Data Structures and Algorithms Lab

Lab 11 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

Binary Search Tree (BST) for Student Management

Implement a class for **Binary Search Trees (BST)**. Each node of this tree will store the **id**, **name**, and **fee** of a student existing in a text file named **input.txt**. The data in the input file is formatted as follows: each new line contains the student's **id**, followed by a blank space, the student's **name**, another blank space, and finally, the student's **fee**.

```
class Student
{
      friend class StudentBST;

private:
      int id;             /** student identifier (unique). */
      string name;        /** student name. */
      float fee;          /** student fee. */
      Student* left;      /** left subtree of a node. */
      Student* right;     /** right subtree of a node. */
};

class StudentBST
{
private:
      Student* root;     /** root of the tree. */

      void inOrder(Student* stree);        /** Helper for in-order traversal. */
      void preOrder(Student* stree);       /** Helper for pre-order traversal. */
      void postOrder(Student* stree);      /** Helper for post-order traversal. */
      void destroy(Student* stree);        /** Helper to destroy the tree. */
public:
      StudentBST();     /** constructor. */
      ~StudentBST();    /** destructor. */
};
```

Specifications for Member Functions

Implement the following member functions of the **StudentBST** class:

1. `void insert(int id, string name, float fee)`
   - Inserts a new student with the given **id**, **name**, and **fee** into the BST at the appropriate position.
   - If a student with the same **id** already exists, this function will not insert the new record and will display an error message such as: "Student with ID <id> already exists".
   - **Time Complexity**:
     - **Average Case:** $O(\log N)$
     - **Worst Case:** $O(N)$ (if the tree is unbalanced)

2. `void search(int id)`
   - Searches for a student with the given **id** in the BST. Displays their details (**id**, **name**, **fee**) if found or an appropriate message otherwise.
   - **Time Complexity**:
     - **Average Case:** $O(\log N)$
     - **Worst Case:** $O(N)$

3. `void inOrder()`
    - Performs an in-order traversal of the BST and displays the details (**id**, **name**, **fee**) of each student.
    - Calls the private helper function:
        - `void inOrder(Student* stree)`
            - A recursive function that performs the in-order traversal on the subtree pointed to by **stree**.
    - **Time Complexity**: $O(N)$

4. `void preOrder()`
    - Performs a pre-order traversal of the BST and displays the details (**id**, **name**, **fee**) of each student.
    - Calls the private helper function:
        - `void preOrder(Student* stree)`
            - A recursive function that performs the pre-order traversal on the subtree pointed to by **stree**.
    - **Time Complexity**: $O(N)$

5. `void postOrder()`
    - Performs a post-order traversal of the BST and displays the details (**id**, **name**, **fee**) of each student.
    - Calls the private helper function:
        - `void postOrder(Student* stree)`
            - A recursive function that performs the post-order traversal on the subtree pointed to by **stree**.
    - **Time Complexity**: $O(N)$

6. `void destroy(Student* stree)`
    - A private, recursive function that destroys (deallocates) the nodes of the subtree pointed to by **stree** in a post-order manner. The left and right subtrees of a node are destroyed before deallocating the node itself.
    - **Time Complexity**:
        - **Worst Case:** $O(N)$

7. `void deleteNode(int id)`
    - Removes a particular student from the tree based on their **id**. Displays an appropriate message in either case.
    - Carefully handles all boundary cases (e.g., deleting the root node, nodes with one or two children, or a leaf node).
    - **Time Complexity**:
        - **Average Case:** $O(\log N)$
        - **Worst Case:** $O(N)$

Input File Specifications

The program should read student data from the file **input.txt** in the following format:

```
<id name fee>

<id name fee>

<id name fee>

...
```

Each student's data is separated by a blank line.

Main Function

- Implement the **main()** function to test the functionality of the BST. The program should:
    1) Read data from the **input.txt** file and insert it into the BST.
    2) Perform searches for specific **ids** and display appropriate outputs.
    3) Display the BST contents using in-order, pre-order, and post-order traversals.
    4) Test the deletion functionality.