|  |
| --- |
| **Department of Computer and Software Engineering – ITU** |
| **SE200T: Data Structures & Algorithms** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 10th Oct 2024** |
| **Teaching Assistant: Zainab, Sadia & Ryan** | **Semester: Fall 2024** |
| **Session: 2024-2028** | **Batch: BSSE2023B** |

# **Assignment 8. Tree Traversals (BFS and DFS)**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll number** | **Obtained Marks/35** |
| Muhammad Mukarram Raza | BSSE – 23029 - B |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Submission:**

• Email instructor or TA if there are any questions. You cannot look at others’ solutions or use others’ solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.

• Submission after due time will not be accepted.

**In this assignment you have to do following tasks:**

**Task 1:** Ensure that you have installed all three softwares in your personal computer (Github, Cygwin & CLion). Now, accept the assignment posted in the classroom (e.g Google, LMS etc) and after accepting, clone the repository to your computer. Make sure you have logged into the github app with your account.

**Task 2:** Open Cygwin app, Move to your code directory with following command “cd <path\_of\_folder>”, <path\_of\_folder> can be automatically populated by dragging the folder and dropping it to the cygwin window.

Run the code through Cygwin, use command “make run”, to get the output of the code

**Task 3:** Solve the given problems, write code using **CLion** or any other IDE.

**Task 4:** Keep your code in the respective git cloned folder.

**Task 5:** Commit and Push the changes through the Github App

**Task 5**: Write the code in separate files **(as instructed**). Ensure that file names are in lowercase (e,g **main.cpp**).

**Task 6:** Run ‘**make run**’ to run C++ code

**Task 7:** Run ‘**make test**’ to test the C++ code

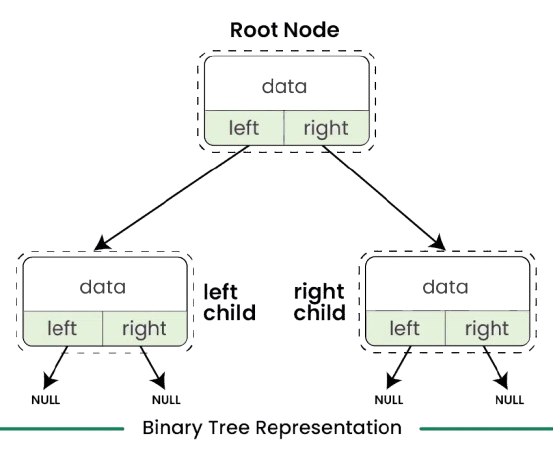
Write code in functions, after completing each part, verify through running code using **“make run”** on Cygwin. Make sure to test the code using **“make test”.**

## **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.
* **Level Order Traversal:** Visits nodes level by level from top to bottom and left to right within each level. This traversal is often implemented using a queue and is useful for scenarios like finding the shortest path in an unweighted tree or printing the tree's nodes in a breadth-first manner.

### Objective for Assignment

The objective of this assignment is to implement a binary tree structure and various traversal methods using the provided Lab 7 code. You will create a TreeNode class and a Tree class with specific attributes and methods as outlined below.

### 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.

**Task 3: Implement Display Functions**

**Member Functions to Implement:**

* **void display\_in():** Displays the nodes of the tree using in-order traversal.
* **void display\_pre():** Displays the nodes of the tree using pre-order traversal.
* **void display\_post():** Displays the nodes of the tree using post-order traversal.
* **void display\_level():** Displays the nodes of the tree using level order traversal.

*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 Assignment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | 3 | 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. |  |
| 1. Conducting experiment | 2 | 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. |  |
| 1. Computer use | 4 | Document submission timely. | Document submission late. | Document submission not done. |  |
| 1. Teamwork | 4 | 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 |  |
| 1. Laboratory safety and disciplinary rules | 2 | 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. |  |
| 1. Data collection | 2 | 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. |  |
| 1. Data analysis | 3 | 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):** | | | | |  |