**Assignment 2:**

**General Tree Implementation and Manipulation**

**Data Structures (CS-UH 1050) — Summer 2024**

# **Code of Conduct**

All assignments are graded and we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we briefly state what is and isn’t allowed when working on an assignment.

Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as **these discussions are restricted to general solution techniques, without sharing the overall or specific details.** Put differently, these discussions should not be about the concrete code you are writing, nor about a specific set of steps or results you wish to submit. Discussions with others should not lead to you possessing the complete or partial solution of others in any form (paper or digital), regardless of who made the solution. You are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that **there is never a valid reason to share your code with fellow students and that there is no valid reason to publish your code online in any form**. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g., if the solution matches with high similarity score that of others), **the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi** (see<https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/>) under the integrity review process.

**Introduction**

In this assignment, you will implement a General Tree data structure. You will download the provided starter code, implement the required methods, and interact with the tree through a command-line interface. The goal is to understand tree operations, traversal techniques, and dynamic memory management in C++.

#### **Instructions**

1. **Download Starter Code:**
   * Download the provided starter code from Brightspace.
   * The starter code includes the **Node** and **Tree** classes with some methods already implemented.
2. **Implement the Following Methods in the Tree Class:**
   1. **Insert Method**
      * **Signature:** void insert(Node\* node, string name)
      * **Description:** Create and insert a new node with the given name as a child of the specified node.
   2. **Remove Method**
      * **Signature:** void remove(Node\* node, string child\_name)
      * **Description:** Remove a specific child node from the given parent node. Print “Child not found” if the specified child node does not exist.
   3. **Is External Method**
      * **Signature:** bool isExternal(Node\* node)
      * **Description:** Check and return **true** if a given node is an external (leaf) node**, false** otherwise.
   4. **Is Internal Method**
      * **Signature:** bool isInternal(Node\* node)
      * **Description:** Check and return **true** if a given node is an internal (non-leaf) node.
   5. **Size Method**
      * **Signature:** int size(Node\* node)
      * **Description:** Calculate and return the number of nodes in the tree/sub-tree rooted at the given node.
   6. **Depth Method**
      * **Signature:** int depth(Node\* node)
      * **Description:** Calculate the depth of the specified node.
   7. **Height Method**
      * **Signature:** int height(Node\* node)
      * **Description:** Calculate the height of the specified node.
   8. **Tree Height Method**
      * **Signature:** int treeHeight()
      * **Description:** Calculate the height (maximum depth) of the entire tree.
   9. **Pre-order Traversal Method**
      * **Signature:** void preorder(Node\* node)
      * **Description:** Traverse and print the tree in pre-order.
   10. **Post-order Traversal Method**
       * **Signature:** void postorder(Node\* node)
       * **Description:** Traverse and print the tree in post-order.
   11. **Destructor for Node Class:**
       * **Description:** Define the destructor for the Node class to delete the node and all its children. Ensure proper memory management to avoid memory leaks.
3. **Enhance the Command-Line Interface:**
   1. Use the provided main function to interact with the tree. Ensure that all commands work as expected with your implementations. See the screenshot below for the expected outputs.
4. **Comments and Documentation:**
   1. Write clear and concise comments throughout your code to explain the logic and flow. Proper documentation is essential for understanding and maintaining the code.

**Submission**

* Ensure that your code is well-documented and follows proper coding standards.
* Submit the source code (cpp file) to Brightspace by the due date. Late submissions will be accepted only up to 3 days late. Afterward, you will receive zero points. For late submissions, 5% will be deducted from the assignment grade per late day.

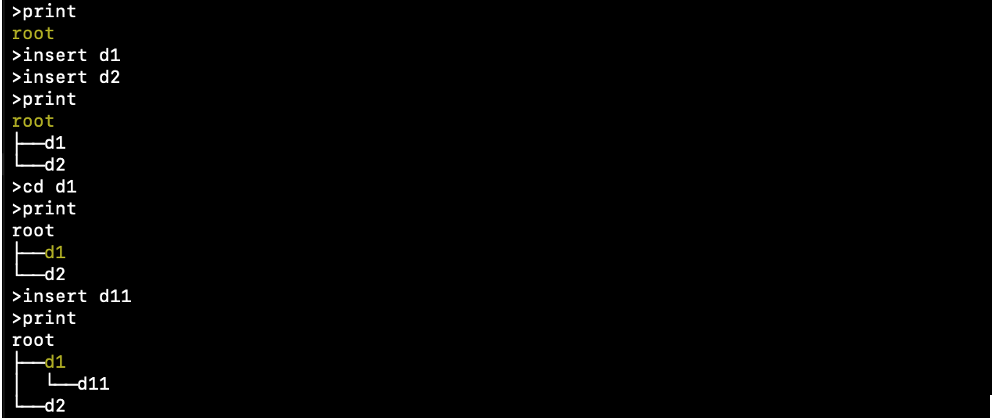
**Additional Notes**

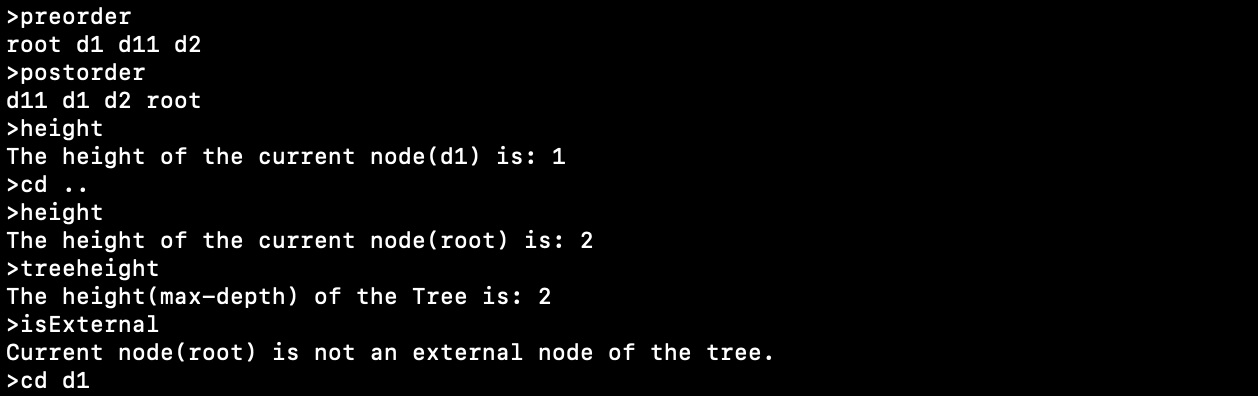
* Pay attention to edge cases and ensure your methods handle them appropriately.
* Test your methods thoroughly to verify their correctness before submission.
* NO SUBMISSIONS OR RESUBMISSIONS VIA EMAIL WILL BE ACCEPTED.
* Your program should be implemented in C++ and must be runnable on the Linux, Unix, or macOS operating system.

**Grading**

|  |  |
| --- | --- |
| **Description** | **Score (20)** |
| Quality in Code Organization & Modularity  - Defining of all required methods correctly  - Writing effiecient and clean code  - Proper initiation and termination of the system | 3 |
| Implementing all required commands correctly | 11 |
| Proper error handling of missing and invalid input etc. | 3 |
| Properly commenting the code (i.e., code documentation) | 3 |

**Sample Output:**



**A black background with a black square

Description automatically generated with medium confidenceA black screen with white text

Description automatically generated**