**Problem Statement**

We have built an interactive Library Management System for the librarian. Here librarian can login in to the system with specific username and password. After the login the librarian can make changes in the library data.

**Functionalities /features supported by project**

* Librarian can –
* Add Books
* Delete books
* Search Books
* Issue Books
* Return Book
* Display Books
* Add New User
* Delete User
* Change Password
* Due to the login feature the system is secured.
* Add and Delete users that a login with their own Username and Password.
* User can change Password anytime.
* User can Issue books.
* Data is stored in files so even after the application is closed data will be saved for future.
* All the data is stored in the file which can be accessed by the librarian for later used.
* Gives the librarian the flexibility to easily add books to data storage of the library, issue books, search for a book, display details of students who have reached their deadline and haven’t returned book.

**Description about selected data structure and reason to select it**

The data structures we used here is Linked List. The reasons for using Linked List are as follows :-

* Makes easy for us to insert and delete a node.
* No memory wastage.
* Faster access time.

**Any special algorithm or method used for implementation**

* We have implemented Files I/O to store our data even after application is closed.
* We have also added a system where users can be added or deleted. New users can even change their passwords just like the admin.

**Code and output of the program**

**/\*BookList.h\*/**

**#ifndef BOOK\_LIST\_H**

**#define BOOK\_LIST\_H**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <time.h>**

**struct BookData**

**{**

**int book\_code;**

**char book\_name[50];**

**char book\_author[50];**

**int book\_shelf\_no;**

**/\*Set Value to 0 or 1 depending on if book is issued\*/**

**int is\_issued;**

**char issue\_student\_name[50];**

**time\_t issue\_time;**

**};**

**struct BookNode**

**{**

**struct BookNode\* next;**

**struct BookData data;**

**};**

**struct BookList**

**{**

**struct BookNode\* m\_head;**

**};**

**typedef void (\*ShowBookData)(struct BookData);**

**void BookListFree(struct BookNode\* del)**

**{**

**free(del);**

**}**

**void BookListInitialise(struct BookList\* p\_book\_list)**

**{**

**p\_book\_list = (struct BookList\*)malloc(sizeof(struct BookList));**

**p\_book\_list->m\_head = NULL;**

**}**

**void BookListDestroy(struct BookList\* p\_list)**

**{**

**struct BookNode\* it;**

**while (p\_list != NULL && p\_list->m\_head != NULL)**

**{**

**it = p\_list->m\_head;**

**p\_list->m\_head = it->next;**

**BookListFree(it);**

**}**

**p\_list->m\_head = NULL;**

**p\_list = NULL;**

**}**

**void BookListReset(struct BookList\* p\_list)**

**{**

**BookListDestroy(p\_list);**

**BookListInitialise(p\_list);**

**}**

**int BookListEmpty(struct BookList\* p\_list)**

**{**

**return p\_list == NULL || p\_list->m\_head == NULL;**

**}**

**int BookListDeleteByCode(struct BookList\* p\_list, const int code)**

**{**

**struct BookNode\* tmp;**

**struct BookNode\* it;**

**if (BookListEmpty(p\_list))**

**return 0;**

**/\*Check Head First\*/**

**if (p\_list->m\_head != NULL && (p\_list->m\_head->data.book\_code == code))**

**{**

**tmp = p\_list->m\_head;**

**p\_list->m\_head = p\_list->m\_head->next;**

**BookListFree(tmp);**

**/\*Node Deleted\*/**

**return 1;**

**}**

**it = p\_list->m\_head;**

**/\* Use this to iterate through the loop\*/**

**while (it != NULL && it->next != NULL)**

**{**

**/\*Condition to Delete found\*/**

**if (it->next->data.book\_code == code)**

**{**

**tmp = it->next;**

**it->next = it->next->next;**

**BookListFree(tmp);**

**/\*Node Deleted\*/**

**return 1;**

**}**

**else**

**{**

**it = it->next;**

**}**

**}**

**/\*Unable to Delete Node\*/**

**return 0;**

**}**

**int BookListFindByCode(struct BookList\* p\_list, const int code, struct BookData\* book)**

**{**

**struct BookNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**if (it->data.book\_code == code)**

**{**

**if (book != NULL)**

**\*book = it->data;**

**/\*Element Found Return True\*/**

**return 1;**

**}**

**}**

**/\*Element Not Found, Return False\*/**

**return 0;**

**}**

**int BookListFindByStudentName(struct BookList\* p\_list,**

**const char\* const student\_name,**

**struct BookData\* book)**

**{**

**struct BookNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**if (strcmp(it->data.issue\_student\_name, student\_name) == 0)**

**{**

**/\*We Shall Pass NULL Here when we do not need the Return Value\*/**

**if (book != NULL)**

**\*book = it->data;**

**/\*Element Found Return True\*/**

**return 1;**

**}**

**}**

**/\*Element Not Found, Return False\*/**

**return 0;**

**}**

**void SwapBookNode(struct BookNode\* left, struct BookNode\* right)**

**{**

**struct BookData\* tmp = (struct BookData\*)malloc(sizeof(struct BookData));**

**struct BookData\* left\_data = &left->data;**

**struct BookData\* right\_data = &right->data;**

**/\*Assign left to tmp\*/**

**{**

**strcpy(tmp->book\_author, left\_data->book\_author);**

**strcpy(tmp->book\_name, left\_data->book\_name);**

**strcpy(tmp->issue\_student\_name, left\_data->issue\_student\_name);**

**tmp->is\_issued = left\_data->is\_issued;**

**tmp->issue\_time = left\_data->issue\_time;**

**tmp->book\_shelf\_no = left\_data->book\_shelf\_no;**

**tmp->book\_code = left\_data->book\_code;**

**}**

**/\*Assign right to left\*/**

**{**

**strcpy(left\_data->book\_author, right\_data->book\_author);**

**strcpy(left\_data->book\_name, right\_data->book\_name);**

**strcpy(left\_data->issue\_student\_name, right\_data->issue\_student\_name);**

**left\_data->is\_issued = right\_data->is\_issued;**

**left\_data->issue\_time = right\_data->issue\_time;**

**left\_data->book\_shelf\_no = right\_data->book\_shelf\_no;**

**left\_data->book\_code = right\_data->book\_code;**

**}**

**/\*Assign Tmp to Right\*/**

**{**

**strcpy(right\_data->book\_author, tmp->book\_author);**

**strcpy(right\_data->book\_name, tmp->book\_name);**

**strcpy(right\_data->issue\_student\_name, tmp->issue\_student\_name);**

**right\_data->is\_issued = tmp->is\_issued;**

**right\_data->issue\_time = tmp->issue\_time;**

**right\_data->book\_shelf\_no = tmp->book\_shelf\_no;**

**right\_data->book\_code = tmp->book\_code;**

**}**

**}**

**void BookListShowByShelfNumber(struct BookList\* p\_list, ShowBookData p\_show)**

**{**

**struct BookNode\* first\_it;**

**struct BookNode\* end\_it = NULL;**

**int is\_swapped;**

**if (p\_list == NULL || p\_list->m\_head == NULL)**

**return;**

**do**

**{**

**is\_swapped = 0;**

**for (first\_it = p\_list->m\_head; first\_it->next != end\_it; first\_it = first\_it->next)**

**{**

**if (first\_it->data.book\_shelf\_no > first\_it->next->data.book\_shelf\_no)**

**{**

**SwapBookNode(first\_it, first\_it->next);**

**is\_swapped = 1;**

**}**

**}**

**end\_it = first\_it;**

**} while (is\_swapped);**

**/\* Use this to iterate through the loop\*/**

**for (first\_it = p\_list->m\_head; first\_it != NULL; first\_it = first\_it->next)**

**{**

**const struct BookData book = first\_it->data;**

**if (!book.is\_issued)**

**p\_show(book);**

**}**

**}**

**void BookListDisplayDelayedList(struct BookList\* p\_list)**

**{**

**struct BookNode\* it;**

**time\_t time\_diff;**

**struct tm spent\_time;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**const struct BookData book = it->data;**

**if (!book.is\_issued)**

**continue;**

**time\_diff = time(NULL) - book.issue\_time;**

**spent\_time = \*localtime(&time\_diff);**

**/\* Minimum Delay Period\*/**

**/\* If less than that, student still has time to return\*/**

**if (spent\_time.tm\_yday <= 7)**

**continue;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Code\t:%d\n", book.book\_code);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Name\t:%s", book.book\_name);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Student Issuer Name\t:%s", book.issue\_student\_name);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Delayed By\t: %d\n", spent\_time.tm\_yday - 7);**

**}**

**}**

**void BookListDisplayByShelfNumber(struct BookList\* p\_list,**

**const int p\_shelf\_no,**

**ShowBookData p\_show)**

**{**

**struct BookNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**const struct BookData book = it->data;**

**if (book.book\_shelf\_no == p\_shelf\_no)**

**p\_show(book);**

**}**

**}**

**void BookListDisplayedAllList(struct BookList\* p\_list, ShowBookData show)**

**{**

**struct BookNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**const struct BookData book = it->data;**

**show(book);**

**}**

**}**

**void BookListFindByString(struct BookList\* p\_list,**

**const char\* const find\_str,**

**ShowBookData show)**

**{**

**struct BookNode\* it;**

**char\* search\_str;**

**struct BookData book;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**book = it->data;**

**search\_str = (char\*)malloc(sizeof(char) \***

**(strlen(book.book\_author) + strlen(book.book\_name) + 2));**

**strcpy(search\_str, book.book\_author);**

**strcat(search\_str, ":");**

**strcat(search\_str, book.book\_name);**

**if (strstr(search\_str, find\_str) != NULL)**

**{**

**show(book);**

**}**

**free(search\_str);**

**}**

**}**

**void BookListInsertInto(struct BookList\* list, struct BookData data)**

**{**

**struct BookNode\* newnode = (struct BookNode\*)(malloc(sizeof(struct BookNode)));**

**newnode->data = data;**

**newnode->next = NULL;**

**/\* As we are Inserting at Front\*/**

**newnode->next = list->m\_head;**

**list->m\_head = newnode;**

**}**

**int BookListBookDeIssue(struct BookList\* p\_list,**

**const int book\_code,**

**struct BookData\* p\_book)**

**{**

**struct BookNode\* it;**

**struct BookData tmp;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**if (it->data.book\_code == book\_code)**

**{**

**if (!it->data.is\_issued)**

**/\*Cannot DeIssue Books which have not been issued\*/**

**return 0;**

**strcpy(tmp.book\_author, it->data.book\_author);**

**strcpy(tmp.book\_name, it->data.book\_name);**

**strcpy(tmp.issue\_student\_name, "");**

**tmp.is\_issued = 0;**

**tmp.issue\_time = 0;**

**tmp.book\_shelf\_no = it->data.book\_shelf\_no;**

**tmp.book\_code = it->data.book\_code;**

**BookListDeleteByCode(p\_list, book\_code);**

**BookListInsertInto(p\_list, tmp);**

**if (p\_book != NULL)**

**\*p\_book = tmp;**

**return 1;**

**}**

**}**

**return 0;**

**}**

**int BookListBookIssue(struct BookList\* p\_list,**

**const char\* const book\_issuer\_student,**

**const int book\_code)**

**{**

**struct BookNode\* it;**

**struct BookData tmp;**

**/\*Check if Student Has not issued another book\*/**

**if (BookListFindByStudentName(p\_list, book\_issuer\_student, NULL))**

**return 0;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**if (it->data.book\_code == book\_code)**

**{**

**if (it->data.is\_issued)**

**/\*Cannot Issue Books Twice To Different People\*/**

**return 0;**

**strcpy(tmp.book\_author, it->data.book\_author);**

**strcpy(tmp.book\_name, it->data.book\_name);**

**strcpy(tmp.issue\_student\_name, book\_issuer\_student);**

**tmp.is\_issued = 1;**

**/\*Get Current Time to set as time of issue\*/**

**tmp.issue\_time = time(NULL);**

**tmp.book\_shelf\_no = it->data.book\_shelf\_no;**

**tmp.book\_code = it->data.book\_code;**

**BookListDeleteByCode(p\_list, book\_code);**

**BookListInsertInto(p\_list, tmp);**

**return 1;**

**}**

**}**

**/\*Unable to Insert Book\*/**

**return 0;**

**}**

**int BookListAdd(struct BookList\* list,**

**const int book\_code,**

**const char\* const book\_name,**

**const char\* const book\_author,**

**const int book\_shelf\_no)**

**{**

**struct BookData book;**

**/\* Books with the same code cannot be inserted\*/**

**if (BookListFindByCode(list, book\_code, NULL))**

**return 0;**

**strcpy(book.issue\_student\_name, "");**

**strcpy(book.book\_author, book\_author);**

**strcpy(book.book\_name, book\_name);**

**book.book\_code = book\_code;**

**book.book\_shelf\_no = book\_shelf\_no;**

**book.issue\_time = 0;**

**book.is\_issued = 0;**

**BookListInsertInto(list, book);**

**return 1;**

**}**

**#endif**

**/\*UserList.h\*/**

**#ifndef USERLIST\_H**

**#define USERLIST\_H**

**#include <stdlib.h>**

**#include <string.h>**

**enum UserAuthority**

**{**

**USER = 0,**

**ADMIN = 1**

**};**

**struct UserData**

**{**

**char m\_user\_name[50];**

**char m\_passwd[20];**

**enum UserAuthority m\_authority;**

**};**

**struct UserNode**

**{**

**struct UserData data;**

**struct UserNode\* next;**

**};**

**struct UserList**

**{**

**struct UserNode\* m\_head;**

**};**

**void UserNodeFree(struct UserNode\* del)**

**{**

**free(del);**

**}**

**int UserListEmpty(struct UserList\* p\_user\_list)**

**{**

**return p\_user\_list->m\_head == NULL;**

**}**

**int UserListHasNoAdmin(struct UserList\* p\_user\_list)**

**{**

**struct UserNode\* it;**

**if (p\_user\_list == NULL)**

**return 1;**

**for (it = p\_user\_list->m\_head; it != NULL; it = it->next)**

**{**

**if (it->data.m\_authority == ADMIN)**

**return 0;**

**}**

**return 1;**

**}**

**void UserListInitialise(struct UserList\* p\_user\_list)**

**{**

**p\_user\_list = (struct UserList\*)malloc(sizeof(struct UserList));**

**p\_user\_list->m\_head = NULL;**

**}**

**void UserListDestroy(struct UserList\* p\_list)**

**{**

**struct UserNode\* it;**

**if (p\_list == NULL)**

**return;**

**while (p\_list->m\_head != NULL)**

**{**

**it = p\_list->m\_head;**

**p\_list->m\_head = it->next;**

**UserNodeFree(it);**

**}**

**}**

**void UserListReset(struct UserList\* p\_list)**

**{**

**UserListDestroy(p\_list);**

**UserListInitialise(p\_list);**

**}**

**void UserListInsertToNode(struct UserList\* p\_user\_list, struct UserNode\* newnode)**

**{**

**if (p\_user\_list == NULL)**

**UserListInitialise(p\_user\_list);**

**/\* As we are Inserting at Front\*/**

**newnode->next = p\_user\_list->m\_head;**

**p\_user\_list->m\_head = newnode;**

**}**

**/\*Use this to verify Existence of user\*/**

**int UserListAuthenticateUser(struct UserList\* p\_list,**

**const char\* const p\_user\_name,**

**const char\* const p\_password,**

**struct UserData\* p\_data)**

**{**

**struct UserNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**/\*User Found\*/**

**if (strcmp(it->data.m\_user\_name, p\_user\_name) == 0)**

**{**

**if (strcmp(it->data.m\_passwd, p\_password) == 0)**

**{**

**if (p\_data != NULL)**

**\*p\_data = it->data;**

**return 1;**

**}**

**else**

**{**

**/\*While User Name matches, password doesn't match\*/**

**/\*User Not Authenticated\*/**

**return 0;**

**}**

**}**

**}**

**/\*User Not Found\*/**

**return 0;**

**}**

**int UserListUserExists(struct UserList\* p\_list, const char\* const p\_user\_name)**

**{**

**struct UserNode\* it;**

**/\* Use this to iterate through the loop\*/**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**/\*User Found\*/**

**if (strcmp(it->data.m\_user\_name, p\_user\_name) == 0)**

**return 1;**

**}**

**/\*User Not Found\*/**

**return 0;**

**}**

**int UserListAdd(struct UserList\* p\_user\_list,**

**const char\* const p\_user\_name,**

**const char\* const p\_passwd,**

**const enum UserAuthority p\_authority)**

**{**

**struct UserNode\* user;**

**if (UserListUserExists(p\_user\_list, p\_user\_name))**

**return 0;**

**user = (struct UserNode\*)malloc(sizeof(struct UserNode));**

**strcpy(user->data.m\_user\_name, p\_user\_name);**

**strcpy(user->data.m\_passwd, p\_passwd);**

**user->data.m\_authority = p\_authority;**

**UserListInsertToNode(p\_user\_list, user);**

**return 1;**

**}**

**int UserListDeleteUserByName(struct UserList\* p\_list, const char\* p\_user\_name)**

**{**

**struct UserNode\* tmp;**

**struct UserNode\* it;**

**/\*Check Head First\*/**

**while (p\_list->m\_head != NULL &&**

**strcmp(p\_list->m\_head->data.m\_user\_name, p\_user\_name) == 0)**

**{**

**tmp = p\_list->m\_head;**

**if (tmp->data.m\_authority == ADMIN)**

**return 0;**

**p\_list->m\_head = p\_list->m\_head->next;**

**UserNodeFree(tmp);**

**return 1;**

**}**

**it = p\_list->m\_head;**

**/\* Use this to iterate through the loop\*/**

**while (it != NULL && it->next != NULL)**

**{**

**/\*Condition to Delete found\*/**

**if (strcmp(it->next->data.m\_user\_name, p\_user\_name) == 0)**

**{**

**tmp = it->next;**

**if (tmp->data.m\_authority == ADMIN)**

**return 0;**

**it->next = it->next->next;**

**UserNodeFree(tmp);**

**it = it->next;**

**return 1;**

**}**

**else**

**{**

**it = it->next;**

**}**

**}**

**return 0;**

**}**

**int UserListChangePassword(struct UserList\* p\_list,**

**const char\* const p\_user\_name,**

**const char\* const p\_old\_password,**

**const char\* const p\_new\_password)**

**{**

**struct UserData old\_data;**

**enum UserAuthority authority;**

**/\*Check if Old password gets authenticated\*/**

**if (!UserListAuthenticateUser(p\_list, p\_user\_name, p\_old\_password, &old\_data))**

**return 0;**

**/\*Delete Old user\*/**

**UserListDeleteUserByName(p\_list, p\_user\_name);**

**/\*Get User Authority\*/**

**authority = old\_data.m\_authority;**

**/\*Add New\*/**

**return UserListAdd(p\_list, p\_user\_name, p\_new\_password, authority);**

**}**

**#endif**

**/\*FileIO.h\*/**

**#ifndef FILEIO\_H**

**#define FILEIO\_H**

**/\* Use this to Interact with the File\*/**

**#include <stdio.h>**

**/\* Returns true if file opened for writing\*/**

**int FileOpenWrite(FILE\*\* f, const char\* p\_file\_name)**

**{**

**\*f = fopen(p\_file\_name, "wb");**

**return (\*f != NULL);**

**}**

**/\* Returns true if file opened for append\*/**

**int FileOpenAppend(FILE\*\* f, const char\* p\_file\_name)**

**{**

**\*f = fopen(p\_file\_name, "ab");**

**return (\*f != NULL);**

**} /\* Returns true if file opened for reading\*/**

**int FileOpenRead(FILE\*\* f, const char\* p\_file\_name)**

**{**

**\*f = fopen(p\_file\_name, "rb");**

**return (\*f != NULL);**

**}**

**void FileClose(FILE\*\* f)**

**{**

**fclose(\*f);**

**}**

**#include "UserList.h"**

**int FileWriteUserList(FILE\*\* f, struct UserList\* p\_list)**

**{**

**struct UserNode\* it;**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**const size\_t ret = fwrite(&it->data, sizeof(struct UserData), 1, \*f);**

**if (ret != 1)**

**{**

**/\*If Writing Failed\*/**

**return 0;**

**}**

**}**

**/\*All Elements were properly written\*/**

**return 1;**

**}**

**void FileReadUserList(FILE\* f, struct UserList\* p\_list)**

**{**

**void\* buf = malloc(sizeof(struct UserData));**

**while (fread(buf, sizeof(struct UserData), 1, f) > 0)**

**{**

**/\*Write All Elements to Buffer\*/**

**struct UserData\* data = (struct UserData\*)buf;**

**UserListAdd(p\_list, data->m\_user\_name, data->m\_passwd, data->m\_authority);**

**}**

**free(buf);**

**/\*All Elements were properly read\*/**

**}**

**#include "BookList.h"**

**int FileAppendBook(FILE\*\* f, struct BookData\* data)**

**{**

**return fwrite(data, sizeof(struct BookData), 1, \*f) == 1;**

**}**

**int FileWriteBookList(FILE\*\* f, struct BookList\* p\_list)**

**{**

**struct BookNode\* it;**

**size\_t ret;**

**for (it = p\_list->m\_head; it != NULL; it = it->next)**

**{**

**ret = fwrite(&it->data, sizeof(struct BookData), 1, \*f);**

**if (ret != 1)**

**{**

**/\*If Writing Failed\*/**

**return 0;**

**}**

**}**

**/\*All Elements were properly written\*/**

**return 1;**

**}**

**void FileReadBookList(FILE\* f, struct BookList\* p\_list)**

**{**

**void\* buf = malloc(sizeof(struct BookData));**

**while (fread(buf, sizeof(struct BookData), 1, f) > 0)**

**{**

**/\*Write All Elements to Buffer\*/**

**struct BookData\* data = (struct BookData\*)buf;**

**BookListInsertInto(p\_list, \*data);**

**}**

**free(buf);**

**/\*All Elements were properly read\*/**

**}**

**#endif**

**/\*Constants.h\*/**

**#ifndef CONSTANTS\_H**

**#define CONSTANTS\_H**

**#define USER\_FILE\_NAME "user\_file.dat"**

**#define BOOK\_FILE\_NAME "book\_file.dat"**

**#endif**

**/\*Login.h\*/**

**#ifndef LOGIN\_H**

**#define LOGIN\_H**

**#include "Constants.h"**

**#include "FileIO.h"**

**#include "UserList.h"**

**struct UserList g\_user\_list;**

**FILE\* g\_user\_file;**

**char g\_current\_user[50];**

**void UserListWriteToFile(void)**

**{**

**if (!FileOpenWrite(&g\_user\_file, USER\_FILE\_NAME))**

**return;**

**FileWriteUserList(&g\_user\_file, &g\_user\_list);**

**FileClose(&g\_user\_file);**

**UserListReset(&g\_user\_list);**

**}**

**void UserListReadFromFile(void)**

**{**

**if (!FileOpenRead(&g\_user\_file, USER\_FILE\_NAME))**

**return;**

**UserListReset(&g\_user\_list);**

**FileReadUserList(g\_user\_file, &g\_user\_list);**

**FileClose(&g\_user\_file);**

**}**

**int Login(enum UserAuthority\* p\_authority)**

**{**

**char username[50];**

**char password[20];**

**struct UserData user;**

**system("cls");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Please Enter Username\n");**

**scanf(" %[^\n]s", username);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Please Enter Password\n");**

**scanf(" %[^\n]s", password);**

**UserListReadFromFile();**

**if (!UserListAuthenticateUser(&g\_user\_list, username, password, &user))**

**{**

**/\*Unable to Login\*/**

**return 0;**

**}**

**/\*Login Successful.\*/**

**strcpy(g\_current\_user, user.m\_user\_name);**

**\*p\_authority = user.m\_authority;**

**return 1;**

**}**

**#endif**

**/\*Menu.h\*/**

**#ifndef MENU\_H**

**#define MENU\_H**

**/\*Define this to reduce error messages about unsafe functions like\*/**

**/\*scanf etc\*/**

**#ifdef \_MSC\_VER**

**# define \_CRT\_SECURE\_NO\_WARNINGS**

**#endif**

**#include <conio.h>**

**#include <stdio.h>**

**#include "BookList.h"**

**#include "Constants.h"**

**#include "FileIO.h"**

**#include "Login.h"**

**void addbook(void);**

**int usermenu(void);**

**int adminmenu(void);**

**void deletebook(void);**

**void searchbook(void);**

**void issuebookadmin(void);**

**void issuebookuser(void);**

**void returnbook(void);**

**void displaybook(void);**

**void adduser(void);**

**void deleteuser(void);**

**void changepassword(void);**

**struct BookList g\_book\_list;**

**FILE\* g\_book\_file;**

**void MenuSetup(void)**

**{**

**/\* Use current time as seed for random generator\*/**

**/\* Required to Generate Random String\*/**

**/\* Generates srand seed by current time\*/**

**srand(time(NULL));**

**UserListInitialise(&g\_user\_list);**

**BookListInitialise(&g\_book\_list);**

**}**

**void MenuDestroy(void)**

**{**

**UserListDestroy(&g\_user\_list);**

**BookListDestroy(&g\_book\_list);**

**}**

**void BookListWriteToFile(void)**

**{**

**if (!FileOpenWrite(&g\_book\_file, BOOK\_FILE\_NAME))**

**return;**

**FileWriteBookList(&g\_book\_file, &g\_book\_list);**

**FileClose(&g\_book\_file);**

**BookListReset(&g\_book\_list);**

**}**

**void BookListReadFromFile(void)**

**{**

**if (!FileOpenRead(&g\_book\_file, BOOK\_FILE\_NAME))**

**return;**

**BookListReset(&g\_book\_list);**

**FileReadBookList(g\_book\_file, &g\_book\_list);**

**FileClose(&g\_book\_file);**

**}**

**char\* GenerateRandomString(const int p\_length)**

**{**

**static const char alpha\_numeric\_charset[] =**

**"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";**

**char\* random = NULL;**

**int key;**

**int i;**

**/\*If String Empty, Allocate Nothing\*/**

**if (p\_length == 0)**

**{**

**return NULL;**

**}**

**/\*Allocate String By Length\*/**

**random = malloc(sizeof(char) \* (p\_length + 1));**

**for (i = 0; i < p\_length; i++)**

**{**

**/\*Get Key to use\*/**

**key = rand() % (int)(sizeof(alpha\_numeric\_charset) - 1);**

**random[i] = alpha\_numeric\_charset[key];**

**}**

**/\*Set Last Elem as NULL\*/**

**random[p\_length] = '\0';**

**return random;**

**}**

**void BookDataDisplay(struct BookData book)**

**{**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Code\t:%d", book.book\_code);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Name\t:%s", book.book\_name);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Author\t:%s", book.book\_author);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Shelf\t:%d", book.book\_shelf\_no);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Book Issued\t:%s\n", book.is\_issued ? "Yes" : "No");**

**}**

**void ErrorMessageDisplay(const char\* const p\_error\_message)**

**{**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Error:- ");**

**printf("%s", p\_error\_message);**

**\_getch();**

**}**

**void SuccessMessageDisplay(const char\* const p\_error\_message)**

**{**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Success:- ");**

**printf("%s", p\_error\_message);**

**\_getch();**

**}**

**int adminmenu(void)**

**{**

**char ch;**

**system("cls");**

**printf(**

**"\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2 MAIN MENU "**

**"\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2\xB2");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 1. Add Books \n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 2. Delete books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 3. Search Books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 4. Issue Books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 5, Return Book\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 6. Display Books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 7. Add New User\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 8. Delete User\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 9. Change Password\n");**

**scanf(" %c", &ch);**

**system("cls");**

**switch (ch)**

**{**

**case '1':**

**addbook();**

**break;**

**case '2':**

**deletebook();**

**break;**

**case '3':**

**searchbook();**

**break;**

**case '4':**

**issuebookadmin();**

**break;**

**case '5':**

**returnbook();**

**break;**

**case '6':**

**displaybook();**

**break;**

**case '7':**

**adduser();**

**break;**

**case '8':**

**deleteuser();**

**break;**

**case '9':**

**changepassword();**

**break;**

**default:**

**return 0;**

**}**

**/\*Menu Worked Successfully\*/**

**return 1;**

**}**

**void deleteuser(void)**

**{**

**char username[50];**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Please Enter Username to delete\n");**

**scanf(" %[^\n]", username);**

**if (strcmp(username, g\_current\_user) == 0)**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 You Cannot Delete your own User Name");**

**return;**

**}**

**UserListReadFromFile();**

**if (!UserListDeleteUserByName(&g\_user\_list, username))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Delete User");**

**return;**

**}**

**UserListWriteToFile();**

**}**

**void changepassword(void)**

**{**

**char old\_pass[50];**

**char new\_pass[50];**

**char confirm\_new\_pass[50];**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter old password\n");**

**scanf(" %[^\n]", old\_pass);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter New Password\n");**

**scanf(" %[^\n]", new\_pass);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Confirm Pass\n");**

**scanf(" %[^\n]", confirm\_new\_pass);**

**if (strcmp(new\_pass, confirm\_new\_pass) != 0)**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Passwords do not match");**

**return;**

**}**

**if (strcmp(old\_pass, new\_pass) == 0)**

**{**

**/\*If Old and new password is same, do nothing\*/**

**return;**

**}**

**UserListReadFromFile();**

**if (!UserListChangePassword(&g\_user\_list, g\_current\_user, old\_pass, new\_pass))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Change Password");**

**return;**

**}**

**UserListWriteToFile();**

**SuccessMessageDisplay("Password Changed Successfully");**

**}**

**void addbook(void)**

**{**

**char book\_name[50], book\_author[50];**

**int book\_shelf\_no, book\_code;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Name\n");**

**scanf(" %[^\n]", book\_name);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Author\n");**

**scanf(" %[^\n]", book\_author);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Code\n");**

**scanf("%d", &book\_code);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Shelf No\n");**

**scanf("%d", &book\_shelf\_no);**

**BookListReadFromFile();**

**if (!BookListAdd(&g\_book\_list, book\_code, book\_name, book\_author, book\_shelf\_no))**

**{**

**ErrorMessageDisplay("Unable to Add Book");**

**return;**

**}**

**BookListWriteToFile();**

**SuccessMessageDisplay("Added To Book");**

**}**

**void adduser(void)**

**{**

**char username[50];**

**char\* password;**

**char is\_admin;**

**enum UserAuthority authority;**

**/\*Get Random Password String\*/**

**password = GenerateRandomString(6);**

**if (password == NULL)**

**{**

**ErrorMessageDisplay("Unable to create Randomised Password");**

**return;**

**}**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Please Enter Username to add\n");**

**scanf(" %[^\n]s", username);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Default Password is %s", password);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Is New User Admin(Y\\N)?");**

**scanf(" %c", &is\_admin);**

**authority = (is\_admin == 'Y' || is\_admin == 'y') ? ADMIN : USER;**

**UserListReadFromFile();**

**if (!UserListAdd(&g\_user\_list, username, password, authority))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Add To List");**

**return;**

**}**

**UserListWriteToFile();**

**/\*Note Password is allocated using malloc in function\*/**

**free(password);**

**SuccessMessageDisplay("Added New User");**

**}**

**void issuebookadmin(void)**

**{**

**int code;**

**char book\_issuer[50];**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Code\n");**

**scanf("%d", &code);**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Issuer\n");**

**scanf(" %[^\n]", book\_issuer);**

**/\*Check if Issuer is actually allowed to do so\*/**

**UserListReadFromFile();**

**if (!UserListUserExists(&g\_user\_list, book\_issuer))**

**{**

**ErrorMessageDisplay(**

**"\n\xDB\xDB\xDB\xDB\xB2 Unable to Issue Book. User Does not Exist");**

**return;**

**}**

**BookListReadFromFile();**

**/\* Check if Student has Already Issued Book or Not\*/**

**if (!BookListBookIssue(&g\_book\_list, book\_issuer, code))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Issue Book");**

**return;**

**}**

**BookListWriteToFile();**

**SuccessMessageDisplay("Issued Book");**

**}**

**void issuebookuser(void)**

**{**

**int code;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book Code\n");**

**scanf("%d", &code);**

**BookListReadFromFile();**

**/\* Check if Student has Already Issued Book or Not\*/**

**if (!BookListBookIssue(&g\_book\_list, g\_current\_user, code))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Issue Book");**

**return;**

**}**

**BookListWriteToFile();**

**SuccessMessageDisplay("Found These Books");**

**}**

**void returnbook(void)**

**{**

**int code;**

**struct BookData book;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter code of Book which is returned\n");**

**scanf("%d", &code);**

**BookListReadFromFile();**

**/\* Check if Book can be Deissued\*/**

**if (!BookListBookDeIssue(&g\_book\_list, code, &book))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Return Book");**

**return;**

**}**

**BookListWriteToFile();**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Returned Book Place to %d Shelf", book.book\_shelf\_no);**

**BookDataDisplay(book);**

**\_getch();**

**}**

**void deletebook(void)**

**{**

**int code;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book code\n");**

**scanf("%d", &code);**

**BookListReadFromFile();**

**if (!BookListDeleteByCode(&g\_book\_list, code))**

**{**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Delete Book");**

**return;**

**}**

**BookListWriteToFile();**

**SuccessMessageDisplay("Deleted Book");**

**}**

**void searchbook(void)**

**{**

**char ch;**

**int code;**

**char text[50];**

**struct BookData book;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 1. Search by Code\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 2. Search by Student Name\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 3. Search by Name/Author\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 4. Search by Shelf No\n");**

**scanf(" %c", &ch);**

**switch (ch)**

**{**

**case '1':**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Book code\n");**

**scanf("%d", &code);**

**BookListReadFromFile();**

**if (BookListFindByCode(&g\_book\_list, code, &book))**

**{**

**BookDataDisplay(book);**

**SuccessMessageDisplay("Found These Books");**

**}**

**else**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Find Book");**

**break;**

**case '2':**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Student Name\n");**

**scanf(" %[^\n]s", text);**

**BookListReadFromFile();**

**if (BookListFindByStudentName(&g\_book\_list, text, &book))**

**{**

**BookDataDisplay(book);**

**SuccessMessageDisplay("Found These Books");**

**}**

**else**

**ErrorMessageDisplay("\n\xDB\xDB\xDB\xDB\xB2 Unable to Find Book");**

**break;**

**case '3':**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Name/Author\n");**

**scanf(" %[^\n]s", text);**

**BookListReadFromFile();**

**BookListFindByString(&g\_book\_list, text, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**case '4':**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Shelf Number\n");**

**scanf("%d", &code);**

**BookListReadFromFile();**

**BookListDisplayByShelfNumber(&g\_book\_list, code, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**}**

**}**

**void displaybook(void)**

**{**

**char ch;**

**printf("\n\xDB\xDB\xDB\xDB\xB2 1. Display details of all books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 2. Display details of Students not returned books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 3. Display books by Shelf Number\n");**

**scanf(" %c", &ch);**

**BookListReadFromFile();**

**switch (ch)**

**{**

**case '1':**

**BookListDisplayedAllList(&g\_book\_list, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**case '2':**

**BookListDisplayDelayedList(&g\_book\_list);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**case '3':**

**BookListShowByShelfNumber(&g\_book\_list, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**}**

**}**

**int usermenu(void)**

**{**

**char ch;**

**char text[50];**

**system("cls");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 1. Issue Books\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 2. Display Books by Shelf No\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 3. Search by Name/Author\n");**

**printf("\n\xDB\xDB\xDB\xDB\xB2 4. Change Password\n");**

**scanf(" %c", &ch);**

**system("cls");**

**switch (ch)**

**{**

**case '1':**

**issuebookuser();**

**break;**

**case '2':**

**BookListReadFromFile();**

**BookListShowByShelfNumber(&g\_book\_list, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**case '3':**

**printf("\n\xDB\xDB\xDB\xDB\xB2 Enter Name/Author\n");**

**scanf(" %[^\n]s", text);**

**BookListReadFromFile();**

**BookListFindByString(&g\_book\_list, text, BookDataDisplay);**

**SuccessMessageDisplay("Found These Books");**

**break;**

**case '4':**

**changepassword();**

**break;**

**default:**

**return 0;**

**}**

**return 1;**

**}**

**#endif**

**/\*Main.c\*/**

**#include "Menu.h"**

**void MainUIShow()**

**{**

**enum UserAuthority authority;**

**/\*Setup The Books Menu\*/**

**MenuSetup();**

**UserListReadFromFile();**

**if (UserListHasNoAdmin(&g\_user\_list))**

**{**

**/\*This Portion Exists for First Login\*/**

**/\*In case of First Login, Set Default UserName and Password for Admin be\*/**

**/\*admin admin\*/**

**UserListAdd(&g\_user\_list, "admin", "admin", ADMIN);**

**SuccessMessageDisplay("Default UserName/Password is admin/admin");**

**UserListWriteToFile();**

**\_getch();**

**strcpy(g\_current\_user, "admin");**

**/\*First User Must be Admin\*/**

**authority = ADMIN;}**

**else**

**{**

**while (!Login(&authority))**

**{**

**ErrorMessageDisplay("Unable to Login. Wrong UserName/Pasword");**

**}**

**}**

**/\*Show the Menu till Terminating Condition called\*/**

**if (authority == ADMIN)**

**{**

**while (adminmenu())**

**{**

**}**

**}**

**else**

**{**

**while (usermenu())**

**{**

**}**

**}**

**MenuDestroy();**

**}**

**int main()**

**{**

**char ch;**

**do**

**{**

**MainUIShow();**

**system("cls");**

**printf("Do you Wish to Login Again(Y\\N)?");**

**scanf(" %c",&ch);**

**} while (ch == 'y' || ch == 'Y');**

**return 0;**

**}**