# COP5536 – Advanced Data Structures

## FALL 2023

## Programming Assignment Report

**Name : Triveni Gudidoddi**
**Email : tgudidoddi@ufl.edu**
**UFID : 61375015**

**Programming Language:** C++
**Running the program:**

**Compile :** make
**Execute :** ./gatorLibrary <input_file>
Here, <input_file> is either the name of the file or the path to file.
**Input files:** test1.txt, test2.txt,test3.txt and test4.txt
**Output :** output of the program is written to <inputFileName>_output_file.txt example (test2_output_file.txt)

**Program Structure:**

The gatorLibrary.cpp program has the below classes and functions:

The program has gatorLibrary class and 3 node structures namely RBTreeNode, Book and ReservationHeap.

- The structure of RBTreeNode is as follows:

```
struct RBTreeNode {
    Book book;
    int color; // 1  for red, 0 for black
    int index;
    RBTreeNode * left;
    RBTreeNode * right;
    RBTreeNode * parent;
    RBTreeNode * linkNodePtr;
};
```

- The structure of Book is as follows:

```
struct Book {
    int bookId;
    int borrowedBy;
    string bookName;
    string authorName;
    int availabilityStatus;

    vector<ReservationNode> reservationHeap;
};
```

- The structure of ReservationHeap is as follows:

```
struct ReservationNode {
    int patronId;
    int priorityNumber;
    long long timeOfReservation;

    ReservationNode(int id, int priority, long long timeStamp) : patronId(id), priorityNumber(priority),
    timeOfReservation(timeStamp) {};
};
```

- The constructor of the class gatorLibrary is as follows:

```
public:
    //constructor
    gatorLibrary() : root(nullptr), flipCount(0) {
        // Create a new node representing NULL in the Red-Black Tree
        NULLNODE = new RBTreeNode;
        NULLNODE->book.bookId = 0;
        NULLNODE->book.bookName = "";
        NULLNODE->book.authorName = "";
        NULLNODE->book.availabilityStatus = false;
        NULLNODE->book.borrowedBy = -1;
        NULLNODE->color = 0; // Assuming black as the default color
        NULLNODE->left = nullptr;
        NULLNODE->right = nullptr;
        NULLNODE->parent = nullptr;

        root = NULLNODE;

        // Open a file for writing output
        outputFile.open(outputFileName);
    }
```

All methods of this class and their definitions is as follows:

1. **void printBook(bookID):** It takes bookID and searches for that bookID and prints the details of that bookID if present: it uses printBookHelper function to print the output in below format.

   BookID: <bookID>
   Title: <bookName>
   Author: <authorName>
   Availability: <availabilityStatus>
   BorrowedBy: <borrowedBy>
   Reservations: [reservations for the book]

   else : prints “Book <bookID> not found in the library”

   **Time and Space complexity:**

   **Time complexity:** In the printBook function we will be searching for the given bookID in a red-black tree which takes O(logN), where N is the number of nodes in the tree. And when printing the details of the book we need to loop the reservationHeap and print them which takes O(R), R denotes number of reservations for that book.

   **T.C = O(logN) + O(R)**

   **Space complexity:** As there is no additional data structures or dynamic memory allocation in this function, the space complexity is **O(1).**

2. **void printBooks(bookID1, bookID2):** it takes two bookIDs i.e., bookID1 and bookID2. I have used a loop starting from bookID1 and increment till we reach bookID2 and for each bookID I am searching that book in the Red-black tree. If that bookID node is present in the tree it prints the details of that book using the printBookHelper function or else does not print anything. In this way we can get the information of the books from bookID1 to bookID2 inclusively.

   **Time and Space complexity:**

   **Time complexity:** For searching the book in red-black tree it takes O(logN) time and we are searching books from bookID1 to bookID2, let (bookID2-bookID1 +1) be K. Then the overall time complexity is K*O(logN) + O(R), where R is the number of reservations.

   **T.C = K*O(logN) + O(R), where K = (bookID2 - bookID1 +1)**

   **Space complexity:** space complexity is **O(1),** as there is no additional space or any dynamic memory allocated in this function.

Functions in the code:

```
void printBookHelper(RBTreeNode* node) {
    outputFile << "BookID: " << node->book.bookId << endl;
    outputFile << "Title: " << node->book.bookName << endl;
    outputFile << "Author: " << node->book.authorName << endl;
    outputFile << "Availability: " << (node->book.availabilityStatus ? "Yes" : "No") << endl;
    outputFile << "BorrowedBy: " << (node->book.borrowedBy == -1 ? "None" : std::to_string(node->book.borrowedBy)) << endl;
    // Print reservation details
    outputFile << "Reservations: [";
    for (auto it = node->book.reservationHeap.begin(); it != node->book.reservationHeap.end(); ++it) {
        outputFile << it->patronId;
        if (std::next(it) != node->book.reservationHeap.end()) {
            outputFile << ", ";
        }
    }
    outputFile << "]" << endl << endl;
}
void printBook(int bookID) {
    RBTreeNode* node = searchRBTree(bookID);
    if (node) {
        printBookHelper(node);
    } else {
        outputFile << "Book " << bookID << " not found in the Library" <<endl<<endl;
        return;
    }
}
void printBooks(int bookID1, int bookID2) {
   for(int i=bookID1;i<=bookID2;i++) {
        RBTreeNode* node = searchRBTree(i);
        if(node){
            printBookHelper(node);
        }
   }
}
```

RBTreeNode *node

3. **void insertBook(bookID, bookName, authorName, availabilityStatus, borrowedBy, ReservationHeap):** This function takes bookID, bookName, authorName, availabilityStatus, borrowedBy and reservations as input parameters. It will insert the node into red-black tree.

   Insert operation into Red black tree as follows:
   **Insert Node:**Insert the node as in a standard binary search tree (BST) and mark it red.
   **Maintain Properties:** Ensure no consecutive red nodes exist (violation of property). Adjust tree structure and node colours if necessary to restore balance using insertRBTreeNodeFix function.
   **Balancing Operations:** Perform rotations and colour changes to preserve the Red-Black Tree properties.Rebalance the tree to satisfy the rules of a Red-Black Tree using rbTreeLeftRotate and rbTreeRightRotate.

   **<u>Time and space complexity:</u>**
   **Time complexity:** Inserting a node involves traversing the tree to find the appropriate position, which takes O(log N) time in the worst case for a balanced tree.

   **Space complexity: O(N)**

**Code for insert operation:**

```
void insertBook(int bookID, const string& bookName, const string& authorName, bool availabilityStatus, int borrowedBy,
    if (searchRBTree(bookID) != nullptr) {
        outputFile << "Error: Duplicate Book ID " << bookID << ". Book not inserted." << endl << endl; // Book with the
        return;
    }
    RBTreeNode* newNode = new RBTreeNode;
    newNode->book.bookId = bookID;
    newNode->book.bookName = bookName;
    newNode->book.authorName = authorName;
    newNode->book.availabilityStatus = availabilityStatus;
    newNode->book.borrowedBy = borrowedBy;
    newNode->book.reservationHeap = reservationHeap;
    newNode->left = NULLNODE;
    newNode->right = NULLNODE;
    newNode->color = 1;
    if (root == NULLNODE) { // If the root is NULLNODE (initially empty tree), set the new node as the root
        root = newNode;
        newNode->color = 0; // Set the color of the root node to black
        return;
    }
    RBTreeNode* parent = nullptr; // Traverse to find the appropriate position to insert the new node
    RBTreeNode* current = root;
    while (current != NULLNODE) {…
    newNode->parent = parent; // Set the parent for the new node based on the traversal
    if (newNode->book.bookId < parent->book.bookId) { // Assign the new node as the appropriate child of its parent
        parent->left = newNode;
    } else {
        parent->right = newNode;
    }
    insertRBTreeNodeFix(newNode); // Function to fix up the tree (if needed)
}
```

4. **void insertRBTreeNodeFix(RBTreeNode* nodeK):** This is a helper function which fixes the violations that occur while inserting a new node into the red black tree. This function takes O(1) time complexity and O(1) space complexity

5. **void rbTreeRightRotate(RBTreeNode* nodeX):** This is a helper function which is used to rotate the tree from the given node (nodeX) to the right.

6. **void rbTreeLeftRotate(RBTreeNode* nodeX):** This is a helper function which is used to rotate the tree from the given node (nodeX) to the left.

7. **void heapifyUp(vector<ReservationNode>& heap, int index):** This function maintains the heap property by moving an element up the heap until it satisfies the heap condition. This is used to implement the priority queue to store the reservations for a bookID.

   **Time and Space complexities:**
   **Time complexity:** O(log N) where N is the number of elements in the heap.
   **Space complexity:** O(1) as the operation is done in place.

8. **void heapifyUp(vector<ReservationNode>& heap, int index):** Similar to heapifyUp, this function maintains the heap property by moving an element down the heap until it satisfies the heap condition.

   **Time and Space complexities:**
   **Time complexity:** O(log N) where N is the number of elements in the heap.
   **Space complexity:** O(1) as the operation is done in place.

9. **void inOrder(RBTreeNode* node, vector<Book>& listOfBooks):** This function performs an in-order traversal (Left, root, right) of a Red-Black Tree and appends the elements to listOfBooks.

   **Time and Space complexities:**
   **Time complexity:** O(N) where N is the number of elements in the heap.
   **Space complexity:** O(N) due to the space used for recursion stack.

10. **RBTreeNode* minimumRBTreeNode(RBTreeNode* rbTreeNode):** Returns the node with the minimum value/key in a Red-Black Tree. This function is used to find the minimum node from right subtree, which is used as rebalance node.

    **Time and Space complexities:**
    **Time complexity:** O(log N) in a balanced tree.
    **Space complexity:** O(1) as it doesn't use extra space proportional to the input.

    Code for minimumRBTreeNode is

```
RBTreeNode* minimumRBTreeNode(RBTreeNode* rbTreeNode) {
    // Follow the left child pointers until we reach a node with no left child.
    while (rbTreeNode->left != NULLNODE) {
        rbTreeNode = rbTreeNode->left;
    }
    // Return the node with the minimum key.
    return rbTreeNode;
}
```

11. **void deleteBook(bookID):** This function takes bookID as input parameter. It will delete the node from red-black tree.

    delete operation into Red black tree as follows:
    **Delete Node:** Find the node to delete using standard Binary Search Tree (BST) deletion algorithms. Identify the node to replace the deleted node. This can be its successor or predecessor in the BST.

**Adjust Tree Structure:** If the node to delete has fewer than two children, replace it with its child (if any). Otherwise, replace it with its successor (or predecessor).
**Restore Balance:** If the node to delete or replace is black, potentially violating Red-Black Tree properties: Fix any potential issues that might arise due to the deletion to maintain Red-Black Tree properties using deleteRBNodeFix function. Adjust the colours of nodes and perform rotations as necessary to ensure the tree remains balanced.
**Balancing Operations:** Perform rotations and colour changes as required to restore Red-Black Tree properties: Recolor nodes or perform rotations (left or right) to maintain the balance of the tree.

**Time and Space complexities:**
**Time complexity:** Deleting a node involves finding the node to delete and then adjusting the tree structure, which includes potential fixes to maintain the Red-Black Tree properties. In the worst case, deletion and rebalancing take O(log N) time for a balanced tree, similar to insertion.

**Space complexity:** O(1) as it doesn't grow with the size of the tree but rather works on existing nodes.

**Code for deleteBook:**

```
void deleteBook(int bookId) {
    RBTreeNode* node = searchRBTree(bookId);
    vector<ReservationNode> re  std::__1::vector<ReservationNode> Book::reservationHeap  objects
    for (auto it = node->book.reservationHeap.begin(); it != node->book.reservationHeap.end(); ++it) {
        reservations.push_back(*it);
    }

    if (!reservations.empty()) {
        outputFile << "Book " << bookId << " is no longer available. ";

        if (reservations.size() > 1) {
            // Book is reserved by multiple patrons
            outputFile << "Reservations made by Patrons: ";
            for (const auto& reservation : reservations) {
                outputFile <<reservation.patronId << " ";
            }
            outputFile << "have been cancelled!" << endl<< endl;
        } else if (reservations.size() == 1) {
            // Book is reserved by one patron
            outputFile << "Reservation made by Patron " << reservations[0].patronId << " has been cancelled!" << endl
        }
    } else {
        outputFile << "Book " << bookId << " is no longer available. "<<endl<<endl;
    }
    // Remove the book and its reservations
    deleteRBTreeNode(bookId);
}
```

12. **void deleteTreeNodeFix(RBTreeNode* nodeX):** Fixes the Red-Black Tree structure after deletion by performing rotations and colour changes.

**Time and Space complexities:**
**Time complexity:** O(log N) where N is the number of nodes in the tree.

**Space complexity:** O(1) as it doesn't use extra space proportional to the input.

13. **void borrowBook(patronID, bookID, patronPriority):** Handles the process of lending a book to a patron based on their priority. It searches if the requested book is there in the library or not, if found it checks for the availabilityStatus of the book. If the book is available we can change the borrowedBY to this partonID else if the book is not available then we can add this partonID to the reservationHeap.

    **Time and Space complexities:**
    **Time complexity:** O(logN), as function we will be searching for the given bookID in a red-black tree, where N is the number of nodes in the tree.
    **Space complexity:** O(1) as it doesn't use extra space proportional to the input.

    Code for borrowBook function:

```
void borrowBook(int patronID, int bookID, int patronPriority) {
    RBTreeNode* bookNode = searchRBTree(bookID);

    if (bookNode) {
        if (bookNode->book.availabilityStatus) {
            // Book is available, update status
            bookNode->book.availabilityStatus = false;
            bookNode->book.borrowedBy = patronID;
            outputFile << "Book " << bookID << " Borrowed by Patron " << patronID << endl << endl;
        } else {
            // Book is not available, add to reservation heap
            bookNode->book.reservationHeap.emplace_back(patronID, patronPriority, getCurrentTimestamp());
            // Assuming you have a function to heapify up in the reservation heap
            heapifyUp(bookNode->book.reservationHeap, bookNode->book.reservationHeap.size() - 1);
            outputFile << "Book " << bookID << " Reserved by Patron " << patronID << endl << endl;
        }
    } else {
        outputFile << "Book " << bookID << " not found in the Library" << endl << endl;
    }
}
```

14. **void returnBook(patronID, bookID):** Manages the return of a book by a patron.

    **Time and Space complexities:**
    **Time complexity:** O(logN), as function we will be searching for the given bookID in a red-black tree, where N is the number of nodes in the tree.
    **Space complexity:** O(1) as it doesn't use extra space proportional to the input.

    Code for returnBook function:

```
void returnBook(int patronID, int bookID) {
    RBTreeNode* bookNode = searchRBTree(bookID);

    if (bookNode) {
        if(bookNode->book.borrowedBy == patronID) {
            bookNode->book.availabilityStatus = true;
            bookNode->book.borrowedBy = -1; // Assuming -1 represents no patron
            outputFile << "Book " << bookID << " Returned by Patron " << patronID << endl << endl;
            if (!bookNode->book.reservationHeap.empty()) {
                // Book has reservations, assign to the highest priority patron
                auto highestPriorityReservation = bookNode->book.reservationHeap.front();
                bookNode->book.reservationHeap.erase(bookNode->book.reservationHeap.begin());
                bookNode->book.availabilityStatus = false;
                bookNode->book.borrowedBy = highestPriorityReservation.patronId;

                outputFile << "Book " << bookID << " Allotted to Patron " << highestPriorityReservation.patronId << endl
            }
        } else {
            outputFile << "Patron " << patronID << " did not borrow Book " << bookID << endl << endl;
        }
    } else {
        outputFile << "Book " << bookID << " not found in the Library" << endl << endl;
    }
}
```

15. **void findClosestBook(targetID):** Finds the closest book to a given target, possibly using a data structure like a tree or graph.

**<u>Time and Space complexities:</u>**
**Time complexity:**
**Space complexity:**

Code for findClosestBook:

```
RBTreeNode* findClosestRBTreeNode(int targetID) {
    RBTreeNode* closest = nullptr;
    int minDiff = INT_MAX;

    while (root != nullptr) {
        int diff = abs(root->book.bookId - targetID);

        if (diff < minDiff) {
            minDiff = diff;
            closest = root;
        }

        if (root->book.bookId == targetID) {
            // Found an exact match, no need to search further
            return root;
        } else if (root->book.bookId < targetID) {
            root = root->right;
        } else {
            root = root->left;
        }
    }

    return closest;
}
```

16. **void colorFlipCount():** Calculates or maintains the count of colour flips, potentially related to an algorithm or specific data structure. Even though the code is printing the flipCount directly, the flipCOunt is incremented in insertion and deletion operations whenever there is a colour change from red to green and vice versa.

    **Time and Space complexities:**
    **Time complexity:** O(1), as it just returns the count.
    **Space complexity:** O(1) as it doesn't use extra space.

```
void getColorFlipCount() {
    outputFile << "Color Flip Count: " << flipCount << endl << endl;
    return;
}
```

17. **Quit():** This is not a function; rather it is implemented when we encounter this in the input file; we break from reading the input file and close the inputFile and exit the program with success status code.

```
// If the line contains "Insert", extract the ride information and call Insert function
if(eachLine.find("Quit()") != string::npos) {
    outputFile << "Program Terminated!!" << endl << endl;
    break;
}
```

Output against test cases provided taken from thunder.cise.ufl server:

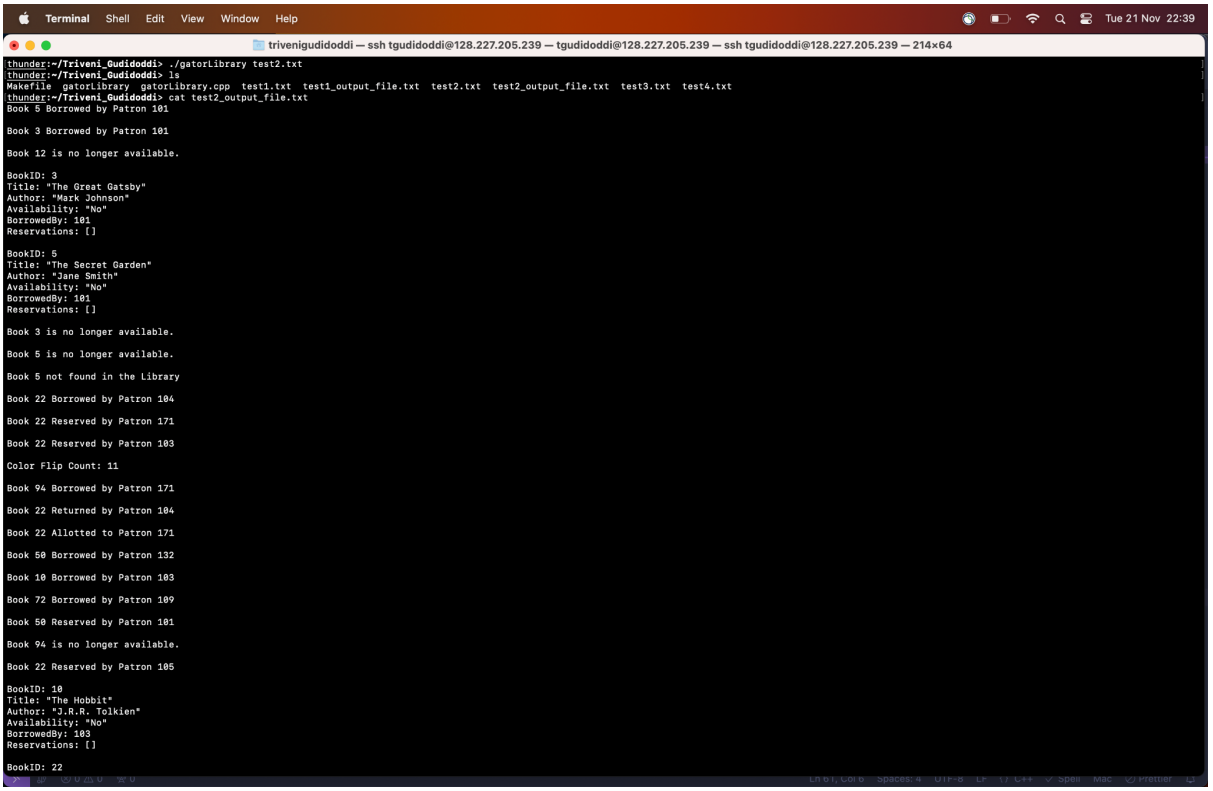
**Test1:** (test1_output_file.txt)

```
thunder:~> cd Triveni_Gudidoddi
thunder:~/Triveni_Gudidoddi> make
g++ -std=c++11 -Wall -o gatorLibrary gatorLibrary.cpp
thunder:~/Triveni_Gudidoddi> ./gatorLibrary test1.txt
thunder:~/Triveni_Gudidoddi> ls
Makefile  gatorLibrary  gatorLibrary.cpp  test1.txt  test1_output_file.txt  test2.txt  test3.txt  test4.txt
thunder:~/Triveni_Gudidoddi> cat test1_output_file.txt
BookID: 48
Title: "Data Structures and Algorithms"
Author: "Sartaj Sahni"
Availability: "Yes"
BorrowedBy: None
Reservations: []

Book 48 Borrowed by Patron 120

Book 101 Borrowed by Patron 132

Book 48 Reserved by Patron 144

Book 48 Reserved by Patron 140

Book 48 Reserved by Patron 142

Book 12 Borrowed by Patron 138

Book 12 Reserved by Patron 150

Book 12 Reserved by Patron 162

Book 48 Returned by Patron 120

Book 48 Allotted to Patron 142

BookID: 6
Title: "Database Management Systems"
Author: "Raghu Ramakrishnan"
Availability: "Yes"
BorrowedBy: None
Reservations: []

BookID: 12
Title: "Artificial Intelligence: A Modern Approach"
Author: "Stuart Russell"
Availability: "No"
BorrowedBy: 138
Reservations: [162, 150]

Book 12 is no longer available. Reservations made by Patrons: 162 150 have been cancelled!

Color Flip Count: 10

Book 73 Borrowed by Patron 111

Book 73 Reserved by Patron 52

Book 25 Borrowed by Patron 153
```

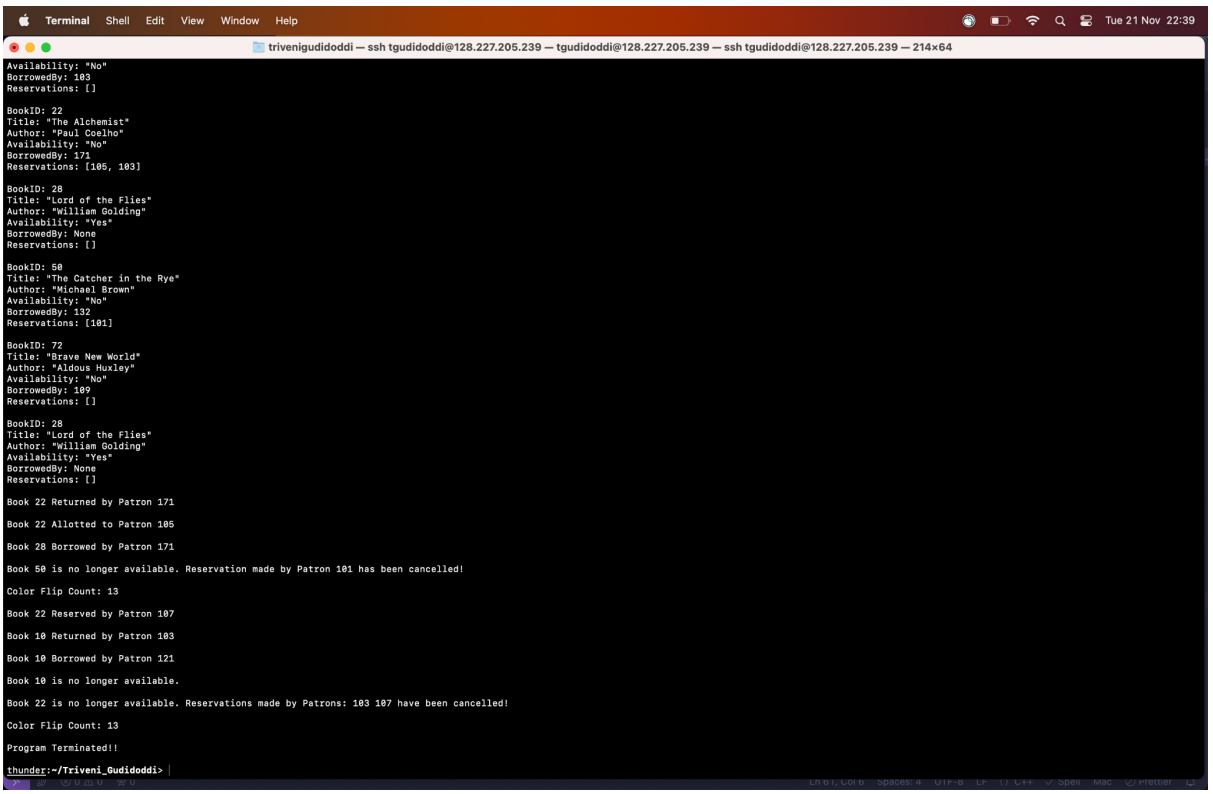
```
BookID: 25
Title: "Computer Networks"
Author: "Andrew S. Tanenbaum"
Availability: "No"
BorrowedBy: 153
Reservations: []

BookID: 48
Title: "Data Structures and Algorithms"
Author: "Sartaj Sahni"
Availability: "No"
BorrowedBy: 142
Reservations: [140, 144]

BookID: 73
Title: "Introduction to the Theory of Computation"
Author: "MichaelSipser"
Availability: "No"
BorrowedBy: 111
Reservations: [52]

BookID: 101
Title: "Introduction to Algorithms"
Author: "Thomas H. Cormen"
Availability: "No"
BorrowedBy: 132
Reservations: []

BookID: 115
Title: "Operating Systems: Internals and Design Principles"
Author: "William Stallings"
Availability: "Yes"
BorrowedBy: None
Reservations: []

BookID: 125
Title: "Computer Organization and Design"
Author: "David A. Patterson"
Availability: "Yes"
BorrowedBy: None
Reservations: []

BookID: 132
Title: "Operating System Concepts"
Author: "Abraham Silberschatz"
Availability: "Yes"
BorrowedBy: None
Reservations: []

Book 25 Reserved by Patron 171

Book 132 Borrowed by Patron 2

BookID: 48
Title: "Data Structures and Algorithms"
Author: "Sartaj Sahni"
Availability: "No"
BorrowedBy: 142
Reservations: [140, 144]

Book 101 Reserved by Patron 18

Book 210 Borrowed by Patron 210
```

```
Book 73 Reserved by Patron 43

BookID: 210
Title: "Machine Learning: A Probabilistic Perspective"
Author: "KevinP. Murphy"
Availability: "No"
BorrowedBy: 210
Reservations: []

Book 210 Reserved by Patron 34

Color Flip Count: 24

Book 125 is no longer available.

Book 115 is no longer available.

Book 210 is no longer available. Reservation made by Patron 34 has been cancelled!

Color Flip Count: 26

Book 25 is no longer available. Reservation made by Patron 171 has been cancelled!

Book 80 is no longer available.

Color Flip Count: 26

Program Terminated!!

thunder:~/Triveni_Gudidoddi>
```

## **Test2:** (test2_output_file.txt)

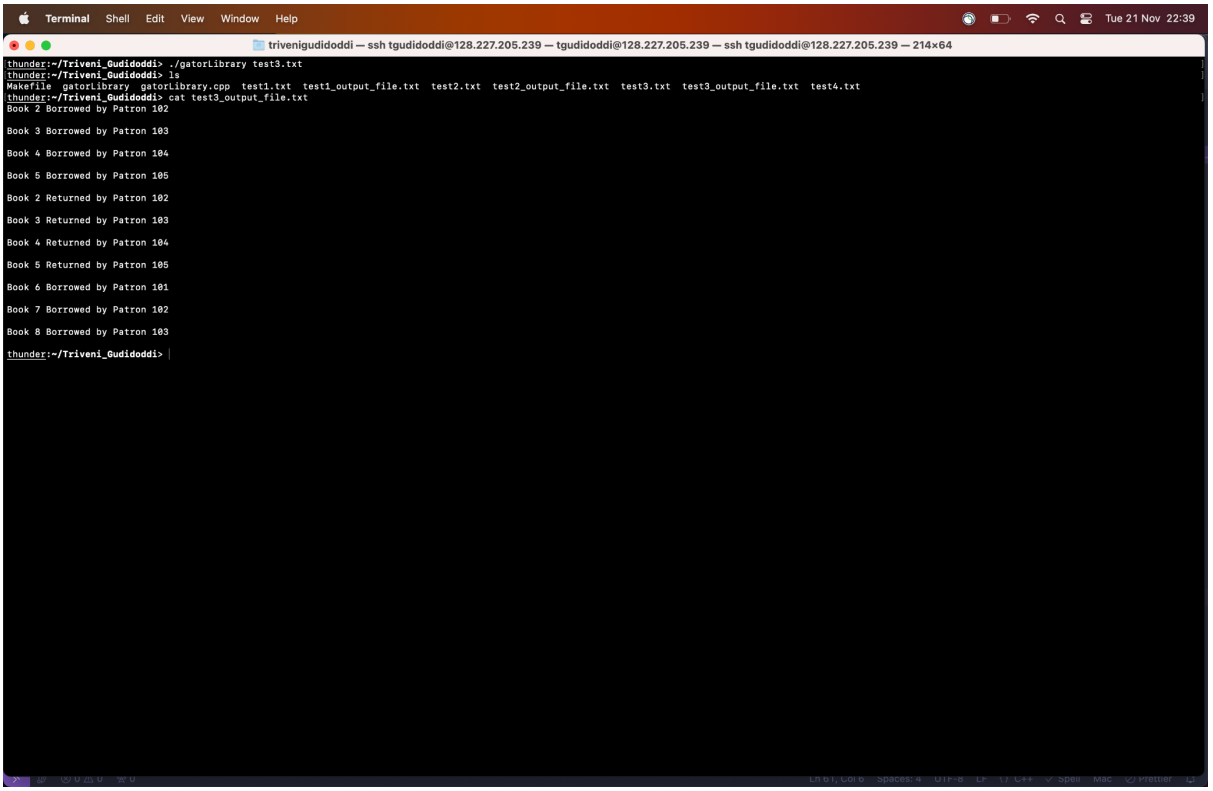


```
Terminal  Shell  Edit  View  Window  Help                                                    Tue 21 Nov  22:39
trivenigudidoddi — ssh tgudidoddi@128.227.205.239 — tgudidoddi@128.227.205.239 — ssh tgudidoddi@128.227.205.239 — 214×64

thunder:~/Triveni_Gudidoddi> ./gatorLibrary test2.txt
thunder:~/Triveni_Gudidoddi> ls
Makefile  gatorLibrary  gatorLibrary.cpp  test1.txt  test1_output_file.txt  test2.txt  test2_output_file.txt  test3.txt  test4.txt
thunder:~/Triveni_Gudidoddi> cat test2_output_file.txt
Book 5 Borrowed by Patron 101

Book 3 Borrowed by Patron 101

Book 12 is no longer available.

BookID: 3
Title: "The Great Gatsby"
Author: "Mark Johnson"
Availability: "No"
BorrowedBy: 101
Reservations: []

BookID: 5
Title: "The Secret Garden"
Author: "Jane Smith"
Availability: "No"
BorrowedBy: 101
Reservations: []

Book 3 is no longer available.

Book 5 is no longer available.

Book 5 not found in the Library

Book 22 Borrowed by Patron 104

Book 22 Reserved by Patron 171

Book 22 Reserved by Patron 103

Color Flip Count: 11

Book 94 Borrowed by Patron 171

Book 22 Returned by Patron 104

Book 22 Allotted to Patron 171

Book 50 Borrowed by Patron 132

Book 10 Borrowed by Patron 103

Book 72 Borrowed by Patron 109

Book 50 Reserved by Patron 101

Book 94 is no longer available.

Book 22 Reserved by Patron 105

BookID: 10
Title: "The Hobbit"
Author: "J.R.R. Tolkien"
Availability: "No"
BorrowedBy: 103
Reservations: []

BookID: 22
```



```
Terminal  Shell  Edit  View  Window  Help                                                    Tue 21 Nov  22:39
trivenigudidoddi — ssh tgudidoddi@128.227.205.239 — tgudidoddi@128.227.205.239 — ssh tgudidoddi@128.227.205.239 — 214×64

Availability: "No"
BorrowedBy: 103
Reservations: []

BookID: 22
Title: "The Alchemist"
Author: "Paul Coelho"
Availability: "No"
BorrowedBy: 171
Reservations: [105, 103]

BookID: 28
Title: "Lord of the Flies"
Author: "William Golding"
Availability: "Yes"
BorrowedBy: None
Reservations: []

BookID: 50
Title: "The Catcher in the Rye"
Author: "Michael Brown"
Availability: "No"
BorrowedBy: 132
Reservations: [101]

BookID: 72
Title: "Brave New World"
Author: "Aldous Huxley"
Availability: "No"
BorrowedBy: 109
Reservations: []

BookID: 28
Title: "Lord of the Flies"
Author: "William Golding"
Availability: "Yes"
BorrowedBy: None
Reservations: []

Book 22 Returned by Patron 171

Book 22 Allotted to Patron 105

Book 28 Borrowed by Patron 171

Book 50 is no longer available. Reservation made by Patron 101 has been cancelled!

Color Flip Count: 13

Book 22 Reserved by Patron 107

Book 10 Returned by Patron 103

Book 10 Borrowed by Patron 121

Book 10 is no longer available.

Book 22 is no longer available. Reservations made by Patrons: 103 107 have been cancelled!

Color Flip Count: 13

Program Terminated!!

thunder:~/Triveni_Gudidoddi>
```

**Test3:** (test3_output_file.txt)




**Test4:** (test4_output_file.txt)

```
thunder:~/Triveni_Gudidoddi> ./gatorLibrary test4.txt
thunder:~/Triveni_Gudidoddi> ls
Makefile  gatorLibrary  gatorLibrary.cpp  test1.txt  test1_output_file.txt  test2.txt  test2_output_file.txt  test3.txt  test3_output_file.txt  test4.txt  test4_output_file.txt
thunder:~/Triveni_Gudidoddi> cat test4_output_file.txt
Book 1 Borrowed by Patron 101

Book 2 Borrowed by Patron 102

Book 3 Borrowed by Patron 103

Book 4 Borrowed by Patron 104

Book 5 Borrowed by Patron 105

Book 1 Returned by Patron 101

Book 1 Borrowed by Patron 106

Book 2 Reserved by Patron 107

Book 3 Reserved by Patron 108

Book 4 Reserved by Patron 109

Book 5 Reserved by Patron 110

Book 2 Returned by Patron 102

Book 2 Allotted to Patron 107

Book 1 is no longer available.

BookID: 10
Title: "Book10"
Author: "Author10"
Availability: "Yes"
BorrowedBy: None
Reservations: []

Book 3 Returned by Patron 103

Book 3 Allotted to Patron 108

Book 11 Borrowed by Patron 112

Book 11 is no longer available.

thunder:~/Triveni_Gudidoddi>
```

Thank you!!