K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Assignment 2

Library Book Management System

Department: CSE

Session: 2025-26

Programme: B.Tech

Semester: 3rd

Course Code: ENCS205

Course: Data Structures

Assignment No: 02

Submitted by: Hemant Saini

Roll no. 2401420051

Submitted to: Dr Swati

---

1. Introduction

This assignment focuses on the implementation of a Library Book Management System using Linear Data Structures – specifically Single Linked Lists and Stacks in Python. The system is designed to manage book records dynamically, maintain borrowing history, and track undo operations related to book issue or return transactions. It demonstrates the practical application of linked list operations (insert, delete, search) and stack-based undo mechanisms to simulate real-world library management scenarios.

2. Problem Statement

The task is to develop a console-based software solution in Python that organizes and manages library book records using single linked lists. The system must support operations such as adding books, removing books, searching for books, issuing books to borrowers, returning books, and undoing the last transaction. The undo functionality is implemented using a stack data structure to revert recent issue or return operations.

3. Book Record ADT Design

Attributes:

- BookID: Integer (unique identifier for the book)
- BookTitle: String (title of the book)
- AuthorName: String (author of the book)
- Status: String (Available / Issued)

Methods:

- insertBook(data): Add a new book record to the linked list
- deleteBook(BookID): Remove a book record using its BookID
- searchBook(BookID): Retrieve details of a specific book
- displayBooks(): Display all books currently in the library

4. Transaction Management System Design

Attributes:

- BookList: A linked list to store all book records dynamically
- TransactionStack: A stack to record issue and return operations for undo functionality

Methods:

- issueBook(BookID): Mark a book as issued and push this transaction onto the stack
- returnBook(BookID): Mark a book as returned and push this transaction onto the stack
- undoTransaction(): Revert the last transaction using stack pop operation
- viewTransactions(): Display all recent transactions

5. Implementation (Python Code)

```
# --- Book node for the singly linked list ---
class Book:
    def __init__(self, book_id, title, author, status="Available"):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.status = status
        self.next = None

# --- Singly linked list for books management ---
class LinkedList:
    def __init__(self):
        self.head = None

    def insertBook(self, book_id, title, author):
        new_book = Book(book_id, title, author)
        new_book.next = self.head
        self.head = new_book
        print(f"Book '{title}' added.")

    def deleteBook(self, book_id):
        current = self.head
        prev = None
        while current:
            if current.book_id == book_id:
                if prev:
                    prev.next = current.next
                else:
                    self.head = current.next
                print(f"Book '{current.title}' deleted.")
                return current
            prev = current
```

```
            current = current.next
            print("Book not found.")
            return None

        def searchBook(self, book_id):
            current = self.head
            while current:
                if current.book_id == book_id:
                    return current
                current = current.next
            return None

        def displayBooks(self):
            books = []
            current = self.head
            while current:
                books.append((current.book_id, current.title,
                              current.author, current.status))
                current = current.next
            return books

    # --- Stack class for undo functionality ---
    class Stack:
        def __init__(self):
            self.stack = []

        def push(self, data):
            self.stack.append(data)

        def pop(self):
            if self.stack:
                return self.stack.pop()
            return None

        def is_empty(self):
            return len(self.stack) == 0

        def viewTransactions(self):
            return self.stack[::-1]

    # --- Library System integrating list and stack ---
    class LibrarySystem:
        def __init__(self):
            self.book_list = LinkedList()
            self.trans_stack = Stack()

        def insertBook(self, book_id, title, author):
            self.book_list.insertBook(book_id, title, author)
```

```
def deleteBook(self, book_id):
    book = self.book_list.deleteBook(book_id)
    if book:
        self.trans_stack.push(("delete", book.book_id,
                               book.title, book.author, book.status))

def searchBook(self, book_id):
    book = self.book_list.searchBook(book_id)
    if book:
        return (book.book_id, book.title, book.author, book.status)
    return None

def displayBooks(self):
    return self.book_list.displayBooks()

def issueBook(self, book_id):
    book = self.book_list.searchBook(book_id)
    if book and book.status == "Available":
        book.status = "Issued"
        self.trans_stack.push(("issue", book_id))
        print(f"Book '{book.title}' issued.")
    else:
        print("Book not available for issue.")

def returnBook(self, book_id):
    book = self.book_list.searchBook(book_id)
    if book and book.status == "Issued":
        book.status = "Available"
        self.trans_stack.push(("return", book_id))
        print(f"Book '{book.title}' returned.")
    else:
        print("Book not currently issued.")

def undoTransaction(self):
    last_action = self.trans_stack.pop()
    if not last_action:
        print("No transactions to undo.")
        return

    action = last_action[0]
    if action == "issue":
        book_id = last_action[1]
        book = self.book_list.searchBook(book_id)
        if book:
            book.status = "Available"
            print(f"Undo: Book '{book.title}' status set to Available.")
    elif action == "return":
```

```
                book_id = last_action[1]
                book = self.book_list.searchBook(book_id)
                if book:
                    book.status = "Issued"
                    print(f"Undo: Book '{book.title}' status set to Issued.")
            elif action == "delete":
                _, book_id, title, author, status = last_action
                self.book_list.insertBook(book_id, title, author)
                inserted_book = self.book_list.searchBook(book_id)
                if inserted_book:
                    inserted_book.status = status
                print(f"Undo: Book '{title}' restored.")

    def viewTransactions(self):
        return self.trans_stack.viewTransactions()

# --- Sample usage ---
if __name__ == "__main__":
    ls = LibrarySystem()
    ls.insertBook(1, "DSA Fundamentals", "Alice")
    ls.insertBook(2, "Python Basics", "Bob")
    ls.insertBook(3, "Algorithms", "Carol")
    print("Initial books:", ls.displayBooks())

    ls.issueBook(2)
    ls.returnBook(2)
    ls.deleteBook(3)
    print("Books after operations:", ls.displayBooks())

    ls.undoTransaction()
    ls.undoTransaction()
    print("Books after undo:", ls.displayBooks())
    print("Transactions:", ls.viewTransactions())
```

6. Data Structure Analysis

Singly Linked List

The singly linked list is used to manage the dynamic collection of book records. Each node in the list represents a book with attributes such as BookID, Title, Author, and Status. The linked list allows efficient insertion at the head, deletion by traversing and unlinking nodes, and search operations by sequential traversal.

- Advantages:
    - Dynamic memory allocation (no fixed size)
    - Efficient insertion and deletion operations
    - Suitable for collections with frequent modifications

Stack for Undo Mechanism

The stack data structure is used to implement the undo functionality. Each transaction (issue, return, or delete) is pushed onto the stack. When an undo operation is requested, the most recent transaction is popped from the stack and reverted accordingly.

- Advantages:
    - LIFO (Last In First Out) property perfectly suits undo operations
    - Simple and efficient push/pop operations
    - Maintains transaction history chronologically

7. Complexity Analysis

Time Complexity:

- insertBook(): O(1) – insertion at the head of linked list
- deleteBook(): O(n) – requires traversal to find the book
- searchBook(): O(n) – sequential search through linked list
- displayBooks(): O(n) – traverses entire list
- issueBook() / returnBook(): O(n) – involves search operation
- undoTransaction(): O(1) for stack pop + O(n) for search
- Stack push/pop: O(1)

Space Complexity:

- Linked List: O(n), where n is the number of books
- Stack: O(k), where k is the number of transactions
- Overall: O(n + k)

8. Implementation Steps Followed

1. Defined the Book class as a node structure for the singly linked list
2. Implemented LinkedList class with insert, delete, search, and display methods
3. Created Stack class with push, pop, and view operations
4. Developed LibrarySystem class integrating both data structures
5. Implemented issueBook() and returnBook() with stack recording
6. Developed undoTransaction() functionality using stack pop and state reversal
7. Added comprehensive testing with sample book records and transactions
8. Validated undo operations for issue, return, and delete actions

9. Conclusion

This assignment demonstrates the design and implementation of a Library Book Management System using Single Linked Lists and Stacks in Python. It highlights the practical application of fundamental linear data structures in managing dynamic data collections and implementing undo mechanisms. The system successfully fulfills all assignment objectives including dynamic book management, transaction tracking, and efficient undo operations. The implementation showcases algorithmic thinking in designing insert, delete, search, and undo operations while simulating real-world library management scenarios.