Bonus Practical

COS132

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Official Deadline: 08/05/2024 at 17:30

Extended Deadline: 09/05/2024 at 23:59

Marks: 170

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your practical well before the deadline, as no extension will be granted.
- **The extended deadline has been put in place in case of loadshedding or other unforeseen circumstances. No further extension will be granted.**
- **Note that no tutor or lecturer will be available after the official deadline**
- You may not import any libraries that have not been imported for you, except `<iostream>` and `cmath`.
- You may `use namespace std;`
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire practical before you start coding.

- You will be afforded 20 upload opportunities per task.
- **You should no longer be using the online compiler. You should compile, run and test your code locally on your machine using terminal commands or makefiles**.
- **You will be awarded a mark of 0 should you use any loops to achieve functionality. An exception will be made for testing in your main function.**
- **You have to use C++98 in order to get marks**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.**

3 Outcomes

Upon successful completion of this practical, students will be able to:

- Understand how to use structs.
- Understand how to use recursion.
- Use pointers to traverse a data structure.

4 Structure

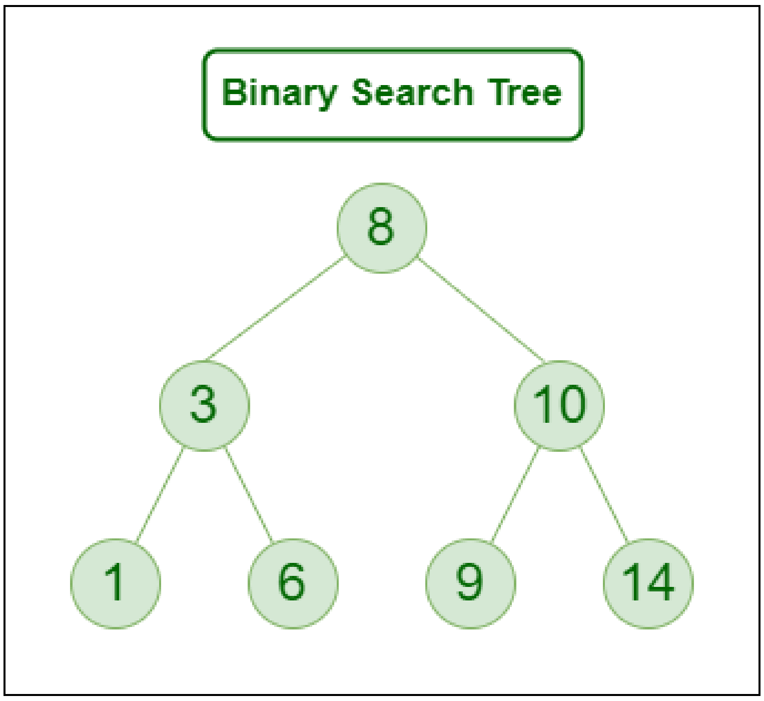
This practical will consist of one task. This task consists of 1 task. You can assume that all input to functions will be valid (i.e. of the correct data type and within range)

5 Mark Distribution

Activity	**Mark**
Task 1 - BST	170
Total	**170**

Table 1: Mark Distribution

6 Background



- In this practical, you will be implementing a binary search tree along with an insert and a search algorithm.
- Binary search trees are used for efficiently storing and retrieving sorted data, allowing for fast lookup, insertion, and deletion operations.
- For any binary search tree, the following properties hold:
  - Every node, including the first node in the tree has two child nodes (or subtrees).
  - The right child node contains data greater than the parent node.
  - The left child node contains data less than the parent node.
  - No duplicate data is contained within the tree.

6.1 Insertion

- If the tree is empty, create a node with the required data and set that as the root node.

- If the tree is not empty:
  - Traverse the tree using the properties mentioned above(e.g: if the data to be inserted is less than the current node, traverse down the left subtree).
  - At each node in the tree: Check if the appropriate child is null, if yes then insert the data by assigning the new node to that child node. If not, then continue traversing the tree via the appropriate child node.

6.2 Searching

- Once you have the insertion working, searching is trivial.
- Traverse the tree until the data is found (by using the properties of a Binary search tree).
- If the data being looked for is found, return true. If not return false.

7 Task 1

After the defeat at the G20 summit, you received a cryptic folder from FooBar asking you to implement a certain tree...

You have been given a `BinaryTree.h`, `BinaryTree.cpp`, as well as `Node.h` and `main.cpp`. Your task is to complete the function implementations in `BinaryTree.cpp` and `BinaryTree.h` according to the instructions.

7.1 Binary Tree Header

The `BinaryTree.h` header file includes the declarations for function that will implement a Binary Tree.

```
#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include "Node.h"
#include <iostream>

struct BinaryTree{
    Node* root;
    BinaryTree(){
        root = NULL;
    }
    void insertNode(int data);
    void recursiveInsert(Node* curr, Node* newNode);
    bool searchNode(int data);
    bool recursiveSearch(Node* curr, int data);
};

#endif
```

Create `BinaryTree.cpp` and include the `BinaryTree.h` header.

```
#include "BinaryTree.h"

//Implement Binary Tree functions here
```

7.2 void BinaryTree::insertNode(int data)

- Purpose: This function will be called when data is to be inserted into the binary tree.
- The passed in parameter contains the data to be inserted.
- Create a node containing the data by calling the constructor in the Node.h file.

- Check if the tree is empty (root will be NULL if it is empty).
- If the tree is not empty, call **recursiveInsert** with root as curr parameter and newly created node as the newNode parameter.

7.3 void BinaryTree::recursiveInsert(Node* curr, Node* newNode)

- Purpose: This is a helper function for the insert.
- This function does the exact same thing as the insert function above, however instead of checking the root node, it checks the curr variable.
- You must first check if the data exists, if it does print: "No duplicate insertions are permitted.", with a newline at the end.

7.4 bool BinaryTree::searchNode(int data)

- Purpose: This function is called when the tree is to be searched for a specific data item.
- This function first checks the root node to see if the tree is empty.
- If the tree is empty, print: "Tree is empty.", with a newline at the end and then return false.
- If the tree is not empty call **recursiveSearch** with root as curr and data as data.

7.5 bool BinaryTree::recursiveSearch(Node* curr, int data)

- Purpose: This is a helper function for the search.
- Check if the curr variable is NULL, if it is then print: "Data not found.", with a newline at the end and return false.
- If the data is equal to the current node's data value, use the following statement: **std::cout « "Current Node value is: " « curr->data « ", the required value is: " « data « std::endl;** and return true.
- If both the above cases have not been fulfilled then print out the following statement: **std::cout « "Current node value is: " « curr->data « ".";**
- If the data is less than the current node's value, use the following statement: **std::cout « "The desired data is less than the current node's value, searching down the left child..."«std::endl;**, then return a recursive call with the appropriate child as the curr parameter.
- If the data is greater than the current node's value, use the following statement: **std::cout « "The desired data is greater than the current node's value, searching down the right child..."«std::endl;**, then return a recursive call with the appropriate child as the curr parameter.
- Place a false return at the bottom of the function so the compiler does not scream at you.

8 Submission checklist

For Task 1:

- Archive (zip) the file used for Task 1 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - BinaryTree.cpp
  - BinaryTree.h
  - Node.h
  - main.cpp
- Upload the archive to FitchFork Bonus Practical Task 1 before the deadline