**Sir Syed CASE Institute of Technology**



**Department of Computer Science**

**Project Report**

SUBMITTED TO:

**Yasir Nayzi**

SUBMITTED BY:

|  |  |  |
| --- | --- | --- |
| 1 | Abdullah Ammar | 2510-4005 |
| 2 | Usman Aslam | 2330-0150 |
| 3 | M.Talha Ramzan | 2330-0141 |

**Project Report**

**Smart Library Management System**

# 1. Introduction

This report documents the final implementation of the "Smart Library Management System." It is a C++ console-based project where we design a library system that can add, search, display books, and save/load data from files. The project uses advanced Object-Oriented Programming (OOP) concepts such as inheritance, polymorphism, encapsulation, abstraction, static members, friend functions, operator overloading, mutable variables, and file handling. The system manages both normal books as well as special books that have additional details.

# 2. Objectives

* To create a library system that can efficiently store and manage book data.
* To practically implement core OOP concepts.
* To provide functionality to save and load book information to/from files.
* To prevent duplicate book IDs ensuring each book is unique.
* To use polymorphism to differentiate the display of normal and special books.
* To provide a user-friendly interface for adding, searching, and displaying books.

# 3. Technologies Used

* Programming Language: C++
* Compiler: GCC / Code::Blocks / Dev-C++
* Platform: Console (Text-based UI)
* File Handling: Reading and writing text files

# 4. Project Features

* Ability to add normal and special books.
* Ensures each book has a unique ID; shows error if a duplicate ID is entered.
* Displays the list of books on the console with polymorphic behavior.
* Saves books to a file and loads books from a file.
* Searches books by title and shows the book’s location (row number) in the list along with search results.
* Uses a static counter to track the total number of books currently in the library.
* Uses a friend function to print book information.
* Overloads the equality operator (==) to compare books by their IDs.
* Uses a mutable member variable to track how many times a book has been viewed (displayed).

# 5. OOP Concepts Used

* **Class and Object**:  
  Book, SpecialBook, and Library classes are created where objects are made to manage books and their operations.
* **Inheritance**:  
  The SpecialBook class inherits from the Book class and adds its own genre data.
* **Polymorphism:**  
  A virtual function display() is used to allow different display behavior for base and derived classes.
* **Encapsulation:**  
  Book data members are protected/private and accessed via public getter/setter methods.
* **Abstraction:**  
  The Library class hides direct data manipulation from users and only provides methods such as addBook, showAllBooks, and searchBook.
* **Static Members:**  
  The static member totalBooks in Book class tracks how many books are currently in the library.
* **Friend Function**:  
  A friend function printBookInfo() is created which accesses private members to print book details.
* **Operator Overloading:**  
  The equality operator (==) is overloaded to compare book objects based on their IDs.
* **Mutable Variable:**  
  The viewCount variable is declared mutable so it can be updated inside const methods; it tracks how many times a book has been displayed.
* **File Handling:**  
  The library’s data can be saved to a file and loaded back from a file to ensure data persistence.

# 6. Conclusion

In this project, advanced OOP concepts were implemented through a real-world example. The library system provides users with an efficient way to manage their book collections, along with important features such as data persistence and validation. This project serves as a good demonstration of OOP principles and C++ file handling capabilities.

**Complete Code**

|  |
| --- |
| #include <iostream>  #include <fstream>  using namespace std;  const int MAX\_BOOKS = 100;  // Abstract Base Class (Abstraction)  class IBook {  public:  virtual void display() const = 0; // pure virtual function  virtual string getTitle() const = 0; // pure virtual function  virtual ~IBook() {}  };  // Book class - Base class  class Book : public IBook {  protected:  int id;  string title;  string author;  mutable int viewCount; // mutable allows modification in const functions  public:  static int totalBooks; // static member  Book() : id(0), title(""), author(""), viewCount(0) {  totalBooks++;  }  Book(int id, const string& title, const string& author)  : id(id), title(title), author(author), viewCount(0) {  totalBooks++;  }  // Copy constructor  Book(const Book& other) {  this->id = other.id;  this->title = other.title;  this->author = other.author;  this->viewCount = other.viewCount;  totalBooks++;  }  // Destructor  virtual ~Book() {  totalBooks--;  }  // Using 'this' pointer in assignment operator (Operator Overloading)  Book& operator=(const Book& other) {  if (this != &other) {  this->id = other.id;  this->title = other.title;  this->author = other.author;  this->viewCount = other.viewCount;  }  return \*this;  }  bool operator==(const Book& other) const {  return this->id == other.id;  }  virtual void display() const override {  viewCount++; // mutable member changed in const function  cout << "Book ID: " << id << endl;  cout << "Title: " << title << endl;  cout << "Author: " << author << endl;  cout << "View Count: " << viewCount << endl;  }  string getTitle() const override {  return title;  }  string getAuthor() const {  return author;  }  int getId() const {  return id;  }  friend void printBookInfo(const Book& b);  };  // Initialize static member  int Book::totalBooks = 0;  // Friend function example  void printBookInfo(const Book& b) {  cout << "[Friend Function] Book: " << b.title << " by " << b.author << endl;  }  // Derived class SpecialBook (Inheritance + Polymorphism)  class SpecialBook : public Book {  private:  string genre;  public:  SpecialBook(int id, const string& title, const string& author, const string& genre)  : Book(id, title, author), genre(genre) {}  void display() const override {  cout << "[Special Book]" << endl;  Book::display();  cout << "Genre: " << genre << endl;  }  string getGenre() const {  return genre;  }  };  class Library {  private:  Book\* books[MAX\_BOOKS];  int count;  // Check if book id already exists (for unique IDs)  bool isIdUsed(int id) const {  for (int i = 0; i < count; i++) {  if (books[i]->getId() == id) {  return true;  }  }  return false;  }  public:  Library() : count(0) {}  ~Library() {  for (int i = 0; i < count; i++)  delete books[i];  }  // Add book if id unique  void addBook(Book\* b) {  if (count >= MAX\_BOOKS) {  cout << "Library is full.\n";  delete b;  return;  }  if (isIdUsed(b->getId())) {  cout << "Error: Book ID " << b->getId() << " already exists.\n";  delete b;  return;  }  books[count++] = b;  cout << "Book added successfully!\n";  }  void showAllBooks() const {  if (count == 0) {  cout << "No books to display.\n";  return;  }  for (int i = 0; i < count; i++) {  cout << "Location (index): " << i << endl;  books[i]->display();  cout << "---------------------\n";  }  }  void searchBook(const string& title) const {  bool found = false;  for (int i = 0; i < count; i++) {  if (books[i]->getTitle() == title) {  cout << "Book found at location (index): " << i << endl;  books[i]->display();  found = true;  break;  }  }  if (!found) {  cout << "Book not found.\n";  }  }  void saveToFile(const string& filename) const {  ofstream fout(filename);  if (!fout) {  cout << "Error opening file for writing.\n";  return;  }  fout << count << endl;  for (int i = 0; i < count; i++) {  fout << books[i]->getId() << endl;  fout << books[i]->getTitle() << endl;  fout << books[i]->getAuthor() << endl;  SpecialBook\* spb = dynamic\_cast<SpecialBook\*>(books[i]);  if (spb) {  fout << "Special" << endl;  fout << spb->getGenre() << endl;  } else {  fout << "Normal" << endl;  }  }  fout.close();  cout << "Books saved to file.\n";  }  void loadFromFile(const string& filename) {  ifstream fin(filename);  if (!fin) {  cout << "Error opening file for reading.\n";  return;  }  // Delete existing books  for (int i = 0; i < count; i++) {  delete books[i];  }  count = 0;  int n;  fin >> n;  fin.ignore();  for (int i = 0; i < n; i++) {  int id;  string title, author, type, genre;  fin >> id;  fin.ignore();  getline(fin, title);  getline(fin, author);  getline(fin, type);  if (type == "Special") {  getline(fin, genre);  addBook(new SpecialBook(id, title, author, genre));  } else {  addBook(new Book(id, title, author));  }  }  fin.close();  cout << "Books loaded from file.\n";  }  };  int main() {  Library lib;  int choice;  do {  cout << "\n=== Library Menu ===\n";  cout << "1. Add Book\n";  cout << "2. Add Special Book\n";  cout << "3. Show All Books\n";  cout << "4. Search Book\n";  cout << "5. Save Books to File\n";  cout << "6. Load Books from File\n";  cout << "7. Show Total Books (Static Member)\n";  cout << "8. Exit\n";  cout << "Enter choice: ";  cin >> choice;  if (choice == 1) {  int id;  string title, author;  cin.ignore();  cout << "Enter Book ID: ";  cin >> id;  cin.ignore();  cout << "Enter Title: ";  getline(cin, title);  cout << "Enter Author: ";  getline(cin, author);  Book\* b = new Book(id, title, author);  lib.addBook(b);  printBookInfo(\*b); // friend function call  }  else if (choice == 2) {  int id;  string title, author, genre;  cin.ignore();  cout << "Enter Book ID: ";  cin >> id;  cin.ignore();  cout << "Enter Title: ";  getline(cin, title);  cout << "Enter Author: ";  getline(cin, author);  cout << "Enter Genre: ";  getline(cin, genre);  Book\* sb = new SpecialBook(id, title, author, genre);  lib.addBook(sb);  }  else if (choice == 3) {  lib.showAllBooks();  }  else if (choice == 4) {  string searchTitle;  cin.ignore();  cout << "Enter Title to Search: ";  getline(cin, searchTitle);  lib.searchBook(searchTitle);  }  else if (choice == 5) {  string filename;  cin.ignore();  cout << "Enter filename to save: ";  getline(cin, filename);  lib.saveToFile(filename);  }  else if (choice == 6) {  string filename;  cin.ignore();  cout << "Enter filename to load: ";  getline(cin, filename);  lib.loadFromFile(filename);  }  else if (choice == 7) {  cout << "Total books in library (static): " << Book::totalBooks << endl;  }  else if (choice == 8) {  cout << "Exiting...\n";  }  else {  cout << "Invalid choice.\n";  }  } while (choice != 8);  return 0;  } |

## Summary

The Smart Library Management System is a C++ console application designed to help users manage a collection of books efficiently. It allows users to add normal and special books, search for books by title, display all books with detailed information, and save/load the book data to and from a file. The system ensures each book has a unique ID, preventing duplicates. It uses Object-Oriented Programming principles to make the code organized, reusable, and easy to maintain. Features like polymorphism enable different types of books to be displayed appropriately, while static members keep track of total books. The system also keeps track of how many times a book has been viewed.