Library Management System

TECHNICAL REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Bachelor of Technology
in Computer Science and Engineering

Submitted by
Balkrishan Singh
URN 2302492
Diljot Singh
URN 2302509
Ayush Mishra
URN 2302490

Submitted To
ER.Shailja

Department of Computer Science and Engineering
Guru Nanak Dev Engineering College
Ludhiana, 141006

Chapter 1

main.cpp

```cpp
#include <iostream>
#include "library.h"

int main()
{

    library lib;
    std::cout << "Welcome to Library management System! ";
    char choice;
    std::cout << "Registered Already? (y/n)";
    std::cin >> choice;
    toupper(choice);
    if (choice == 'y')
    {
        Menu::Login(lib);
    }
    else if (choice == 'n')
    {
        Menu::Registration(lib);
        Menu::Login(lib);
    }
    return 0;
}
```

Chapter 2

Library.h

```
1
2  #ifndef LIBRARY_H
3  #define LIBRARY_H
4  #include <fstream>
5  #include <iostream>
6  #include <string>
7  #include <vector>
8  #include <memory>
9  class Book;
10 class User;
11 class library;
12
13 class Book {
14     bool isIssued;
15     std::string bookName;
16     std::string author;
17     int bookID;
18
19 public:
20     std::shared_ptr<User> bookBorrower;
21     Book(int bookID, std::string bookName, std::string author);
22
23     Book(std::ifstream &inFile);
24
25     void Save(std::ofstream &outFile);
26
27     void BookInformation();
28
29     int getBookID() {
30         return bookID;
31     }
32
33     bool isBookIssued() const { return isIssued; }
34     void setIssued(bool status) { isIssued = status; }
35 };
```

```
36
37  class User {
38  protected:
39      std::string userName;
40      int password;
41
42      User(std::string userName, int password);
43
44  public:
45      virtual ~User() = default;
46
47      std::string getUserName() const { return userName; }
48      int getPassword() const { return password; }
49  };
50
51  class Student : public User {
52      int userID; //Roll no
53      //TODO Save which book has been borrowed.
54      std::shared_ptr<Book> borrowedBook;
55      int borrowedBookID;
56
57  public:
58      void SaveStudent(std::ofstream &outFile);
59
60      Student(int userID, std::string userName, int password);
61
62      void DisplayIssuedBook();
63
64      std::shared_ptr<Book> getBorrowedBook() const { return borrowedBook; }
65
66      void setBorrowedBook(std::shared_ptr<Book> borrowedBook) {
67          this->borrowedBook = borrowedBook;
68      }
69
70      bool hasIssuedBook() {
71          return borrowedBook != nullptr;
72      }
73
74      Student(std::ifstream &inFile);
75
76      void studentInformation();
77  };
78
79  class FileManager {
80  public:
81      static void SaveBooks(const std::vector<std::shared_ptr<Book> > &books, const std
82
83      static void LoadBooks(std::vector<std::shared_ptr<Book> > &books, const std::stri
```

```
84
85      static void SaveUsers(const std::vector<std::shared_ptr<User> > &users, const std
86
87      static void LoadUsers(std::vector<std::shared_ptr<User> > &users, const std::stri
88  };
89
90  class Administrator : public User {
91  public:
92      Administrator(std::string userName, int password);
93  };
94
95  class library {
96      std::shared_ptr<Administrator> administrator;
97      std::vector<std::shared_ptr<User> > users;
98      std::vector<std::shared_ptr<Book> > books;
99
100 public:
101     library();
102
103     ~library();
104
105     void RegisterStudent(int userID, std::string userName, int password);
106
107     void LoginUser(bool isAdmin);
108
109     void issueBook(std::shared_ptr<Student> student);
110
111     void addBook();
112
113     void displayBooks();
114
115     void searchBook();
116
117     void displayIssuedBooksForStudent();
118
119     void displaystudents();
120
121     void returnBook(std::shared_ptr<Student> student);
122
123     void AddDummyStudentsToBinaryFile();
124
125     void AddDummyBooksToBinaryFile();
126 };
127
128 class Menu {
129 public:
130     static void Registration(library &lib);
131
```

```
132     static void Login(library &lib);
133
134     static void StudentDashboard(library &lib, std::shared_ptr<Student> activeStudent
135
136     static void AdministratorDashboard(library &lib);
137 };
138
139 #endif // LIBRARY_H
140
```

Chapter 3

library.cpp

```cpp

#include "library.h"
#include "string"
#include <iostream>

void library::AddDummyBooksToBinaryFile()
{
    books.push_back(std::make_shared<Book>(101, "C++ Programming", "Bjarne Stroustrup
    books.push_back(std::make_shared<Book>(102, "Introduction to Algorithms", "Thomas
    books.push_back(std::make_shared<Book>(103, "Clean Code", "Robert C. Martin"));
    books.push_back(std::make_shared<Book>(104, "The Pragmatic Programmer", "Andrew H
    books.push_back(std::make_shared<Book>(105, "Design Patterns", "Erich Gamma"));
}

// TODO Add input validation for username (Capitalization ignore).
void library::AddDummyStudentsToBinaryFile()
{
    // Add some dummy students
    users.push_back(std::make_shared<Student>(101, "Alice", 1234));
    users.push_back(std::make_shared<Student>(102, "Bob", 5678));
    users.push_back(std::make_shared<Student>(103, "Charlie", 91011));
}

library::library()
{
    administrator = std::make_shared<Administrator>("Diljot", 1234);
    FileManager::LoadBooks(books, "./books.dat");
    FileManager::LoadUsers(users, "./students.dat");
}

library::~library()
{
    FileManager::SaveBooks(books, "./books.dat");
    FileManager::SaveUsers(users, "./students.dat");
}
```

```cpp

Book::Book(int bookID, std::string bookName, std::string author)
{
    this->bookID = bookID;
    this->bookName = bookName;
    this->author = author;
}

void Book::BookInformation()
{
    std::cout << "Book ID: " << bookID
              << "\nTitle: " << bookName
              << "\nAuthor: " << author
              << "\nIssued: " << (isIssued ? "Yes" : "No") << '\n';
}

User::User(std::string userName, int password)
{
    this->userName = userName;
    this->password = password;
}

Student::Student(int userID, std::string userName, int password) : User(userName, pas
{
    this->userID = userID;
}

Administrator::Administrator(std::string userName, int password) : User(userName, pas
{
}

void FileManager::SaveBooks(const std::vector<std::shared_ptr<Book>> &books, const st
{
    std::ofstream outFile(filename, std::ios::binary);
    if (!outFile)
    {
        std::cerr << "Error opening file for saving.\n";
        return;
    }

    size_t count = books.size();
    outFile.write(reinterpret_cast<const char *>(&count), sizeof(count));

    for (const auto &book : books)
    {
        book->Save(outFile);
    }

```

```
84      if (!outFile)
85      {
86          std::cerr << "Error writing to file!\n";
87      }
88  }
89
90  void FileManager::LoadBooks(std::vector<std::shared_ptr<Book>> &books, const std::str
91  {
92      std::ifstream inFile(filename, std::ios::binary);
93      if (!inFile)
94      {
95          std::cerr << "Error opening file for loading book data.\n";
96          return;
97      }
98      size_t count;
99      inFile.read(reinterpret_cast<char *>(&count), sizeof(count));
100     books.clear();
101     for (size_t i = 0; i < count; ++i)
102     {
103         auto book = std::make_shared<Book>(inFile);
104         books.push_back(book);
105     }
106     if (!inFile)
107     {
108         std::cerr << "Error reading from book data file.!\n";
109     }
110 }
111
112 void FileManager::SaveUsers(const std::vector<std::shared_ptr<User>> &users, const st
113 {
114     std::ofstream outFile(filename, std::ios::binary);
115     if (!outFile)
116     {
117         std::cerr << "Error opening file for saving users.\n";
118         return;
119     }
120
121     size_t count = users.size();
122     outFile.write(reinterpret_cast<const char *>(&count), sizeof(count));
123
124     for (const auto &user : users)
125     {
126         if (auto student = std::dynamic_pointer_cast<Student>(user))
127         {
128             student->SaveStudent(outFile);
129         }
130     }
131
```

```
132     if (!outFile)
133     {
134         std::cerr << "Error writing to user data file!\n";
135     }
136 }
137 
138 void FileManager::LoadUsers(std::vector<std::shared_ptr<User>> &users, const std::str
139 {
140     std::ifstream inFile(filename, std::ios::binary);
141     if (!inFile)
142     {
143         std::cerr << "Error opening file for loading users.\n";
144         return;
145     }
146 
147     size_t count;
148     inFile.read(reinterpret_cast<char *>(&count), sizeof(count));
149 
150     users.clear();
151     for (size_t i = 0; i < count; ++i)
152     {
153         auto student = std::make_shared<Student>(inFile);
154         users.push_back(student);
155     }
156 
157     if (!inFile)
158     {
159         std::cerr << "Error reading from user data file!\n";
160     }
161 }
162 
163 Student::Student(std::ifstream &inFile) : User("", 0)
164 {
165     size_t nameSize;
166     inFile.read(reinterpret_cast<char *>(&nameSize), sizeof(nameSize));
167     userName.resize(nameSize);
168     inFile.read(&userName[0], nameSize);
169 
170     inFile.read(reinterpret_cast<char *>(&password), sizeof(password));
171 
172     inFile.read(reinterpret_cast<char *>(&userID), sizeof(userID));
173     //
174 }
175 
176 Book::Book(std::ifstream &inFile)
177 {
178     inFile.read(reinterpret_cast<char *>(&bookID), sizeof(bookID));
179 
```

```
    size_t nameSize;
    inFile.read(reinterpret_cast<char *>(&nameSize), sizeof(nameSize));
    bookName.resize(nameSize);
    inFile.read(&bookName[0], nameSize);

    size_t authorSize;
    inFile.read(reinterpret_cast<char *>(&authorSize), sizeof(authorSize));
    author.resize(authorSize);
    inFile.read(&author[0], authorSize);

    inFile.read(reinterpret_cast<char *>(&isIssued), sizeof(isIssued));
}

void Book::Save(std::ofstream &outFile)
{

    outFile.write(reinterpret_cast<const char *>(&bookID), sizeof(bookID));

    size_t nameSize = bookName.size();
    outFile.write(reinterpret_cast<const char *>(&nameSize), sizeof(nameSize));
    outFile.write(bookName.c_str(), nameSize);

    size_t authorSize = author.size();
    outFile.write(reinterpret_cast<const char *>(&authorSize), sizeof(authorSize));
    outFile.write(author.c_str(), authorSize);

    outFile.write(reinterpret_cast<const char *>(&isIssued), sizeof(isIssued));
}

void Student::DisplayIssuedBook()
{
    if (borrowedBook)
    {
        std::cout << "Issued Book Details:\n";
        borrowedBook->BookInformation();
    }
    else
    {
        std::cout << "No book has been issued to you.\n";
    }
}

void library::RegisterStudent(int userID, const std::string userName, int password)
{
    auto student = std::make_shared<Student>(userID, userName, password);
    users.push_back(student);
    std::cout << "Student registered successfully.\n";
}
```

```
228
229 void library::LoginUser(bool isAdmin)
230 {
231     std::string userName;
232     int password;
233     std::cout << "Enter Username: ";
234     std::cin.ignore();
235     getline(std::cin, userName);
236     std::cout << "Enter Password: ";
237     std::cin >> password;
238
239     if (!isAdmin)
240     {
241         for (const auto &user : users)
242         {
243             if (dynamic_cast<Student *>(user.get()))
244             {
245                 if (user->getUserName() == userName && user->getPassword() == passwor
246                 {
247                     std::cout << "Student login successful!\n";
248                     Menu::StudentDashboard(*this, std::dynamic_pointer_cast<Student>(
249                     return;
250                 }
251             }
252         }
253     }
254     else
255     {
256         if (administrator->getUserName() == userName && administrator->getPassword()
257         {
258             std::cout << "Admin login successful!\n";
259             Menu::AdministratorDashboard(*this);
260             return;
261         }
262     }
263     std::cout << "Invalid credentials!\n";
264 }
265
266 void library::issueBook(std::shared_ptr<Student> student)
267 {
268     int bookID;
269     std::cout << "Enter Book ID to issue: ";
270     std::cin >> bookID;
271     if (!student->hasIssuedBook())
272     {
273         for (auto &book : books)
274         {
275             if (book->getBookID() == bookID && !(book->isBookIssued()))
```

```
            {
                book->setIssued(true);
                student->setBorrowedBook(book);
                std::cout << "Book issued successfully.\n";
                return;
            }
        }
    }

    std::cout << "Book not available.\n";
}

void library::returnBook(std::shared_ptr<Student> student)
{
    if (student->hasIssuedBook())
    {
        student->getBorrowedBook()->setIssued(true);
        student->setBorrowedBook(nullptr);
        std::cout << "Book returned successfully.\n";
    }
    else
    {
        std::cout << "You don't have any issued books..\n";
    }
}

void library::addBook()
{
    int bookID;
    std::string bookName, author;

    std::cout << "Enter Book ID: ";
    std::cin >> bookID;
    std::cin.ignore();
    std::cout << "Enter Book Name: ";
    getline(std::cin, bookName);
    std::cout << "Enter Author Name: ";
    getline(std::cin, author);

    auto newBook = std::make_shared<Book>(bookID, bookName, author);
    books.push_back(newBook);

    std::cout << "Book added successfully.\n";
}

void library::displayBooks()
{
    std::cout << "Books in the library:\n";
```

```
324     for (const auto &book : books)
325     {
326         book->BookInformation();
327     }
328 }
329 
330 void library::displaystudents()
331 {
332     for (const auto &user : users)
333     {
334         std::dynamic_pointer_cast<Student>(user)->studentInformation();
335     }
336 }
337 
338 void Student::studentInformation()
339 {
340     std::cout << "Student ID: " << userID
341               << "\nName: " << userName
342               << "\nPassword: " << password << std::endl;
343 }
344 
345 void Student::SaveStudent(std::ofstream &outFile)
346 {
347 
348     size_t nameSize = userName.size();
349     outFile.write(reinterpret_cast<const char *>(&nameSize), sizeof(nameSize));
350     outFile.write(userName.c_str(), nameSize);
351     outFile.write(reinterpret_cast<const char *>(&password), sizeof(password));
352 
353     outFile.write(reinterpret_cast<const char *>(&userID), sizeof(userID));
354 }
355 // unused
356 void library::searchBook()
357 {
358     int bookID;
359     std::cout << "Enter Book ID to search: ";
360     std::cin >> bookID;
361 
362     for (const auto &book : books)
363     {
364         if (book->isBookIssued())
365         {
366             book->BookInformation();
367             return;
368         }
369     }
370 
371     std::cout << "Book not found.\n";
```

```
372 }
373
374 void library::displayIssuedBooksForStudent()
375 {
376     std::string studentName;
377     std::cout << "Enter your username: ";
378     getline(std::cin, studentName);
379
380     for (const auto &user : users)
381     {
382         if (auto student = std::dynamic_pointer_cast<Student>(user))
383         {
384             if (student->getUserName() == studentName)
385             {
386                 student->DisplayIssuedBook();
387                 return;
388             }
389         }
390     }
391     std::cout << "Student not found or no books issued.\n";
392 }
393
394 // Menu Class Definitions
395 void Menu::Registration(library &lib)
396 {
397     std::cout << "Registration Menu\n1. Register as Student\n2. Exit\n";
398     int choice;
399     std::cin >> choice;
400
401     if (choice == 1)
402     {
403         int userID, password;
404         std::string userName;
405         std::cout << "Enter User ID: ";
406         std::cin >> userID;
407         std::cin.ignore();
408         std::cout << "Enter Username: ";
409
410         getline(std::cin, userName);
411         std::cout << "Enter Password: ";
412         std::cin >> password;
413         std::cin.ignore();
414         lib.RegisterStudent(userID, userName, password);
415     }
416 }
417
418 void Menu::Login(library &lib)
419 {
```

```
420     std::cout << "Login Menu\n1. Student Login\n2. Admin Login\n3. Exit\n";
421     int choice;
422     std::cin >> choice;
423     if (choice == 1)
424     {
425         lib.LoginUser(false);
426     }
427     else if (choice == 2)
428     {
429         lib.LoginUser(true);
430     }
431 }
432
433 void Menu::StudentDashboard(library &lib, std::shared_ptr<Student> activeStudent)
434 {
435     bool running = true;
436     while (running)
437     {
438         std::cout << "Student Dashboard\n1.Display Books\n2.Issue Book\n3.Return Book
439         int choice;
440         std::cin >> choice;
441         switch (choice)
442         {
443         case 1:
444             lib.displayBooks();
445             break;
446         case 2:
447             lib.issueBook(activeStudent);
448             break;
449         case 3:
450             lib.returnBook(activeStudent);
451             break;
452         case 4:
453             activeStudent->DisplayIssuedBook();
454             break;
455
456         default:
457             running = false;
458             break;
459         }
460     }
461 }
462
463 void Menu::AdministratorDashboard(library &lib)
464 {
465     bool running = true;
466     while (running)
467     {
```

```
468             std::cout << "Administrator Dashboard\n1. Add Book\n2. Display Users\n3. Disp
469             int choice;
470             std::cin >> choice;
471             switch (choice)
472             {
473             case 1:
474                 lib.addBook();
475                 break;
476             case 2:
477                 lib.displaystudents();
478                 break;
479             case 3:
480                 lib.displayBooks();
481                 break;
482             default:
483                 running = false;
484             }
485         }
486     }
```

Chapter 4

Outputs