Data Structures and Algorithms Lab

Lab 10 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

Book Management System with Doubly Linked List Operations

You are required to implement a **BookList** class that stores **Books** in an **unsorted order**. The class declarations are provided below:

```
class Book
{
    friend class BookList;

private:
    int id;                 /** ID of a book. */
    string title;           /** Name of a book. */
    float price;            /** Price of a book. */
    Book* next;             /** Address of the next node. */
    Book* pre;              /** Address of the previous node. */

public:
    /** Constructor */
    Book(int id, string title, float price, Book* next = nullptr, Book* pre = nullptr);
    void bookDetails(); /** Displays the book information */
};

class BookList
{
private:
    Book* head;             /** Pointer to the first node in the list. */
    Book* cursor;           /** Pointer to the current node in the list. */
    Book* last;             /** Pointer to the last node in the list. */

public:
    BookList();             /** Constructor */
    ~BookList();            /** Destructor */

    void insert(const Book& newItem);         /** Insert a new book. */
    void remove();                            /** Remove the book at the cursor. */
    void search(string title);                /** Search for a book by title. */
    void replace(const Book& newItem);        /** Replace the book at the cursor. */
    bool isEmpty() const;                     /** Check if the list is empty. */
    void gotoBeginning();                     /** Move the cursor to the beginning. */
    void gotoEnd();                           /** Move the cursor to the end. */
    bool gotoNext();                          /** Move the cursor to the next node. */
    bool gotoPrior();                         /** Move the cursor to the previous node. */
    Book getCursor() const;                   /** Get a copy of the book at the cursor. */
    void showStructureForward() const;        /** Display all books from head to tail. */
    void showStructureReverse() const;        /** Display all books from tail to head. */
};
```

Specifications for Member Functions

Each member function should follow the given behavior:

1. `insert(const Book& newItem)`
   - Inserts **newItem** into the list. If the list is not empty, inserts **newItem after the cursor**. Otherwise, inserts it as the **first and only book**. In either case, moves the cursor to **newItem**.
   - **Time Complexity**: $O(1)$

2. `remove()`
    - Removes the book at the cursor. If the list is not empty afterward, moves the cursor to the next book. If the deleted book was the last in the list, moves the cursor to the first book.
    - **Time Complexity**: $O(1)$
3. `search(string title)`
    - Searches for book(s) by **title** in the list. Displays the details of all matching books or an appropriate message if no matches are found.
    - **Time Complexity**: $O(N)$
4. `replace(const Book& newItem)`
    - Replaces the book at the cursor with newItem. The cursor remains on newItem.
    - **Time Complexity**: $O(N)$ (search for id) or $O(1)$ (if appending at the end)
5. `isEmpty() const`
    - Returns **true** if the list is empty, otherwise returns **false**.
    - **Time Complexity**: $O(1)$
6. `gotoBeginning()`
    - Moves the cursor to the beginning of the list.
    - **Time Complexity**: $O(1)$
7. `gotoEnd()`
    - Moves the cursor to the end of the list.
    - **Time Complexity**: $O(N)$
8. `gotoNext()`
    - Moves the cursor to the next node if it's not at the end, returning **true**. Otherwise, returns **false**.
    - **Time Complexity**: $O(1)$
9. `gotoPrior()`
    - Moves the cursor to the previous node if it's not at the beginning, returning **true**. Otherwise, returns **false**.
    - **Time Complexity**: $O(1)$
10. `getCursor() const`
    - Returns a copy of the book at the cursor.
    - **Time Complexity**: $O(1)$
11. `showStructureForward() const`
    - Displays the books in the list from **head to tail**. If the list is empty, outputs "Empty list".
    - **Time Complexity**: $O(N)$
12. `showStructureReverse() const`
    - Displays the books in the list from **tail to head**. If the list is empty, outputs "Empty list".
    - **Time Complexity**: $O(N)$

Input File Specifications

The program should read book data from the file **input.txt** in the following format:

```
<id>
<book name>
<price>

<id>
<book name>
<price>
...
```

Each book's data is separated by a blank line.

Main Function

- Load the data from **input.txt** and store each book in the **BookList**.
- Test all public member functions of the **BookList** class. Display outputs for each operation to demonstrate functionality.