Craig Truitt
Project 1 Design Documentation
CS5393-002 (Data Structures in C++)
Collaborators: None

Output:

```
Enter your username, email address, or name: Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
Successfully reserved. We have 0 copies of The Eagle Has Landed left.
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: a
What is the title? Harry Potter
Performing Binary Search ...
Successfully reserved. We have 2 copies of Harry Potter left.
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 0 reservations in front of you.
The current list of people who have reserved this book is: Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 1 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: b
Returning Harry Potter
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: b
Returning The Eagle Has Landed
Your reservation for The Eagle Has Landed is available. Would you like to a) retrieve or b) forfeit? <a/b>: a
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 1 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 2 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 3 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 4 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 5 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 6 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 7 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: a
Query by: a) Title b) ISBN <a/b>: b
What is the ISBN? 1209834
Performing hash on ISBN ...
All copies of The Eagle Has Landed taken.
You have reserved this book. There are 8 reservations in front of you.
The current list of people who have reserved this book is: Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous, Anonymous
Would you like to a) borrow a book, b) return a book, c) or quit? <a/b/c>: c

C:\Users\ctlus\source\repos\DS_Projects\x64\Debug\BigProject1.exe (process 29032) exited with code 0.
Press any key to close this window . . .
```

NOTE: This output was made on the old dataset.

Output details:
The majority of this output depicts 10 reservations on books. For both time and space, I decided to make these reservations all on the same book.
After making my first 2 reservations on this book, I returned a book twice. The reason that there were two returns was that borrowed books were tracked by a stack. Thus, the last successful borrow, which was a Harry Potter book, had to be returned before returning “The Eagle Has Landed.”
I did not count successful borrows as reservations for this assignment.

Assumptions made when coding:
-I assumed that reservations and borrowing were different things, especially since the CSV had a quantity in stock segment.
-Because the assignment said to keep track of borrowing with a stack, I assumed that books had to be returned in LIFO manner.
-Because a reservation occurs for a specific book, I determined that each book would have an associated reservation queue.

Summary:
The Library Management System (LMS) implemented in this project leverages multiple data structures to manage various library operations. A hash table stores books based on their ISBN for quick lookup, while an AVL tree (a self-balancing binary search tree) organizes books by title, allowing efficient retrieval and modification. Book reservations are handled using a queue, where users can reserve books, display the queue, and manage availability when multiple users request the same book. A stack is employed to track recently borrowed books, ensuring that books can be returned and re-lent efficiently. Memory management is centralized through a garbage collection system, preventing memory leaks by safely deleting unneeded objects. The project also incorporates search functionality, with algorithms such as binary search used to quickly locate specific records, demonstrating the practical application of different data structures to manage complex systems.

Coding architecture:
The key custom data type in this implementation is `bookInfo`, which represents and stores essential information about each book in the library.

1. `bookInfo` Struct
   - Purpose: Stores information about each book.
   - Data Members:
     - `ISBN number` (int)
     - `Title` (char*)

    - `Author` (char*)
    - `Price` (double)
    - `Quantity` (int)
    - `Reservations` (queue)
- Design Choices:
    - Implemented as a `struct`, not a `class`.
    - Includes a `print` method for testing and adding new features easily.
    - Memory Management: The struct contains pointers but doesn't allocate or deallocate memory directly. A separate data structure is responsible for managing memory, which simplifies tracking and ensures proper deletion.

2. `Queue` for Book Reservations
- Purpose: Stores the names of people reserving books.
- Implementation: Built on a doubly linked list.
- Data Members: Stores `const char*` values representing the reserver's name.
- Methods:
    - `enqueue`: Adds a `char*` to one end of the queue.
    - `dequeue`: Removes and returns a `char*` from the opposite end.
    - `front`: Retrieves the `char*` to be dequeued next.
    - `isEmpty`: Checks if the queue is empty.
    - `length`: Returns the length of the queue.
    - `displayAll`: Displays all reservations. This method was added to meet the project requirements.
- Memory Management: Since the queue stores `const` values, it does not manage memory. The names are managed externally.

3. `Garbage` Class
- Purpose: Handles memory allocation and deallocation.
- Implementation: Built on a singly linked list.
- Key Functionality:
    - No methods to view or remove specific values.
    - All memory is deleted when the destructor is called.

This class centralizes memory management, ensuring that memory allocated elsewhere in the program is properly cleaned up.

4. `AVL` Tree for Book Storage
- Purpose: Stores books sorted by title in a self-balancing binary search tree.
- Design:
    - Uses nodes that store pointers to `bookInfo`, allowing updates across different structures.
    - Self-balancing with AVL properties.
- Public Methods:

    - `insert`: Adds a book to the tree.
    - `retrieve`: Finds and returns the information associated with a title.
    - `remove`: Deletes a book from the tree.
- Private Methods:
    - `insertRec`, `retrieveRec`, `removeRec`: Recursive versions of the public methods.
    - `rotateRight`, `rotateLeft`: Helper functions for balancing the tree.
    - `balance`, `height`, `getBalance`: Ensure the tree remains balanced.
    - `findMin`: Retrieves the node with the smallest key.
    - `deleteTree`: Recursively deletes all nodes in the tree (but not the `bookInfo*`).
- Memory Management: Since `bookInfo*` is shared across multiple structures, memory is not managed within this class. Instead, the `garbage` class handles deletion when no longer needed.

5. `hashTable` for ISBN Storage
- Purpose: Stores books based on their ISBN numbers.
- Design: Utilizes an array of `sortedList` objects to handle collisions.
- Methods:
    - `insert`: Hashes the ISBN and inserts `bookInfo*` into the correct list.
    - `get`: Hashes the ISBN and retrieves the corresponding book.
    - `remove`: Deletes a book by ISBN.
- Constructor: The array size is based on an estimated number of books, rounded up to the nearest prime number to ensure even distribution of hash values.
- Memory Management: The hash table does not delete `bookInfo*` entries; this is handled by the `garbage` class.

6. `intHash` Class
- Purpose: Computes consistent yet random hash values for ISBNs.
- Hashing Method:
    - Uses pre-generated random values optimized for 10-13 digit ISBNs.
    - Hash function multiplies the ISBN by a large prime number, XORs with the original number, and takes a modulus with another prime constant.

7. `sortedList` Class for Collision Handling
- Purpose: Handles collisions in the hash table.
- Implementation: Singly linked list.
- Methods:
    - `insert`: Adds `bookInfo*` at the correct position in the sorted list.
    - `get`: Finds and returns the desired node, or `nullptr` if not found.
    - `remove`: Deletes the appropriate node.

- Memory Management: As with other classes, memory deletion is handled externally by the `garbage` class.

8. `stack` for Tracking Borrowed Books
   - Purpose: Keeps track of borrowed books, although a stack may not be the best data structure for this.
   - Implementation: Doubly linked list.
   - Methods:
     - `push`: Adds a book to the top of the stack.
     - `pop`: Removes and returns the book from the top.
     - `peep`: Returns the book at the top without removing it.

9. `LMS` Class
   - Purpose: Manages user interactions with the library system.
   - Functionality:
     - Initializes system data.
     - Provides an interface for users to interact with the library.

This architecture efficiently separates data structures and memory management responsibilities, ensuring flexibility and correctness throughout the library system

Sources

https://chatgpt.com/g/g-2DQzU5UZl-code-copilot