Project 1: Design Documentation

Ruth Assefa

Professor Chakrabarty

CS 5393 002

30 September 2024

Project 1: Library Management System

Project 1 asks to create a library management system that is implemented with multiple data structures such as hash tables, queues, stacks, and binary trees. The object of using the data structures in the code is not only to demonstrate an understanding of how they work but also how these structures are best used in certain situations. For the project, a dataset was given with a list of books that are made available in the library. In order to properly use this dataset, I first started by creating a function that could properly read and initialize the book information and creating a BookNode class that will be used to create objects of the Books. The public variables of the class are the ISBN, title, author, price, and quantity. I also created two pointers: 'left' and 'right' which will be used for the binary tree structure. The Book class also has multiple functions: print(), decreaseQuantity(), increaseQuantity(), and returnQuantity(). The quantity functions are helpful when books are being reserved or returned. After completing the Book class, I worked on the initializeBooks function. Because some of the book titles contain quotes, it was important to make sure that every variable is read properly when iterating through the file. When a line is read, a vector of strings is used to store all the info for each book, a string named 'field' is used to temporarily hold string for each field, a boolean value that checks if the string is in quotes, and also a string 'quotedfield' which is used to get the title that has quotes. A conditional statement will check to see if the current field has a quotation mark at the beginning of the string. If it does, then the boolean inQuotes variable will be marked as true and the 'quotefField' variable will become the current 'field' variable. Then the 'quotedfield' will be continually added to until it is detected that the last character of the current 'field' string is a quotation mark. The boolean variable will be marked as false, and the 'quotedField' will be added to the vector. This process is only used when the string has quotation marks(which is usually the title), the rest of the strings

are just regularly added to the vector. After one line is read, all of the information is initialized to a BookNode object which will be added to the vector. Once all the lines are read, the file is closed and the vector full of BookNode objects will be used to initialize the hashtable and binary tree.

The next stop in creating a fully functioning library management system was creating the data structures that properly sort the data. I first started by creating a hash table class that contains two private variables: a static const int variable which is initialized to 7 and keeps track of the address size, and an array of BookNodes that will hold pointers to each node. The class also has 3 functions: hash(string ISBN), set(BookNode* book), and get(string ISBN). The hash function will receive an ISBN for a book from the user. The string will be looped through in order to receive the key for the Book. Each character at 'i' will have its ASCII value calculated, and when it is officially calculated, it will be added to the hash value. The function will then return the hash value. This function is called upon in the set function. The set function will receive a book object from the main class. The hash function is called upon and the value that is returned will be the index address for the book object in the hash table. If there are no nodes at the given index, then the node will be added to it. If there are nodes at the index, then a linked list will be created at the index in which the new node is added to the end of the list. The get function will receive an ISBN from the user and the hash function will be called upon to find the index of the key. The location on the hash table at the given index will be found and will be looped through until an ISBN match is found. If it is found, the book object will be returned. If not, a nullptr will be returned.

After creating the hash table class, I moved on to the binary tree class where I declared one private variable that keeps track of the root of the data structure, and I created 2 public

functions: insert(BookNode* book) and binarySearch(string title). The insert function will receive a book node object from the user. The function will first check if the root is null. If it is, then the new book object will become the root. If not, then a temporary book object will be initialized to the root. A while loop will iterate through the binary tree and can only be broken with the 'return' statement. In order to add the given book object to the binary tree, a '.compare()' function is needed to compare the titles within the tree alphabetically. The function will give an integer as the result of the comparison. If the result is 0, then the temporary object's title and the given book object's title are equal. This will result in a false boolean being returned since no duplicates are allowed in the binary tree. If the result is less than zero, the new book object's title comes BEFORE the temporary node's title in the alphabet. It will be checked within this conditional statement to see if the temporary node's left node is empty. If it is, then the new book object's node will be inserted and it will return true. If it isn't, then the next spot on the left will be moved and the iteration through the tree will continue. If the result is greater than zero, then that means the new book object's title comes AFTER the temporary node's title in the alphabet. It will be checked within this conditional statement to see if the temporary node's right node is empty. If it is, then the new book object's node will be inserted and it will return true. If it isn't, then the next spot on the right will be moved and the iteration through the tree will continue. The binary search function implements similar features to the binary search algorithm. The function is given a key to search for within the tree, in this case, it is given a title of a book to search for. A temporary book node object is created and starts at the root value of the tree. Similar to the insert function, the .compare() function will compare the given title to the title of the temporary object. If the result is less than zero, then that means the given title is before the temporary node's title, so the temporary node will be initialized to the left node. If the result is

greater than zero, then that means the given title is after the temporary node's title, so the temporary node will be initialized to the right node. When the titles are equal, the quantity of the node will first be checked. If it is zero or less, then a nullptr will be returned, but if it is not then the book will be returned. If the title is not found within the binary tree, then a nullptr will be returned. In the main class, a hash table and binary tree are created as objects and the set and insert functions are called in order to add the books to both data structures.

After I created the binary tree and hash table classes, I then created the queues and stacks classes. The queue data structure is used to store reserved books. A vector is used to implement the queue since it is much simpler than using a linked list. I created two functions, the first was the reserveBook(BookNode* book) function. A book node is passed to the function to reserve it. The quantity of the book will decrease, and the book will be added to the beginning of the vector. The second function was the linearSearchReturnBook(BookNode* book) function. The linear search function implements similar features to the linear search algorithm. A book node is passed to the function to return the book. The vector will be looped through, and it will check if the value at a vector matches the passed book node. If it does, then the node will be removed from the vector and its quantity will be increased. The stack data structure is used to keep track of the most recently borrowed books at the library. I used a vector to implement the stack in this case since it was much quicker than creating a linked list. I created two functions, the first was addBorrowedBook(BookNode* book). A book node is passed to the function and is added to the very end of the vector. The second function, returnLastBook(), basically takes the last book from the stack(which is the most recently borrowed book) and prints it to the user.

After implementing all of the needed data structures for the library management system, I then asked the user to reserve 10 books. The user has the choice to type the ISBN of the book or

the title. When the user types their input, the binary tree will first be searched to see if the user's input matches the title of any of the books. If a valid title is received then the loop will be broken out of, and the user can type the next book. If an invalid title is received, then either the given input was an ISBN or not available. The hash table will then be searched to see if the user's input matches the ISBN of any of the books. If a valid ISBN is received then the loop will be broken out of, and the user can type the next book. If an invalid ISBN is received, then that means the user either typed a book that was not available in the library OR they typed a book that is currently out of stock at the moment.

```
*Reserving 10 books*

Please type the ISBN or the title for the book you are looking for.
The Sea of Tranquility
1981625:
         Title: The Sea of Tranquility
         Author: Mark Haddon
         Price: 91.99
         Quantity: 5

Please type the ISBN or the title for the book you are looking for.
1961721
1961721:
         Title: PADDINGTON GOES TO SALES L/CUB (Collins Colour Cubs)
         Author: Michael Bond
         Price: 28.99
         Quantity: 7

Please type the ISBN or the title for the book you are looking for.
Friend Monkey
1952404:
         Title: Friend Monkey
         Author: P. L Travers
         Price: 8.99
         Quantity: 0

Please type the ISBN or the title for the book you are looking for.
1952404

You typed a book that is NOT available in our library OR a book that is out of stock at the moment.
Please try again...
```

When an invalid input is typed the user can retype a new input. After the 10 books are reserved, then the user must type the ISBN or title of a book they wish to receive 2 copies of. If the user tries to choose a book that has less than two copies or is unavailable, then they will have to pick a different book. When reserved, the next availability for the book is then printed to the user.

```
*Reserving 2 of the same books*

Please type the ISBN or the title for the book you are looking for.
2005395
2005395:
        Title: Deafening
        Author: Frances Itani
        Price: 35.99
        Quantity: 2
2005395:
        Title: Deafening
        Author: Frances Itani
        Price: 35.99
        Quantity: 1
*Next availability for 'Deafening': 1 books left.*
```

Then the stack will return our most recently borrowed book, which in this case is “Deafening”.

```
Last Book Reserved:
2005395:
        Title: Deafening
        Author: Frances Itani
        Price: 35.99
        Quantity: 1
```

One copy of the book will then be returned, and then the same book is reserved again.

```
*Returning book*
2005395:
        Title: Deafening
        Author: Frances Itani
        Price: 35.99
        Quantity: 2


*Reserving same book*
2005395:
        Title: Deafening
        Author: Frances Itani
        Price: 35.99
        Quantity: 1
```

References

C++ Data Structures & Algorithms | Udemy. (n.d.).

https://www.udemy.com/course/data-structures-algorithms-cpp/

GeeksforGeeks. (2024a, July 5). *Introduction to linear search algorithm*.

https://www.geeksforgeeks.org/linear-search/

GeeksforGeeks. (2024, September 25). *Searching in binary search tree (BST)*.

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

OpenAI. (2024). *ChatGPT* (Sep 30 version) [Large language model].

https://chat.openai.com