Group Members: Alec Lempert (113316032), Dylan Li (114228567), Brian Fong (113319367)

Final Project report

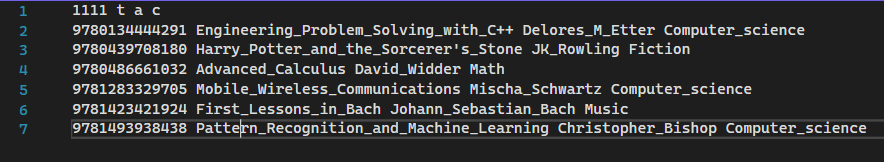
System: Windows

Compiler: Visual Studio 2022

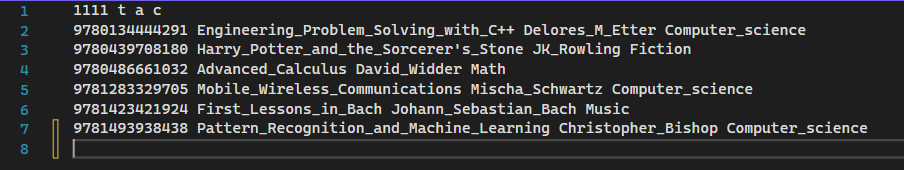
Description: Recreating how a Library works in a c++ program, such as borrowing books, searching books, and returning books. The program asks for authentication via a username and password before the user can proceed to access the Library. Further added functions include reservations for books, a new librarian class, and all data is now holded by binary trees.

\*Note: Make sure text files have no extra lines or spaces at the end, it may mess up the program

Good



Bad



Major Contributions:

Dylan - Reader functions, sorting and search functions, accessors and mutators for each class, debugging, main\_key search implementation, virtual function implementation

Alec - Binary tree creation, file reading/writing, Librarian class and functions, overloaded operators, template functions in main and header, debugging

Brian - User interface and authentication/main, Librarian functions, print functions, debugging

**How it works:**

1. **Initialization**

All information on books, copies, and all users are stored in 6 binary trees: Book, Copy, Student, Teacher, Librarian, User. The readFiles() function located in main.cpp opens and reads the text files and stores it in each node of their respective binary tree. The book file is first read and stored in the respective tree first followed by the copy file and user file. Data from the file is stored by utilizing the addElement() function, located in library.h, which uses recursion to traverse the binary tree until an empty node is found.The data of each tree can be accessed by using the printtree() function with the tree we want to see.

1. **User Authentication**

User authentication works by first requesting a user input for the username and another input for the password. Once a user has entered the username and password, by traversing the student, teacher, and librarian trees to see if a user and password match. The search(TreeNode<classType>\* head, searchType search\_key) function works by taking in the root of a tree and the user input value and uses recursively calls itself to find the search\_key input.

1. **Binary Trees**

There are 6 binary trees in this program: one for Books, Copies. Students, Teachers, Librarians, and Users (which is composed of students, teachers, librarians).

1. **ReadFiles()**

When this function is called, the text files are opened and the binary trees are filled in via the addElement() function. The global variable *last\_id* is incremented each time a new Copy is added to the Copy tree. At the end, removeElement() is called for each tree because readFiles() makes the heads of the trees NULL, so I want to remove that.

1. **addElement() and addElementUsers()**

addElement() is used to add an element to any tree except to the users tree. This is because these functions work by taking in two parameters of an abstract type (via templates): a value (this value is an object) and a node from one of the trees. The Users tree, however, has objects of type Student, Teacher, and Librarian inside it, which caused difficulties within the function during the comparison operators. That’s why the addElementUsers() function exists for the Users tree. Essentially, what these functions are doing is comparing the two main keys of the two parameters through operators ‘<’ and ‘>’ and traversing down the corresponding tree via recursion depending on the result. It returns nodes in ascending order, and sorts the trees each time.

* **Bool User::operator > (User& right) and bool operator <(User& right)**

These operators are located in the user class. Similar operators are present in Book.cpp and Copy.cpp. They work by comparing the main key of some lefthand object with the main key of the righthand object. The ‘<’ operator returns true if the lefthand is smaller. The ‘>’ returns true if the lefthand is bigger. The “main\_key” is a general term used to refer to the variable by which these trees are sorted/accessed. ISBN for books, ID for copies, username for users.

1. **search()**

Takes in a node and an object as parameters. If the object entered is the same as the object at the node, returns the node. If the object entered is smaller than the object at the node (compare the main keys), traverse to the left child and recur. If larger, traverse to the right child and recur.

1. **removeElement() and removeElementUsers()**

Takes in a node and an object as parameters. Searches for the object in the tree just like in the search() function. If the object is in the tree, the node is deleted via the *delete* key-word.

1. **printValueAtNode() and printTree()**

Takes in a node as parameters. printValueAtNode does an inorder traversal to print out the main key of every node of the corresponding tree. printTree checks what type its parameter was to let the user know what tree it's about to print, and then prints that tree.

1. **The Book Class**

The Book class represents a set of book copies with the same ISBN, title, author, and category. The only differences between the copies of the Book class are their IDs which the Book class stores in *list<int> copyList.* The Book class also contains a pointer to a linked list that will represent the list of reservers and can be traversed using the mutator and accessor functions. The last variable is *favor\_count* which tracks how much the Book class is liked throughout all the copies, but we don’t use it for any other purpose as we didn’t do the Bonus “I’m Feeling Lucky” function. The *main\_key* used for searching is the ISBN. The Book class has no notable functions beyond its accessors, mutators, and operator overloads.

1. **The Copy Class**

The Copy class represents a copy of a Book class with the IDs being distinct for each instance of Copy. The Copy class contains the variables to take in a reader if it is currently being borrowed and the variables to take in a reserver if it is currently being reserved. The *main\_key* used for searching is the ID. The Copy class has no notable functions beyond its accessors, mutators, and operator overloads.

1. **The User Base Class and Derived Classes**
   1. **User Class**

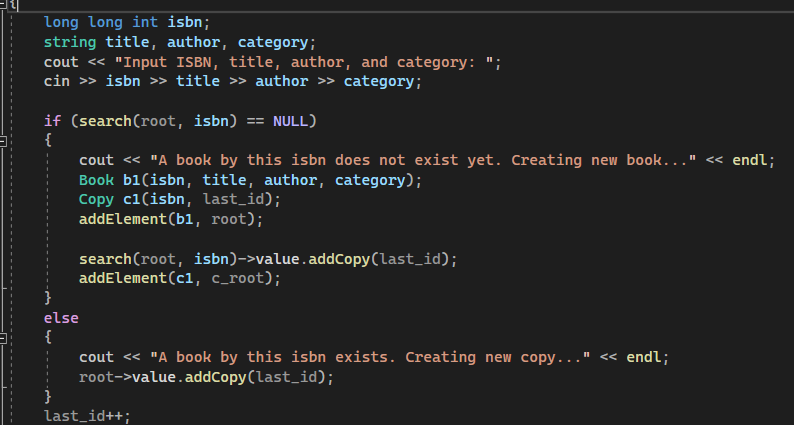
The User class serves as the base class for all the other classes(Librarian and Reader). The variables for the User class are the username, password, and role. For the role, a 0 represents an instance of Student class, a 1 represents an instance of Teacher class, and a 2 represents an instance of Librarian class. Besides the accessor and mutator functions for each of these variables, the User class also has a virtual print function that prints differently depending whether or not the class calling the function is a Librarian or a Reader(Student/Teacher class) class. The username and password is only printed out for the Librarian, while the Reader class prints out the username, password, list of books they borrowed, and whether or not they are a Student or Teacher depending on their role. Each of the derived classes have overloaded operators “<<” and “>>” that take in or print out the information of the class. The *main\_key* used for searching is the username.

* 1. **Librarian Class**

The Librarian class is derived from the User class and can do actions, such as adding or deleting a Book and searching for other users, that the Reader class can not do.

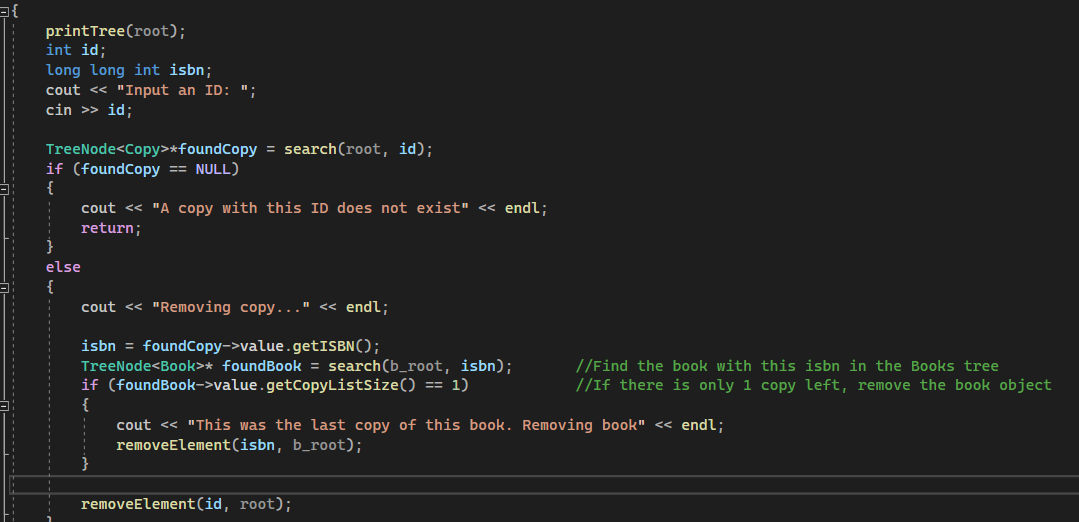
-void addBooks(TreeNode<Book>\* root, TreeNode<Copy>\* c\_root, int& last\_id)

This function adds a book copy by asking the Librarian for the ISBN, title, author, and category of the new copy. The function also requires a reference to a parameter *last\_id* to know which ID the new Copy should have. If the inputted ISBN matches with an existing instance of Book class, then the Copy gets added to that Book class’s copyList. Otherwise, a new instance of Book class is created for that new Copy.



-void deleteBooks(TreeNode<Copy>\* root, TreeNode<Book>\* b\_root)

This function asks for a book ID and deletes the copy associated with that ID(if there is one). The function also checks if the Copy instance being deleted is the last copy of that Book class through the size of the copyList. If it is, then the Book class also gets deleted.



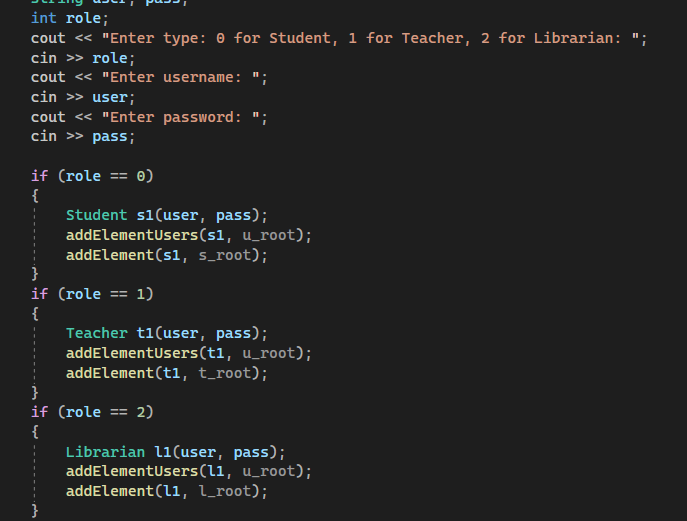
-void searchUsers(TreeNode<User>\* u\_root, TreeNode<Student>\* s\_root, TreeNode<Teacher>\* t\_root, TreeNode<Librarian>\* l\_root, TreeNode<Book>\* b\_root, TreeNode<Copy>\* c\_root)

This function searches for Users through the imputed username. If the matching username is a Reader, then the function prints out the username, password, the information of the copies they borrowed, and whether they are a Student or a Teacher. If it is a Librarian, then the username and password are only printed out.



-void addUsers(TreeNode<User>\* u\_root, TreeNode<Student>\* s\_root, TreeNode<Teacher>\* t\_root, TreeNode<Librarian>\* l\_root)

This function adds a new User by asking for their role, username, and password for the new User. It also checks if the username is different from the other users. Otherwise, the function does nothing.



-void deleteUsers(TreeNode<User>\* u\_root, TreeNode<Student>\* s\_root, TreeNode<Teacher>\* t\_root, TreeNode<Librarian>\* l\_root, TreeNode<Book>\* b\_root)

This function deletes a User after the Librarian enters their username(if that User exists). If the User to be deleted is a Reader, it also checks if they have any borrowed books. If the Reader does have borrowed books, then the function returns since it can’t delete a Reader with borrowed books. If the Reader has no borrowed books, then they are deleted from the binary tree and all their reservations are also removed. The reservations are checked through the Reader’s reserved list’s ISBNs before the Reader gets deleted.



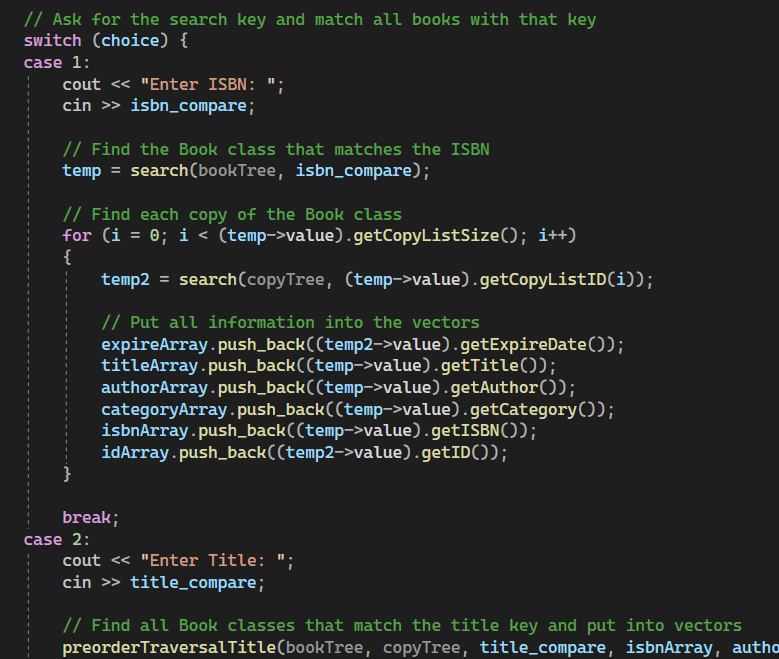
* 1. **Reader Class(Student/Teacher)**

The Reader class contains two derived classes (Student/Teacher), but they share most of the same functions from the Reader base class with only slight differences. The Reader class has functions that allow it to search books, borrow books, return books, and a few other functions.

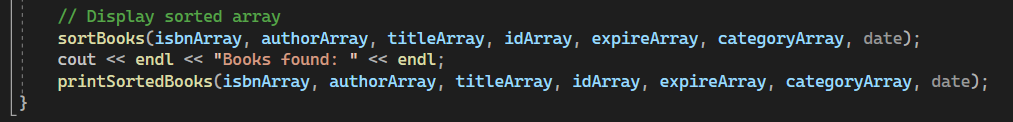
- Variables: The variables for the Reader class (aside the ones from the User class) is *list<int> borrow\_list* , which stores the IDs of each Copy that the Reader has borrowed and *list<long long int> reserved\_list* , which stores the ISBNs of each Book class the the Reader has reserved. These variables make it possible to look for the information of the Copy that the reader has borrowed through the ID or look for the information of the Book that the reader has reserved through the ISBN.

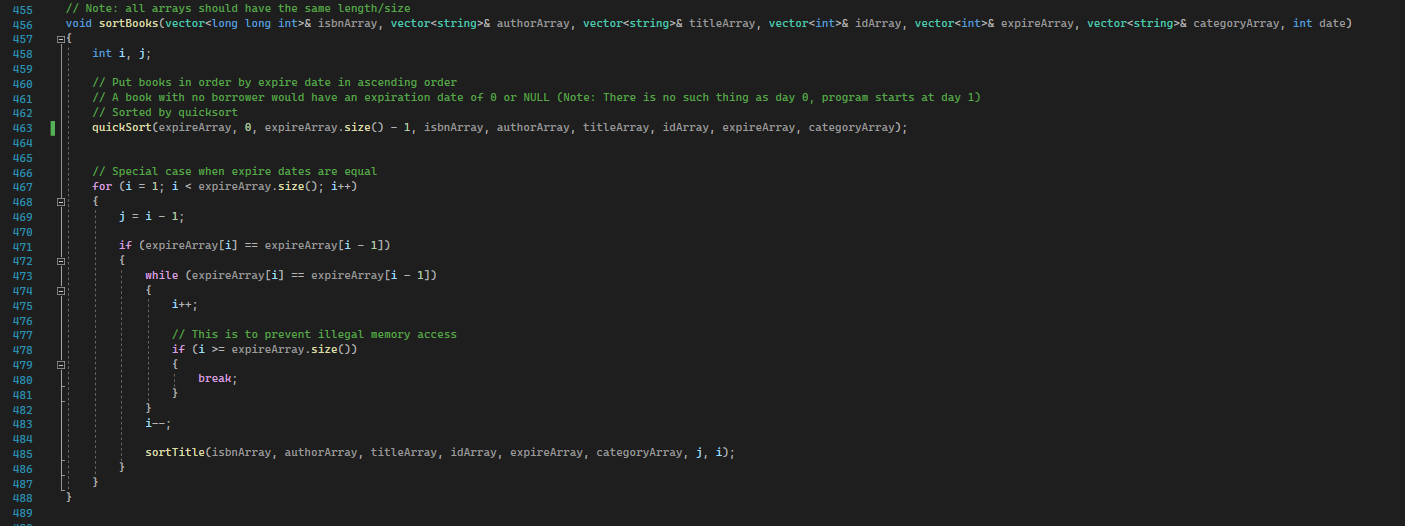
- void Reader::searchBook(TreeNode<Book>\* bookTree, TreeNode<Copy>\* copyTree, int date)

This function first asks for a search key to search Books with. After the user inputs a search key, the program finds all Books that match with the search key and puts all of the info from the Book, including the Copies, into vectors that store an attribute of the Book/Copy that will later be used for sorting, such as expiration date and title.



After putting all information needed from the matching Books into the vectors, the vectors are sorted using quicksort in priority order from the expireArray, which contains each copy’s expiration date, to the idArray, which contains the unique IDs of each copy. During quicksort, the changes in one vector are also reflected on the other vectors to keep the information of each copy match so that they can be printed in ascending order. After each quicksort, the vector that was sorted will check for elements that have the same value. For example, if two elements in titleArray have the same title “Harry\_Potter”, then it limits the range of elements needed to be sorted to the range of elements that are equal to each other and passes the vectors to another function that will sort by author instead. After the sorting is done on all the vectors, the final result should print all the vectors in ascending elemental order.

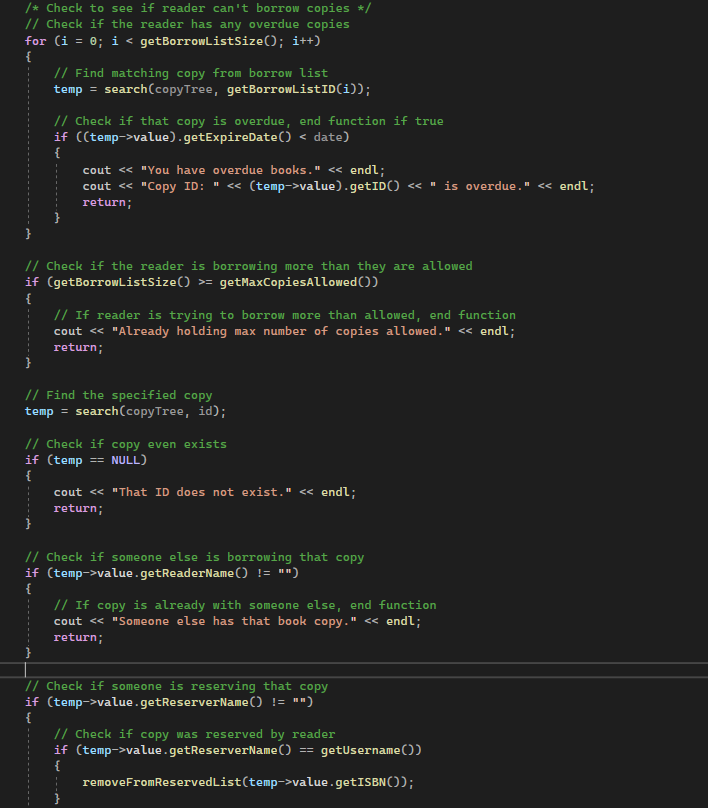




-void Reader::borrowBook(TreeNode<Copy>\* copyTree, int id, int date)

This function allows the Reader to borrow a book copy by ID. Before it lets the Reader borrow a book, the function runs through a series of checks to make sure the Reader is eligible to borrow a book copy. The checks are:

1. Does the ID that the Reader wants to borrow exist?
2. Does the Reader have any overdue copies?
3. Is the Reader trying to borrow more than they are allowed to borrow?
4. Is someone else borrowing that copy right now?
5. Is someone else reserving that copy right now?



After each of these checks have passed, the Reader will be allowed to borrow the book copy by ID. The borrower name of the Copy will become the reader’s name, the borrow start date will be the current date, and the expiration date will be determined by whether the Reader is a Student or a Teacher(30 days for Students, 50 days for Teachers). In the end, a successful message should be displayed.

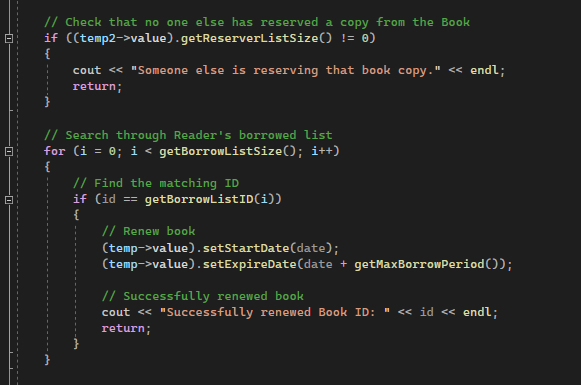
-void Reader::returnBook(TreeNode<Book>\* bookTree, TreeNode<Copy>\* copyTree, int id, int date)

This function allows the Reader to return a book copy that they are currently borrowing to make available for other Readers. The ID is removed from the Reader’s borrowed book list as well as the Reader’s name being removed from the Copy. If there is a reserver, then the Copy that was returned sets the reserver’s name as that reserver’s username. That way, the Copy is now “reserved”. In the end, the program also asks the Reader if they liked the Book and increments the favor count if they say yes.



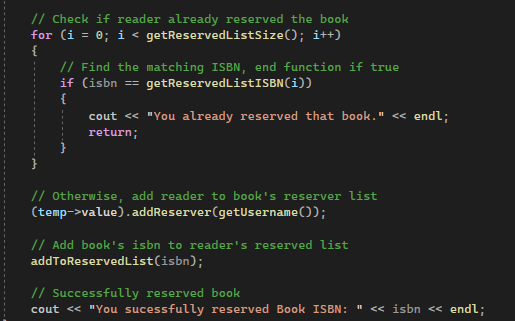
-void Reader::renewBook(TreeNode<Book>\* bookTree, TreeNode<Copy>\* copyTree, int id, int date)

This function renews a book copy that the Reader is currently borrowing as long as there is no reserver in the related Book class’s reserver linked list. If there is a reserver for a copy of that Book class, then the Reader can not renew the book. If there is no reserver, then the Copy’s start date is set as the current date and the expiration date is changed.



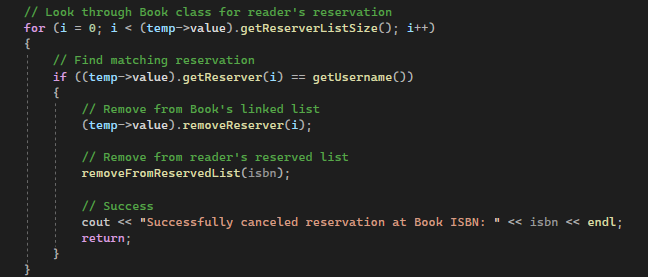
-void Reader::reserveBook(TreeNode<Book>\* bookTree, long long int isbn)

If there are no copies of a Book available to borrow, then the Reader can reserve the Book. The Reader gets added to the Book’s reserver list and is removed from the list once the reserver node reaches the head of the linked list and a copy of that Book is returned.



-void Reader::cancelReservation(TreeNode<Book>\* bookTree, long long int isbn)

If the Reader wishes to cancel a reservation(if there is one), then they can enter the ISBN of the Book they reserved which will remove them from that Book’s reserver linked list. The Book’s ISBN is also removed from the Reader’s list of books they reserved.

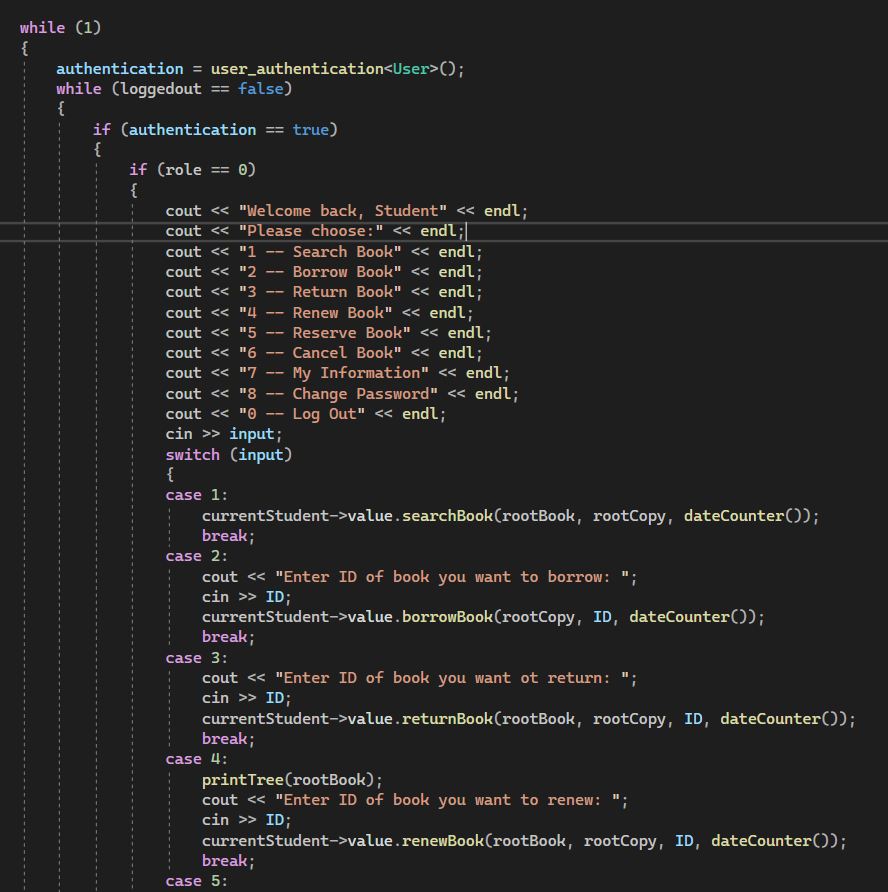


-int getMaxCopiesAllowed() and int getMaxBorrowPeriod()

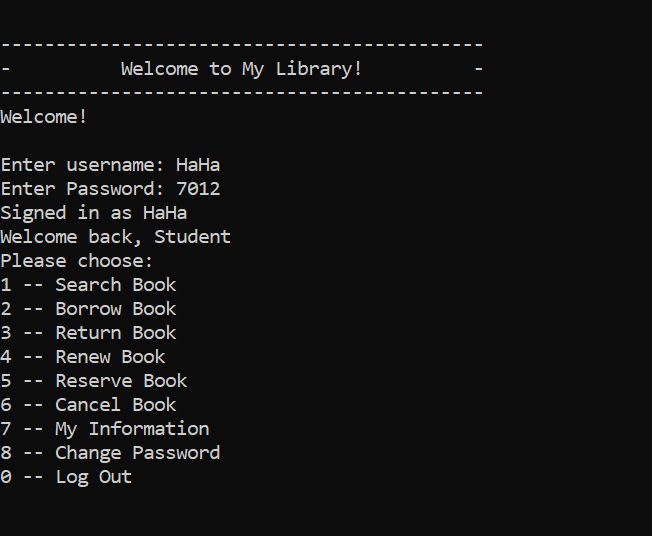
These are the only functions that differ between the Student and Teacher class. The Student class returns 5 for getMaxCopiesAllowed() and 30 for getMaxBorrowPeriod(). The Teacher class returns 10 for getMaxCopiesAllowed() and 50 for getMaxBorrowPeriod().

**7) Main**

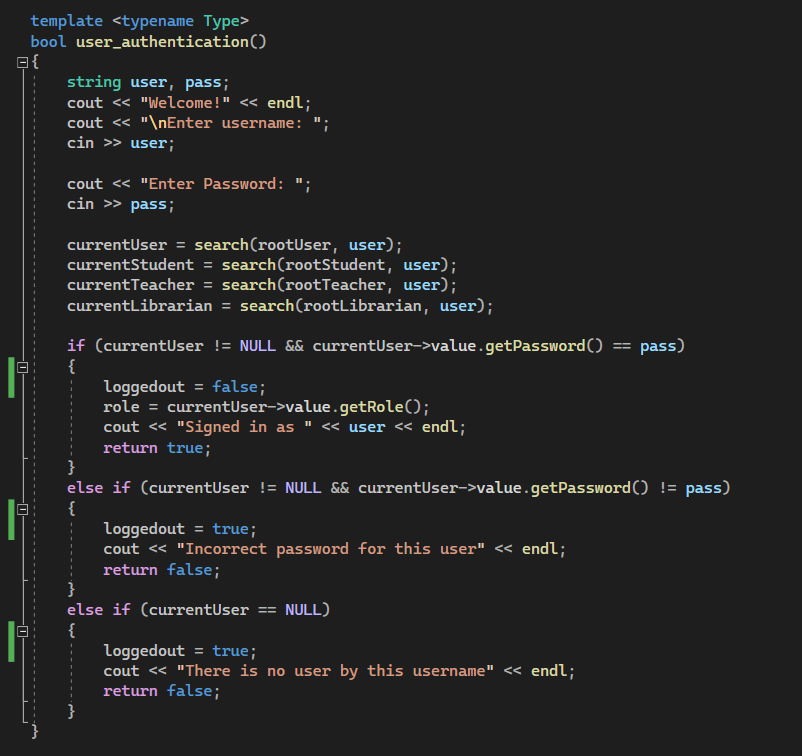
The main function is the frontend aspect of the code that the user interacts with. It first starts out by printing all the binary trees. After printing the data, the program asks for two user inputs: one of the username and one of the password.



At this point, the program enters an endless while loop that will only exit when the user closes the console. After receiving both inputs, it checks if the input matches with a user in the tree. If no user exits, the program outputs “There is no user by this username”. If the user exists but the password is incorrect, the program outputs “Incorrect password for this user”. If both the username and password are correct, the program logs the user in to their respective user interface; either Student, Teacher, or Librarian, and displays their available commands.



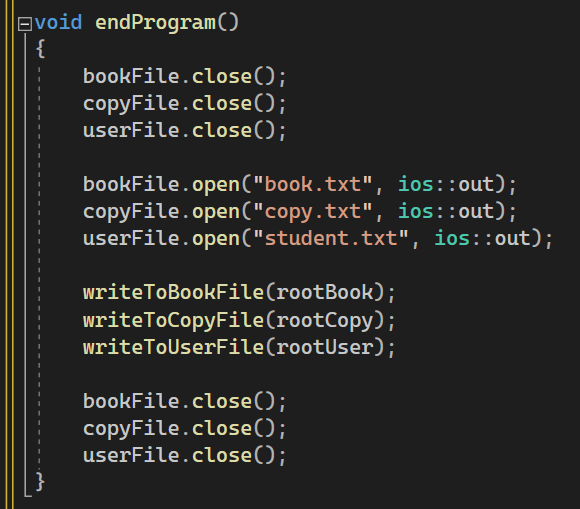
Students, for example, have the ability to search, borrow, return, renew, reserve, cancel reserve books , find their information, change password, or log out. Users can execute the available commands by entering the number of the command they want to execute which would then be executed through a switch statement. For most of the commands, the switch case would call the respective function that reads or alters data from the binary tree. The entire login section of the code is inside another while loop which only occur when the variable “loggedout” is false. The variable is true by default and when the use successfully logs in, function user\_authentication turns it false.



When the user logs out, the switch case for “logout” turns the loggedout variable true and ends the loop. When the endless while loop loops again, user\_authentication() will be called again, asking the user to input a username and password again, completing the loop.

**8) Ending the program**

When a librarian has logged in, they have the option to terminate the program. When the librarian terminates the program, endProgram() is called.



This function closes all the files and then reopens them in write mode, but also empties them via ios::out. Then, 3 functions writeToBookFile(), writeToCopyFile(), writeToUserFile() are called. These functions take in a node as a parameter and perform an inorder traversal of the corresponding tree. They write the contents of the object at the node into the corresponding file at every iteration. (NOTE: The contents do successfully get written onto the file, but with an unnecessary blank line at the end of them, caused by an *endl.* As the program is only realistically going to be run one time, this issue was not resolved. That extra line causes an issue when the readFiles() function is called if one chooses to run the program a second time. Remove the blank line manually from the files to prevent this).