Assignment: Singly Linked List Implementation

Objective

This assignment is designed to challenge students to implement and manipulate singly linked lists dynamically. Students will demonstrate their understanding of linked lists, dynamic memory allocation, and algorithmic problem-solving.

Instructions

Problem Statement

Design and implement a console-based **Library Catalog** that allows users to:

1. **Add a Book**:
    - Prompt the user for book details: a unique ID, Title, Author, and Publication Year.
    - Validate the ID to ensure it is unique. If a duplicate ID is entered, display an error message and do not add the book.
    - Dynamically allocate memory for the book and add it to the singly linked list.
    - The new book should be added to the end of the list.
2. **View All Books**:
    - Traverse the singly linked list and display the details of all books in the catalog.
    - If the list is empty, display an appropriate message indicating that there are no books in the catalog.
3. **Update a Book**:
    - Prompt the user to enter the ID of the book they want to update.
    - If the book with the specified ID exists, allow the user to modify its Title, Author, and Publication Year.
    - If no book with the given ID is found, display an appropriate message.
4. **Delete a Book**:
    - Prompt the user to enter the ID of the book they want to delete.
    - Search for the book in the singly linked list and, if found, remove it from the list and free the associated memory.
    - If no book with the given ID exists, display an error message.
5. **Search for a Book**:
    - Prompt the user to enter a Title or partial Title to search for.
    - Traverse the singly linked list and display the details of all books whose Title contains the search string (case-insensitive).
    - If no matching books are found, display an appropriate message.

Requirements

- Use a **singly linked list** to manage the books.
- Implement **dynamic memory allocation** for creating and managing books.
- The program should handle edge cases such as adding duplicate book IDs, invalid inputs, or an empty catalog gracefully.
- Provide a menu-driven interface for all operations.

Program Structure

1. Define a `Book` structure with the following fields:
    - `int id`
    - `char title[100]`
    - `char author[100]`
    - `int publication_year`
    - `struct Book* next` (pointer to the next book in the list)
2. Implement the following functions:
    - `void addBook(Book** head, int id, const char* title, const char* author, int publication_year)`
        - Adds a new book to the end of the linked list.
    - `void viewBooks(Book* head)`
        - Traverses the linked list and prints details of all books.
    - `void updateBook(Book* head, int id)`
        - Searches for a book by ID and updates its details.
    - `void deleteBook(Book** head, int id)`
        - Removes a book by ID from the linked list and frees the memory.
    - `void searchBooks(Book* head, const char* title)`
        - Searches for books containing the specified Title or partial Title.
3. Ensure proper memory management with functions to free memory when books are deleted or the program exits.

Deliverables

1. A `LibraryCatalog.c` file containing the implementation.
2. A short report (1-2 pages) explaining the approach, challenges faced, and how they were overcome.

- Submit all files in a compressed folder named `LibraryCatalog_<YourName>.zip`.
- Include detailed comments in your code.

Grading Rubric (10 Points Total)

Criteria	Excellent (2 points)	Good (1.5 points)	Average (1 point)	Needs Improvement (0.5 points)
Code Functionality	All features implemented correctly with no bugs.	Most features implemented, minor bugs.	Some features implemented, significant bugs.	Minimal features implemented, major issues.
Code Efficiency	Efficient algorithms and data structures used.	Algorithms are mostly efficient.	Inefficient algorithms used in places.	Poorly designed and inefficient code.
Dynamic Memory Usage	Proper use of dynamic memory allocation and deallocation.	Minor issues with memory management.	Significant memory management issues.	No use of dynamic memory or major leaks.
User Interface	Clear, user-friendly, and intuitive interface.	Mostly clear interface with minor issues.	Interface is functional but not user-friendly.	Poorly designed interface, hard to use.
Report Quality	Comprehensive and well-written report.	Adequate report covers most aspects.	Report lacks details or clarity.	Report missing or very poorly written.

Total Score: /10

If you need clarification or assistance, feel free to reach out during office hours or via email. Good luck!