INFORMATION TECHNOLOGY UNIVERSITY

Software Engineering Department - ITU

SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 2/10/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 7. Implementation of Binary Tree

Name	Roll number	Report (out of 35)

Checked on: _____

Signature: _____

1.1 Objective

The objective of this lab is to practice problems related to binary tree implementation.

1.2 Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

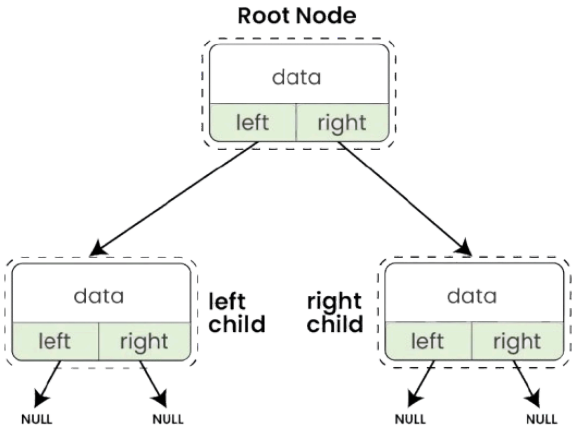
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A **binary tree** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child. Binary trees are are widely used in various applications such as expression parsing, sorting algorithms, and data storage.

Tree Properties:

- **Height**: The length of the longest path from the root to a leaf node.
- **Depth**: The distance of a node from the root, measured in edges.
- **Degree**: The number of children a node has.



Traversal: Various traversal methods can be used to visit nodes in a binary tree:

- **In-Order Traversal**: Visits the left child, the parent node, and then the right child, resulting in sorted order for binary search trees.
- **Pre-Order Traversal**: Visits the root before its subtrees (left then right), useful for creating a copy of the tree or generating prefix expressions.
- **Post-Order Traversal**: Visits the subtrees (left then right) before the root, ideal for deleting the tree or generating postfix expressions.

INFORMATION TECHNOLOGY UNIVERSITY

Lab Tasks

Task 1:

Implement the **TreeNode** class

Data Members:

- **int data**: The value stored in the node.
- **TreeNode* left**: Pointer to the left child.
- **TreeNode* right**: Pointer to the right child.

Member Functions to Implement:

- **TreeNode(int value)**: Constructor to initialize the node with a given value.
- **TreeNode* getLeftChild()**: Returns a pointer to the left child.
- **TreeNode* getRightChild()**: Returns a pointer to the right child.
- **int getData()**: Returns the data stored in the node.
- **void addLeftChild(TreeNode* node)**: Adds a left child to the current node.
- **void addRightChild(TreeNode* node)**: Adds a right child to the current node.

Task 2:

Implement the **Tree** class

Data Members:

- **TreeNode* root**: Pointer to the root node of the tree.

Member Functions to Implement:

- **Tree()**: Constructor to initialize the tree.
- **void insertNode(int value)**: Inserts a node into the tree using level order traversal.
- **void deleteNode(int value)**: deletes the node with the given value by replacing it with the last leaf node and then deleting it.
- **void printTree()**: Prints the tree in an in-order traversal.
- **int getTreeHeight()**: Returns the height of the tree.
- **int getDegree(int data)**: Returns the degree of a given node (number of children).
- **int getHeight(int data)**: Returns the height of a given node.
- **TreeNode* findNode(int data):** Finds and returns the node with the data otherwise return nullptr
- **TreeNode* getRoot():** Returns the root node of the tree.

Implementation Instructions:

- **TreeNode Class**:
  - Implement the TreeNode class with its members and functions.
  - Ensure proper memory management when adding or removing child nodes.
- **Tree Class**:
  - Implement the Tree class.
  - **insertNode**:
    - Use a queue for level order traversal to insert a new node.
    - Ensure that the tree maintains its structure as a binary tree.
    - You can modify and use the queue class from the last assignment which would be helpful
  - **deleteNode**:
    - The node to be deleted will be found by matching the value.
    - Replace the node to be deleted with the deepest leaf node in the tree.
    - The deepest leaf node is then removed.
  - **printTree**:
    - Implement in-order traversal to print the tree nodes.
  - **getTreeHeight**, **getDegree**, and **getHeight**:
    - Implement these functions to compute the respective values based on the given criteria.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					