

COP 5536: Advanced Data Structures

Fall 2023

Project Report



Name: Yash Bhalla
UFID: 26116597
Email ID: yash.bhalla@ufl.edu

Introduction

The project's objective is to create the fictional library called GatorLibrary, utilizing a Red-Black Tree to organize books based on their book-ids. GatorLibrary is designed to offer functionalities such as adding and deleting books, as well as enabling patrons to borrow, return, and reserve books.

How to run the program?

The program is made using C++, and thus requires a makefile to create the executable. The makefile makes the executable file 'gatorLibrary' which can be executed using terminal.

To run the executable, type:

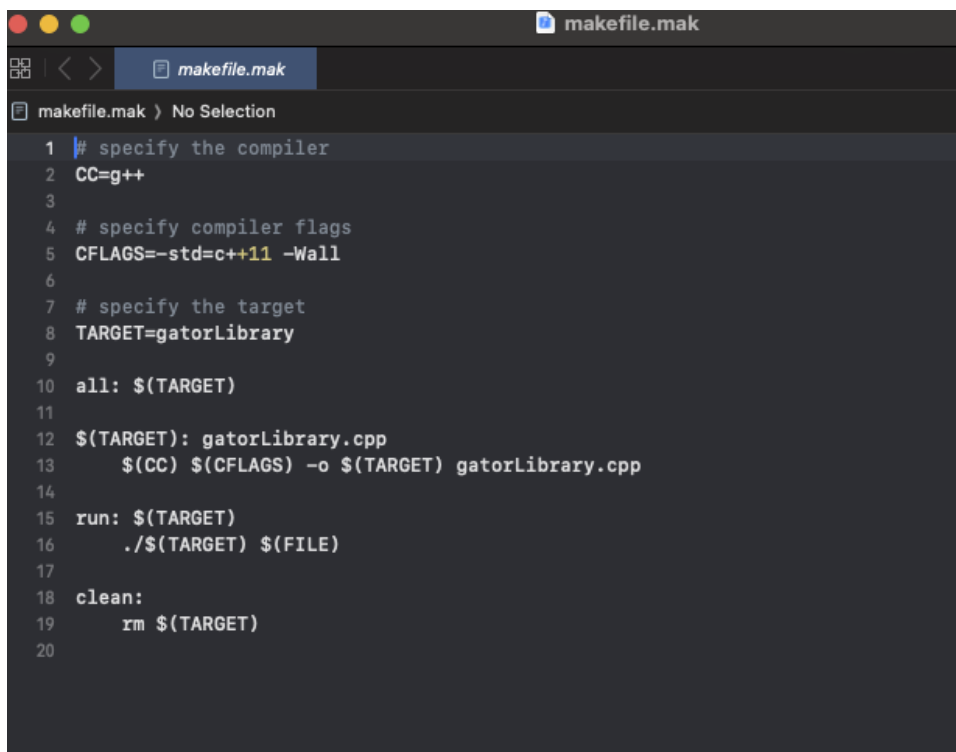
```
.\ gatorLibrary <input_filename>
```

Replace the <input_filename> with the file name, for example, the input file is 'input1.txt' the command should look like:

```
.\ gatorLibrary input1.txt
```

What is a Makefile?

The makefile serves the purpose of generating a target executable file. Its primary function is to articulate a series of tasks essential for the execution of a program. Essentially, the makefile establishes a connection between the source code and the final executable by specifying the steps and dependencies involved in the build process. This enables efficient compilation and linking of source code, ensuring that the executable is created in a systematic and reproducible manner.

A screenshot of a code editor window titled 'makefile.mak'. The editor shows the content of the Makefile with line numbers 1 through 20. The code specifies the compiler as g++, compiler flags as -std=c++11 -Wall, the target as gatorLibrary, and includes rules for building the target, running it, and cleaning up.

```
1 # specify the compiler
2 CC=g++
3
4 # specify compiler flags
5 CFLAGS=-std=c++11 -Wall
6
7 # specify the target
8 TARGET=gatorLibrary
9
10 all: $(TARGET)
11
12 $(TARGET): gatorLibrary.cpp
13     $(CC) $(CFLAGS) -o $(TARGET) gatorLibrary.cpp
14
15 run: $(TARGET)
16     ./$$(TARGET) $(FILE)
17
18 clean:
19     rm $(TARGET)
20
```

Concepts Utilized

A Red-Black Tree is a self-balancing binary search tree in computer science. In this tree structure, each node has an additional attribute called color, which can be either red or black, and plays a crucial role in maintaining balance.

The rotations executed in the program deviate from those illustrated in the lecture slides. Instead, they align with rotations observed in an online tree simulator. This approach enhances the visualization of the tree at each step, aiding in the identification of potential mistakes and ensuring a more accurate implementation.

A Min Heap is a binary tree variant where the value of each parent node is less than or equal to the values of its children. This characteristic holds true throughout the entire tree. Consequently, the root node invariably holds the smallest (minimum) value in the heap.

Program Structure, Class & Methods Prototypes

Structure

Reservations {}: This structure node holds all the necessary reservation information such as resPatronID, resPriority and resTimeStamp.

Node {}: This structure node holds all the Book data such as BookId, BookName, AuthorName, AvailabilityStatus, BorrowedBy, ReservationHeap, color, parent of the node, left child of the node, right child of the node and patronsReserved.

Class

RedBlackTree {}: Implementation of Red-Black Tree.

Private Members:

- void initializeNode (nodepointer, nodepointer): To initialize the nodes of the tree to NULL.
- pair<int, int> getGreatestAndSmallest (vector<int>, int, int): To get the closest smallest greater and greatest smaller values than the given key value.
- void printNode(nodepointer): To print a node value.
- void printBooksFind(nodepointer, int, int): To iterate through and print all the books as per the range specified.
- nodepointer searchTreeFind(nodepointer, int): To find a book as per the given ID.
- void fixMinHeap(vector<reservations>&): To fix the reservation min-heap after a new reservation has been added.
- void inorderTraversal(nodepointer, vector<int>&): To perform inorder traversal.
- void insertion(nodepointer): To fix the tree after insertion.

- void deletion(nodepointer): To fix the tree after deletion.
- void redBlackTreeTransplant(nodepointer, nodepointer): Exchanges a subtree as a child of its parent with another subtree.
- void deleteNodeFind(nodepointer, int): To delete a book as per the given ID.
- void deleteMinimum(vector<reservations>&): To delete the topmost value from the heap and replace it with a value that does not violate the rules of the heap.
- void printTreeFind(nodepointer, string, bool): To print the complete tree.

Public Members:

- RedBlackTree(): Constructor to initialize the Red-Black tree.
- void searchClosestBookID(int): Returns the book as per the given key.
- vector<int> inOrder(): Returns the inorder traversal array of the tree.
- nodepointer searchTree(int): To search a given node in the tree.
- nodepointer maximum(nodepointer): To get the max value at a given node.
- nodepointer minimum(nodepointer): To get the min value at a given node.
- void leftRotate(nodepointer): To left rotate the tree at the given node.
- void rightRotate(nodepointer): To right rotate the tree at the given node.
- void insertBook(int, string, string, string): To insert a new book.
- void deleteNode(int): To delete the book as per the given key.
- void printTree(): To print the tree.
- void borrowBook(int, int, int): Allows to borrow a book and gives results if the book is unavailable, and reserves the book as per the priority.
- void returnBook(int, int): Allows return of the book and allots patron as per reservation list, who is at the top of the list.
- void printBooks(int, int): To print the all books between the range given
- void printBook(int): To print the book data as per the given key.