COS 212 Assignment

by Naing Min Khant

Submission date: 29-Jun-2023 01:23PM (UTC+0800)
Submission ID: 2123288929
File name: COS_212_Assignment_Documentation_-_Naing_Min_Khant.docx (1.79M)
Word count: 1017
Character count: 4726

British University College

2023-2024 Academic Year

Assignment

COS212 – Object-Oriented Programming with C++

by

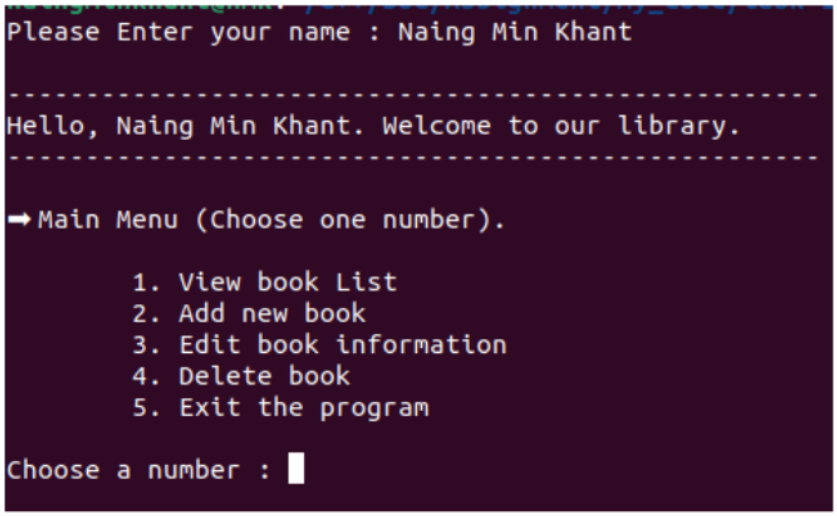
Naing Min Khant (HDIT Class - D)

Presented On – June 30 2023

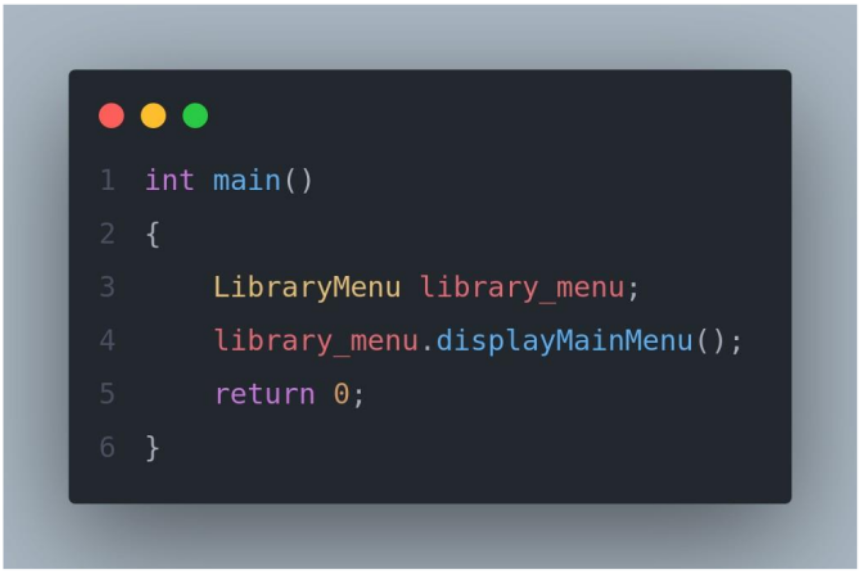
Table of Content

Task 1 - I/O File Manipulation

A simple library management system with file manipulation including read, write, edit, and delete features. Once we run a program, it will ask for the guest's name and print the greeting message as shown in Fig (1.1). Then, Main Menu will display the options. We can see the code behind this output in Fig (1.2), Fig (1.2.1), and Fig (1.3).



Fig(1.1)



```
int main()
{
    LibraryMenu library_menu;
    library_menu.displayMainMenu();
    return 0;
}
```

Fig(1.2)

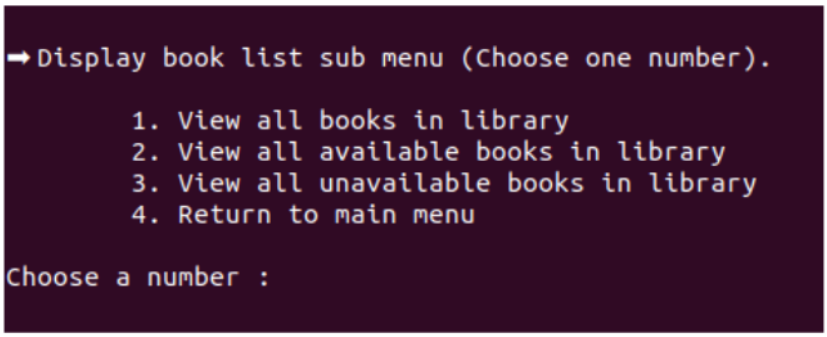
```
Menu::Menu() // constructor definition outside class
{
    cout << "Please Enter your name : ";
    getline(cin, guest_name);
    cout << endl
         << "------------------------------------------------" << endl
         << "Hello, " << guest_name << ". Welcome to our library." << endl
         << "------------------------------------------------" << endl;
}
```

Fig(1.2.1)

```cpp
void LibraryMenu::displayMainMenu() // for main menu
{
    cout << endl
         << "-> Main Menu (Choose one number)." << endl
         << endl;

    displayMenuList(MAIN_MENUS, sizeof(MAIN_MENUS) / sizeof(string)); // displaying main menus
    cout << endl
         << "Choose a number : ";
    cin >> selected_menu_no;

    switch (selected_menu_no)
    {
    case 1:
        displaySubMenu();
        break;
    case 2:
        library.addNewBook();
        cout << "_____" << endl;
        displayMainMenu();
        break;
    case 3:
        library.editBook();
        cout << "_____" << endl;
        displayMainMenu();
        break;
    case 4:
        library.deleteBook();
        cout << "_____" << endl;
        displayMainMenu();
        break;
    case 5:
        cout << endl
             << "Good Bye, " << guest_name << ". See you again." << endl;
        break;
    default:
        cout << "Invalid number!" << endl;
        cout << "Choose again!" << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        displayMainMenu();
        break;
    }
}
```

Fig(1.3)

If we choose the number 1, a sub-menu will display. There are more options for viewing books. The sub-menu will be shown so that the user can choose the option to view the book list as shown in Fig (1.4).



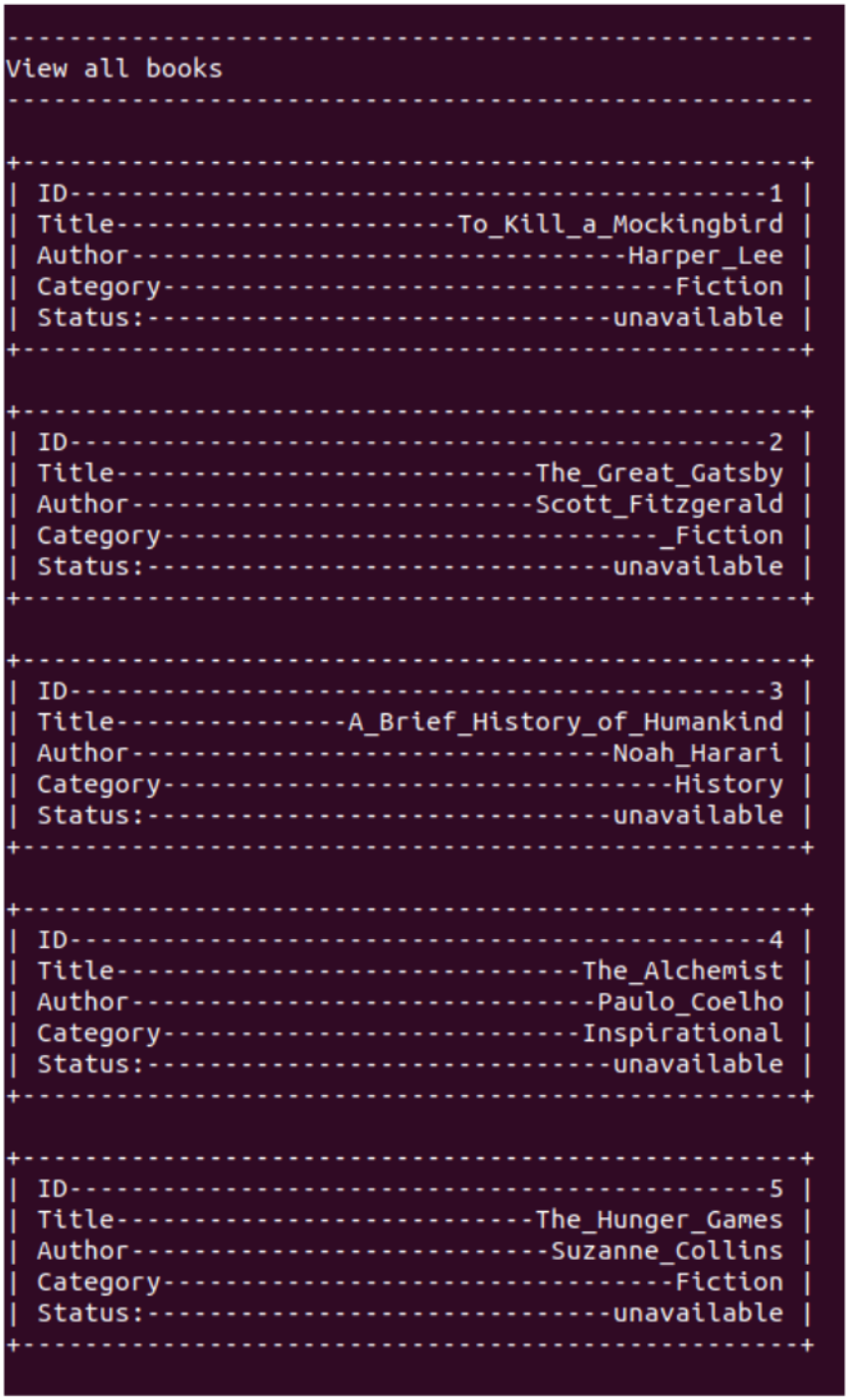
Fig(1.4)

The code working behind this output can be seen in Fig (1.5).

```cpp
void LibraryMenu::displaySubMenu() // for sub menu
{
    cout << endl
         << "➥ Display book list sub menu (Choose one number)." << endl
         << endl;

    displayMenuList(SUB_MENUS, sizeof(SUB_MENUS) / sizeof(string)); // displaying sub menus
    cout << endl
         << "Choose a number : ";
    cin >> selected_menu_no;

    switch (selected_menu_no)
    {
    case 1:
        library.viewBookList("all"); // viewing all books' info
        cout << "_____" << endl;
        displaySubMenu();
        break;
    case 2:
        library.viewBookList("a"); // viewing available books' info
        cout << "_____" << endl;
        displaySubMenu();
        break;
    case 3:
        library.viewBookList("ua"); // viewing unavailable books' info
        cout << "_____" << endl;
        displaySubMenu();
        break;
    case 4:
        cout << "_____" << endl;
        displayMainMenu();
        break;
    default:
        cout << "Invalid number! " << endl
             << "Choose again." << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        displaySubMenu();
        break;
    }
}
```

Fig(1.5)

If we choose the number 1, we will see all of the books in the library like in Fig(1.6). Numbers 2 and 3 will work the same as number 1. Number 2 will only show the available books in the library while number 3 does the opposite. It will show the unavailable books. The code behind this feature can be seen in Fig(1.7) and Fig(1.8).


View all books
ID-----1
Title-----To_Kill_a_Mockingbird
Author-----Harper_Lee
Category-----Fiction
Status:-----unavailable
ID-----2
Title-----The_Great_Gatsby
Author-----Scott_Fitzgerald
Category-----_Fiction
Status:-----unavailable
ID-----3
Title-----A_Brief_History_of_Humankind
Author-----Noah_Harari
Category-----History
Status:-----unavailable
ID-----4
Title-----The_Alchemist
Author-----Paulo_Coelho
Category-----Inspirational
Status:-----unavailable
ID-----5
Title-----The_Hunger_Games
Author-----Suzanne_Collins
Category-----Fiction
Status:-----unavailable

Fig(1.6)

```
// methods definition outsie Library class
void Library::viewBookList(string type) // display books' info according to type
{
    string header, temp[5], tmpStr, line;

    if (type == "all")
        header = "all";
    else if (type == "a")
        header = "available";
    else if (type == "ua")
        header = "unavailable";

    cout << endl
         << "-------------------------------------------------" << endl
         << "View " << header << " books" << endl
         << "-------------------------------------------------" << endl;

    ifstream bookFile("book.txt"); // open book file

    if (bookFile.is_open())
    {
        while (getline(bookFile, line))
        {
            stringstream ss(line);
            int i = 0;

            while (getline(ss, tmpStr, ' '))
            {
                temp[i] = tmpStr;
                i++;
            }

            if (type == temp[4] || type == "all") // display book's info base on type
            {
                Book book;
                book.setId(temp[0]);
                book.setTitle(temp[1]);
                book.setAuthor(temp[2]);
                book.setCategory(temp[3]);
                book.setStatus(temp[4]);
                book.displayInfo();
            }
        }
        bookFile.close(); // close book file
    }
    else
        cout << "Cannot open file!" << endl;
}
```

Fig(1.7)

```cpp
// methods definition outside Book class
void Book::displayInfo()
{
    // output book info
    cout << endl
         << setfill('-')
         << "+" << right << setw(51) << "+" << endl
         << left << setw(20) << "| ID" << right << setw(30) << id << " |" << endl
         << left << setw(20) << "| Title" << right << setw(30) << title << " |" << endl
         << left << setw(20) << "| Author" << right << setw(30) << author << " |" << endl
         << left << setw(20) << "| Category" << right << setw(30) << category << " |" << endl
         << left << setw(20) << "| Status:" << right << setw(30) << getStatus() << " |" << endl
         << "+" << right << setw(51) << "+" << endl;
}
```

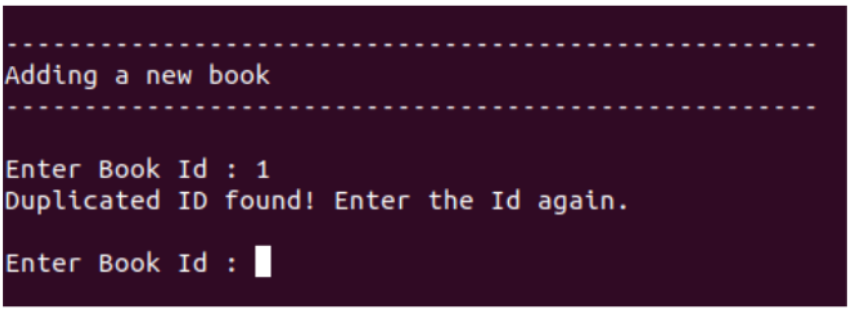
Fig(1.8)

If we click number 2 of the main menu, the program will ask for the data for a new book. As we can see in Fig(1.9).

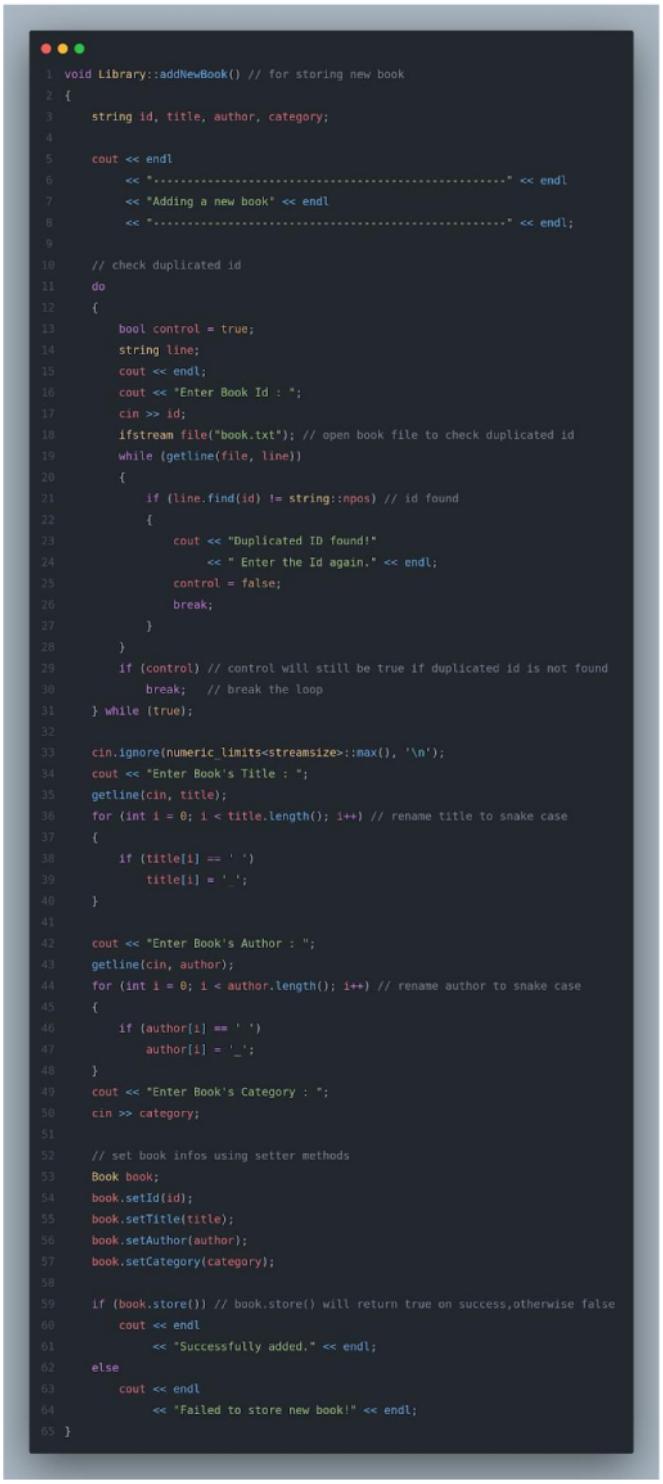
```
---------------------------------------------------
Adding a new book
---------------------------------------------------

Enter Book Id : 8
Enter Book's Title : new title
Enter Book's Author : apollo
Enter Book's Category : fiction

Successfully added.
-----------------------------------------------------------
```

Fig(1.9)

If we add the id that already exists, the program will show the warning message and ask for the id again as shown in Fig (1.10). We can see the code for this feature in Fig (1.11) and Fig(1.12).



Fig(1.10)



```cpp
void Library::addNewBook() // for storing new book
{
    string id, title, author, category;

    cout << endl
         << "-------------------------------------------------" << endl
         << "Adding a new book" << endl
         << "-------------------------------------------------" << endl;

    // check duplicated id
    do
    {
        bool control = true;
        string line;
        cout << endl;
        cout << "Enter Book Id : ";
        cin >> id;
        ifstream file("book.txt"); // open book file to check duplicated id
        while (getline(file, line))
        {
            if (line.find(id) != string::npos) // id found
            {
                cout << "Duplicated ID found!"
                     << " Enter the Id again." << endl;
                control = false;
                break;
            }
        }
        if (control) // control will still be true if duplicated id is not found
            break;   // break the loop
    } while (true);

    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "Enter Book's Title : ";
    getline(cin, title);
    for (int i = 0; i < title.length(); i++) // rename title to snake case
    {
        if (title[i] == ' ')
            title[i] = '_';
    }

    cout << "Enter Book's Author : ";
    getline(cin, author);
    for (int i = 0; i < author.length(); i++) // rename author to snake case
    {
        if (author[i] == ' ')
            author[i] = '_';
    }
    cout << "Enter Book's Category : ";
    cin >> category;

    // set book infos using setter methods
    Book book;
    book.setId(id);
    book.setTitle(title);
    book.setAuthor(author);
    book.setCategory(category);

    if (book.store()) // book.store() will return true on success,otherwise false
        cout << endl
             << "Successfully added." << endl;
    else
        cout << endl
             << "Failed to store new book!" << endl;
}
```

Fig(1.11)

```cpp
bool Book::store() // return true on success, false on fail
{
    try
    {
        ofstream bookFile("book.txt", ios::app); // open book.txt file using ofstream
        if (!bookFile.is_open())
        {
            string err_message = "Could not open file!";
            throw(err_message); // throw exception if file can't open
        }
        // write new book's info to file
        bookFile << id << " " << title
                 << " " << author << " " << category << " " << status << endl;

        bookFile.close(); // file close
        return true;
    }
    catch (string error_message)
    {
        cerr << endl
             << error_message << endl; // display error message
        return false;
    }
}
```

Fig(1.12)

Number 3 is for editing the data of the book using book id. Firstly, we need to input the id of the book that we want to edit. So, the program will find the book using the id that we provided. If we enter the id which does not exist in the library, the program will say a 'Not Found' message as we can see in Fig (1.13).

```
--------------------------------------------------
Editing Book
--------------------------------------------------

Enter Book ID to edit book : 111

Not found!

Data not found for id 111
```

Fig(1.13)

If we enter the invalid input for the book's status in editing, the program will print a message and ask for the input again. That feature is shown in Fig (1.14). The code that works behind this feature is shown in Fig (1.15) and Fig (1.16).

```
--------------------------------------------------
Editing Book
--------------------------------------------------

Enter Book ID to edit book : 1

We found the following data.

+------------------------------------------------+
| ID-----------------------------------------1 |
| Title----------------------To_Kill_a_Mockingbird |
| Author---------------------------------Harper_Lee |
| Category-------------------------------Fiction |
| Status:-------------------------------unavailable |
+------------------------------------------------+

Choose the number below to edit.

Choose 1 to edit title.
Choose 2 to edit author.
Choose 3 to edit category.
Choose 4 to edit status.
Choose 5 to return to main menu.

Enter number : 4

Enter new status ('a' for available / 'ua' for unavailable) : fd
Invalid Input! Choose again.
Enter new status ('a' for available / 'ua' for unavailable) :
```

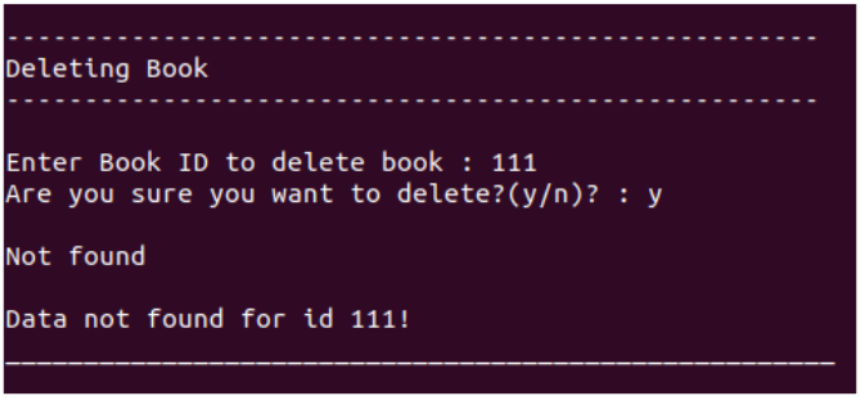
Fig(1.14)

```cpp
void Library::editBook()
{
    string id;
    Book book;
    cout << endl
         << "--------------------------------------------------" << endl
         << "Editing Book" << endl
         << "--------------------------------------------------" << endl;
    cout << endl
         << "Enter Book ID to edit book : ";
    cin >> id;
    if (!book.edit(id)) // book.edit(id) will return true on success,otherwise false
        cout << endl
             << "Data not found for id " << id << endl;
}
```

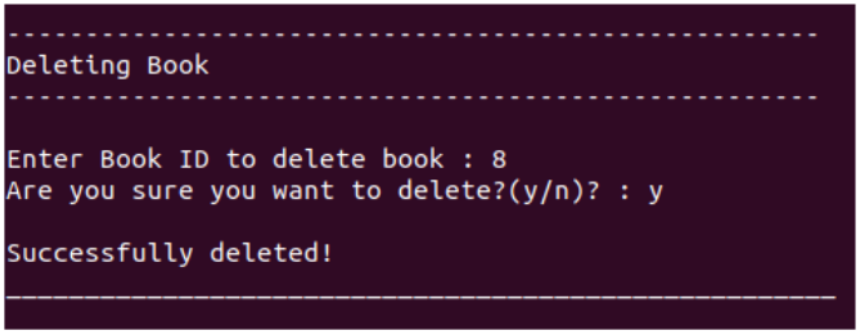
Fig(1.15)

Fig(1.16)

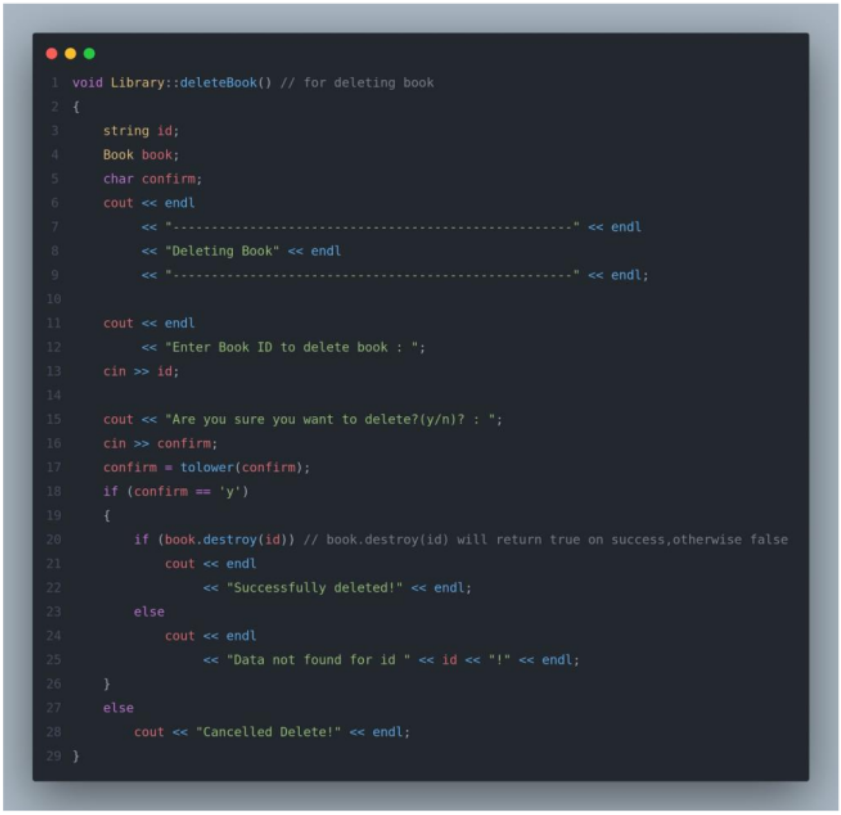
Number 4 is for deleting a book using its id. Firstly, we need to provide the id of the book that we want to delete. So, the program will find the book using the id that we provided. The program will also print the confirmation message which asks the user whether he really wants to delete. If we enter the id that does not exist in the library, the program will say a 'Not Found' message. Otherwise, it will be successfully deleted. For this feature, Fig (1.17), Fig (1.18), Fig(1.19), and Fig (1.20) are shown below.



Fig(1.17)



Fig(1.18)



```cpp
void Library::deleteBook() // for deleting book
{
    string id;
    Book book;
    char confirm;
    cout << endl
         << "------------------------------------------------" << endl
         << "Deleting Book" << endl
         << "------------------------------------------------" << endl;

    cout << endl
         << "Enter Book ID to delete book : ";
    cin >> id;

    cout << "Are you sure you want to delete?(y/n)? : ";
    cin >> confirm;
    confirm = tolower(confirm);
    if (confirm == 'y')
    {
        if (book.destroy(id)) // book.destroy(id) will return true on success,otherwise false
            cout << endl
                 << "Successfully deleted!" << endl;
        else
            cout << endl
                 << "Data not found for id " << id << "!" << endl;
    }
    else
        cout << "Cancelled Delete!" << endl;
}
```

Fig(1.19)

```cpp
bool Book::destroy(string id) // find and delete book with id // return true on success,otherwise false
{
    string line, book_id;
    fstream file("book.txt", ios::in | ios::out);

    if (!file.is_open())
    {
        cerr << endl
             << "Could not open file!" << endl;
        return false;
    }

    while (getline(file, line))
    {
        stringstream ss(line);

        getline(ss, book_id, ' '); // get id from line

        if (book_id == id) // id is found
        {
            file.close();

            // create and open temporary file
            ofstream outfile;
            file.open("book.txt");
            outfile.open("temp.txt", ios::app);

            while (getline(file, line))
                if (line.find(id) == string::npos) // rewriting existing data expect the given id
                    outfile << line << endl;

            outfile.close();
            file.close();
            remove("book.txt");            // delete book file
            rename("temp.txt", "book.txt"); // rename temp.txt to book.txt

            return true;
        }
    }
    // if the search id is not found, the code will reach here, and print 'Not found' and return false
    cout << endl
         << "Not found" << endl;
    return false;
}
```

Fig (1.20)

To exit the program, choose number 5. The program will print the goodbye message to the user with his name which was provided at the start of the program as shown in Fig (1.21).

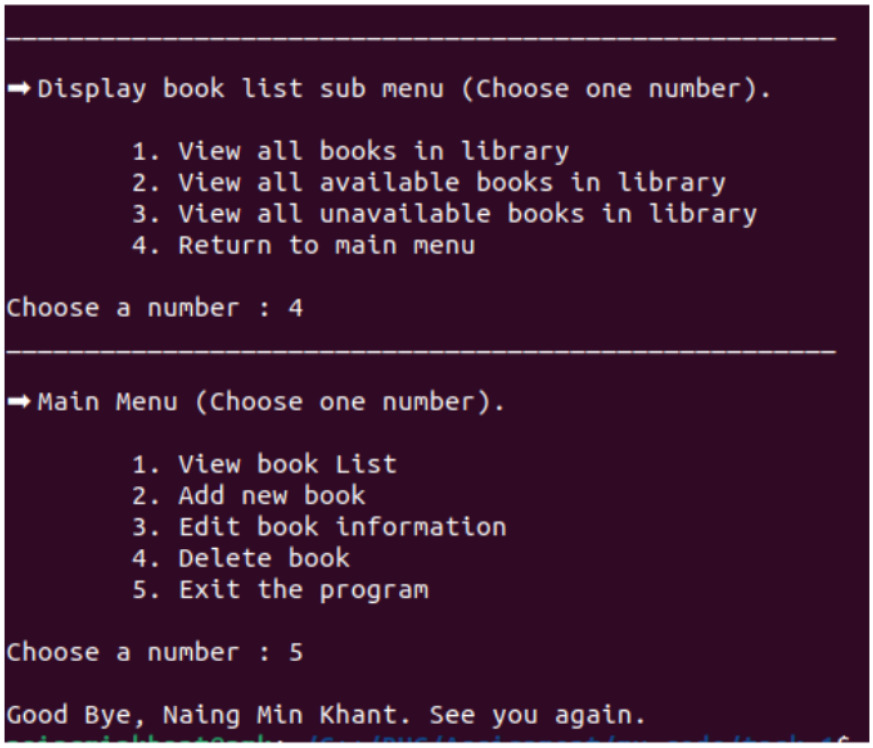


Fig (1.21)

The code behind this feature is shown in Fig (1.22).

```
Menu::~Menu() // destructor definition outside class
{
    cout << endl
         << "Good Bye, " << guest_name << ". See you again." << endl;
}
```

Fig (1.22)

Task 2 - Inheritance

One class cannot access the properties and methods of another class without a relationship, although it can be done by instantiating an object of another. In this case, Inheritance can be used as a relationship between classes. Inheritance is one of the most important features of Object-Oriented Programming. It is just like a Parent-Child relationship which means Child has the ability to derive the parent's possession. By implementing Inheritance, the derived class(child class) will have the ability to access the properties of the base class (parent class). However, the members of the base class can be controlled by the access specifiers and will not be affected by the derived class. The derived class now is said to be inherited from the base class.

There are basically three types of Inheritance in C++, which are Single Inheritance, Multi-Level Inheritance, and Multiple Inheritance. If a derived class has only one base class(parent class), it can be called Single Inheritance. One derived class can inherit from more than one class, and it is known as Multiple Inheritance. If a derived class is created from another derived class, it is said to be Multi-Level Inheritance.

I implement the Single Inheritance in the program. Implementation of Single Inheritance in the program can be seen in Fig (2.1) and Fig (2.2).

```
class Menu
{
public:
    Menu();  // constructor declaration
    ~Menu(); // destructor declaration
protected:
    string guest_name;
    int selected_menu_no;
    void displayMenuList(const string menus[], int size);
};

Menu::Menu() // constructor definition outside class
{
    cout << "Please Enter your name : ";
    getline(cin, guest_name);
    cout << endl
         << "-----------------------------------------------------" << endl
         << "Hello, " << guest_name << ". Welcome to our library." << endl
         << "-----------------------------------------------------" << endl;
}
Menu::~Menu() // destructor definition outside class
{
    cout << endl
         << "Good Bye, " << guest_name << ". See you again." << endl;
}
```

Fig (2.1)

In this figure, the Menu class, which has a constructor, destructor, and three protected members, is the base class (parent class).

```
class LibraryMenu : public Menu
{
    const string MAIN_MENUS[5] = {"1. View book List",
                                  "2. Add new book",
                                  "3. Edit book information",
                                  "4. Delete book",
                                  "5. Exit the program"};
    const string SUB_MENUS[4] = {"1. View all books in library",
                                 "2. View all available books in library",
                                 "3. View all unavailable books in library",
                                 "4. Return to main menu"};
    Library library;

public:
    void displayMainMenu();

    void displaySubMenu();
};
```

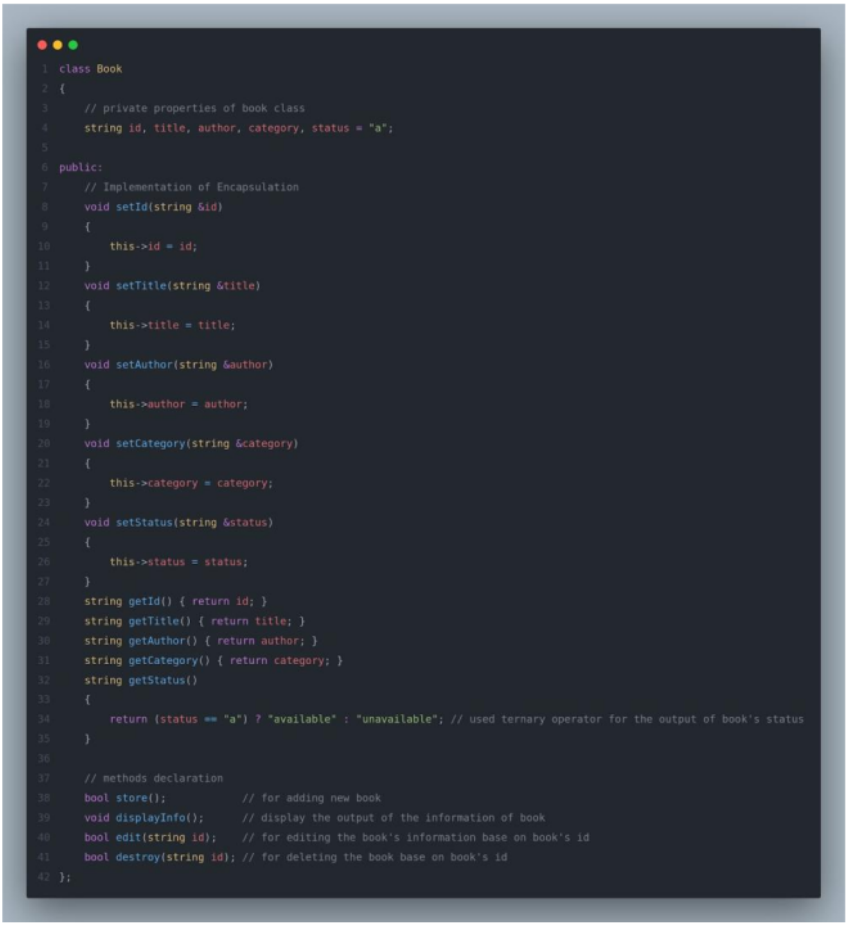
Fig (2.2)

In Fig (2.2), the Library class is said to be inherited publically from the Menu class (the base class). The Library class can, now, have access to manipulate the protected members of the Menu class.

Task 3 - Encapsulation

Encapsulation is one of the features of Object-Oriented Programming. Encapsulation can be defined as setting the data of a class as private properties and manipulating them by using class functions. Those class functions can only be accessed by an object of its class and must use only member variables. If there is no function inside the class that is using the member variable of the class, it cannot be called encapsulation. By using encapsulation, the security of class data can be increased and it also helps to control the modification of class data members.

Implementation of Encapsulation, which is used in the program, is shown in Fig (3.1) and Fig (3.2).



```
class Book
{
    // private properties of book class
    string id, title, author, category, status = "a";

public:
    // Implementation of Encapsulation
    void setId(string &id)
    {
        this->id = id;
    }
    void setTitle(string &title)
    {
        this->title = title;
    }
    void setAuthor(string &author)
    {
        this->author = author;
    }
    void setCategory(string &category)
    {
        this->category = category;
    }
    void setStatus(string &status)
    {
        this->status = status;
    }
    string getId() { return id; }
    string getTitle() { return title; }
    string getAuthor() { return author; }
    string getCategory() { return category; }
    string getStatus()
    {
        return (status == "a") ? "available" : "unavailable"; // used ternary operator for the output of book's status
    }

    // methods declaration
    bool store();           // for adding new book
    void displayInfo();     // display the output of the information of book
    bool edit(string id);   // for editing the book's information base on book's id
    bool destroy(string id); // for deleting the book base on book's id
};
```

Fig (3.1)

The Book class has the private properties and those can only be manipulated by the getters and setters functions of the class.

```
if (type == temp[4] || type == "all") // display book's info base on type
          {
              Book book;
              book.setId(temp[0]);
              book.setTitle(temp[1]);
              book.setAuthor(temp[2]);
              book.setCategory(temp[3]);
              book.setStatus(temp[4]);
              book.displayInfo();
          }
```

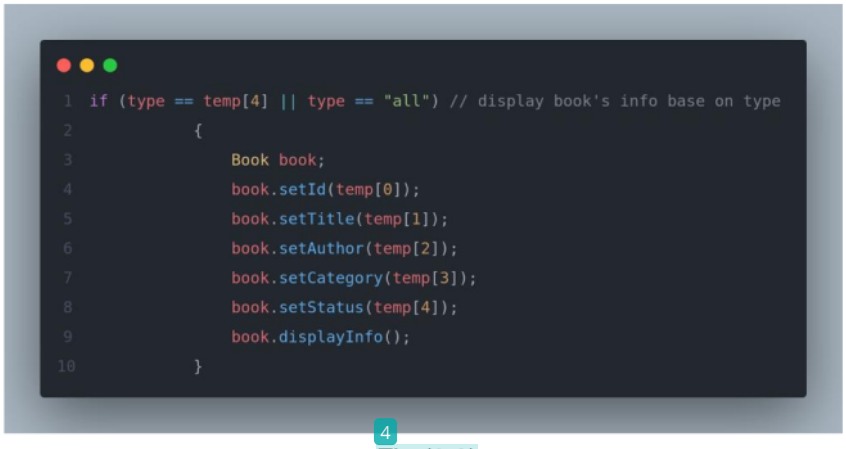


Fig (3.2)

Fig (3.2) shows the private members of the Book class are being set by the setters methods of its class.

COS 212 Assignment

ORIGINALITY REPORT

19%	11%	16%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Li Zheng, Yuan Dong, Fang Yang. "7. Inheritance And Derivation", Walter de Gruyter GmbH, 2019 Publication	4%
2	etheses.whiterose.ac.uk Internet Source	3%
3	Ivor Horton's Beginning ANSI C++ The Complete Language, 2004. Publication	2%
4	digital.lib.washington.edu Internet Source	2%
5	data.epo.org Internet Source	2%
6	www.joiebaby.com Internet Source	2%
7	Kishori Sharan, Adam L. Davis. "Chapter 20 Inheritance", Springer Science and Business Media LLC, 2022 Publication	2%
	csharprescue.wordpress.com	

8	Internet Source	1%
9	uhra.herts.ac.uk Internet Source	1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		