Design (Inspiration taken heavily from my previous 133C work)

main.cpp:

Include book.h and test.h

* int main()
    - o test_add
    - o test_peek
    - o test_delete

test.h

Declares the functions:

* test_add
* test_peek
* test_delete

test.cpp

* test_add
    - o add 3 books always adding to the back [add_book(&head, book1)]
    - o print books added in order
* test_peek
    - o add books
    - o peek at top (should show first book)
* test_delete
    - o deletes books from the front until empty
    - o deletes first book first

book.h

Contains:

* The struct for Book
    - o *Title (pointer to a string (title))
    - o Book id (integer)
* The struct for BookList
    - o Allows for linked list
* Declares the functions
    - o Create_node
    - o Add_book
    - o Delete_book
    - o Peek_a_book

book.cpp

* Create_node(BookList **head_ref, Book book)
    - o Creates new node for linked list of books
* Add_book(BookList **head_ref)
    - o create_node
    - o if empty head_ref points to new node
    - o else traverse queue until last node and once last node is found set its next ptr to point to new node
* Delete_book(BookList **head_ref)
    - o if queue is empty, error message
    - o else delete
    - o return value of deleted book

* peek_a_book(BookList **head_ref)
    - o if empty error
    - o else peek

Requirements:

1. Based on what we know about linked lists, stacks, and queues, design a linked queue (a queue using a linked-list to store the data in the structure)

    - See above design

Design (Inspiration taken heavily from my previous 133C work)

main.cpp:

Include book.h and test.h

* int main()
    - o test_add
    - o test_peek
    - o test_delete

test.h

Declares the functions:

* test_add
* test_peek
* test_delete

test.cpp

* test_add
    - o add 3 books always adding to the back [add_book(&head, book1)]
    - o print books added in order

Note: I used a similar method to my 133C class because I understand it. I will attempt to learn this new style better.

2. Design, implement, and test a Queue data structure that:
    1. uses a **linked-list** to store values in the queue

```
typedef struct Node
{
    Book book;
    struct Node *next;
}
BookList;
```

    2. has an **enqueue** method that will appropriately **add** a value to the **back** of the queue as an appropriate element.

```
int add_book(BookList **head_ref, Book book) {
    //creates a new node
    BookList* newNode = create_node(book);

    //if head_ref is empty set head_ref to point to new node
    if (*head_ref == nullptr) {
        *head_ref = newNode;
    } else {
        //If the queue is not empty, traverse the queue to find the last node
        BookList* temp = *head_ref;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        // Once the last node is found, set its next pointer to point to the newly create
        temp->next = newNode;
    }
```

3. has a **dequeue** method that will appropriately **remove** an element from the **front** of the queue **and return its value.**

```
39  // Deletes the book at the front of the queue and returns its value
40  Book delete_book(BookList **head_ref) {
41      if (*head_ref == nullptr) {
42          return {"", -1};
43      } else {
44          BookList* temp = *head_ref;
45          *head_ref = (*head_ref)->next;
46
47          // Store the value of the deleted book
48          Book deletedBook = temp->book;
49
50          // No need to free memory for title (handled by std::string automatically)
51          // Free the memory allocated for the BookList node
52          delete temp;
53
54          // Return the value of the deleted book
55          return deletedBook;
56      }
57  }
```

4. Optionally has a **peek** method that **returns the value at the front** of the queue **without removing it**. **Bonus** if you also create an array based Queue!

```
// Peeks at front of queue and returns the values without removing it
Book peek_a_book(BookList **head_ref) {
    if (*head_ref == nullptr) {
        return {"", -1};
    } else {
        // Return the value of the front book
        return (*head_ref)->book;
    }
}
```

3. Tests: Be sure to include at least one test for each piece of functionality that should verify that your code is working!

```
// Test function for adding three books and printing them in order
void test_add() {
    BookList* head = nullptr;

    // Test adding books
    Book book1 = {"Murder Bot", 1234567};
    Book book2 = {"Eragon", 2345678};
    Book book3 = {"The Martian", 3456789};
    add_book(&head, book1);
    add_book(&head, book2);
    add_book(&head, book3);

    // Print the books in the order they were added
    cout << "Books in the order they were added:" << endl << endl;
    BookList* current = head;  // Start from the head
    while (current != nullptr) {
        cout << "Title: " << current->book.title << ", ID: " << current->book.id << endl;
        current = current->next;  // Move to the next node
```

```
Books in the order they were added:

Title: Murder Bot, ID: 1234567
Title: Eragon, ID: 2345678
Title: The Martian, ID: 3456789

Expected: Murder Bot, Eragon, The Martian
```

```
void test_peek() {
    BookList* head = nullptr;

    // Adding books
    Book book1 = {"Murder Bot", 1234567};
    Book book2 = {"Eragon", 2345678};
    Book book3 = {"The Martian", 3456789};
    add_book(&head, book1);
    add_book(&head, book2);
    add_book(&head, book3);

    // Test peeking at the front of the queue
    cout << "Peek at the top book: " << endl << endl;
    Book topBook = peek_a_book(&head);
    if (topBook.title != "" && topBook.id != -1) {
        cout << "Title: " << topBook.title << ", ID: " << topBook.id << endl << endl;
    } else {
        cout << "No book found at the top." << endl << endl;
    }
}//Expected: Murder Bot
```

```
void test_delete() {
    cout << "Testing delete_book until queue is empty:" << endl;
    BookList* head = nullptr;

    // Test adding books
    Book book1 = {"Murder Bot", 1234567};
    Book book2 = {"Eragon", 2345678};
    Book book3 = {"The Martian", 3456789};
    add_book(&head, book1);
    add_book(&head, book2);
    add_book(&head, book3);

    // Delete the front book and peek until the queue is empty
    while (head != nullptr) {
        // Print the top book's information before deleting
        cout << endl << "Peek at the top book:" << endl;
        Book topBook = peek_a_book(&head);
        if (topBook.title != "" && topBook.id != -1) {
            cout << "Title: " << topBook.title << ", ID: " << topBook.id << endl << endl
        } else {
            cout << "No book found at the top." << endl << endl;
        }

        // Delete the front book
        cout << "Deleting the front book!!!" << endl;

        // Print the deleted book's information
        Book deletedBook = delete_book(&head);
        cout << "Deleted book: " << deletedBook.title << " (" << deletedBook.id << ")" <

    }

    cout << "Queue is empty!" << endl;
}
```

```
Peek at the top book:

Title: Murder Bot, ID: 1234567


Expected: Murder Bot

Testing delete_book until queue is empty:

Peek at the top book:
Title: Murder Bot, ID: 1234567

Deleting the front book!!!
Deleted book: Murder Bot (1234567)

Peek at the top book:
Title: Eragon, ID: 2345678

Deleting the front book!!!
Deleted book: Eragon (2345678)

Peek at the top book:
Title: The Martian, ID: 3456789

Deleting the front book!!!
Deleted book: The Martian (3456789)
Queue is empty!
```

Note: test_delete() sent me on a bit of a memory allocation spiral with deleting the titles returning poorly and I relied on ChatGBT to help me find out where I went wrong.