and changed through the screen format consisting of defined fields. Four arrow keys are defined to move the cursor in between the defined fields on the screen format. [PgDn] and [PgUp] are also defined for the user to recall the previous record or request the next record to be shown on the screen. [Esc] is defined for exiting the editing mode.

4. VERIFICATION

Verification is done, before any function selection which could physically change the content of a data file can occur. For example, when saving changes, appending new records, or deleting records, the user will be prompted to verify his/her decision by the message "Are you sure (Y/N)?". This prevents accidental lost of data.

The following sections will give a high level description of the overall system design. The first section will cover organization of the file structures and the second section will discuss the functions.

2.4 FILE STRUCTURE AND THE DESIGN JUSTIFICATION

The REFWELL bibliographic system consists of zero or more data files, a dictionary for keywords and one for author names, and one ledger file.

The ledger file is created when the first data file is created and exists as long as there is a data file exists. The ledger file contains three fields, namely, FILE NUMBER, FILE NAME, and FILE SIZE. The file number is a unique integer number given by the system when the file is created. The file name consists of the conventional one to eight character DOS file name with BBF as its extension standing for a bibliographic data file. The file size is also an integer number representing how many records are in the data file. The format of ledger file is given in Figure 1. Before beginning to execute a function, the user is required to supply the file name s/he wants to work on. The file name will be checked against the ledger file. Only an existing file can be accessed for the purposes other than creation.

```
+-----+
| FILE NUMBER ( 2 CHARS ) |
+-----+
| FILE NAME ( 12 CHARS ) |
+-----+
| FILE SIZE ( 2 CHARS ) |
+-----+
```

Figure 1: Format of the Ledger File

There are two dictionaries, one is an author dictionary, the other is a keyword dictionary. Dictionaries are created when the first data file is saved. The data items of the dictionaries come from the data file. The author dictionary contains five fields, namely, FILE NUMBER, AUTHOR NAME, RECORD NUMBER, TYPE OF

PUBLICATION, and PUBLICATION DATE. The keyword dictionary contains three fields, namely, FILE NUMBER, KEYWORD, RECORD NUMBER. The dictionaries are kept sorted in ascending order by keyword. The file numbers in the dictionaries are used instead of the file name to identify the desired data file. The record number is used to indicate the position of the record in the file. The formats of author and keyword dictionaries are given in Figure 2 and Figure 3, respectively.

FILE NUMBER (2 CHARS)
AUTHOR NAME (30 CHARS)
RECORD NUMBER (2 CHARS)
TYPE OF PUBLICATION (1 CHAR)
DATE (15 CHARS)

Figure 2: AUTHOR DICTIONARY

FILE NUMBER (2 CHARS)
KEYWORD (30 CHARS)
RECORD NUMBER (2 CHARS)

Figure 3: KEYWORD DICTIONARY

The data file is unsorted and contains the detailed information on each reference stored. Each record corresponds to one publication which may be a journal or a book. The formats for journals and books are given in Figure 4 and Figure 5, respectively.

TYPE OF PUBLICATION (1 CHAR)
AUTHOR NAME COUNT (2 CHARS)
AUTHOR NAME 1 (31 CHARS)
AUTHOR NAME 2 (31 CHARS)
AUTHOR NAME 3 (31 CHARS)
AUTHOR NAME 4 (31 CHARS)
AUTHOR NAME 5 (31 CHARS)
AUTHOR NAME 6 (31 CHARS)
KEYWORD COUNT (2 CHARS)
KEYWORD 1 (21 CHARS)
KEYWORD 2 (21 CHARS)
KEYWORD 3 (21 CHARS)
KEYWORD 4 (21 CHARS)
KEYWORD 5 (21 CHARS)
KEYWORD 6 (21 CHARS)
PUBLICATION NAME (127 CHARS)
PUBLICATION TOPIC (127 CHARS)
PAGES (21 CHARS)
DATE (21 CHARS)

Figure 4: JOURNAL STRUCTURE

TYPE OF PUBLICATION (1 CHAR)
AUTHOR NAME COUNT (2 CHARS)
AUTHOR NAME 1 (31 CHARS)
AUTHOR NAME 2 (31 CHARS)
AUTHOR NAME 3 (31 CHARS)
AUTHOR NAME 4 (31 CHARS)
AUTHOR NAME 5 (31 CHARS)
AUTHOR NAME 6 (31 CHARS)
BOOKNAME (127 CHARS)
CITY (31 CHARS)
PUBLISHER (51 CHARS)
PAGES (21 CHARS)
DATE (17 CHARS)

Figure 5: BOOK STRUCTURE

The ledger file maintains the essential information of the data files for validating the entered file name. The dictionaries provide the direct retrieval of the data file records without slow sequentially searching of the file.

Only one data file can be processed at a time. So the user is always required to supply the file name to a function. The file name will be checked against the ledger file.

References could be retrieved upon a given criteria. A criteria may consist of combination of author names, dates, and keywords. If the criteria contains author names, dates, or both, the author dictionary will be used. If the criteria includes

keywords then the keyword dictionary will be also used.

Retrieval begins with checking the entered file name against the ledger file. If it is an existing file, the file will be loaded from the disk to the RAM memory and the file number is made known to the function. Next, the dictionary is sequentially searched.

First of all, the first field of the dictionary, that is, the file number, must be matched with the file number of the input file. If they matched, then the names, dates, or keywords will be checked against the items on the criteria. Only when all the items on the criteria are found in the dictionary records having the same record number, then the record number will be recorded for further process.

Finally, the selected references can be displayed, deleted, or extracted. The access diagram showing the relationships between the ledger file, author dictionary, and data file is given in Figure 6.

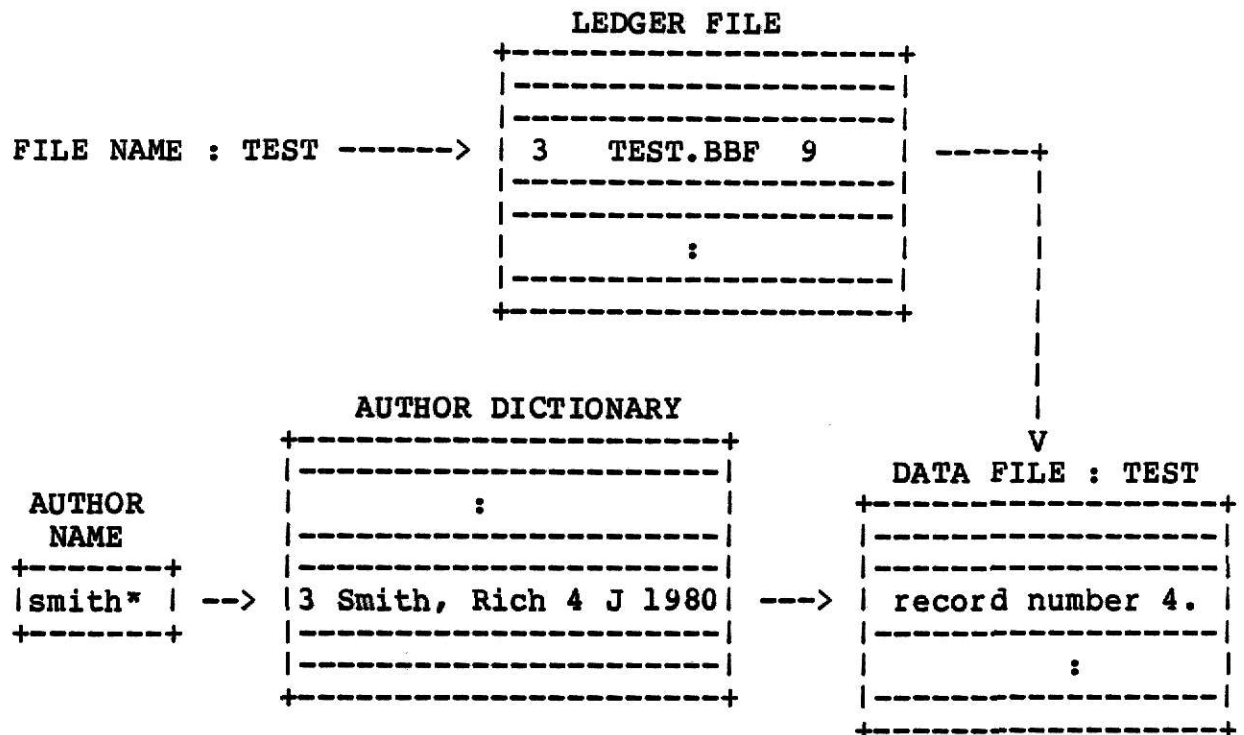


Figure 6: Access Diagram

2.4 FUNCTIONS

This section contains the detailed description of the individual system function. Warnier-Orr diagrams are used to illustrate the function activities and relationships between functions.

2.4.0 MAIN

Function MAIN will bring up the INTRODUCTION of this system to the user when the REFWELL program is loaded. INTRODUCTION contains the general information about the REFWELL system followed by LISTING showing the content of the ledger file. A general help message and other major function selections are displayed through the main menu which will be brought up after

the display of the LISTING messages.

The INTRODUCTION and LISTING are only displayed once when the system is loaded. The other major functions can be called zero or more times. Every function will return control to the function MAIN. The Warnier-Orr diagram for the function MAIN is given in Figure 7.

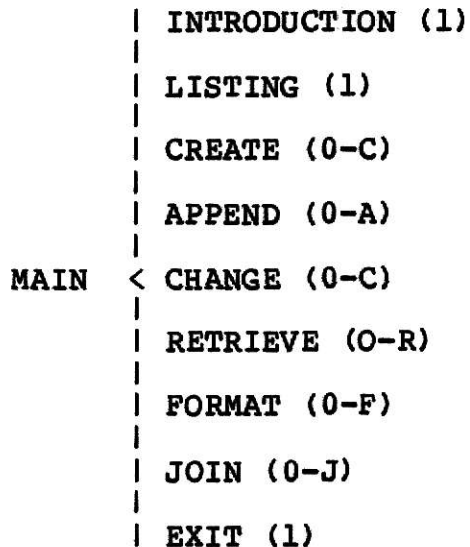


Figure 7: Warnier-Orr Diagram for Function MAIN

2.4.1 CREATE

When CREATE is chosen from the main menu, the user will be prompted for a file name. Only a non-existent file name is considered as a valid file name and will be opened and created. Otherwise, the system will abort the creation and go back to the main menu. A valid file name will bring up the next menu which allows the user to select the screen entry format which could be a journal or a book, or exiting the creation mode.

Four arrow keys are defined for cursor movement on the

screen entry format. The rest of the function keys on the keyboard are undefined to the REFWELL program. Hitting any of the undefined function keys does not input anything into the file.

To exit the entry mode, move the cursor to the last line and hit return. The user will be again prompted to select the screen format or type "x" to exit. After exiting the entry mode, the user has the option to save the entries. If answer is no, message "Are you sure (Y/N)? " will be shown to ask the user for verification. Any answer other than (Y)es and (N)o will be considered as Yes. A unique integer number is assigned to the newly created file as its file number, and together with its file name and file size will be recorded into the ledger file. The dictionaries will also be updated. If not saved, all the entries will be lost.

The journal and book screen entry formats are given in Figure 8 and 9, respectively. The Warnier-Orr Diagram for the function CREATE is given in Figure 10. In the top left of Figure 8 and 9, "FILE NAME" shows the name of the file being created and "RECORD #" indicates the number of the record being entered.

```

FILE NAME : test.bbf
RECORD # : 1

NAME 1  : [ ]
NAME 2  : [ ]
NAME 3  : [ ]
NAME 4  : [ ]
NAME 5  : [ ]
NAME 6  : [ ]

TOPIC   : [ ]
        [ ]

JOURNAL : [ ]
        [ ]

DATE    : [ ]

KEYWORD : [ ] [ ]
        [ ] [ ]
        [ ] [ ]

PAGE #  : [ ]

```

Figure 8: Journal Screen Entry Format

```

FILE NAME : test.bbf
RECORD # : 2

NAME 1  : [ ]
NAME 2  : [ ]
NAME 3  : [ ]
NAME 4  : [ ]
NAME 5  : [ ]
NAME 6  : [ ]

BOOKNAME : [ ]
          [ ]

YEAR : [ ] PAGE : [ ]

CITY : [ ]

PUBLISHER : [ ]

```

Figure 9: Book Screen Entry Format

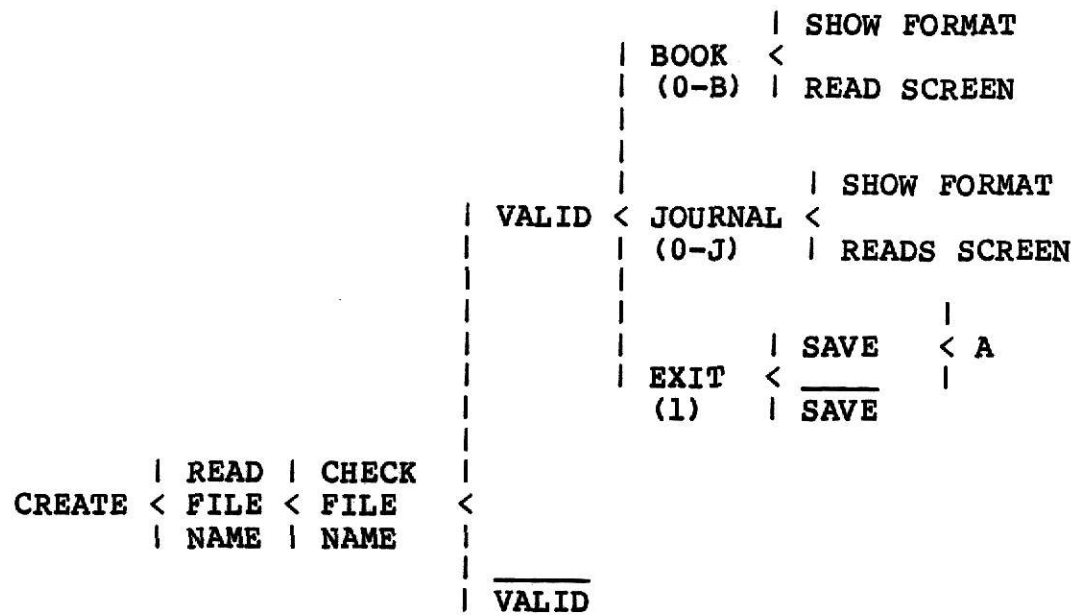


Figure 10: Warnier-Orr Diagram for Function CREATE

2.4.2 APPEND

When APPEND is chosen from the main menu, the user is prompted for the name of the file to be appended. If it exists in the ledger file, the rest of the transactions are the same as in the function CREATE. The Warnier-Orr diagram for the function APPEND is given in Figure 11.

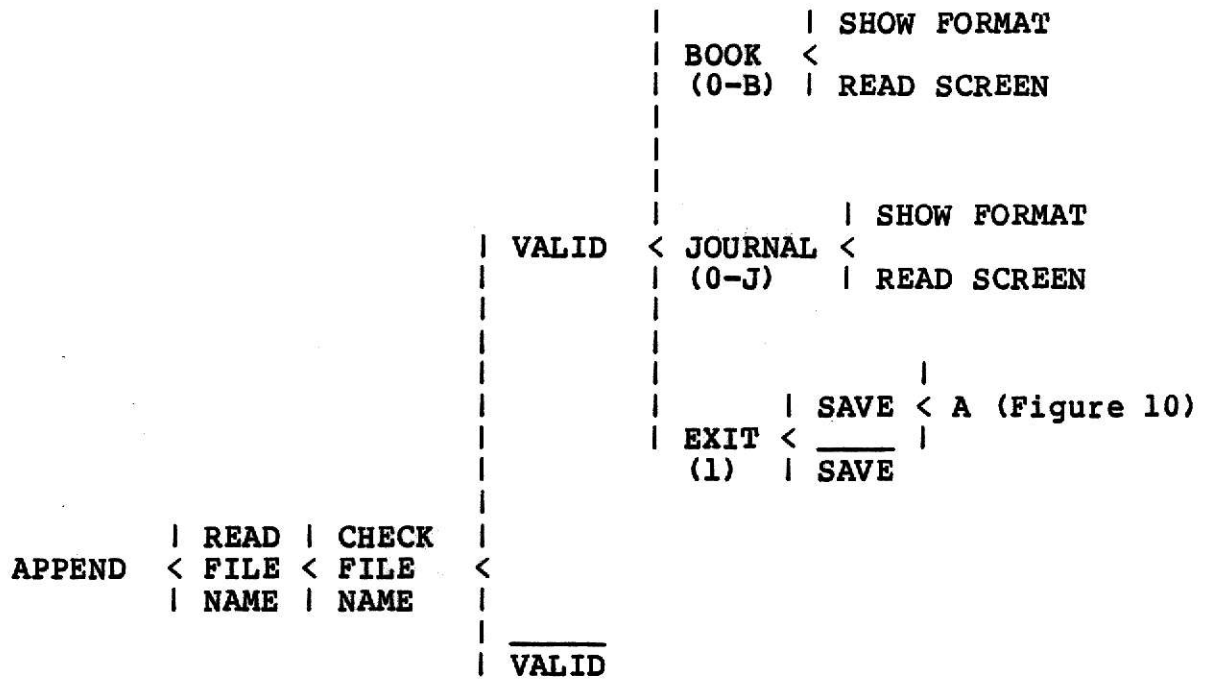


Figure 11: Warnier-Orr Diagram for Function APPEND

2.4.3 CHANGE

CHANGE allows the user to recall a desired data file back to the screen, make corrections, and save all the changes. The four arrow keys, [PgUp], and [PgDn] are defined to allow the user to view and make changes easily. [PgUp] brings back the previous record and [PgDn] bring up the next record to the screen. [Esc] is defined to exit the editing mode.

After exiting from the editing mode, just like CREATE or APPEND, if the file is not saved, the data file will remain unchanged. The Warnier-Orr diagram for the function CHANGE is given in Figure 12.

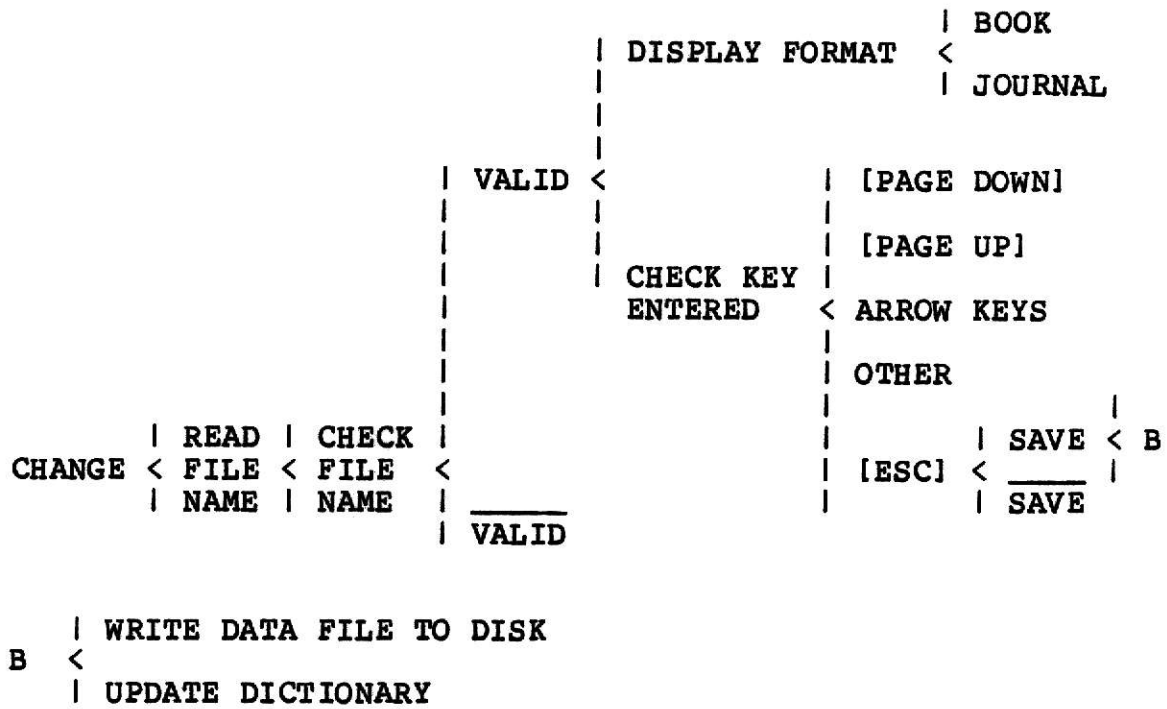


Figure 12: Warnier-Orr Diagram for Function CHANGE

2.4.4 RETRIEVE

Function RETRIEVE allows the user to receive a listing of records which meet the criteria entered by the user. The criteria may have up to six author names, two dates (one beginning and one ending dates), and six keywords. This is an Intersection Selection, which means that each of the selected references must contain all the items in the criteria. The screen format for entry of the criteria given in Figure 13. In the top left of Figure 13, "FILE NAME" shows the name of the file to be searched and "FILE SIZE" is the number of records in the file.

```
FILE NAME : test.bbf
FILE SIZE : 7
```

```
NAME1 : [ ]
NAME2 : [ ]
NAME3 : [ ]
NAME4 : [ ]
NAME5 : [ ]
NAME6 : [ ]
```

```
DATE FROM : [ ]
          TO : [ ]
```

```
KEYWORD1 : [ ]
KEYWORD2 : [ ]
KEYWORD3 : [ ]
KEYWORD4 : [ ]
KEYWORD5 : [ ]
KEYWORD6 : [ ]
```

Figure 13: Criteria Screen Entry Format

For the convenience of comparing the content of the criteria with the content of the dictionaries, comparison of names and keywords is not case-sensitive. There are two date fields in the criteria, namely, FROM and TO. The dates should consist of four digits representing the year, for example, 1986. If criteria contains only a FROM date, the system will search all the references which were published after that date. If only TO exists, searching will focus on the references which were published prior to the given date. If both exist, the references which were published in between will be selected.

A wildcard "*" matches every combination of characters for the search parameters. For instance, "smith*" in the criteria stands for searching all the references in which one of the authors has "Smith" as the last name (independent of first name is). This, in combining with published dates, and keywords,

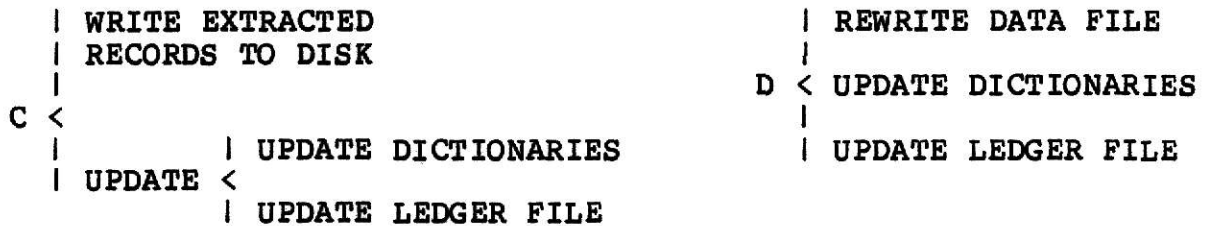


Figure 14: Warnier-Orr Diagram for Function RETRIEVE

2.4.5 DELETE

DELETE is imbedded in function RETRIEVE and allows the user to delete only the references retrieved through the given criteria. The ledger file and dictionaries will be updated.

2.4.6 EXTRACT

EXTRACT is also imbedded in function RETRIEVE and allows creation of a new data file for the references retrieved through the given criteria. A unique integer number is assigned to the newly created file as its file number, and together with its file name and file size will be recorded into the ledger file. The dictionaries will be also updated and the data file remain unchanged.

2.4.7 JOIN

The function JOIN concatenates two data files into a new data file. The original two files remain unchanged. A unique integer number is assigned to the newly created file as its file number, and together with its file name and file size will be recorded into the ledger file. The dictionaries will be also updated. The Warnier-Orr diagram for the function JOIN is given

in Figure 15.

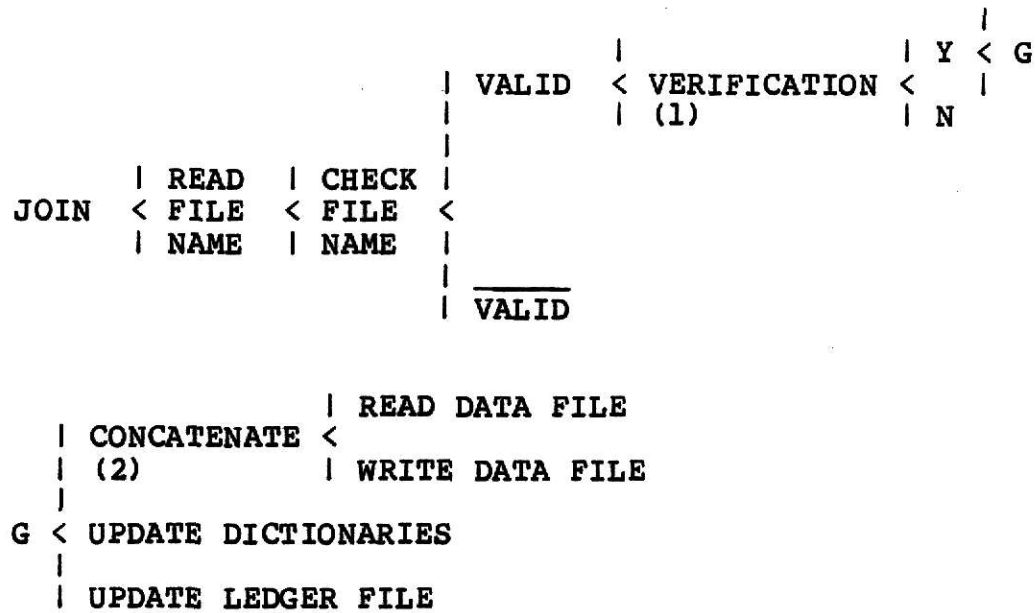


Figure 15: Warnier-Orr Diagram for Function JOIN

2.4.8 FORMAT

The content of a data file may be automatically formatted into either one of the two predefined formats. These emulate the formats used in the ACM Computing Surveys and ACM Transactions on Office Information Systems. Sample listings are given in Figure 16.

or writing the formatted listing to the screen or disk, the control will return to the function MAIN. So, each execution of the function FORMAT only provides either a view or creating an output file for the formatted listing. The output file contains the sorted and formatted reference listing which has different structure from a data file. It will not be included in the ledger file.

After the destination of the output is decided, the format should be selected. The default format is formatl, that is the reference format used in the ACM Computing Surveys. The last selection needed is to choose the order of the listing, which could be ascending, descending, or unsorted order by author name. The default order is unsorted.

The width of the formatted listing will be 65 characters, double-spaced between two reference, single-spaced within each reference. The system will automatically adjust the blanks between words to align both left and right margins (see Figure 16).

To sort the references on author name is done by means of the indexing method. The system will create a temporary index file, containing each record number of the file, for this purpose. The author names in the data file are compared to each other, but only the record number in the index file will be swapped and sorted. So the data file will not be physically sorted. The shellsort is used to perform the sorting. The Warnier-Orr diagram for the function FORMAT is given in Figure

CHAPTER 3. SYSTEM IMPLEMENTATION

3.1 OVERVIEW

Chapter two discussed the system design through a discussion of the its file structures and functions. This chapter will discuss the project implementation, a brief description of the software and hardware used, and the interesting problems which occurred during programming. The justification of the implementation will be discussed in the last section of this chapter.

3.2 SOFTWARE AND HARDWARE USED

3.2.1 THE SUPPORT SOFTWARE

As mentioned in Chapter 1, the C language is a versatile and powerful high level programming language. The Lifeboat version 2.13 Lattice C compiler is used for the implementation of this project. The Lattice portable C compiler accepts a program written in the C programming language, determines the elementary actions specified by that program, and eventually translates those actions into machine language instructions[1]. The definition of the C language by the Lattice C compiler using Kernighan and Ritchie text The C Programming Language as a reference point[7]. Though existing some differences from the standard, the vast majority of programs will not encounter those potential troublespots.

The Lattice C compiler supports the full set of pre-processor commands in Kernighan & Ritchie; the standard

extensions leading to various kinds of derived objects, including pointers, functions, arrays, and structures and unions; the full complement of statement flow constructs. All the standard operators are supported by the Lattice compiler, in the standard order of precedence (see P. 49 of Kernighan & Ritchie).

This version of the C compiler provides four memory models. The model S is used, because it produces compact, efficient code limited to addressing a 64K data area. The model S is the most efficient of the four models.

C provides the fundamental flow-control constructions required for well-structured programs. The functions of a C program may be compiled separately [7]. This entire system consists of several programs; they are separately coded, compiled, tested and finally, linked together. The advantage of this modularity is that a simple modification made on one function only causes the recompiling of the program containing the function and relinking the new compiled object file with other object files.

This project consists of one header file and ten program files. HEADER.H is the header file containing all the structure declarations and constant definitions. Each program file performs one specific system function.

Program files have ".C"'s as their extensions. They are compiled separately into object files with the extension ".OBJ"'s, and linked together creating an executable file with

the extension ".EXE". The link program is the Microsoft 8086 Object Linker, Version 2.40.

Three general classes of I/O functions are provided. First, the level 2 functions define a buffered text file interface which implements the single character I/O functions as macros rather than function calls. Second, the level 1 functions define a byte stream-oriented file interface. Finally, a special set of functions allows single character I/O directly to the user's terminal, as well as formatted and string I/O.

Two file types are provided; either an untranslated or binary file, or a translated or text file. In a binary file, as all components of the file are of equal length, the position of a specific component can be calculated. Thus, the file pointer can be moved to any component in the file, providing random access to any element of the file. The basic components of a text file are characters, they are structured into lines. As the length of lines may vary, the position of a given line in a file can not be calculated. Text files can therefore only be processed sequentially. Furthermore, input and output cannot be performed simultaneously to a text file. The level 2 functions are designed to work primarily with text files. The actual I/O operations are performed by the level 2 functions through calls to the level 1 I/O functions.

This project uses text files and level 2 I/O functions. The decision to choose text file I/O is made because of its ease-of-use. It is easy to understand how the file is read and written,

what the file structures are, and how the programs work. Thus, It is easy for the user to understand, debug, and modify the programs using text file I/O. Besides, text file I/O is used in the programming classes and even in some advanced classes in our department.

Inside a C program, DOS system calls provide an easy way to execute the DOS resident commands and the executable files[1]. For example, `SYSTEM("CLS")` clears the screen, `SYSTEM("PAUSE")` suspends the execution until any key except CTRL BREAK is pressed. These DOS function calls can be embedded in source programs and compiled together, which are useful when a program needs the services of the DOS utilities or executing an external file.

The number of lines of code and the size of the source and object code for each file is listed in Table 1. Each of these modules was discussed in Chapter 2 in terms of functionality.

File Name	Lines	Source code	Object code
APPEND.C	78	1791	1450
CHANGE.C	1105	21928	20167
COMMON.C	1168	24417	18280
CREATE.C	92	1790	1519
DELETE.C	131	3119	2691
FORM.C	677	15404	12985
FORMAT.C	57	3078	3028
JOIN.C	229	6076	4830
MAIN.C	177	5803	6135
SEARCH.C	748	14428	11625
HEADER.H	107	1888	
TOTAL	4569	99722	77280
REFWELL.EXE			63360

Table 1. Actual sizes of the REFWELL programs

3.2.2 THE HARDWARE

This project is implemented on a Zenith 150 personal computer, with dual 320K floppy drives, 320K bytes RAM under networking through an Ethernet server.

3.3 PROBLEMS IN IMPLEMENTATION

This section contains some random observations about the process of programming this project.

Since full-screen data entry is designed as one of the features of REFWELL, the essential requirement is to manipulate cursor movement. Lattice C is a plain compiler providing only all elementary functions but not including this function. A an 8088 assembly program is developed for this purpose. The cursor can be moved by function call `cursor(x, y)`, where `x` is a value on the X-axis and `y` is a value on the Y-axis on the screen.

Screen cursor positioning may be controlled by issuing a special character sequence from within the program[12]. The character sequence is converted into a macro (i.e., `#define`) and included in the header file. Macros execute faster than their equivalent functions, because function-call and return processing are eliminated, but macros are more difficult to debug than functions. In order to support the full-screen data entry and displaying the menus, cursor movement is greatly used. The replacement of the cursor function by a macro has increased overall system performance.

The library functions `scanf()` and `fscanf()` are the two level 2 input functions used. According to the manual, they should perform formatted input conversion. But these two functions don't work as they should at this version of Lattice C Compiler. White space is used as the field delimiter instead of the predefined formats.

Because `fscanf()` and `scanf()` always treat blanks as field delimiters, the problem occurs when the data item is a string with imbedded blanks. Therefore, the input and output of a data file have to be converted into line-oriented format.

The author dictionary contains five fields which must be read in by one input function. There is at least one space inside the AUTHOR NAME and DATE fields, these imbedded blanks create the appearance of more than five fields. The input function will scan only the first five fields. The remaining fields will be unread and cause the problems.

How to cause the unconsecutive character strings to be treated as one field is troublesome. The author dictionary is heavily used and the problem needs to be solved efficiently. There are two possible alternatives can be used to solve this problem. First, is to separate each field of a record as a line. In this case, an author dictionary record can be read in by five input statements. When the author dictionary is growing larger, this kind of arrangement would greatly increase the overhead by sequential read and write. Second, is to make the blanks within a field "disappeared". Such that, a record can be read in by only

one input statement. With the respect of efficiency, the second alternative is chosen. Before the author names and dates are written into the dictionary, all the spaces within a field are replaced with "*"s. This links the unconsecutive words of a field become one field which allows application of the input functions without any problem. For example, "John, Smith" is converted into "John,*Smith".

One of the unique features of C is its dynamic memory allocation capability. Arrays and linked lists are the two commonly used data structures in some high level languages, for example, PASCAL. Array is a fixed size data structure and whose size must be determined when declaring the variable name. For an application software, fixed size data structures may result in wasted memory or memory shortage. This is inflexible and impractical. Although a linked list is a dynamic data structure, when the list is large, the overhead of the sequential search can greatly influence the performance.

In a C program, an array can be dynamically created. A block of consecutive memory space can be allocated to a pointer type variable by the function call `calloc()`. After the memory has been allocated to the variable, it becomes an array and can be accessed by an index. In REFWELL, when a data file is needed, the size of the data file is determined by the ledger file. If the memory space is available, the data file will be loaded. After data file processing is completed, the memory can be released to the system for other purposes by the function call `free()`.

3.4 JUSTIFICATION

The specification and resulting design of REFWELL resolved many performance problems, although some problems still remain.

Files are sequentially accessed and the I/O speed is limited, the performance of REFWELL is also limited. The dictionaries and ledger file are not loaded and kept in the RAM memory until an update or change is needed. After changes are made, system will write them back onto the diskette. Any change in a data file will also cause rewriting of the data file and the dictionary. When the data files, dictionaries, and ledger file are large, REFWELL will take a while to finish the process. Actually, in any data base management system, when the data base is large, the processing time will be longer.

On-line help messages are available from screen menus. The user may need the information from the help messages to get started on each function. The help messages are not accessible while a function is being processed. The menu contains self-describing function names and help messages, there are only a small set of function key definitions and data formats to be referenced. The arrangement of the help messages is actually convenient and complete enough for the user to use REFWELL without referencing the help messages.

REFWELL accepts all the characters from the screen entry formats. There is no data format checking and data type checking in this version of REFWELL. Unrecognizable characters may interfere with the processing of the REFWELL function. The user

may obtain the unexpected results because of incorrect entries, but will not bring the REFWELL system down.

The current version of REFWELL has no capability to access the data file from the different diskette drive. The REFWELL.EXE, COMMAND.COM, dictionaries, ledger file and data files must be kept in the same drive.

A blank and formatted diskette has 360K free bytes. Excluding space needed for REFWELL.EXE and COMMAND.COM, there will be 277,500 free bytes available. The maximum size of a data file is 64k bytes. The maximum sizes of a journal and book records are 613 and 436 bytes, respectively. We can have 104 fully-entered journal records in a file and four files in a diskette. But, this much space will not be needed for actual use of the system. For example, a sample data file containing one book record and nine journal records takes only 1357 bytes. In this case, one diskette will be able to keep large amounts of information.

Although it contains some minor deficiencies, the REFWELL bibliographic system is a useful tool. REFWELL solves the problems of exhausting, and time-consuming typing, formatting and organizing the large amount of references.

From the time-saving point of view, full-screen data entry of REFWELL makes data entry easier than using a word processor (see figure 8 and 9). The user has no need to worry about the formats while typing the references. There is also no need to

convert from one format to the other manually. REFWELL provides the user with two predefined format options. The bibliographic formatted listing will be automatically generated and can be sorted by the author names.

From the operation point of view, full-screen data entry, menu-driven processing, and appropriate on-line help messages provide the user a very user-friendly environment. The formatted bibliographic listing can be printed through a personal computer on any kind of printer. The user can use a word processor or DOS "PRINT" command to get the listing from the printer. The user can work on the references on an IBM or IBM compatible personal computer which is more easily accessed than a dumb terminal.

From the expense point of view, the cost of using this system is minimal. REFWELL can be used on an IBM or IBM compatible personal computer with either dual or single drive under MS-DOS or PC-DOS. The RAM memory can be in the range between 128K to 640K. In order to use REFWELL on a Zenith 150 personal computer under networking through an Ethernet server at Kansas State University, Computer Science Department, the user only needs to provide formatted diskettes.

CHAPTER 4. CONCLUSION

4.1 SUMMARY

This report describes the design and implementation of the REFWELL bibliographic system and its facilities. The main purpose of this project is to implement a useful tool on the microcomputer for the use of the faculty members in Computer Science Department, Kansas State University.

This first microcomputer version of bibliographic system provides the facilities allowing users to create, update, and retrieve references. Also, it can produce formatted bibliographic listings in the formats which are used in the ACM Transactions on Office Information Systems or in the ACM Computing Surveys.

The code is made as modular as possible, the overall structure is determined by a functional decomposition of the tasks to be implemented. This is aided by working in the C language, in which all program files are independent.

4.2 POSSIBLE FUTURE ENHANCEMENTS

There are still possibilities for future enhancements to the REFWELL bibliographic system. There are improvements that could increase the capabilities of REFWELL. Many of these could be establishing a communication between the system functions and the expanding the capabilities of the system functions.

As the system is currently designed, communications between functions in REFWELL are in between RETRIEVE and DELETE, and

between RETRIEVE and EXTRACT. Functions DELETE and EXTRACT process the references come from the result of RETRIEVE. It would be convenient for the user to edit a set of related references, if the result of retrieval could be also passed to CHANGE. For example, to correct all the misspellings of an author name in a data file. All the references can be selected by RETRIEVE and passed to CHANGE for updating.

References are selected through a given criteria, then the selected references can be deleted or extracted. Additionally, references can be changed consecutively. It might be handy to provide a method where retrieving, deleting, extracting, and updating specific references could be done by designating the target record number.

For some users, the structures of the journal, book, and criteria could be expanded to contain more information. For example, to include additional fields for ISBN number, editor names, and library information into the journal and book; additional field for strings in the criteria.

REFWELL is capable of producing bibliographies in two formats, the one used in the ACM Transactions on Office Information Systems and the other one used in Computing Surveys. It might worthwhile defining more formats such as those used in the IEEE publications. An additional enhancement would be to optionally allow a sort by publication date.

REFWELL has the capabilities to build the author dictionary and performs functions on it. The function RETRIEVE can only

search the author names, publication dates or both on the criteria. It would be useful to have the functions to build the keyword dictionary and allow the function RETRIEVE has the capabilities to search references through combination of the author names, publication dates, and keywords. This would provide the user a powerful and easy way to retrieve references. It could be accomplished simply by adding the functions which are similar to the functions working on the author names.

There are some improvements that could increase the ease-of-use. The help facilities in REFWELL are on the menus, and can be chosen only from the menus. The conversion of the help facilities to context-sensitive help facilities would be very convenient. This means that if the user requests "hlep" while in the creation mode, the help response would relate specifically to the activities involving creation. Another improvement to this system would be to add a utility with the capability to list the directory of the ledger file. It would be very handy, especially, when the user is prompted for entering file name. The user could locate the data file easily.

An essential future enhancement to this system would be the development of error checking procedures. These could be done by using data types and data formats. For example, in the editing mode, any character other than digits would not be accepted in the date field, any entry of undefined function keys could not move the cursor. Available memory checking is also necessary. System exit would occur if the live memory is not big enough to

meet the memory needed. To avoid this, a checking procedure should be able to test the memory allocated. If the memory shortage happens, then return the control to the main program instead of executing the system exit.

REFERENCES :

- 1 Lattice C Compiler Manual. Lifeboat Associates, New York, New York, 1984.
- 2 MS-DOS version 2. Zenith Data Systems Corporation, St. Joseph, Michigan, 1984.
- 3 IBM Disk Operating System Reference Manual. Microsoft, Inc., Boca Raton, Florida 33432, 1980.
- 4 Hanson, O. Design of Computer Data Files. Pitman Publishing Inc., Marshfield, Massachusettes, 1982.
- 5 Harbison, S., and Steele, B. Jr. C: a Reference Manual. printice-Hall, Englewood Cliffs, New Jersey, 1984.
- 6 Higgins, D. Warnier-Orr Diagrams of Auerback Computer Programming in Management. Auerback Publishers, Inc., 1980.
- 7 Kernighan, B., and Richie, D. The C Programming Language. Bell Labs., Murray Hill, New Jersey, 1978.
- 8 Lesk, M. Typing Documents on the UNIX System : Using the -ms Macros with Troff and Nroff . Bell Laboratories 1978.
- 9 Miller, K. An Interactive Bibliographic Reference System. Master Report, Computer Science Department, Kansas State University 1984.
- 10 Orr, K. Introducing Structured System Design. Infotech International Limited, Maidenhead, England, 1980.
- 11 Pressman, R. Software Engineering : A Practitioner's Approach. McGraw Hill, Inc., 1982.
- 12 Schustack, S. Variations in C. Microsoft Press, Bellevue, Washington, 1985.
- 13 Sobell, M. A Practical Guide to the UNIX System. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1984.

APPENDIX A USER'S MANUAL

1. INTRODUCTION

The REFWELL bibliographic system uses a relational data base and allows the storage, retrieve, and format, etc. the references the user is dealing with.

It is a menu-driven system. On-line help messages are available on the menus. Help messages explicitly describe and explain the functions on each menu. Because of this, there will not be detailed descriptions of the individual function in this manual.

This manual assumes that the user knows how to boot the personal computer, how to format a blank diskette, how to copy files from one diskette to another, and how to print the formatted bibliographic listing.

The following sections will tell the user how to get started using REFWELL, how to use the menus, what the meanings of the messages are, and what the precautions that the user should be aware of.

2. GETTING STARTED WITH REFWELL

2.1 WHAT THE USER NEEDS TO USE REFWELL

To use REFWELL, the user needs the following equipment:

- an IBM or IBM compatible personal computer
- the REFWELL.EXE file
- the COMMAND.COM and CONFIG.SYS from the MS-DOS/PC-DOS

- a supply of blank, formatted diskettes

2.2 THE STEPS TO USE REFWELL

- Boot the personal computer with a system disk. The system disk should have a copy of CONFIG.SYS which should contain the following statement: "device=ANSI.SYS".
- copy COMMAND.COM and REFWELL.EXE into the blank, formatted diskette.
- Insert the diskette containing both COMMAND.COM and REFWELL into drive A and type "REFWELL".

3. FUNCTION MODES

In this section, all the menus and messages are described in detailed. This would be very helpful for the user to use REFWELL efficiently and correctly.

3.1 MAIN

After REFWELL was loaded, an introduction is displayed which briefly describes the system functions and the features of REFWELL. By hitting any key, the message "No data file exists!" will be displayed if REFWELL is being used for the first time. Otherwise, the directory of the ledger file will be listed on the screen. The directory will contain the file name and file size.

By hitting any key again, the main menu will be brought up to the screen (see Figure 18). Each function on the menu has an associated number. The user can select the function by entering the appropriate number key.

0> HELP.
1> CREATE.
2> APPEND.
3> CHANGE.
4> RETRIEVE.
5> FORMAT.
6> JOIN.
7> EXIT to DOS.

Enter your choice :

Figure 18: MAIN MENU

3.2 CREATE MODE

Upon the selection of "1" from the main menu, the message "Enter file name : " is displayed. An invalid input of file name, such as only entering a return, will bring up the message "Invalide file name" and redisplay the main menu. If it is an existing file, no further action will go on and the main menu will be displayed.

A valid file name will bring up the menu (see Figure 19) showing the user the options for the screen entry formats or exit the create mode.

[J] = Journal, [B] = Book, [X] = exit

Enter your choice :

Figure 19: Screen Entry Format Selection or Exit

Upon the selection of "J" (or "j") and "B" (or "b"), the screen entry format for a journal and book information is displayed respectively (see Figure 8, 9).

The message "Save or Not (Y/N)? " will be displayed upon the selection of "X" (or "x"). All the answers other than "N" or "n" will be treated as "YES" and all the entries will be saved. If the answer is "N" (or "n"), the message "Are your sure (Y/N)? " is displayed. If the answer of this message is "Y" (or "y"), REFWELL will discard all the entries and redisplay the main menu.

To exit the screen entry by moving the cursor down to the last field of the screen format and entering a carriage return. Again, the menu for selecting screen format (see Figure 19) is redisplayed. The user can select a format for the next entry or exit the creation mode.

3.3 APPEND MODE

All the menus and messages are the same as in the CREATE mode.

3.4 CHANGE MODE

Upon the selection of "3" from the main menu, the user is prompted for the file name. A valid file name will bring up the

menu shown in Figure 20.

```
0> HELP.  
1> EDITING.  
9> RETURN.
```

Enter your choice :

Figure 20: Menu for Editing the References

By entering "1" to start the editing. The references will be shown on the associated screen format. The message "Hit [Esc] to Exit; [PgDn] to next page; [PgUp] to previous page" is displayed on the bottom of each screen format showing the the available function keys. [PgUp] and [PgDn] will bring up the previous and next reference on the screen respectively. By hitting [Esc], the user will be prompted for selecting the save option. If the user does not issue the save selection, all the entries will not be save and the file remain unchanged.

3.5 RETRIEVE MODE

Upon the selection of "4" from the main menu, the user is prompted for entering file name. A valid file name will bring up the criteria screen format (see Figure 13).

To exit the screen entry, the user needs to move the cursor down to the last field of the format and hit the carriage return.

If no reference found by searching, the message "No Record Found" is displayed and the main menu is redisplayed. If there is

any reference been selected, the following menu (Figure 21) is displayed.

```
1> QUICK VIEW.
2> TO NEW FILE.
3> TO DELETE.
9> RETURN.
```

Enter your choice :

Figure 21: Menu for Viewing, Extracting and Deleteng
Selected References

The user can select "1" for a quick view of the selected references. The references will be displayed by a short form (Figure 22), where "Rcd#" stands for the record number, "First Author Name" is the name of the author listed in the first place in a reference, "type" is the type of the publication, which is either a "J" for journal or "B" for book, "Topic or Book Name" is the topic of the publication or the name of a book, and "Date" is the publication date. After the displaying, the previous menu (Figure 21) will be displayed again.

Rcd#	First Author Name	type	Topic or Book Name	Date
------	-------------------	------	--------------------	------

Figure 22: Heading of the Quick View Listing

Upon the selection of "2" (Figure 21), the user is prompted for entering the file name for the output file. After extracting the selected references, the main menu is displayed.

Upon the selection of "3", the user is prompted by the message "Are you sure (Y/N) ?". If "Y" or "y" is entered, the selected references will be deleted and the main menu is redisplayed. If "N" or "n" is entered, the previous menu (Figure 21) is redisplayed.

3.6 FORMAT MODE

Upon the selection of "5" from the main menu, the user is prompted for entering the file name. A valid file name will bring up the menu shown in Figure 23.

```
0> HELP.  
1> CONSOLE.  
2> FILE.  
3> SAMPLE.  
9> RETURN.
```

Enter your choice :

Figure 23: Menu for Selecting the Destination
for the Output Listing

Selection "1" will display the formatted listing to the console, and then redisplay the menu (see Figure 23). Selection "2" will bring up the message to the user for entering the output file name and direct the formatted listing to the file. The listing will not be displayed on the screen. Selection "3" will display a sample listing showing the two formats which are used in the ACM Computing Surveys and Transactions of the Office

Information Systems.

Upon the selection "1" or "2", the menu for selecting the format is displayed is shown in Figure 24.

```
1> FORMAT1.  
2> FORMAT2.  
9> RETURN.
```

Enter your choice :

Figure 24: Menu for Selecting the Format

After selecting the format, the menu for selecting the order of the formatted listing is displayed (Figure 25). Any selection other than "1" or "2" is treated as selection "3".

```
1> ASCENDING ORDER.  
2> DESCENDING ORDER.  
3> RETURN( Unsort ).
```

Enter your choice :

Figure 25: Menu for selecting the Order of the Listing

After either displaying the formatted listing on the screen or writing the listing into a file, the main menu is redisplayed.

3.6 JOIN MODE

Upon the selection of "6" from the main menu, the user will be prompted by the message "Enter the first file name : ", "Enter the second file name : ", and "Enter the target file name : ".

The first and second file names are the names of the two files to be joined, and the target file name is the name of the file that will contain the result of the join.

After entering the file names, the user is prompted by the following message (see Figure 26) for verifying entries.

First file name is _____

Second file name is _____

Target file name is _____

Are your sure (Y/N)?

Figure 26: Messages for Verifying the File Names for Join

If the answer is "Y" or "y", the first and second file names will be checked against the ledger file. If they are existing and not identical files, the join function will proceed. Otherwise, the message "file(s) not found!" will be displayed and the main menu will be redisplayed.

4. MISCELLANIES:

1. Any selection or verification requires only one keystroke. This means that entering the number or the required character without a following carriage return. Extra keystrokes will be considered as the input for the next menu, query, or screen entry.

2. When the user is prompted for a file name, the user needs to supply the file name without the extension and followed by a carriage return.

3. While in the editing mode, the user is recommended not to leave any extra leading or trailing blanks. Only four arrow keys are eligible to move the cursor. Hitting space bar will be treated as entering a white space. Any entry of the undefined function keys won't be considered as an input.

4. REFWELL can only access the data files whose names are included in the ledger file. Any data files copied from other diskettes or not included in the ledger file won't be accessed by REFWELL.

5. It is very important for the user to know that do not use a word processor to edit the data files, ledger file, and dictionaries. This may jeopardize the whole system.

6. To answer a query, "Y" or "y" represents yes, "N" or "n" represents no, and "X" or "x" represents exit.

7. After displaying the help messages for a menu, the

previous menu will be redisplayed.

8. The selection of "return" (number 9) from a menu will return the control to the function MAIN and the main menu will be displayed.

```

/* HEADER.H defines the constants and structures. */
#define LIST      6
#define TRUE      1
#define FALSE     0
#define MAXLEN    75
#define MAXLINE 127

struct NameDic
{
    char *name;
    int  which;
    int  where;
    char what;
    char *date;
    struct NameDic *left;
    struct NameDic *right;
};

struct WordDic
{
    char *word;
    int  which;
    int  where;
    struct WordDic *left;
    struct WordDic *right;
};

struct Ldgr
{
    char filename[20];
    int  filno;
    int  filsize;
};

struct KeywordDic
{
    char word[30];
    int  which;
    int  where;
    struct KeywordDic *next;
};

struct AuthorDic
{
    char name[30];
    int  which;
    int  where;
    char what;
    char date[20];
    struct AuthorDic *next;
};

struct Book
{
    char BookName[128];

```

```

    char Publisher[50];
    char City[50];
};

struct Journal
{
    int KeywordCnt;
    char KeywordList[LIST][30];
    char J_Name[128];
    char Topic[128];
};

struct Reference
{
    int Namecnt;
    char NameList[LIST][30];
    char B_or_J;
    char Date[20];
    char Pages[20];
    struct Book B_Part;
    struct Journal J Part;
    struct Reference *next;
};

struct Ref
{
    int Namecnt;
    char NameList[LIST][30];
    char B_or_J;
    char Date[20];
    char Pages[20];
    struct Book B_Part;
    struct Journal J Part;
};

struct temp
{
    int where;
    struct temp *left;
    struct temp *right;
};

struct Criteria
{
    int Namecnt;
    int Datecnt;
    int Wordcnt;
    char from[15];
    char to[15];
    char NameList[6][40];
    char WordList[6][40];
};

```

```

/* MAIN.C includes all the required header files and declares all
   the global variables.
*/
#include <stdio.h>
#include <ctype.h>
#include <b:header.h>
#define cursor(x,y) fprintf(stderr, "\33[%d;%dH", x,y)
struct AuthorDic *Aroot, *atemp;
struct KeywordDic *Kroot, *ktemp;
struct Reference *Head, *rtemp;
struct Ldgr *lgr;
struct Ref *Ary;
struct Criteria *in;
struct temp *root;
int RcdFound;
int *TempStore; /* ..... keep record # found by searching */
int gg; /* ..... used as index for TempStore */
int NextFileNo; /* ..... for new file */
int NotSave; /* ..... 1 = yes; 0 = no */
int ActCode; /* ..... 65=append; 67=create */
int ndx; /* ..... count new entries */
int rdct;
int again=0,j=0,i=0,x=0,y=0;
int *Rcd; /* .... points to the current working record */
char *fn; /* ..... carries input file name */
char *date;
char *page;
char temps[19][45];
char *aname="AUTHOR.DIC";
char *lname="LEDGER";
char *DefaultDate="1980";
char tempj[20][70]; /* ..... journal screen format */
char tempb[15][70]; /* ..... book screen format */
main()
{
    int done=FALSE;
    Introduction();
    Listing();
    while (!done)
    {
        ShowMenu();
        if (Process(Choose())) ;
        else done=TRUE;
    } /* of while */
} /* of main */

Listing()
{
    FILE *lp,*fopen();
    char *name,count=1;
    int no,size;
    system("cls");
    if (lp=fopen(lname,"r"))
    {
        rewind(lp);

```

```

printf("This diskette contains :\n\n");
name=(char *)calloc(15,sizeof(char));
while (!feof(lp))
{
    fscanf(lp,"%d %s %d\n",&no,name,&size);
    if (!feof(lp))
    {
        if (count%4==0)
            printf("\t%s %d\n",name,size);
        else
            printf("\t%s %d",name,size);
        } /* of outer if */
        ++count;
    } /* of while */
    NextFileNo=count;
    fclose(lp);
}
else
{
    cursor(12,25);
    printf("No data file exists!");
}
printf("\n\n");
system("pause");
} /* of Listing */

```

ShowMenu() /* display main menu */

```

{
    system("cls");
    cursor(2,30);
    printf("0> HELP.");
    cursor(4,30);
    printf("1> CREATE.");
    cursor(6,30);
    printf("2> APPEND. ");
    cursor(8,30);
    printf("3> CHANGE. ");
    cursor(10,30);
    printf("4> RETRIEVE. ");
    cursor(12,30);
    printf("5> FORMAT. ");
    cursor(14,30);
    printf("6> JOIN. ");
    cursor(16,30);
    printf("9> EXIT to DOS.");
    cursor(20,30);
    printf("Enter your choice : ");
}

```

Choose() /* read the choice */

```

{
    int c;
    c=getch();
    putchar(c);
    return(c);
}

```

```

    } /* of Choose */

Process(code) /* process the choice */
int code;
{
    switch(code)
    {
        case '0' : return(MainHelp());
        case '1' : return(create());
        case '2' : return(append());
        case '3' : return(change());
        case '4' : return(search());
        case '5' : return(form());
        case '6' : return(join());
        case '9' : system("cls");
                    return(0);
        default : break;
    } /* of switch */
} /* of Process */

MainHelp() /* display help messages */
{
    system("cls");
    printf("\t* * * * * HELP * * * * *\n\n");
    printf("\t1> CREATE: Create a .BBF file.\n\n");
    printf("\t2> CHANGE: The content of a data file can be ");
    printf("changed. Each reference will be shown on the ");
    printf("\t\t screen with associated format.\n\n");
    printf("\t3> FORMAT: Two defined formats are available ");
    printf("\t\t to format\n\n");
    printf("\t\t a .BBF file into a .FMT file.\n\n");
    printf("\t4> JOIN:   Join two .BBF files onto another new ");
    printf("\t\t data file.\n\n");
    printf("\t5> RETRIEVE: Search reference from .BBF file ");
    printf("\t\t by a given\n\n");
    printf("\t\t criteria. The result of retrieval ");
    printf("\t\t can be deleted, \n\n");
    printf("\t\t extracted into a new .BBF file or displayed ");
    printf("\t\t on screen for a quick view.\n\n");
    system("pause");
    return(1);
} /* of MainHelp */

Introducation()
{
    printf("\t* * * * * INTRODUCTION * * * * *\n\n");
    printf("\t1> REFWELL is a menu-driven system with full-\n\n");
    printf("\t\t -screen data entry and on-line help .\n\n");
    printf("\t2> Function selection from a menu only needs a\n\n");
    printf("\t\t keystroke without a following return.\n\n");
    printf("\t3> REFWELL provides the functions for : \n\n");
    printf("\t\t Creating a data file.\n\n");
    printf("\t\t Updating the content of a data file.\n\n");
    printf("\t\t Appending additional records to a data file.\n\n");

```



```

printf("\t\t Seletcting references by a given criteria.\n");
printf("\t\t Formatting the data file into two defined \n");
printf("\t\t formats in ascending or descending order.\n");
printf("\t\t Joining two data files into a new file.\n\n");
printf("\t4> Each function processes one file at a time,\n");
printf("\t so after selecting a function, a file name \n");
printf("\t should be provided with a following return.\n");
printf("\n\t5> If there is any question, see the help\n");
printf("\t messages on each menu or the user's manual\n");
printf("\t on the appendix of the author's report.\n\n");
printf("\t6> To exit the screen entry formats by moving\n");
printf("\t the cursor down to the last line and hitting ");
printf("\n\t the carriage return.\n\n");
printf("\tWarnings : \n");
printf("\t\t Do not exit REFWELL abnomally, this might ");
printf("\t\t cause lost of data.\n");
printf("\t\t Do not edit data files, ledger file, and ");
printf("\t\t dictionaries for any purpose.\n\n");
printf("\n\n");
system("pause");
} /* of Introduction */

```

```

/* APPEND.C appends additional records to a data base file. */
#include <stdio.h>
#include <b:header.h>
extern struct Ldgr *lgr;
extern char *lname;
extern int ndx;
extern int ActCode;
append()
{
    int count=0;

    ActCode=65;
    Initialization();
    ReadFilename();
    if (CheckFileName()) /* ..... valid filename */
    {
        if ((count=ReadAConsole())>=1) /* .. one or more entry */
        {
            if (WantSave())
            {
                Writing();
                AfterAppend();
            }
            else Release(); /* free all the global variables */
        }
    }
    else FileNotFound();
    return(1);
} /* of Append */

ReadAConsole() /* read reference from console */
{
    int done=FALSE;
    char c;
    ndx=lgr->filesize+1;
    while (!done)
    {
        ShowChoice();
        c=getch();
        putch(c);
        if ((c==88)||(c==120)) done=TRUE;
        else
        {
            if ((c==74)||(c==106)) /* Journal */ ReceiveJ();
            else ReceiveB();
            ++ndx;
        }
    } /* of while */
} /* of ReadConsole */

AfterAppend() /* take care of the rest after appending */
{
    ModiLedger();
    SortName();
} /* AfterAppend */

```

```

ModiLedger() /* modify the file entry from the ledger file */
{
    FILE *lp,*tp,*fopen();
    int cond,no,size;
    char *name,*tmp;

    tmp="ldger";
    lp=fopen(lname,"r");
    tp=fopen(tmp,"w");
    name=(char *)calloc(30,sizeof(char));
    rewind(lp);
    while (!feof(lp))
    {
        fscanf(lp,"%d %s %d\n",&no,name,&size);
        if (((cond=Compare(lgr->filename,name))==0)&&
            (!feof(lp))) /* found */
            fprintf(tp,"%d %s %d\n",no,name,ndx-1);
        else { if (!feof(lp))
            fprintf(tp,"%d %s %d\n",no,name,size);}
    }
    fclose(lp);
    fclose(tp);
    system("del LEDGER");
    rename(tmp,lname);
}/* ModiLedger */

```

```

/* CHANGE.C performs the functions which allow the user to update
the content of a data base file record by record.
*/
#include <stdio.h>
#include <b:header.h>
extern struct Ldgr *lgr;
extern struct Ref *Ary;
extern int x,y,i,j,ndx;
extern char *DefaultDate;
extern char *aname;
extern char tempj[20][70];
extern char tempb[15][70];
static char word[6][30];
static char *cpage;
static char *cdate;
change()
{
    ReadFileName();
    if (CheckFileName())
    {
        Fresh();
        ScanFile();
        if (ReadChanges(ReadChoice()))
        {
            if (WantChanges()) UpdateChanges();
            else FreeVars();
        }
        /* of if file exists */
    }
    else FileNotFound();
    return(1);
} /* of change */

FreeVars()
{
    free(Ary);
    free(cdate);
    free(cpage);
} /* of FreeVars */

ReadChoice()
{
    int right=FALSE,c;
    while (!right)
    {
        system("cls");
        cursor(9,25);
        printf("0> HELP.");
        cursor(11,25);
        printf("1> EDITING.");
        cursor(13,25);
        printf("9> RETURN.");
        cursor(20,20);
        printf("Enter your choice (0, 1, or 9) : ");
        c=getch();
        putch(c);
    }
}

```

```

        if ((c!=49)&&(c!=50)&&(c!=57))      ChangeHelp();
        else if (c==48)      ChangeHelp();
        else      return(c);
    } /* of while */
} /* of ReadChoice */

```

```

ChangeHelp()
{
    system("cls");
    printf("\t* * * * *      HELP      * * * * *\n");
    printf(" * * * * *\n\n");
    printf("\t\t1> CHANGE allows you to change the content \n");
    printf("of the data\n");
    printf("      file. All the information of a record \n");
    printf("will be shown\n");
    printf("      on the screen one at a time. All the \n");
    printf("typing will be\n");
    printf("      treated as the changes. It is recommended\n");
    printf("that user\n");
    printf("      should not leave any unnecessary blank \n");
    printf("to keep the\n");
    printf("      entries correct.\n\n");
    printf("\t\t2> Four arrow keys are defined for user to \n");
    printf("move cursor\n");
    printf("      on the screen. User should not use any \n");
    printf("other unde-\n");
    printf("      fined function key for any purpose.\n\n");
    printf("\t\t3> Hitting PgUp will bring you the next \n");
    printf("record on the\n");
    printf("      screen. Likewise, PgDn will bring you \n");
    printf("the previous\n");
    printf("      record on the screen. When finishing \n");
    printf("editing, hit\n");
    printf("      ESC to exit the edit mode.\n\n");
    system("pause");
} /* ChangeHelp */

```

```

ReadChanges(code)
int code;
{
    int status,done=FALSE;

    if (code==57) return(0);
    else if (code==49) /* editing */
    {
        ndx=0; /* init the value of global index of Ary */
        while (!done)
        {
            CDisplay();
            if ((status=AcceptConsole())==0) /* ESC */ done=TRUE;
        } /* of while */
        return(1);
    }
} /* of ReadChanges */

```

```

AcceptConsole()
{
    if (Ary[ndx].B_or_J=='J')    return(XReceiveJ());
    else return(XReceivB());
} /* of AcceptConsole */

CDisplay() /* copy each reference into a temp structure which is
           be copied onto the screen. */
{
    if (Ary[ndx].B_or_J == 'J')
    {
        Copy2Jtemp();
        ShowJout();
    }
    else
    {
        Copy2Btemp();
        ShowBout();
    }
} /* of CDisplay */

InitWord()
{
    int i,j;
    for (i=0;i<6;i++)
        for (j=0;j<30;j++)
            word[i][j]=0;
} /* of InitWord */

Copy2Jtemp() /* copy the content of a reference into a Jtemp
             which is to be copied onto to screen */
{
    int len,i,j,k;
    /* size of tempj is [20][70]
       0-5 are names;
       7,8 are topic;
       10,11 are journal name;
       13 is date;
       15-17 are keywords;
       19 is pages; */
    InitJ();
    InitWord();
    for (i=0;i<Ary[ndx].Namecnt;i++)
        strcpy(tempj[i],Ary[ndx].NameList[i]);
    len=strlen(Ary[ndx].J_Part.Topic);
    if (len > 60)
    {
        for (i=0;i<60;i++)
            tempj[7][i]=Ary[ndx].J_Part.Topic[i];
        for (j=0,i=60;i<len;i++,j++)
            tempj[8][j]=Ary[ndx].J_Part.Topic[i];
    }
    else strcpy(tempj[7],Ary[ndx].J_Part.Topic);
    len=strlen(Ary[ndx].J_Part.J_Name);

```

```

if (len > 60)
{
    for (i=0;i<60;i++)
        tempj[10][i]=Ary[ndx].J_Part.J_Name[i];
    for (j=0,i=60;i<len;i++,j++)
        tempj[11][j]=Ary[ndx].J_Part.J_Name[i];
}
else strcpy(tempj[10],Ary[ndx].J_Part.J_Name);
strcpy(tempj[13],Ary[ndx].Date);
x=15; y=11;
for (i=0;i<Ary[ndx].J_Part.KeywordCnt;i++)
    if (i%2==0) /* right side */
    {
        if (i==2)
            for (k=37,j=0;k<57;j++,k++)
            {
                word[i][j]=Ary[ndx].J_Part.KeywordList[i][j];
                tempj[15][k]=Ary[ndx].J_Part.KeywordList[i][j];
            }
        if (i==4)
            for (k=37,j=0;k<57;j++,k++)
            {
                word[i][j]=Ary[ndx].J_Part.KeywordList[i][j];
                tempj[16][k]=Ary[ndx].J_Part.KeywordList[i][j];
            }
        if (i==6)
            for (k=37,j=0;k<57;j++,k++)
            {
                word[i][j]=Ary[ndx].J_Part.KeywordList[i][j];
                tempj[17][k]=Ary[ndx].J_Part.KeywordList[i][j];
            }
    }
else /* left side */
{
    if (i==1)
        for (k=0;k<30;k++)
        {
            word[i][k]=Ary[ndx].J_Part.KeywordList[i][k];
            tempj[15][k]=Ary[ndx].J_Part.KeywordList[i][k];
        }
    if (i==1)
        for (k=0;k<30;k++)
        {
            word[i][k]=Ary[ndx].J_Part.KeywordList[i][k];
            tempj[16][k]=Ary[ndx].J_Part.KeywordList[i][k];
        }
    if (i==3)
        for (k=0;k<30;k++)
        {
            word[i][k]=Ary[ndx].J_Part.KeywordList[i][k];
            tempj[17][k]=Ary[ndx].J_Part.KeywordList[i][k];
        }
}
strcpy(tempj[19],Ary[ndx].Pages);
} /* of Copy2temp */

```

```
Copy2Btemp() /* copy the content of a book into Btemp which is to
              be copied onto the screen */
```

```
{
    int i,j,len;

    InitB();
    cpage=(char *)calloc(15,sizeof(char));
    cdate=(char *)calloc(15,sizeof(char));
    for (j=0,i=0;i<Ary[ndx].Namecnt;i++,j++)
        strcpy(tempb[j],Ary[ndx].NameList[i]);
    len=strlen(Ary[ndx].B_Part.BookName);
    if (len > 60)
    {
        for (i=0;i<60;i++)
            tempb[7][i]=Ary[ndx].B_Part.BookName[i];
        for (j=0,i=60;i<len;i++,j++)
            tempb[8][j]=Ary[ndx].B_Part.BookName[i];
    }
    else strcpy(tempb[7],Ary[ndx].B_Part.BookName);
    for (i=0;i<15;i++)
    {
        cdate[i]=Ary[ndx].Date[i];
        tempb[10][i]=Ary[ndx].Date[i];
    }
    for (j=0,i=48;i<63;i++,j++)
    {
        cpage[j]=Ary[ndx].Pages[j];
        tempb[10][i]=Ary[ndx].Pages[j];
    }
    strcpy(tempb[12],Ary[ndx].B_Part.City);
    strcpy(tempb[14],Ary[ndx].B_Part.Publisher);
} /* Copy2Btemp */
```

```
ShowJout() /* display a journal on to the screen */
```

```
{
    int i,j;
    system("cls");
    cursor(0,0);
    printf("FILE NAME : %s",lgr->filename);
    cursor(1,0);
    printf("RECORD # : %d",ndx+1);
    RefFormat();
    x=3; y=11;
    for (i=0;i<6;i++)
    {
        cursor(x++,y);
        printf("%s",tempj[i]);
    } /* print names */
    cursor(10,11);
    printf("%s",tempj[7]); /* Topic */
    cursor(11,11);
    printf("%s",tempj[8]);
    cursor(13,11);
    printf("%s",tempj[10]); /* Journal */
}
```



```

        cursor(14,11);
        printf("%s",tempj[11]);
        cursor(16,11);
        printf("%s",tempj[13]);
        cursor(18,11);
        printf("%s",word[0]);
        cursor(18,37);
        printf("%s",word[1]);
        cursor(19,11);
        printf("%s",word[2]);
        cursor(19,37);
        printf("%s",word[3]);
        cursor(20,11);
        printf("%s",word[4]);
        cursor(20,37);
        printf("%s",word[5]);
        cursor(22,11);
        printf("%s",tempj[19]);
        PrintChoice();
    } /* of ShowJout */

```

```

ShowBout() /* display book on the screen */

```

```

{
    int i,j;
    system("cls");
    cursor(0,0);
    printf("FILE NAME : %s",lgr->filename);
    cursor(1,0);
    printf("RECORD # : %d",ndx+1);
    BookFormat();
    x=3;
    y=11;
    for (i=0;i<6;i++)
    {
        cursor(x++,y);
        printf("%s",tempb[i]);
    } /* print names */
    cursor(10,12);
    printf("%s",tempb[7]);
    cursor(11,12);
    printf("%s",tempb[8]);
    cursor(13,8);
    printf("%s",cdate);
    cursor(13,48);
    printf("%s",cpage);
    cursor(15,8);
    printf("%s",tempb[12]);
    cursor(17,13);
    printf("%s",tempb[14]);
    PrintChoice();
} /* of ShowBout */

```

```

PrintChoice() /* display the definitions of the function key */

```

```

{
    cursor(24,0);

```

```

    printf("Hit [ESC] to Exit; [PgDn] to next page; [PgUp] to
           previous page");
    cursor(3,11);
} /* of PrintChoice */

WantChanges() /* read the decision to update or not */
{
    int c,sure=FALSE;
    while (!sure)
    {
        system("cls");
        cursor(12,20);
        printf("Save or Not (Y/N)? ");
        c=getch();
        putch(c);
        if ((c==78)|| (c==110)) /* No */
        {
            system("cls");
            cursor(12,25);
            printf("Are you sure (Y/N)? ");
            c=getch();
            putch(c);
            if ((c==89)|| (c==121)) return(0); /* Yes */
        }
        else return(1);
    } /* of while */
} /* of WantSave */

UpdateChanges() /* update dictionaries, ledger file, and the data
                base file */
{
    FILE *ap,*tap,*fp,*fopen();
    int i;
    char *tname;
    tname="tmp";
    ap=fopen(aname,"r");
    tap=fopen(tname,"a");
    fp=fopen(lgr->filename,"w");
    Delold(ap,tap);
    fclose(ap);
    Rebuild(tap,fp);
    fclose(fp);
    fclose(tap);
    system("del AUTHOR.DIC");
    rename(tname,aname);
    SortName();
} /* of UpdateChanges */

Delold(ap,tap) /* delete old names from author dictionary */
FILE *ap;
FILE *tap;
{
    char *name,*date,what;
    int c=0,which,where;
    rewind(ap);

```

```

name=(char *)calloc(30,sizeof(char));
date=(char *)calloc(30,sizeof(char));
while (!feof(ap))
{
    fscanf(ap,"%d %s %d %ls %s\n",
           &which,name,&where,&what,date);
    if ((which!=lgr->filno)&&(!feof(ap)))
    {
        ++c; fprintf(tap,"%d %s %d %c %s\n",
                     which,name,where,what,date);
    }
}
free(name);
free(date);
} /* Delold */

Rebuild(ap,ip) /* rebuild the author dictionary */
FILE *ap;
FILE *ip;
{
    int i,h;
    char what,*date;
    for (i=0;i<lgr->filesize;i++)
    {
        strcpy(date,Ary[i].Date);
        fprintf(ip,"%c",Ary[i].B_or_J);
        fprintf(ip,"\n%d",Ary[i].Namecnt);
        for (h=0;h<Ary[i].Namecnt;h++)
        {
            fprintf(ip,"\n%s",Ary[i].NameList[h]);
            MarkBlank(Ary[i].NameList[h]);
            MarkBlank(date);
            fprintf(ap,"\n%d %s %d %c %s",lgr->filno,
                    Ary[i].NameList[h],(i+1),Ary[i].B_or_J,date);
        }
        if (Ary[i].B_or_J == 'J')
        {
            fprintf(ip,"\n%d",Ary[i].J_Part.KeywordCnt);
            for (h=0;h<Ary[i].J_Part.KeywordCnt;h++)
                fprintf(ip,"\n%s",Ary[i].J_Part.KeywordList[h]);
            fprintf(ip,"\n%s",Ary[i].J_Part.J_Name);
            fprintf(ip,"\n%s",Ary[i].J_Part.Topic);
        }
        else
        {
            fprintf(ip,"\n%s",Ary[i].B_Part.BookName);
            fprintf(ip,"\n%s",Ary[i].B_Part.City);
            fprintf(ip,"\n%s",Ary[i].B_Part.Publisher);
        } /* of else */
        fprintf(ip,"\n%s",Ary[i].Pages);
        fprintf(ip,"\n%s\n",Ary[i].Date);
    } /* of for loop */
} /* of Rebuild */

XReceiveJ() /* read journal from the screen entry format */

```

```

{
    int c,more=1;
    x=3; y=11;
    cursor(x,y);
    while (more)
    {
        if (((c=getch()) == 13) && (x == 22))
            { ++ndx; more = 0; }
        else if (c==13)
            { XJRet(); cursor(x,y); }
        else if (c==27)
            { XAssignJ(); return(0); }
        else if (c==0) /* Arrow Keys */
            {
                c=getch();
                if ((c==81)|| (c==73)) /* PgDn or PgUp */
                {
                    XAssignJ();
                    if (c==81)
                    {
                        if ((ndx+1)==lgr->filesize) /*nothing after*/
                            putch(7);
                        else ++ndx;
                    }
                    else
                    {
                        if (ndx==0) putch(7);/* nothing before */
                        else --ndx;
                    }
                }
                return(1);
            }
        else if (c==72) XJUp();/* UpArrow */
        else if (c==80) XJDown();/* DnArrow */
        else if (c==75) XJLeft();/* LfArrow */
        else if (c==77) XJRight();/* RtArrow */
        cursor(x,y);
        } /* of Arrow Keys */
        else XJEat(c);
    } /* of while more */
    XAssignJ();
    return(1);
} /* of XReceiveJ */

XJRight() /* right arrow key is pressed */
{
    if ((x>=3)&&(x<=8))
    {
        if (y<40) ++y;
        else if ((y==40)&&(x==8))
            { x=x+2; y=11; }
        else if (y==40)
            { ++x; y=11; }
    } /* of 3 to 8 */
    else if ((x==10)|| (x==13))
    {

```

```

        if (y<70) ++y;
        else { ++x; y=11; }
    } /* of 10,13 */
else if ((x==11)|| (x==14))
{
    if (y<70) ++y;
    else { x=x+2; y=11; }
    } /* of 11,14 */
else if ((x==16)|| (x==22))
{
    if (y<30) ++y;
    else if (y==30)
    {
        if (x==16)
        { x=x+2; y=11; }
        else putch(7);
    }
    } /* of 16,22 */
else if ((x==18)|| (x==19)|| (x==20))
{
    if (y==30) y=37;
    else if (y==56)
    {
        if (x==20) { x=22; y=11; }
        else { y=11; ++x; }
    }
    else if ((y<31)|| (y<56)) ++y;
    } /* of 18,19,20 */
} /* of XJRight */

XJUp() /* up arrow key is pressed */
{
    if (((x>3)&&(x<=8))|| (x==11)|| (x==14)|| (x==20)|| (x==19))
    { --x; y=11; }
    else if ((x==10)|| (x==13)|| (x==16)|| (x==18)|| (x==22))
    { x=x-2; y=11; }
    else if (x==3)
    { x=24; y=11; }
    } /* of XJUp */

XJDown() /* down arrow key is pressed */
{
    if (((x<8) && (x>=3))|| (x==10)|| (x==13)|| (x==18)|| (x==19))
    { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==14)|| (x==16)|| (x==20))
    { x=x+2; y=11; }
    else if (x==22) { x=3; y=11; }
    } /* of XJDown */

XJRet() /* return is pressed */
{
    if (((x>=3) && (x<=7)) || (x==10) || (x==13))
    { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==14)|| (x==16)|| (x==20))
    {

```

```

        if ((x==20)&&(y>=11)&&(y<=30)) y=37;
        else { x=x+2; y=11; }
    }
    else if ((x==18) || (x==19))
    {
        if ((y<=31) && (y>=11)) y=37;
        else if ((y>=37) && (y <=59))
            { ++x; y=11; }
    }
} /* of XJRet */

XJLeft() /* left arrow key is pressed */
{
    if ((x>3)&&(x<=8))
    {
        if (y>11) --y;
        else { --x; y=40; }
    } /* of 3 to 8 */
    else if (x==3)
    {
        if (y==11) putch(7);
        else --y;
    }
    else if ((x==10)|| (x==13))
    {
        if (y>11) --y;
        else
        {
            if (x==10) y=40;
            else y=70;
            x=x-2;
        }
    } /* of 10,13 */
    else if ((x==11)|| (x==14))
    {
        if (y>11) --y;
        else { --x; y=70; }
    } /* of 11,14 */
    else if ((x==16)|| (x==22))
    {
        if (y>11) --y;
        else
        {
            if (x==16) y=70;
            else y=56;
            x=x-2;
        }
    } /* of 16,22 */
    else if ((x==18)|| (x==19)|| (x==20))
    {
        if (y==37) y=30;
        else if (y==11)
        {
            if (x==18) { x=16; y=30; }
            else { y=56; --x; }
        }
    }
}

```

```

        }
        else if ((y>11)|| (y>37)) --y;
    } /* of 18,19,20 */
} /* of XJLeft */

XJEat(c) /* accept the legal characters */
char c;
{
    if (((x>=3)&&(x<=8))&&(y<=40)) ||
        ((x==10)|| (x==11)|| (x==13)|| (x==14))&&(y<=70)) ||
        ((x==16)&&(y<31)) ||
        ((x==22)&&(y<31))) XJShow(c);
    else if ((x==18)|| (x==19)|| (x==20))
    {
        if ((y==31)|| (y==57)) putch(7);
        else XJShow(c);
    }
    else putch(7);
} /* of XJEat */

XJShow(c) /* display the characters onto the screen, move the
           cursor, and assign them into tempj[][]
           */
char c;
{
    putch(c);
    i=x-3;
    j=y-11;
    tempj[i][j++]=c;
    cursor(x,++y);
} /* of JShow */

XAssignJ() /* copy from tempj[][] to the Ary[] */
{
    int wordcnt=0,namecnt=0,len,h,v,l,k;
    char *stpblk(),*big,word[30],*p;
    system("cls");
    Ary[ndx].B_or_J='J';
    strcpy(Ary[ndx].Date,stpblk(tempj[13]));
    for (k=0;k<6;k++)
    {
        p=stpblk(tempj[k]);
        if ((len=strlen(p)) >=1)
        {
            ++namecnt;
            strcpy(Ary[ndx].NameList[k],p);
        }
    }
    Ary[ndx].Namecnt=namecnt;
    big=(char *)calloc(MAXLINE,sizeof(char));
    strcat(big,stpblk(tempj[7]));
    strcat(big,stpblk(tempj[8]));
    strcpy(&Ary[ndx].J_Part.Topic,big);
    free(big);
    big=(char *)calloc(MAXLINE,sizeof(char));

```

```

strcat(big, stpblk(tempj[10]));
strcat(big, stpblk(tempj[11]));
strcpy(&Ary[ndx].J_Part.J_Name, big);
free(big);
for (v=15, h=0; v<18; v++) /* get keywords */
{
    for (k=0; k<20; k++)
        word[k] = tempj[v][k];
    strcpy(p, stpblk(word));
    if ((len=strlen(p))>1)
    {
        strcpy(Ary[ndx].J_Part.KeywordList[h], p);
        ++wordcnt;
    }
    ++h;
    for (k=26, l=0; k<57; l++, k++)
        word[l] = tempj[v][k];
    strcpy(p, stpblk(word));
    if ((len=strlen(p))>1)
    {
        strcpy(Ary[ndx].J_Part.KeywordList[h], p);
        ++wordcnt;
    }
    ++h;
}
Ary[ndx].J_Part.KeywordCnt=wordcnt;
strcpy(Ary[ndx].Pages, stpblk(tempj[19]));
} /* of XAssignJ */

```

XReceivB() /* read book entry */

```

{
    int c, more=TRUE;
    x=3;
    y=11;
    cursor(x, y);
    while (more)
    {
        if (((c=getch()) == 13) && (x == 17))
            { XAssignB(); more = 0; }
        else if (c==13) { XBRet(); cursor(x, y); }
        else if (c==27) { XAssignB(); return(0); }
        else if (c==0) /* Arrow Keys */
        {
            c=getch();
            if ((c==81) || (c==73)) /* PgDn or PgUp */
            {
                XAssignB();
                if (c==81)
                {
                    if ((ndx+1)==lgr->filesize) /* nothing after */
                        putch(7);
                    else ++ndx;
                }
            }
            else
            {

```



```

        if (ndx==0) putch(0);
        else --ndx;
    }
    return(1);
}
else if (c==72) XBUUp();/* UpArrow */
else if (c==80) XBDown();/* DnArrow */
else if (c==75) XBLeft();/* LfArrow */
else if (c==77) XBRight();/* RtArrow */
cursor(x,y);
} /* of Arrow Keys */
else XBEat(c);
} /* of while more */
XAssignB();
return(1);
} /* of ReceiveB */

XBRet() /* return is entered */
{
    if ((x>=3)&&(x<8)) { ++x; y=11; }
    else if (x==8) { x=10; y=12; }
    else if (x==10) { x=11; y=12; }
    else if (x==11) { x=13; y=8; }
    else if (x==13)
    {
        if (y<=24) y=48;
        else { x=15; y=8; }
    }
    else if (x==15)
    { x=17; y=13; }
} /* of XBReturn */

XBUP() /* up arrow key is entered */
{
    if ((x>3)&&(x<=8)) { --x; y=11; }
    else if (x==3) { x=17; y=13; }
    else if (x==11) { --x; y=12; }
    else if ((x==10)|| (x==13)|| (x==15)|| (x==17))
    {
        if (x==13) y=12;
        else if ((x==15)|| (x==17)) y=8;
        else if (x==10) y=11;
        x=x-2;
    }
} /* of XBUP */

XBDown() /* down arrow key is entered */
{
    if ((x<8)&&(x>=3)) { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==13)|| (x==15))
    {
        if (x==8) y=12;
        else if ((x==11)|| (x==13)) y=8;
        else if (x==15) y=13;
        x=x+2;
    }
}

```

```

    }
    else if (x==10) { x=11; y=12; }
    else if (x==17) { x=3; y=11; }
} /* of XBDOWN */

XBRight() /* right arrow key is entered */
{
    if ((x>=3)&&(x<=8))
    {
        if (y<40) ++y;
        else if ((y==40)&&(x==8))
            { x=10; y=12; }
        else if (y==40)
            { ++x; y=11; }
    }
    else if ((x==10)|| (x==11))
    {
        if (y<70) ++y;
        else if (y==70)
        {
            if (x==10) { x=11; y=12; }
            else { x=13; y=8; }
        }
    }
    else if (x==13)
    {
        if (((y>=8)&&(y<23)) || ((y>=48)&&(y<62))) ++y;
        else if (y==23) y=48;
        else if (y==62) { x=15; y=8; }
    }
    else if (x==15)
    {
        if (y<37) ++y;
        else if (y==37) { x=17; y =13; }
    }
    else if (x==17)
    {
        if (y<62) ++y;
        else putchar(7);
    }
} /* of XBRIGHT */

XBLeft() /* left arrow key is entered */
{
    if ((x>3)&&(x<=8))
    {
        if (y>11) --y;
        else if (y==11) { --x; y=40; }
    }
    else if (x==3)
    {
        if (y==11) putchar(7);
        else --y;
    }
    else if ((x==10)|| (x==11))

```

```

    {
        if (y>12) --y;
        else if (y==12)
        {
            if (x==10) { x=8; y=40; }
            else if (x==11) { x=10; y=70; }
        }
    }
    else if (x==13)
    {
        if (((y>8)&&(y<=23))||((y>48)&&(y<=62))) --y;
        else if (y==8) { x=11; y=70; }
        else if (y==48) y=23;
    }
    else if (x==15)
    {
        if (y>8) --y;
        else if (y==8) { x=13; y=62; }
    }
    else if (x==17)
    {
        if (y>13) --y;
        else if (y==13) { x=15; y=37; }
    }
} /* of XBLEFT */

XBEat(c) /* accept the legal characters */
char c;
{
    if (((x>=3)&&(x<=8))&&(y<=40)) ||
        ((x==10)|| (x==11))&&(y<=70)) ||
        ((x==15)&&(y<=37)) ||
        ((x==17)&&(y<=62))) XBSHow(c);
    else if (x==13)
    {
        if ((y==24)|| (y==63)) putch(7);
        else XBSHow(c);
    }
    else putch(7);
} /* of XBEat */

XBSHow(c) /* display the characters, move the cursor, and assign
           the characters into tempb[ ][ ]. */
char c;
{
    int i=0;
    putch(c);
    i=x-3;
    if ((x>=3)&&(x<=8)) j=y-11;
    else if ((x==10)|| (x==11)) j=y-12;
    else if ((x==13)|| (x==15)) j=y-8;
    else if (x==17) j=y-13;
    tempb[i][j++]=c;
    cursor(x,++y);
} /* of XBSHow */

```

```

XAssignB() /* copy from tempb[][] to Ary[] */
{
    int wordcnt=0,namecnt=0,len,h,v,m,l,k;
    char *stpblk(),*big,word[30],*p;
    system("cls");
    Ary[ndx].B_or_J='B';
    for (k=0;k<6;k++)
    {
        p=stpblk(tempb[k]);
        if ((len=strlen(p)) >=1)
        {
            ++namecnt;
            strcpy(Ary[ndx].NameList[k],p);
        }
    }
    Ary[ndx].Namecnt=namecnt;
    big=(char *)calloc(MAXLINE,sizeof(char));
    strcat(big,stpblk(tempb[7]));
    strcat(big,stpblk(tempb[8]));
    if ((len=strlen(big))>=1)
        strcpy(Ary[ndx].B_Part.BookName,big);
    else strcpy(Ary[ndx].B_Part.BookName,NULL);
    strcpy(Ary[ndx].B_Part.City,stpblk(tempb[12]));
    strcpy(Ary[ndx].B_Part.Publisher,stpblk(tempb[14]));
    for (k=0,h=0;h<15;h++,k++)
        word[h] = tempb[10][k];
    strcpy(p,stpblk(word));
    if ((len=strlen(p))>=1) strcpy(Ary[ndx].Date,p);
    else strcpy(Ary[ndx].Date,DefaultDate);

    for (k=40,h=0;k<54;h++,k++)
        word[h] = tempb[10][k];
    strcpy(p,stpblk(word));
    if ((len=strlen(p))>=1) strcpy(Ary[ndx].Pages,p);
    else strcpy(Ary[ndx].Pages,NULL);
} /* XAssignB */

```

```

/* SEARCH.C selects references by a given criteria from a data
base file. The result of retrieval can be either extracted
or deleted.
*/
#include <stdio.h>
#include <b:header.h>
extern struct Ref *Ary;
extern struct Ldgr *lgr;
extern struct temp *root;
extern struct Criteria *in;
extern int *TempStore;
extern int RcdFound;
extern int *Rcd;
extern int NextFileNo;
extern int i,j,x,y,gg;
extern char temps[19][45];
extern char *aname;
extern char *lname;
static int xx=0;
static char ofn[15]; /* output file name */
static int TimePeriod;
search() /* Search.c */
{
    ReadFileName();
    if (CheckFileName())
    {
        Rcd=(int *)calloc(lgr->filesize,sizeof(int));
        if (Searching(ReadCriteria())) /* if found anything */
        {
            Ary=(struct Ref *)calloc(lgr->filesize,
                                      sizeof(struct Ref));
            TempStore=(int *)calloc(RcdFound,sizeof(int));
            TmpTreeprint(root);
            ScanFile();
            if (Choice())
            {
                ToDest();
                AfterExtract();
            }
        }
        else /* not found */
        {
            system("cls");
            cursor(12,25);
            printf("No record not found");
        }
    }
    else FileNotFound();
    SetFree();
    return(1);
} /* of Search */

SetFree() /* free useless variables */
{
    free(Ary);

```

```

    free(root);
    free(Rcd);
    free(lgr);
    free(TempStore);
} /* of SetFree */

Choice() /* read the choice for the selected references */
{
    int done=FALSE,c;
    while (!done)
    {
        system("cls");
        cursor(9,25);
        printf("1> QUICK VIEW.");
        cursor(11,25);
        printf("2> TO NEW FILE.");
        cursor(13,25);
        printf("3> TO DELETE.");
        cursor(15,25);
        printf("9> RETURN.");
        cursor(20,25);
        printf("Enter your choice : ");
        c=getch();
        putchar(c);
        if (c==49)
        {
            ShortHeading();
            ShortDisplay();
        }
        else if (c==50) return(1);
        else if (c==51)
        {
            system("cls");
            cursor(12,25);
            printf("Are you sure (Y/N)? ");
            c=getch();
            putchar(c);
            if ((c==121)|| (c==89)) return(delete());
        }
        else return(0);
    } /* of while */
} /* of Choice */

ToDest() /* save the selected references into a file */
{
    system("cls");
    cursor(12,25);
    printf("Enter output file name : ");
    gets(ofn,15,stdin);
    strcat(ofn,".bbf");
    if (strlen(ofn)==0)
    {
        system("cls");
        cursor(12,25);
        printf("\nInvalid output file name.");
    }
}

```

```

        return(1); /* null file name, return to menu */
    }
    else SaveToFile();
} /* ToDest */

ShortHeading() /* display the heading for the quick view */
{
    system("cls");
    cursor(0,0);
    printf("Rcd#   Fisrt Author Name   type           Topic \n");
    printf("or Book Name           Date");
    cursor(1,0);
    printf("~~~~~ ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~\n");
    printf("~~~~~ ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~");
} /* of ShortHeading */

ShortDisplay() /* display the content of the quick view */
{
    int yy,c;
    xx=2;
    for (c=0;c<RcdFound;c++)
    {
        yy=TempStore[c];
        cursor(xx,0);
        if (Ary[yy].B_or_J=='J')
            printf("%-6.4d%-19.17s%-6.1c%-37.35s%-11.10s",
                yy+1,Ary[yy].NameList[0],Ary[yy].B_or_J,
                Ary[yy].J_Part.Topic,Ary[yy].Date);
        else
            printf("%-6.4d%-19.17s%-6.1c%-37.35s%-11.10s",
                yy+1,Ary[yy].NameList[0],Ary[yy].B_or_J,
                Ary[yy].B_Part.BookName,Ary[yy].Date);
        xx=xx+1;
    }
    printf("\n\n");
    system("pause");
} /* of ShortDisplay */

InitRoot() /* initialize root for building tree */
{
    root=NULL;
    RcdFound=0;
    gg=0;
} /* of InitRoot */

Searching(code) /* search references */
int code;
{
    FILE *ap,*fopen();
    int check=0,more=TRUE,done=FALSE,which=0,where=0,i=0;
    char *come,*bak,*name,*date,what;
    struct temp *TmpTree();
    if (code==0) /* empty criteria */ return(0);
    else
    {

```

```

InitRoot();
ap=fopen(aname,"r");
rewind(ap);
name=(char *)calloc(30,sizeof(char));
date=(char *)calloc(15,sizeof(char));
while (!feof(ap)&&!done)
{
    if (more) /* more to be compared */
    {
        fscanf(ap,"%d %s %d %ls %s\n",&which,name,&where,
            &what,date);
    }
    if (which==lgr->filno)
    {
        if (code==3) /* name and date */
        {
            check=CmpChar(in->NameList[i],name);
            if (check==0) /* FOUND */
            {
                if (CheckDate(date))
                {
                    Mark(where-1,i+1,code);
                    more=TRUE;
                }
            }
            else if (check<0) /* */
            {
                if ((i+1)>=in->Namecnt) done=TRUE;
                else
                {
                    ++i;
                    more=FALSE;
                }
            }
            else if (check>0) /* read more or done */
            more=TRUE;
        }
        else if (code==1) /* name only */
        {
            check=CmpChar(in->NameList[i],name);
            if (check==0)
            {
                Mark(where-1,i+1,code);
                more=TRUE;
            }
            else if (check<0)
            {
                if ((i+1)>=in->Namecnt) done=TRUE;
                else
                {
                    ++i;
                    more=FALSE;
                }
            }
            else more=TRUE;
        }
    }
}

```



```

        }
        else if (code==2) /* date only */
        {
            if (CheckDate(date))
                Mark(where-1,where+1,code);
        }
    } /* of outter if */
} /* of while */
} /* of outter else */
free(name);
free(date);
return(RcdFound);
} /* of Searching */

Mark(i,j,c) /* put down the record number found into Rcd[] and
            build it into the TmpTree for further process. */
int i,j,c;
{
    if (Rcd[i]==j) /* already there */ return;
    else
    {
        Rcd[i]=j;
        if (((Rcd[i]==in->Namecnt)&&((c==1)||(c==3))) || (c==2))
        {
            root=TmpTree(root,i+1);
            ++RcdFound;
        }
    }
} /* of Mark */

CheckDate(d) /* check the publication date */
char *d;
{
    int c1=1,c2=2;
    switch(TimePeriod)
    {
        case 1 : return(Valid(in->from,d,TimePeriod));
        case 2 : return(Valid(in->to,d,TimePeriod));
        case 3 : if ((Valid(in->from,d,c1)) &&
                    (Valid(in->to,d,c2))) return(1);
        default : return(0);
    } /* of swich */
} /* of CheckDate */

Valid(ii,dd,code) /* determine date is FROM, TO, or both */
char *ii, *dd;
int code;
{
    int cond=0;
    cond=CmpDate(ii,dd);
    if ((cond<=0)&&(code==1)) return(1); /* from */
    else if ((cond>=0)&&(code==2)) return(1); /* to */
    else return(0);
} /* of Valid */

```

```

CmpDate(a,b) /* perform the comparsion of dates */
char a[]; /* ..... input criteria date */
char b[]; /* ..... date been compared */
{
    int i,j,lena=0,lenb=0;
    char *stpblk();
    lena = strlen(a);
    lenb = strlen(b);
    if (lena==lenb) /* both are only have year field */
        return(CmpChar(a,b));
    else /* b is longer than a */
    {
        i=lenb-lena;
        for (j=0; j<4; j++,i++)
        {
            if (a[j]>b[i]) return(1);
            else if (a[j]<b[i]) return(-1);
            else if ((a[j]==b[i])&&(j==3)) return(0);
        }
    }
} /* of Cmpdate */

CmpChar(s,t) /* compare two character strings */
char *s;
char *t;
{
    for (;toupper(*s)==toupper(*t);s++,t++)
        if (*s=='\0') return(0);
    if (*s=='\0') return(0);
    return(toupper(*s)-toupper(*t));
} /* CmpChar */

ReadCriteria() /* display criteria screen entry format and
               read the criteria. */
{
    int c,more=1,size;
    system("cls");
    cursor(0,0);
    printf("SERACHING : %s",lgr->filename);
    cursor(1,0);
    printf("FILE SIZE : %d RECORDS",lgr->filsiz);
    SearchFormat();
    InitS();
    cursor(x,y);
    while (more)
    {
        if (((c=getch()) == 13) && (x == 18)) more = 0;
        else if (c==13)
        {
            SRet();
            cursor(x,y);
        }
        else if (c==0) /* Arrow Keys */
        {
            c=getch();

```

```

        if (c==72)      /* UpArrow */ SUP();
        else if (c==80) /* DnArrow */ SDown();
        else if (c==75) /* LfArrow */ SLeft();
        else if (c==77) /* RtArrow */ SRight();
        cursor(x,y);
    } /* of Arrow Keys */
    else SEat(c);
} /* of while more */
return(AssignS());
} /* of ReadCriteria */

InitS() /* initialize temps[11] for criteria format entry */
{
    int i,j;
    x=3;
    y=9;
    for (i=0;i<19;i++)
        for (j=0;j<45;j++)
            temps[i][j]=0;
} /* of Init */

SRet() /* return is entered */
{
    if ((x>=3)&&(x<8))
        { ++x; y=9; }
    else if (x==8)
        { x=10; y=13; }
    else if (x==10)
        { x=11; y=13; }
    else if (x==11)
        { x=13; y=12; }
    else if ((x>=13)&&(x<18))
        { ++x; y=12; }
} /* of SRet */

SUP() /* up arrow key is entered */
{
    if ((x>3)&&(x<=8))
        { --x; y=9; }
    else if (x==3)
        { x=18; y=12; }
    else if (x==10)
        { x=8; y=9; }
    else if ((x==10)|| (x==11))
        {
            if (x==10)
                { x=8; y=13; }
            else if (x==11)
                { x=10; y=13; }
        }
    else if (x==13)
        { x=11; y=13; }
    else if ((x<=18)&&(x>13))
        { --x; y=12; }
} /* of SUP */

```

```
SDown() /* down arrow key is entered */
```

```
{
    if ((x<8)&&(x>=3))
        { ++x; y=9; }
    else if ((x==8)|| (x==11))
        {
            if (x==8) y=13;
            else if (x==11) y=12;
            x=x+2;
        }
    else if (x==10)
        { x=11; y=13; }
    else if ((x<18)&&(x>=13))
        { ++x; y=12; }
    else if (x==18)
        { x=3; y=9; }
} /* of SDOWN */
```

```
SRight() /* right arrow key is entered */
```

```
{
    if ((x>=3)&&(x<=8))
        {
            if (y<38) ++y;
            else if ((y==38)&&(x==8))
                { x=10; y=13; }
            else if (y==38)
                { ++x; y=9; }
        }
    else if ((x==10)|| (x==11))
        {
            if (y<28) ++y;
            else if (y==28)
                {
                    if (x==10)
                        { x=11; y=13; }
                    else
                        { x=13; y=12; }
                }
        }
    else if ((x>=13)&&(x<=18))
        {
            if ((x==18)&&(y==41)) putchar(7);
            else if (y<41) ++y;
            else if (y==41)
                { ++x; y=12; }
        }
} /* of SRIGHT */
```

```
SLeft() /* left arrow key is entered */
```

```
{
    if ((x>3)&&(x<=8))
        {
            if (y>9) --y;
            else if (y==9)
```

```

        { --x; y=38; }
    }
    else if (x==3)
    {
        if (y==9) putch(7);
        else --y;
    }
    else if ((x==10)|| (x==11))
    {
        if (y>13) --y;
        else if (y==13)
        {
            if (x==10)
                { x=8; y=38; }
            else if (x==11)
                { x=10; y=28; }
        }
    }
    else if ((x==13)&&(y==12))
    { x=11; y=28; }
    else if ((x>=13)&&(x<=18))
    {
        if (y>12) --y;
        else if (y==12)
            { --x; y=41; }
    }
} /* of SLEFT */

SEat(c) /* accept all the legal characters */
char c;
{
    if ( (((x>=3)&&(x<=8))&&(y<=38)) ||
          (((x==10)|| (x==11))&&(y<=28)) ||
          (((x>=13)&&(x<=18))&&(y<=41)) )
        SShow(c);
    else putch(7);
} /* of SEat */

SShow(c) /* display and copy the character accepted and
          move the cursor */
char c;
{
    putch(c);
    i=x-3;
    if ((x>=3)&&(x<=8)) j=y-9;
    else if ((x==10)|| (x==11)) j=y-13;
    else j=y-12;
    temps[i][j++]=c;
    cursor(x,++y);
} /* of SShow */

AssignS() /* copy the content from temps[][] to in for
          further process */
{
    int wordcnt=0,namecnt=0,lenf=0,lent=0,len,m,k;

```

```

char *stpblk(),*p;
system("cls");
for (k=0;k<6;k++)
{
    p=stpblk(temps[k]);
    if ((len=strlen(p)) >=1)
    {
        ++namecnt;
        strcpy(in->NameList[k],p);
    }
}
in->Namecnt=namecnt;
strcpy(in->to,stpblk(temps[8]));
strcpy(in->from,stpblk(temps[7]));
lenf=strlen(in->from);
lent=strlen(in->to);
if ((lenf>0)&&(lent>0))
{
    TimePeriod=3;
    in->Datecnt=2;
}
else if ((lenf>0) || (lent>0))
{
    if (lenf>0) TimePeriod=1;
    else TimePeriod=2;
    in->Datecnt=1;
}
else
{
    TimePeriod=0;
    in->Datecnt=0;
}
for (m=0,k=10;k<16;m++,k++)
{
    p=stpblk(temps[k]);
    if ((len=strlen(p)) >=1)
    {
        ++wordcnt;
        strcpy(in->WordList[m],p);
    }
}
in->Wordcnt=wordcnt;
return(Classify());
} /* of AssignS */

```

```

Classify() /* determine what type of process should perform */
{
    if ((in->Namecnt==0)&&(in->Datecnt==0)) return(0); /* none */
    if ((in->Namecnt>0)&&(in->Datecnt==0)) return(1); /* name */
    if ((in->Namecnt==0)&&(in->Datecnt>0)) return(2); /* date */
    if ((in->Namecnt>0)&&(in->Datecnt>0)) return(3); /* n + d */
    else return(7);
} /* of Classify */

```

```

/* build the selected references into a binary tree */

```

```

struct temp *TmpTree(root,where)
struct temp *root;
int where;
{
    struct temp *OneMore();
    int cond;

    if (root == NULL)
    {
        root = OneMore();
        root->where = where;
        root->left = NULL;
        root->right = NULL;
    }
    else if (where > root->where)
        root->right = TmpTree(root->right,where);
    else root->left = TmpTree(root->left,where);
    return(root);
} /* of TmpTree */

SaveToFile() /* extract the selected references into a file */
{
    FILE *ap,*ip,*kp,*fopen();
    char *tdate;
    int c,p;
    ip=fopen(ofn,"w");
    ap=fopen(aname,"a");
    kp=fopen("KEYWORD.DIC","a");
    for (p=0;p<RcdFound;p++)
    {
        c=TempStore[p];
        strcpy(tdate,Ary[c].Date);
        fprintf(ip,"%c",Ary[c].B_or_J);
        fprintf(ip,"\n%d",Ary[c].Namecnt);
        MarkBlank(tdate);
        for (i=0;i<Ary[c].Namecnt;i++)
        {
            fprintf(ip,"\n%s",Ary[c].NameList[i]);
            MarkBlank(Ary[c].NameList[i]);
            fprintf(ap,"%d %s %d %c %s\n",NextFileNo,
                Ary[c].NameList[i], (p+1),Ary[c].B_or_J,tdate);
        }
        if (Ary[c].B_or_J == 'J')
        {
            fprintf(ip,"\n%d",Ary[c].J_Part.KeywordCnt);
            for (i=0;i<Ary[c].J_Part.KeywordCnt;i++)
            {
                fprintf(ip,"\n%s",Ary[c].J_Part.KeywordList[i]);
                MarkBlank(Ary[c].J_Part.KeywordList[i]);
                fprintf(kp,"%d %s %d\n",NextFileNo,
                    Ary[c].J_Part.KeywordList[i],(p+1));
            }
            fprintf(ip,"\n%s",Ary[c].J_Part.J_Name);
            fprintf(ip,"\n%s",Ary[c].J_Part.Topic);
        }
    }
}

```

```

        else
        {
            fprintf(ip, "\n%s", Ary[cl].B_Part.BookName);
            fprintf(ip, "\n%s", Ary[cl].B_Part.City);
            fprintf(ip, "\n%s", Ary[cl].B_Part.Publisher);
        } /* of else */
        fprintf(ip, "\n%s", Ary[cl].Pages);
        fprintf(ip, "\n%s\n", Ary[cl].Date);
    }
    fclose(ap);
    fclose(kp);
    fflush(ip);
    fclose(ip);
} /* of SaveToFile */

struct temp *OneMore() /* allocate a block memory for temp */
{return((struct temp *)calloc(1,sizeof(struct temp)));}

TmpTreeprint(root) /* print out the binary tree */
struct temp *root;
{
    if (root != NULL)
    {
        TmpTreeprint(root->left);
        CopytoTemp(root->where);
        TmpTreeprint(root->right);
    }
} /* of TmpTreeprint */

CopytoTemp(where)
int where;
{
    int c;
    c=where-1;
    TempStore[gg++]=c;
} /* of CopytoTemp */

AfterExtract() /* modify the ledger file */
{
    AddToLedger();
    SortName();
} /* of AfterExtract */

AddToLedger()
{
    FILE *lp,*fopen();
    lp=fopen(lname,"a");
    fprintf(lp,"%d %s %d\n",NextFileNo,ofn,RcdFound);
    ++NextFileNo;
    fclose(lp);
} /* of AddToLedger */

```



```

/* DELTE.C deletes the references selected by SEARCH.C */
#include <stdio.h>
#include <b:header.h>
extern struct Ldgr *lgr;
extern int *TempStore;
extern struct Ref *Ary;
extern char *aname;
extern char *lname;
extern int RcdFound;

delete()
{
    DelRef();
    DelName();
    AfterDelete();
}

DelRef() /* delete the references from the data base file */
{
    FILE *ip,*fopen();
    int done=FALSE,i=0,k=0;
    ip=fopen(lgr->filename,"w");
    while (!done)
    {
        if (k==lgr->filsize)
            done=TRUE;
        else if (k==TempStore[i])
            { ++k; ++i; }
        else if (k!=TempStore[i])
            {
                fprintf(ip,"%c",Ary[k].B_or_J);
                fprintf(ip,"\n%d",Ary[k].Namecnt);
                for (i=0;i<Ary[k].Namecnt;i++)
                    fprintf(ip,"\n%s",Ary[k].NameList[i]);
                if (Ary[k].B_or_J == 'J')
                {
                    fprintf(ip,"\n%d",Ary[k].J_Part.KeywordCnt);
                    for (i=0;i<Ary[k].J_Part.KeywordCnt;i++)
                        fprintf(ip,"\n%s",Ary[k].J_Part.KeywordList[i]);
                    fprintf(ip,"\n%s",Ary[k].J_Part.J_Name);
                    fprintf(ip,"\n%s",Ary[k].J_Part.Topic);
                }
                else
                {
                    fprintf(ip,"\n%s",Ary[k].B_Part.BookName);
                    fprintf(ip,"\n%s",Ary[k].B_Part.City);
                    fprintf(ip,"\n%s",Ary[k].B_Part.Publisher);
                } /* of else */
                fprintf(ip,"\n%s",Ary[k].Pages);
                fprintf(ip,"\n%s\n",Ary[k].Date);
                ++k;
            } /* of else */
    } /* of while */
    fclose(ip);
} /* of DelRef */

```

```

DelName() /* delete the names from author dictionary */
{
    FILE *ap,*tap,*fopen();
    int which,where,less,i,match=FALSE;
    char *tmpname,*tmpdate,what,*tname;
    tname="tmp";
    ap=fopen(aname,"r");
    tap=fopen(tname,"w");
    rewind(ap);
    tmpname=(char *)calloc(30,sizeof(char *));
    tmpdate=(char *)calloc(30,sizeof(char *));
    while(!feof(ap))
    {
        fscanf(ap,"%d %s %d %ls %s\n",&which,tmpname,
                                           &where,&what,tmpdate);
        if (which==lgr->filno)
        {
            i=0;
            match=FALSE;
            less=0;
            while ((!match)&&(i<RcdFound))
            {
                if (where==(TempStore[i]+1)) match=TRUE;
                else if (where>(TempStore[i]+1)) ++less;
                ++i;
            } /* of inner while */
            if ((!match)&&(strlen(tmpname) != 0))
                fprintf(tap,"%d %s %d %c %s\n",which,tmpname,
                    (where-less),what,tmpdate);
            } /* of outter if */
        else if (!feof(ap))
            fprintf(tap,"%d %s %d %c %s\n",which,tmpname,where,
                what,tmpdate);
    } /* of while */
    fclose(ap);
    fclose(tap);
    free(tmpname);
    free(tmpdate);
    system("del AUTHOR.DIC");
    rename(tname,aname);
} /* of DelName */

AfterDelete() /* update the ledger file */
{
    FILE *tlp,*lp,*fopen();
    char *name,*tname;
    int no,size;

    tname="l1";
    lp=fopen(lname,"r");
    tlp=fopen(tname,"w");
    rewind(lp);
    name=(char *)calloc(15,sizeof(char *));
    while(!feof(lp))

```

```

{
    fscanf(lp,"%d %s %d\n",&no,name,&size);
    if (!feof(lp))
    {
        if (no==lgr->filno)
            fprintf(tlp,"%d %s %d\n",no,name,
                    (lgr->filesize-RcdFound));
        else fprintf(tlp,"%d %s %d\n",no,name,size);
    }
} /* of while */
fclose(lp);
fclose(tlp);
system("del ledger");
rename(tname,lname);
free(name);
} /* of AfterDelete */

```

```

/* FORM.C formats the references of a data base file into two
   defined formats, namely, ACM Computing Surveys and Trans. on
   Office Info. Syst.. The formatted listing could be sorted in
   ascending or descending order by author names.
*/
#include <stdio.h>
#include <b:header.h>
#define EndLine      65
#define FORMLENGTH 65
#define BUFLen      260
extern struct Ldgr *lgr;
extern struct Ref *Ary;
static char OutBuf[4][65];
static char *Buffer;
static char out[15];
static int Here,Current,Inline,ActualLength,go;
static int ToWhere; /* 49 to Console, 50 to File */
static int WhatFormat; /* 49 form1, 50 form2 */
static int WhatOrder; /*48-original,49-ascending,50-descending*/
static int *index;
static int cnt; /* count output index */

form() /* form.c */
{
    ReadFileName();
    if (CheckFileName())
    {
        Init();
        ScanFile();
        if (((ToWhere==Where())==57) ||
            ((ToWhere==50)&&(!ToFile()))) return(1);
        else
        {
            if ((WhatFormat==ReadFormat())!=57)
            {
                if ((WhatOrder==ReadOrder())!=48)
                {
                    InitIndex();
                    Sorting();
                    system("cls.");
                }
                Reform();
                free(Ary);
                free(index);
            }
        }
    }
    else FileNotFound();
    return(1);
} /* of form */

Init() /* initialize the variables */
{
    cnt=1;
    go=0; /* index for Ary[], it's a global variable */

```

```

    ToWhere=49; /* default to Console */
    WhatFormat=49; /* default to format 1 */
    WhatOrder=48; /* default to original order */
    Ary=(struct Ref *)calloc(lgr->filesize,sizeof(struct Ref));
    index=(int *)calloc(lgr->filesize,sizeof(int));
} /* of Init */

ReadOrder() /* read the order for the formatted listing */
{
    int c;
    system("cls");
    cursor(11,25);
    printf("1> ASCENDING ORDER.");
    cursor(13,25);
    printf("2> DESCENDING ORDER.");
    cursor(15,25);
    printf("9> RETURN( Unsort ).");
    cursor(20,25);
    printf("Enter your choice( 1, 2, or 9) : ");
    c=getch();
    putchar(c);
    if ((c!=49)&&(c!=50))
        return(48); /* no sorting */
    else return(c);
} /* of ReadOrder */

Where() /* display menu and read the destination */
{
    int more=1,c;
    while (more)
    {
        system("cls");
        cursor(7,25);
        printf("0> HELP.");
        cursor(9,25);
        printf("1> CONSOLE.");
        cursor(11,25);
        printf("2> FILE.");
        cursor(13,25);
        printf("3> SAMPLE.");
        cursor(15,25);
        printf("9> RETURN.");
        cursor(20,25);
        printf("Enter your choice : ");
        c=getch();
        putchar(c);
        if (c==48) FormHelp();
        else if (c==51) Demo();
        else if ((c==49)|| (c==50)|| (c==57)) return(c);
    }
} /* of Where */

ReadFormat() /* display menu and read the format selection */
{
    int c,more=1;

```

```

while (more)
{
    system("cls");
    cursor(9,25);
    printf("0> SAMPLE.");
    cursor(11,25);
    printf("1> FORMAT 1.");
    cursor(13,25);
    printf("2> FORMAT 2.");
    cursor(15,25);
    printf("9> RETURN TO PREVIOUS MENU.");
    cursor(20,25);
    printf("Enter your choice : ");
    c=getch();
    putch(c);
    if (c==48) /* 0 for sample */ Demo();
    else if (c==52) /* HELP */ FormHelp();
    else return(c);
} /* of while */
} /* of ReadFormat */

Demo() /* display sample listing */
{
    system("cls");
    cursor(1,30);
    printf("FORMAT 1");
    cursor(2,30);
    printf("~~~~~");
    cursor(4,5);
    printf("Liu78   Liu, M. T. \"Distributed loop computer \");
    printf("networks,\" in Ad");
    cursor(5,5);
    printf("        vances in Computers, M. C. Yovits (Ed.), ");
    printf("Academic Press,");
    cursor(6,5);
    printf("        New York, 1978, pp. 163-221.");
    cursor(8,5);
    printf("Kamo81   Kamoun, F. \"A drop and throttle flow \");
    printf("control policy for");
    cursor(9,5);
    printf("        computer networks,\" IEEE Trans. Commun. ");
    printf("COM-29   ( April");
    cursor(10,5);
    printf("        1981), 444-452.");
    cursor(13,30);
    printf("FORMAT 2");
    cursor(14,30);
    printf("~~~~~");
    cursor(16,5);
    printf("1. Ellis, P., and Nutt, G. Office information ");
    printf("systems and");
    cursor(17,5);
    printf("        computer science. ACM Comput. Surv. 12, 1 (Mar.");
    printf("1980), 27-60.");
    cursor(19,5);

```

```

printf("10. Newman, W. Office models and office systems.");
printf(" design. In");
cursor(20,5);
printf("    Intgrated Office Systems. N. Naffah, Ed.,");
printf(" North-Holland,");
cursor(21,5);
printf("    Amsterdam, 1980, 3-10.");
cursor(23,0);
system("pause");
} /* of Demo */

```

FillBuffer() /* feed Buffer[] from Ary[] ready to be formatted */

```

{
    int m,len,i=0,j,k;
    if (WhatFormat==49) /* format one */
    {
        for (i=0;i<4;i++)
            Buffer[i]=Ary[go].NameList[0][i];
        len=strlen(Ary[go].Date);
        len=len-2;
        for (i=4;i<6;i++,len++)
            Buffer[i]=Ary[go].Date[len];
        Buffer[6]=' ';
        Buffer[7]=' ';
        i=8;
    }
    if (Ary[go].Namecnt > 1)
    {
        m=Ary[go].Namecnt-1;
        for (j=0;j<m;j++) /* n-1 Author names */
        {
            len=strlen(Ary[go].NameList[j]);
            for (k=0;k<len;k++,i++)
                Buffer[i]=Ary[go].NameList[j][k];
            Buffer[i++]=' ';
            Buffer[i++]=' ';
        }
        Buffer[i++]='a';
        Buffer[i++]='n';
        Buffer[i++]='d';
        Buffer[i++]=' ';
        len=strlen(Ary[go].NameList[m]);
        for (k=0;k<len;i++,k++)
            Buffer[i]=Ary[go].NameList[m][k];
    }
    else /* only one author */
    {
        len=strlen(Ary[go].NameList[0]);
        for (k=0;k<len;k++,i++)
            Buffer[i]=Ary[go].NameList[0][k];
    } /* the end of handling author names */
    Buffer[i++]=' ';
    if (WhatFormat==49)
        Buffer[i++]='"'; /* double quote, star of topic */
    if (Ary[go].B_or_J=='J')

```

```

        {
            len=strlen(Ary[go].J_Part.Topic);
            for (k=0;k<len;i++,k++)
                Buffer[i]=Ary[go].J_Part.Topic[k];
        }
    else
    {
        len=strlen(Ary[go].B_Part.BookName);
        for (k=0;k<len;k++,i++)
            Buffer[i]=Ary[go].B_Part.BookName[k];
    }
    if (WhatFormat==49)
    {
        Buffer[i++]=','; /* the end of topic */
        Buffer[i++]='"';
    }
    else Buffer[i++]='.';
    Buffer[i++]=' ';
    if (Ary[go].B_or_J=='J')
    {
        len=strlen(Ary[go].J_Part.J_Name);
        for (k=0;k<len;i++,k++)
            Buffer[i]=Ary[go].J_Part.J_Name[k];
        Buffer[i++]=' ';
    }
    else
    {
        if ((len=strlen(Ary[go].B_Part.Publisher))!=0)
        {
            for (k=0;k<len;k++,i++)
                Buffer[i]=Ary[go].B_Part.Publisher[k];
            Buffer[i++]=',';
            Buffer[i++]=' ';
        }
        if ((len=strlen(Ary[go].B_Part.City))!=0)
        {
            for (k=0;k<len;k++,i++)
                Buffer[i]=Ary[go].B_Part.City[k];
            Buffer[i++]=',';
            Buffer[i++]=' ';
        }
    }
    len=strlen(Ary[go].Date);
    if (len>4) /* long date */
    {
        Buffer[i++]='(';
        for (k=0;k<len;k++,i++)
            Buffer[i]=Ary[go].Date[k];
        Buffer[i++]=')';
    }
    else
    {
        for (k=0;k<len;k++,i++)
            Buffer[i]=Ary[go].Date[k];
    }
}

```



```

    if ((len=strlen(Ary[go].Pages))!=0)
    {
        Buffer[i++]=' ';
        Buffer[i++]=' ';
        for (k=0;k<len;k++,i++)
            Buffer[i]=Ary[go].Pages[k];
    }
    Buffer[i]='.'; /* ..... The End */
    ActualLength=i;
} /* of FillBuffer */

ToFile() /* if destination is a plain file */
{
    int len;
    system("cls");
    cursor(12,25);
    printf("Enter ouput file name : ");
    gets(out,15,stdin);
    len=strlen(out);
    strcat(out, ".fmt");
    return(len);
} /* of ToFile */

FormHelp() /* display help messages */
{
    system("cls");
    printf("\n\t\t* * * * * HELP * * * * *");
    printf(" * * * * *\n\n");
    printf("\t\t1> There are two formats available. FORMAT 1");
    printf(" is used in \n");
    printf("\t\t the ACM Computing Surveys and FORMAT 2 ");
    printf("which is used \n");
    printf("\t\t the ACM Trans. on Offince Info. Syst.\n\n");
    printf("\t\t2> The listings could be sorted in ascending");
    printf(" or descending\n");
    printf("\t\t order. The option should be chosen before ");
    printf("the re-\n");
    printf("\t\t ference is properly formatted.\n\n");
    printf("\t\t3> The extension of the output file is .FMT ");
    printf("which will \n");
    printf("\t\t not be included in LEDGER file.\n\n");
    printf("\t\t4> The listings are 65 characters long, double");
    printf(" space \n");
    printf("\t\t between references, single space within a");
    printf(" reference.\n\n");
    printf("\t\t5> Select SAMPLE for detailed format layouts.");
    cursor(23,0);
    system("pause");
} /* of FormHelp */

InitBuffer() /* initialize Buffer[] */
{
    int i,j;
    for (i=0;i<4;i++)
        for (j=0;j<FORMLENGTH;j++)

```

```

        OutBuf[i][j]=0;
Buffer=(char *)calloc(BUFLen,sizeof(char));
InLine=0;
if (WhatFormat==49)
    Here=8;
else Here=3;
Current=0;
} /* of InitBuffer */

Heading(op) /* print out the heading of the listing */
FILE *op;
{
    if (ToWhere==49)
    {
        system("cls");
        printf("REFERENCES : \n\n");
    }
    else fprintf(op,"REFERENCES : \n\n");
} /* of Heading */

Arrange() /* copy the content of Buffer[] to a template */
{
    int done=FALSE,
        Need=0,
        Extra=0;
    while (!done)
    {
        if ((Need=GetWord())==0) done=TRUE;
        else if (Need<=(Extra=SpaceLeft())) CopyInto(Need);
        else Adjust(Extra);
    } /* of while */
    done=FALSE;
} /* of Arrange */

Reform() /* format the references */
{
    FILE *op,*fopen();
    int j=0,i=1,done=FALSE;
    op=fopen(out,"w");
    Heading(op);
    while (!done)
    {
        InitBuffer();
        if (WhatOrder==49) /* ascending */
            go=index[j++];
        else if (WhatOrder==50) /* descending */
            go=index[lgr->filesize-i++];
        else go=j++;
        FillBuffer();
        if (WhatFormat==49) CopyHead();
        else CopyNo();
        Arrange();
        PrintOut(op);
        if ((WhatOrder!=50)&&(WhatOrder!=49)) /* original order */
            ++go;
    }
}

```

```

        FreeBuffer();
        if ((j>=lgr->filesize)|| (i>lgr->filesize)||
            (go>=lgr->filesize)) done=TRUE;
    } /* of while */
    printf("\n\n");
    system("pause.");
    fclose(op);
    return(l);
} /* of Reform */

CopyNo() /* copy number of the sequence number for the listing */
{
    int i;
    for (i=0;i<3;i++)
        OutBuf[l][i]=' ';
} /* of CopyNo */

FreeBuffer() /* release the memory of Buffer[] */
{ free(Buffer); } /* of FreeBuffer */

CopyHead() /* copy the heading from Buffer[] to OutBuf[][] */
{
    int i;
    for (i=0;i<8;i++)
        OutBuf[l][i]=Buffer[Current++];
} /* of CopyHead */

SpaceLeft() /* determine the available space of a line */
{ return(EndLine-Here); } /* of SpaceLeft */

GetWord() /* return the length of next input word */
{
    int dummy;
    dummy=Current;
    if (dummy>ActualLength) /* done */ return(0);
    if (Buffer[dummy]==' ') return(1);
    while ((Buffer[dummy]!=' ')&&(dummy<=ActualLength))
        ++dummy;
    return(dummy-Current);
} /* of GetWord */

CopyInto(m) /* copy into OutBuf[][] */
int m;
{
    int i;

    if ((m==1)&&(Buffer[Current]==' ')) /* Just a blank */
    {
        ++Current;
        if ((WhatFormat==49)&&(Here!=8))
            OutBuf[InLine][Here++]=' ';
        else if ((WhatFormat==50)&&(Here!=3))
            OutBuf[InLine][Here++]=' ';
    }
    else

```

```

        {
            for (i=0;i<m;i++)
                OutBuf[InLine][Here++]=Buffer[Current++];
        }
    } /* CopyInto */

Adjust(n) /* insert blanks if needed */
int n; /* n is the spaces needed to be inserted blanks */
{
    int i;
    if (OutBuf[InLine][Here-1]==' ')
    {
        ++n;
        OutBuf[InLine][Here-1]='*';
    }
    else if ((OutBuf[InLine][Here]==0)&&(Here==64)) /* blank */
    {
        n=1;
        OutBuf[InLine][Here]='*';
    }
    BackWardsandMark(n);
    ++InLine;
    if (WhatFormat==49)
        Here=8;
    else Here=3;
}

FindBlank() /* insert blanks */
{
    int back;
    back=Here-1;
    while ((OutBuf[InLine][back])!=' ')
        --back;
    Here=back; /* update Here position */
    return(back);
} /* FindBlank */

Insert6() /* if blanks needed are more than 6 */
{
    int back,i;
    for (i=0;i<3;i++)
    {
        back=FindBlank();
        if (i==0) OutBuf[InLine][back]='%'; /* three more */
        else if (i==1) OutBuf[InLine][back]='&'; /* two more */
        else if (i==2) OutBuf[InLine][back]='*'; /* one more */
    } /* of for */
} /* of Insert6 */

InsertRest(k) /* insert blanks less or equal to 3 */
int k;
{
    int back;
    back=FindBlank();
    if (k%5==0)

```

```

        {
            OutBuf[InLine][back]='%'; /* three more spaces */
            back=FindBlank();
            OutBuf[InLine][back]='&'; /* two more spaces */
        }
    else if (k%4==0)
    {
        OutBuf[InLine][back]='%'; /* three more spaces */
        back=FindBlank();
        OutBuf[InLine][back]='&'; /* two more spaces */
    }
    else if (k%3==0)
        OutBuf[InLine][back]='%'; /* three more spaces */
    else if (k%2==0)
        OutBuf[InLine][back]='&'; /* two more spaces */
    else if (k%1==0)
        OutBuf[InLine][back]='*'; /* one more space */
} /* of InsertRest */

BackWardsandMark(n) /* mark where blanks are to be inserted */
int n; /* n is the spaces need to be filled with '*' */
{
    int i,q,r,back,done=FALSE;
    if (n>=8)
    {
        q=n/6;
        r=n%6;
        while (q-- > 0)
            Insert6();
        if (r>0)
            InsertRest(r);
    }
    else
    {
        for (i=0;i<n;i++)
        {
            back=FindBlank();
            OutBuf[InLine][back]='*'; /* one more space */
        } /* of for */
    } /* of else */
} /* OF BACKWARDSANDMARK */

PrintOut(op) /* print out the formatted listing */
FILE *op;
{
    int j,i;
    PrintHead(op);
    for (i=0;i<(InLine+1);i++) /* print outbuf lines */
        PrintLine(i,op);
    if (ToWhere==49)
        printf("\n\n");
    else fprintf(op,"\n\n");
} /* PrintOut */

PrintHead(op) /* print either sequence number or the other */

```

```

FILE *op;
{
    int i;
    if (WhatFormat==49) /* format 1 */
    {
        for (i=0;i<8;i++)
        {
            if (ToWhere==49)
                printf("%c",OutBuf[11][i]);
            else /* to a file */
                fprintf(op,"%c",OutBuf[11][i]);
        }
    }
    else /* format 2 */
    {
        if (ToWhere==49)
            printf("%2d ",cnt++);
        else fprintf(op,"%2d ",cnt++);
    }
} /* of PrintHead */

PrintLine(j,op) /* print one line at a time */
int j;
FILE *op;
{
    int k,i;
    if (WhatFormat==49) i=8;
    else if (WhatFormat==50) i=3;
    for (;i<FORM1LENGTH;i++)
    {
        if (OutBuf[j][i]==' ') /* one more space */
        {
            for (k=0;k<2;k++)
            {
                if (ToWhere==49) printf("\ ");
                else fprintf(op,"\ ");
            }
        } /* of ' ' */
        else if (OutBuf[j][i]=='%') /* three more spaces */
        {
            for (k=0;k<4;k++)
            {
                if (ToWhere==49) printf("\ ");
                else fprintf(op,"\ ");
            }
        } /* of '%' */
        else if (OutBuf[j][i]=='&') /* two more spaces */
        {
            for (k=0;k<3;k++)
            {
                if (ToWhere==49) printf("\ ");
                else fprintf(op,"\ ");
            }
        } /* of '&' */
    }
    else

```

```

        {
            if (ToWhere==49)
                printf("%c",OutBuf[j][i]); /* normal char */
            else fprintf(op,"%c",OutBuf[j][i]);
        }
    } /* of for */
    if (ToWhere==49) /* to Console */
    {
        if (WhatFormat==49) /* format 1 */
            printf("\n\t");
        else printf("\n   ");
    }
    else
    {
        if (WhatFormat==49)
            fprintf(op,"\n\t");
        else
            fprintf(op,"\n   ");
    }
} /* of PrintLine */

InitIndex()
{
    int i;
    for (i=0;i<lgr->filesize;i++)
        index[i]=i;
} /* of InitIndex */

Sorting() /* Shell Sort */
{
    int n,cond,done=FALSE,gap,i,j,time,x;
    n=lgr->filesize;
    for (gap=n/2;gap>0;gap/=2)
    {
        for (i=gap;i<n;i++)
            for (j=i-gap;j>=0;j-=gap)
            {
                if (Ary[index[j]].Namecnt >
                    Ary[index[j+gap]].Namecnt)
                    time=Ary[index[j+gap]].Namecnt;
                else time=Ary[index[j]].Namecnt;
                x=0; /* index of NameList */
                done=FALSE;
                while (!done)
                {
                    if ((cond=Compare(Ary[index[j]].NameList[x],
                                        Ary[index[j+gap]].NameList[x]))==0)
                    {
                        ++x;
                        --time;
                    }
                    else if (cond>0)
                    {
                        Swap(&index[j],&index[j+gap]);
                        done=TRUE;
                    }
                }
            }
    }
}

```

```

        }
        else done=TRUE;
        if (time==0)
            done=TRUE;
    } /* of while */
    if ((cond<=0)|| (time==0)) /* no swap any more */
        break;
    } /* of inner for */
} /* of outter for */
} /* of sorting */

Swap(a,b) /* swap the record numbers if needed */
int *a;
int *b;
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
} /* of swap */

```



```

/* JOIN.C joins two different data base files into another
   file, the original files remain the same.
*/
#include <stdio.h>
#include <b:header.h>
extern int NextFileNo;
extern char *aname;
extern char *lname;
static struct Ldgr *lgr1,*lgr2;
static char tfn[15];
static char fn1[15];
static char fn2[15];
join()
{
    if (ReadJoin())
    {
        if (ReadNames())
        {
            if (CheckNames()) /* both are valid */
            {
                Waiting();
                JoinFiles();
                RenewDic();
                AddJoinToLedger();
            }
            else
            {
                system("cls");
                cursor(12,25);
                fprintf(stderr,"File(s) not found!");
            }
        }
        } /* of outter if */
    return(1);
} /* of Join */

Waiting() /* a waiting message */
{
    system("cls");
    cursor(12,25);
    printf("Writing . . . Please wait!");
} /* of wait */

ReadJoin() /* display menu and read the choice */
{
    int more=TRUE,c;
    while (more)
    {
        system("cls");
        cursor(9,20);
        printf("0> HELP.");
        cursor(11,20);
        printf("1> JOIN TWO FILES INTO A NEW FILE.");
        cursor(13,20);
        printf("9> RETURN.");
    }
}

```



```

        putchar(c);
        if ((c==121)|| (c==89)) /* Yes */ return(1);
        else /* No, return to main menu */ return(0);
    }
    else
    {
        system("cls");
        cursor(12,20);
        fprintf(stderr,"Invalid file names.");
        return(0);
    }
} /* of while */
} /* of ReadNames */

CheckNames()
{
    FILE *lp, *fopen();
    char *name;
    int no,done=FALSE,count=0,cond,size;

    lgr1=(struct Ldgr *)calloc(1,sizeof(struct Ldgr));
    lgr2=(struct Ldgr *)calloc(1,sizeof(struct Ldgr));
    name=(char *)calloc(15,sizeof(char));
    lp=fopen(lname,"r");
    rewind(lp);
    while (!feof(lp) && !done)
    {
        fscanf(lp,"%d %s %d\n",&no,name,&size);
        if ((cond=Compare(fn1,name))==0)
        {
            lgr1->filno=no;
            strcpy(lgr1->filename,name);
            lgr1->filsz=size;
            ++count;
        }
        else if ((cond=Compare(fn2,name))==0)
        {
            lgr2=(struct Ldgr *)calloc(1,sizeof(struct Ldgr));
            lgr2->filno=no;
            strcpy(lgr2->filename,name);
            lgr2->filsz=size;
            ++count;
        }
        if (count==2) done=TRUE;
    }
    fclose(lp);
    if (done) return(1);
    else return(0);
} /* of CheckNames */

JoinFiles() /* perform the concatenation */
{
    FILE *in1,*in2,*out,*fopen();
    int count=0;
    char *line;

```

```

in1=fopen(lgr1->filename,"r");
in2=fopen(lgr2->filename,"r");
out=fopen(tfn,"w");
rewind(in1);
/* copy line by line. Write in1 first, then in2. */
while (!feof(in1))
{
    ++count;
    line=(char *)calloc(MAXLINE,sizeof(char));
    fgets(line,MAXLINE,in1);
    fputs(line,out);
}
fclose(in1);
rewind(in2);
while (!feof(in2))
{
    ++count;
    fgets(line,MAXLINE,in2);
    fputs(line,out);
}
fclose(in2);
fclose(out);
free(line);
} /* of JoinFiles */

```

```

RenewDic() /* update the author dictionary */
{
    FILE *ap, *tap, *fopen();
    char *tname,*name,*date,what;
    int which,where,find=0;
    tname="TMP";
    ap=fopen(aname,"r");
    tap=fopen(tname,"w");
    rewind(ap);
    name=(char *)calloc(30,sizeof(char));
    date=(char *)calloc(30,sizeof(char));
    while (!feof(ap))
    {
        fscanf(ap,"%d %s %d %ls %s\n",
            &which,name,&where,&what,date);
        fprintf(tap,"%d %s %d %c %s\n",
            which,name,where,what,date);
        if ((which==lgr1->filno) || (which==lgr2->filno))
        {
            if (which==lgr2->filno) where+=lgr1->filsize;
            ++find;
            which=NextFileNo;
            fprintf(tap,"%d %s %d %c %s\n",
                which,name,where,what,date);
        } /* of if */
    } /* of while */
    fclose(tap);
    fclose(ap);
    if (find!=0)
    {

```

```

        system("del AUTHOR.DIC");
        rename(tname,aname);
        SortName();
    } /* of if */
    return(1);
} /* of RenewDic */

AddJoinToLedger() /* update the ledger file */
{
    FILE *lp,*fopen();
    lp=fopen(lname,"a");
    fprintf(lp,"%d %s %d\n",NextFileNo,tfn,
        (lgr1->filesize+lgr2->filesize));
    ++NextFileNo;
    fclose(lp);
    free(lgr1);
    free(lgr2);
} /* of AddJoinToLedger */

```

```

/* FORMAT.C defines three screen entry formats : journal
book, and criteria */
RefFormat() /* reference format */
{
    cursor(3,0);
    printf("NAME 1 : [ ]\n");
    printf("NAME 2 : [ ]\n");
    printf("NAME 3 : [ ]\n");
    printf("NAME 4 : [ ]\n");
    printf("NAME 5 : [ ]\n");
    printf("NAME 6 : [ ]\n\n");
    printf("TOPIC : [ ]");
    printf(" [ ]\n");
    printf(" [ ]");
    printf(" [ ]\n\n");
    printf("JOURNAL : [ ]");
    printf(" [ ]\n");
    printf(" [ ]");
    printf(" [ ]\n\n");
    printf("DATE : [ ]\n\n");
    printf("KEYWORD : [ ]");
    printf("[ ]\n");
    printf(" [ ]");
    printf("[ ]\n");
    printf(" [ ]");
    printf("[ ]\n\n");
    printf("PAGE # : [ ]");
} /* of RefFormat */

BookFormat() /* book format */
{
    cursor(3,0);
    printf("NAME 1 : [ ]\n");
    printf("NAME 2 : [ ]\n");
    printf("NAME 3 : [ ]\n");
    printf("NAME 4 : [ ]\n");
    printf("NAME 5 : [ ]\n");
    printf("NAME 6 : [ ]\n\n");
    printf("BOOKNAME : [ ]");
    printf(" [ ]\n");
    printf(" [ ]");
    printf(" [ ]\n\n");
    printf("YEAR : [ ]\t\t\tPAGE : ");
    printf("[ ]\n\n");
    printf("CITY : [ ]\n\n");
    printf("PUBLISHER : [ ]");
    printf(" [ ]");
} /* of BookFormat */

SearchFormat() /* criteria format */
{
    cursor(3,0);
    printf("NAME1 : [ ]\n");
    printf("NAME2 : [ ]\n");
    printf("NAME3 : [ ]\n");

```

```

printf("NAME4 : [                                     ]\n");
printf("NAME5 : [                                     ]\n");
printf("NAME6 : [                                     ]\n\n");
printf("DATE FROM : [                               ]\n");
printf("      TO      : [                               ]\n\n");
printf("KEYWORD1 : [                                     ]\n");
printf("KEYWORD2 : [                                     ]\n");
printf("KEYWORD3 : [                                     ]\n");
printf("KEYWORD4 : [                                     ]\n");
printf("KEYWORD5 : [                                     ]\n");
printf("KEYWORD6 : [                                     ]\n");
} /* of SearchFormat */

```

```

/* COMMON.C includes the functions which are shared by the
   systems functions.
*/
#include <stdio.h>
#include <b:header.h>
extern int *Rcd;
extern struct temp *root;
extern struct Ref *Ary;
extern struct AuthorDic *Aroot, *atemp;
extern struct Reference *Head, *rtemp;
extern struct KeywordDic *Kroot, *ktemp;
extern struct Ldgr *lgr;
extern char *fn;
extern char *aname;
extern char *lname;
extern char *DefaultDate;
extern int x,y,i,j;
extern int ndx;
extern int NextFileNo;
extern int ActCode;
extern char tempj[20][70];
extern char tempb[15][70];

struct NameDic *Nroot;
struct WordDic *Wroot;

Fresh() /* allocate memory */
{
    Rcd=(int *)calloc(lgr->filsize,sizeof(int));
    Ary=(struct Ref *)calloc(lgr->filsize,sizeof(struct Ref));
} /* of Fresh */

FileNotFound() /* error message, when file found */
{
    system("cls");
    cursor(12,25);
    fprintf(stderr,"File : %s not found.",fn);
} /* of FileNotFound */

Initialization()
{
    struct Reference *New();
    struct AuthorDic *NewName();
    struct KeywordDic *NewWord();
    Aroot = NewName();
    Aroot->next = NULL;
    atemp = Aroot;
    Kroot = NewWord();
    Kroot->next = NULL;
    ktemp = Kroot;
    Head = New();
    Head->next = NULL;
    rtemp = Head;
    ndx=0;
} /* of Initialization */

```



```

ReadFilename() /* read file name */
{
    int len;
    fn=(char *)calloc(15,sizeof(char));
    system("cls");
    cursor(12,20);
    printf("Enter file name : ");
    gets(fn,15,stdin);
    if ((len=strlen(fn))>0)
        strcat(fn, ".bbf");
    return(len);
} /* of ReadFilename */

CheckFileName() /* check file name against the ledger file */
{
    FILE *lp,*fopen();
    int count=0,filno,filsize,found=FALSE,cond;
    char filename[20];
    lgr = (struct Ldgr *)calloc(1,sizeof(struct Ldgr));
    if (lp=fopen(lname,"r"))
    {
        rewind(lp);
        while (!feof(lp) && !found)
        {
            fscanf(lp,"%d %s %d\n",&filno,filename,&filsize);
            ++count;
            if ((cond = Compare(fn,filename))== 0)
            {
                found = TRUE;
                lgr->filno = filno;
                strcpy(lgr->filename, filename);
                lgr->filsize = filsize;
            }
        } /* of while */
        fclose(lp);
    } /* of exist */
    if (found) return(lgr->filno);
    else if ((lp==NULL)||(!found)) /* file or LEDGER not exist */
    {
        strcpy(lgr->filename,fn);
        lgr->filsize=0;
        if ((!found)&&(lp!=NULL))
        {
            lgr->filno=count;
            NextFileNo=count;
        }
        else /* ledger not exists */
        {
            lgr->filno=1;
            NextFileNo=1;
        }
        return(0);
    }
} /* of Check */

```

```

SortName() /* sort author dictionary */
{
    FILE *ap,*tap,*fopen();
    struct NameDic *Ntree();
    int count=0,len=0,which,where;
    char *pseudo,*tdate,what,*tname;
    pseudo="AUTHOR.ORD";
    ap=fopen(aname,"r");
    tap=fopen(pseudo,"w");
    Nroot = NULL;
    rewind(ap);
    while (!feof(ap))
    {
        tname = (char *)calloc(30,sizeof(char));
        tdate = (char *)calloc(15,sizeof(char));
        fscanf(ap,"%d %s %d %ls %s\n",&which,tname,&where,
            &what,tdate);
        if ((len=strlen(tname))>0)
        {
            ++count;
            Nroot = Ntree(Nroot,tname,which,where,what,tdate);
        }
    } /* of while */
    if (count>0) Ntreeprint(tap,Nroot);
    fclose(ap);
    fclose(tap);
    free(tname);
    free(tdate);
    if (count>0)
    {
        system("del AUTHOR.DIC");
        rename(pseudo,aname);
    }
} /* of SortName */

/* build the names into a binary tree */
struct NameDic *Ntree(root,name,which,where,what,date)
struct NameDic *root;
char *name;
int which,where;
char what, *date;
{
    struct NameDic *NewNameDic();
    int cond;
    if (root == NULL)
    {
        root = NewNameDic();
        root->name = name;
        root->which = which;
        root->where = where;
        root->what = what;
        root->date = date;
        root->left = NULL;
        root->right = NULL;
    }
}

```

```

    }
    else if ((cond=Compare(name,root->name))>0) root->right =
        Ntree(root->right,name,which,where,what,date);
    else root->left=Ntree(root->left,name,which,where,what,date);
    return(root);
} /* of Ntree */

Ntreeprint(op,root) /* print the binary tree and write to disk */
FILE *op;
struct NameDic *root;
{
    if (root != NULL)
    {
        Ntreeprint(op,root->left);
        fprintf(op,"%d %s %d %c %s\n",root->which,root->name,
            root->where,root->what,root->date);
        Ntreeprint(op,root->right);
    }
} /* of Ntreeprint */

struct NameDic *NewNameDic()
{
    return((struct NameDic *)calloc(1,sizeof(struct NameDic)));
} /* of NewNameDic */

WantSave() /* read the choice to save or not */
{
    int c,sure=FALSE;
    while (!sure)
    {
        system("cls");
        cursor(12,20);
        printf("Save or Not (Y/N)? ");
        c=getch();
        putchar(c);
        if ((c==78)||(c==110)) /* No */
        {
            system("cls");
            cursor(12,25);
            printf("Are you sure (Y/N)? ");
            c=getch();
            putchar(c);
            if ((c==89)||(c==121)) return(0);/* Yes */
        }
        else return(1);
    } /* of while */
} /* of WantSave */

Showchoice() /* display choices */
{
    system("cls");
    cursor(11,15);
    printf("[J] = Journal, [B] = Book, [X] = Exit");
    cursor(22,15);
    printf("Enter your choice(J, B, X) : ");
}

```

```

    } /* of Showchoice */

Writing() /* write to a disk file */
{
    FILE *ip, *ap, *fopen();
    switch(ActCode)
    {
        case 65 : ip = fopen(lgr->filename,"a"); break;
        case 67 : ip = fopen(lgr->filename,"w"); break;
        default : return(1);
    }
    ap = fopen(aname,"a");
    Traverse(Head, ip);
    TvsNameDic(Aroot,ap);
    fclose(ip);
    fclose(ap);
} /* of Writing */

MarkBlank(s) /* replace blank with a '*' in names and dates */
char *s;
{
    char *stpblk(),*q;
    int len,i;
    q=stpblk(s);
    len=strlen(q);
    for (i=0;i<len;i++,q++)
        if (isspace(*q)) *q='*';
} /* of MarkBlank */

/* traverse the binary tree and write to disk file */
TvsNameDic(ahead,ap)
struct AuthorDic *ahead;
FILE *ap;
{
    struct AuthorDic *temp;
    temp = ahead->next;
    while (temp != NULL)
    {
        MarkBlank(temp->name);
        MarkBlank(temp->date);
        fprintf(ap,"%d %s %d %c %s\n",temp->which,temp->name,
            temp->where,temp->what,temp->date);
        temp = temp->next;
    }
} /* of TvsNameDic */

Traverse(head,ip) /* Write into diskette */
struct Reference *head;
FILE *ip;
{
    int i;
    struct Reference *temp;
    temp = head->next;
    while (temp!=NULL)
    {

```

```

        fprintf(ip,"%c",temp->B_or_J);
        fprintf(ip,"\n%d",temp->Namecnt);
        for (i=0;i<temp->Namecnt;i++)
            fprintf(ip,"\n%s",temp->NameList[i]);

        if (temp->B_or_J == 'J')
        {
            fprintf(ip,"\n%d",temp->J_Part.KeywordCnt);
            for (i=0;i<temp->J_Part.KeywordCnt;i++)
                fprintf(ip,"\n%s",temp->J_Part.KeywordList[i]);
            fprintf(ip,"\n%s",temp->J_Part.J_Name);
            fprintf(ip,"\n%s",temp->J_Part.Topic);
        }
        else
        {
            fprintf(ip,"\n%s",temp->B_Part.BookName);
            fprintf(ip,"\n%s",temp->B_Part.City);
            fprintf(ip,"\n%s",temp->B_Part.Publisher);
        } /* of else */
        fprintf(ip,"\n%s",temp->Pages);
        fprintf(ip,"\n%s\n",temp->Date);
        temp = temp->next;
    } /* of while */
} /* of Traverse */

Release() /* free memory */
{
    free(Aroot);
    free(Kroot);
    free(Head);
    free(atemp);
    free(ktemp);
    free(rtemp);
    free(lgr);
} /* Release */

/* Allocate memory for linked list nodes */
struct Reference *New()
{ return((struct Reference *)calloc(1,
    sizeof(struct Reference))); } /* of New */

struct AuthorDic *NewName() /* Allocate for AUTHOR dictionary */
{ return((struct AuthorDic *)calloc(1,
    sizeof(struct AuthorDic))); } /* of NewName */

ReceiveJ() /* Read journal info from screen format */
{
    int c,more=1;
    InitJ(); /* Initialize tempj[ ][ ], and x,y */
    system("cls");
    cursor(0,0);
    printf("FILE NAME : %s",lgr->filename);
    cursor(1,0);
    printf("RECORD # : %d",ndx);
}

```

```

RefFormat();
cursor(x,y);
while (more)
{
    if (((c=getch()) == 13) && (x == 22)) more = 0;
    else if (c==13) { JRet(); cursor(x,y); }
    else if (c==0) /* Arrow Keys */
    {
        c=getch();
        if (c==72) JUp(); /* UpArrow */
        else if (c==80) JDown(); /* DnArrow */
        else if (c==75) JLeft(); /* LfArrow */
        else if (c==77) JRight(); /* RtArrow */
        cursor(x,y);
    } /* of Arrow Keys */
    else JEat(c);
} /* of while more */
AssignJ();
x=3; y=11;
} /* of ReceiveJ */

InitJ()
{
    int i,j;
    x=3;
    y=11;
    for (i=0;i<20;i++)
        for (j=0;j<70;j++)
            tempj[i][j]=0;
} /* of InitJ */

JRight() /* right arrow key is entered */
{
    if ((x>=3)&&(x<=8))
    {
        if (y<40) ++y;
        else if ((y==40)&&(x==8))
            { x=x+2; y=11; }
        else if (y==40)
            { ++x; y=11; }
    } /* of 3 to 8 */
    else if ((x==10)|| (x==13))
    {
        if (y<70) ++y;
        else { ++x; y=11; }
    } /* of 10,13 */
    else if ((x==11)|| (x==14))
    {
        if (y<70) ++y;
        else { x=x+2; y=11; }
    } /* of 11,14 */
    else if ((x==16)|| (x==22))
    {
        if (y<30) ++y;
        else if (y==30)

```

```

        {
            if (x==16)
                { x=x+2; y=11; }
            else putch(7);
        }
    } /* of 16,22 */
else if ((x==18)|| (x==19)|| (x==20))
{
    if (y==30) y=37;
    else if (y==56)
        {
            if (x==20) { x=22; y=11; }
            else { y=11; ++x; }
        }
    else if ((y<31)|| (y<56)) ++y;
} /* of 18,19,20 */
} /* of JRight */

JUp() /* up arrow key is entered */
{
    if (((x>3)&&(x<=8))|| (x==11)|| (x==14)|| (x==20)|| (x==19))
        { --x; y=11; }
    else if ((x==10)|| (x==13)|| (x==16)|| (x==18)|| (x==22))
        { x=x-2; y=11; }
    else if (x==3)
        { x=24; y=11; }
} /* of JUp */

JDown() /* down key is entered */
{
    if (((x<8) && (x>=3))|| (x==10)|| (x==13)|| (x==18)|| (x==19))
        { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==14)|| (x==16)|| (x==20))
        { x=x+2; y=11; }
    else if (x==22)
        { x=3; y=11; }
} /* of JDown */

JRet() /* return is entered */
{
    if (((x>=3) && (x<=7)) || (x==10) || (x==13))
        { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==14)|| (x==16)|| (x==20))
        {
            if ((x==20)&&(y>=11)&&(y<=30)) y=37;
            else { x=x+2; y=11; }
        }
    else if ((x==18) || (x==19))
        {
            if ((y<=31) && (y>=11)) y=37;
            else if ((y>=37) && (y <=59)) { ++x; y=11; }
        }
} /* of JRet */

JLeft() /* left arrow is entered */

```

```

{
    if ((x>3)&&(x<=8))
    {
        if (y>11) --y;
        else { --x; y=40; }
    } /* of 3 to 8 */
    else if (x==3)
    {
        if (y==11) putch(7);
        else --y;
    }
    else if ((x==10)|| (x==13))
    {
        if (y>11) --y;
        else
        {
            if (x==10) y=40;
            else y=70;
            x=x-2;
        }
    } /* of 10,13 */
    else if ((x==11)|| (x==14))
    {
        if (y>11) --y;
        else { --x; y=70; }
    } /* of 11,14 */
    else if ((x==16)|| (x==22))
    {
        if (y>11) --y;
        else
        {
            if (x==16) y=70;
            else y=56;
            x=x-2;
        }
    } /* of 16,22 */
    else if ((x==18)|| (x==19)|| (x==20))
    {
        if (y==37) y=30;
        else if (y==11)
        {
            if (x==18) { x=16; y=30; }
            else { y=56; --x; }
        }
        else if ((y>11)|| (y>37)) --y;
    } /* of 18,19,20 */
} /* of JLeft */

JEat(c) /* accept the legal characters */
char c;
{
    int i;

    if (((x>=3)&&(x<=8))&&(y<=40)) ||
        ((x==10)|| (x==11)|| (x==13)|| (x==14))&&(y<=70)) ||

```



```

        ((x==16)&&(y<31)) ||
        ((x==22)&&(y<31)))
        JShow(c);
    else if ((x==18)|| (x==19)|| (x==20))
    {
        if ((y==31)|| (y==57)) putch(7);
        else JShow(c);
    }
    else putch(7);
} /* of JEat */

/* display and assign the characters and move the cursor */
JShow(c)
char c;
{
    putch(c);
    i=x-3;
    j=y-11;
    tempj[i][j++]=c;
    cursor(x,++y);
} /* of JShow */

AssignJ() /* copy from tempj[] to rtemp */
{
    struct Reference *New();
    struct AuthorDic *NewName();
    struct KeywordDic *NewWord();
    int wordcnt=0,namecnt=0,len,h,v,m,l,k;
    char *stpblk(),*big,word[30],*p;
    system("cls");
    rtemp->next=New();
    rtemp=rtemp->next;
    rtemp->B_or_J='J';
    if ((len=strlen(stpblk(tempj[13])))==0)
        strcpy(rtemp->Date,DefaultDate);
    else strcpy(rtemp->Date,stpblk(tempj[13]));
    for (k=0;k<6;k++)
    {
        p=stpblk(tempj[k]);
        if ((len=strlen(p)) >=1)
        {
            atemp->next=NewName();
            atemp=atemp->next;
            ++namecnt;
            strcpy(rtemp->NameList[k],p);
            strcpy(atemp->name,p);
            atemp->which=lgr->filno;
            atemp->where=ndx;
            atemp->what='J';
            strcpy(atemp->date,stpblk(rtemp->Date));
            atemp->next=NULL;
        }
    }
    rtemp->Namecnt=namecnt;
    big=(char *)calloc(MAXLINE,sizeof(char));

```

```

    strcat(big, stpblk(tempj[7]));
    strcat(big, stpblk(tempj[8]));
    strcpy(&rtemp->J_Part.Topic, big);
    free(big);
    big=(char *)calloc(MAXLINE, sizeof(char));
    strcat(big, stpblk(tempj[10]));
    strcat(big, stpblk(tempj[11]));
    strcpy(&rtemp->J_Part.J_Name, big);

    for (v=15; h=0; v<18; v++) /* get keywords */
    {
        for (k=0; k<20; k++)
            word[k] = tempj[v][k];
        strcpy(p, stpblk(word));
        if ((len=strlen(p))>=1)
        {
            ktemp->next=NewWord();
            ktemp=ktemp->next;
            strcpy(rtemp->J_Part.KeywordList[h], p);
            strcpy(ktemp->word, p);
            ktemp->which=lgr->filno;
            ktemp->where=ndx;
            ktemp->next=NULL;
            ++wordcnt;
        }
        ++h;
        for (k=26, l=0; k<57; l++, k++)
            word[l] = tempj[v][k];
        strcpy(p, stpblk(word));
        if ((len=strlen(p))>=1)
        {
            ktemp->next=NewWord();
            ktemp=ktemp->next;
            strcpy(rtemp->J_Part.KeywordList[h], p);
            strcpy(ktemp->word, p);
            ktemp->which=lgr->filno;
            ktemp->where=ndx;
            ktemp->next=NULL;
            ++wordcnt;
        }
        ++h;
    }
    rtemp->J_Part.KeywordCnt=wordcnt;
    strcpy(rtemp->Pages, stpblk(tempj[19]));
} /* of AssignJ */

```

```

/* s is the input string, t is the string been scanned */
Compare(s, t) /* compare two character strings */
char *s;
char *t;
{
    for (; toupper(*s)==toupper(*t); s++, t++)
        if (*s == '\0') return(0);
    return(toupper(*s)-toupper(*t));
}

```

```

    } /* of Compare */

ReceiveB() /* read book info from screen format */
{
    int c, more=TRUE;
    system("cls");
    cursor(0,0);
    printf("FILE NAME : %s", lgr->filename);
    cursor(1,0);
    printf("RECORD # : %d", ndx);
    BookFormat();
    InitB();
    cursor(x,y);
    while (more)
    {
        if ((c=getch()) == 13) && (x == 17) more = 0;
        else if (c==13) { BRet(); cursor(x,y); }
        else if (c==0) /* Arrow Keys */
        {
            c=getch();
            if (c==72) BUp(); /* UpArrow */
            else if (c==80) BDown(); /* DnArrow */
            else if (c==75) BLeft(); /* LfArrow */
            else if (c==77) BRight(); /* RtArrow */
            cursor(x,y);
        } /* of Arrow Keys */
        else BEat(c);
    } /* of while more */
    AssignB();
} /* of ReceiveB */

InitB()
{
    int i,j;
    x=3;
    y=11;
    for (i=0; i<15; i++)
        for (j=0; j<70; j++)
            tempb[i][j]=0;
} /* InitB */

BRet() /* return is entered */
{
    if ((x>=3)&&(x<8)) { ++x; y=11; }
    else if (x==8) { x=10; y=12; }
    else if (x==10) { x=11; y=12; }
    else if (x==11) { x=13; y=8; }
    else if (x==13)
    {
        if (y<=24) y=48;
        else { x=15; y=8; }
    }
    else if (x==15) { x=17; y=13; }
} /* of BReturn */

```

```

BUp() /* up arrow key is entered */
{
    if ((x>3)&&(x<=8)) { --x; y=11; }
    else if (x==3) { x=17; y=13; }
    else if (x==11) { --x; y=12; }
    else if ((x==10)|| (x==13)|| (x==15)|| (x==17))
    {
        if (x==13) y=12;
        else if ((x==15)|| (x==17)) y=8;
        else if (x==10) y=11;
        x=x-2;
    }
} /* of BUP */

BDown() /* down arrow key is entered */
{
    if ((x<8)&&(x>=3)) { ++x; y=11; }
    else if ((x==8)|| (x==11)|| (x==13)|| (x==15))
    {
        if (x==8) y=12;
        else if ((x==11)|| (x==13)) y=8;
        else if (x==15) y=13;
        x=x+2;
    }
    else if (x==10) { x=11; y=12; }
    else if (x==17) { x=3; y=11; }
} /* of BDOWN */

BRight() /* right arrow key is entered */
{
    if ((x>=3)&&(x<=8))
    {
        if (y<40) ++y;
        else if ((y==40)&&(x==8)) { x=10; y=12; }
        else if (y==40) { ++x; y=11; }
    }
    else if ((x==10)|| (x==11))
    {
        if (y<70) ++y;
        else if (y==70)
        {
            if (x==10) { x=11; y=12; }
            else { x=13; y=8; }
        }
    }
    else if (x==13)
    {
        if (((y>=8)&&(y<23)) || ((y>=48)&&(y<62))) ++y;
        else if (y==23) y=48;
        else if (y==62) { x=15; y=8; }
    }
    else if (x==15)
    {
        if (y<37) ++y;
    }
}

```

```

        else if (y==37) { x=17; y =13; }
    }
    else if (x==17)
    {
        if (y<62) ++y;
        else putch(7);
    }
} /* of BRIGHT */

BLeft() /* left arrow key is down */
{
    if ((x>3)&&(x<=8))
    {
        if (y>11) --y;
        else if (y==11) { --x; y=40; }
    }
    else if (x==3)
    {
        if (y==11) putch(7);
        else --y;
    }
    else if ((x==10)|| (x==11))
    {
        if (y>12) --y;
        else if (y==12)
        {
            if (x==10) { x=8; y=40; }
            else if (x==11) { x=10; y=70; }
        }
    }
    else if (x==13)
    {
        if (((y>8)&&(y<=23))||((y>48)&&(y<=62))) --y;
        else if (y==8) { x=11; y=70; }
        else if (y==48) y=23;
    }
    else if (x==15)
    {
        if (y>8) --y;
        else if (y==8) { x=13; y=62; }
    }
    else if (x==17)
    {
        if (y>13) --y;
        else if (y==13) { x=15; y=37; }
    }
} /* of BLEFT */

BEat(c)
char c;
{
    if (((x>3)&&(x<=8))&&(y<=40)) ||
        ((x==10)|| (x==11))&&(y<=70)) ||
        ((x==15)&&(y<=37)) ||
        ((x==17)&&(y<=62)))

```

```

        BShow(c);
    else if (x==13)
    {
        if ((y==24)|| (y==63)) putch(7);
        else BShow(c);
    }
    else putch(7);
} /* of BEat */

/* display and assign the characters and move the cursor */
BShow(c)
char c;
{
    int i=0;

    putch(c);
    i=x-3;
    if ((x>=3)&&(x<=8)) j=y-11;
    else if ((x==10)|| (x==11)) j=y-12;
    else if ((x==13)|| (x==15)) j=y-8;
    else if (x==17) j=y-13;
    tempb[i][j++]=c;
    cursor(x,++y);
} /* of BShow */

AssignB() /* assign from tempb[] to rtemp */
{
    struct Reference *New();
    struct AuthorDic *NewName();
    int wordcnt=0,namecnt=0,len,h,v,m,l,k;
    char *stpblk(),"big,temp[30],*p;
    system("cls");
    rtemp->next=New();
    rtemp=rtemp->next;
    rtemp->B_or_J='B';

    for (h=0;h<15;h++) /* copy date */
        temp[h] = tempb[10][h];
    strcpy(p,stpblk(temp));
    if ((len=strlen(p))>=4) strcpy(rtemp->Date,p);
    else strcpy(rtemp->Date,DefaultDate);

    for (k=0;k<6;k++)
    {
        p=stpblk(tempb[k]);
        if ((len=strlen(p)) >=1)
        {
            atemp->next=NewName();
            atemp=atemp->next;
            ++namecnt;
            strcpy(rtemp->NameList[k],p);
            strcpy(atemp->name,p);
            atemp->which=lgr->filno;
            atemp->where=ndx;
            atemp->what='B';
        }
    }
}

```

```

        strcpy(atemp->date,rtemp->Date);
        atemp->next=NULL;
    }
}
rtemp->Namecnt=namecnt;
big=(char *)calloc(MAXLINE,sizeof(char));
strcat(big,stpbk(tempb[7]));
strcat(big,stpbk(tempb[8]));
strcpy(&rtemp->B_Part.BookName,big);
free(big);
strcpy(rtemp->B_Part.City,stpbk(tempb[12]));
strcpy(rtemp->B_Part.Publisher,stpbk(tempb[14]));

for (k=40,h=0;k<54;h++,k++) /* copy page */
    temp[h] = tempb[10][k];
strcpy(p,stpbk(temp));
strcpy(rtemp->Pages,p);
} /* of AssignB */

ScanFile() /* scan input file */
{
    FILE *fp,*fopen();
    char *FetchLine();
    char bj[2];
    int z,i,c=0,k;

    fp = fopen(lgr->filename,"r");
    while (c < lgr->filesize)
    {
        fgets(bj,MAXLEN,fp);
        Ary[c].B_or_J = bj[0];
        fgets(z,MAXLEN,fp);
        Ary[c].Namecnt = atoi(z);
        for (i=0; i < Ary[c].Namecnt; i++)
        {
            strcpy(Ary[c].NameList[i],FetchLine(fp));
        }
        if (Ary[c].B_or_J == 'J')
        {
            fgets(k,MAXLEN,fp);
            Ary[c].J_Part.KeywordCnt = atoi(k);
            for (i=0; i<Ary[c].J_Part.KeywordCnt; i++)
                strcpy(Ary[c].J_Part.KeywordList[i],FetchLine(fp));

            strcpy(Ary[c].J_Part.J_Name,FetchLine(fp));
            strcpy(Ary[c].J_Part.Topic,FetchLine(fp));
        }
        else
        {
            strcpy(Ary[c].B_Part.BookName,FetchLine(fp));
            strcpy(Ary[c].B_Part.City,FetchLine(fp));
            strcpy(Ary[c].B_Part.Publisher,FetchLine(fp));
        }
        strcpy(Ary[c].Pages,FetchLine(fp));
        strcpy(Ary[c].Date,FetchLine(fp));
    }
}

```

```

        ++c;
    } /* of while */
    fclose(fp);
} /* of ScanFile */

char *FetchLine(ip) /* read one line from disk file */
FILE *ip;
{
    char line[MAXLINE], *q;
    int len;
    if ((q=(char *)malloc(len=strlen(fgets(line,MAXLINE,ip))))
        !=NULL)
    {
        line[len-1] = '\0';
        return(line);
    }
    else
    {
        system("cls");
        cursor(12,25);
        printf("Insufficient memory!");
        exit(1);
    }
} /* of FetchLine */

```


BIBLIOGRAPHIC SYSTEM FOR MICROCOMPUTER ENVIRONMENTS

By

Wei Lee

B.S., National Chung Hsiung University, Taiwan, R.O.C., 1980

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**

1986

ABSTRACT

A bibliographic system, REFWELL, has been designed and implemented for the use of the faculty members in the Kansas State University, Computer Science Department. REFWELL provides on microcomputer system the capabilities necessary for them to manage their large number of bibliographic references.

The system is implemented on an IBM compatible personal computer using the C programming language.

The menu-driven system uses full-screen data entry and has on-line help messages which render REFWELL a user-friendly system. REFWELL provides six major functions which are: CREATE, APPEND, CHANGE, RETRIEVE, FORMAT, and JOIN. Functions DELETE and EXTRACT are embedded in RETRIEVE. One of the unique and powerful features of REFWELL is the selection and subsequent formatting of a subset of references into one of the two predefined formats. These two formats are used by the ACM Computing Surveys and ACM Transactions on Office Information Systems.

Appendices contain a copy of the user's manual together with the source codes.

1430-78
00-53