**Library Management System**

**Data Structures**

* **Structs** ***Book***, ***BookQuery***, ***Member***, and ***BorrowingRecord*** are implemented as structs because they represent simple data entities that are immutable and can be compared easily.
  + ***Book***: Represents a book entity with attributes such as ISBN, Title, Author, Year, Genre, and Quantity. Each book has a unique, self-incremented ID managed using a static counter.
  + ***Member***: Represents a user entity with attributes like Name and a list of borrowed books. Each member is assigned a unique, self-incremented ID.
  + ***BorrowingRecord***: Represents a borrowing record, capturing the relationship between a Member and a Book. It includes attributes like Borrow Date, Due Date, and Returned status.
  + ***BookQuery***: Represent search criteria for books, making it easy to extend with new search parameters.
* **Enums** are used to define menu options clearly, enhancing code readability and maintainability.
  + ***ELMSChoice***: Represents the main actions users can perform, such as adding books, viewing books, searching for books, borrowing books, returning books, and exiting the system.
  + ***ELMSSearchChoice***: Defines search options for books, including by ID, Title, and Author, facilitating easy extension for additional search criteria.
* **List<T>** collection is used for dynamically storing books, members, and borrowing records, allowing flexible sizing and easy iteration over the collections.
  + **Books**: ***List<Book>*** stores the collection of books.
  + **Members**: ***List<Member>*** maintains the collection of library members.
  + **Borrowing Records*: List<BorrowingRecord>*** holds the records of all book borrowings.

**Algorithms Implemented**

* **Binary Search**: Used for quickly finding books by their ID, which is efficient and reduces the search time compared to linear search.
* **Linear Search**: Used for searching books by Title and Author due to the flexibility of partial and case-insensitive matches.

**Challenges**

* **Unique ID Management**: Ensuring that each book, borrowing records and member had a unique ID was crucial. This was addressed using static self-increased counters (***IDCounter***).
* **Book Quantity Updates**: Managing the quantity of books required careful handling to avoid inconsistencies. Books are checked for existence before adding, and quantities are updated appropriately.
* **Borrowing Logic**: Ensuring that books could not be borrowed if not available for borrowing and managing the borrowing records accurately was a challenge that required clear logic and validation.

### **Test Cases**

1. **Adding Books**:
   1. Test adding a book with new details.
   2. Verify that adding an existing book updates its quantity.
2. **Viewing Books**:
   1. Test viewing all books.
   2. Test filtering books by genre.
3. **Searching Books**:
   1. Test searching for a book by ID.
   2. Test searching for books by Title and Author.
4. **Borrowing Books**:
   1. Test borrowing a book when it is available.
   2. Test borrowing a book when it is not available.
5. **Returning Books**:
   1. Test returning a book that was borrowed.
   2. Test attempting to return a book that was not borrowed.

**Future Development**

* **Member Management:** Implement functionalities for adding, removing, and viewing members to provide a more complete system.
* **Book Reservation:** Add features for reserving books that are currently not available.
* **Extended Search:** Enhance the ***BookQuery*** struct to support more sophisticated search queries, including by publication year or genre.