Data Structures and Algorithms Lab

Lab 13 **Marks 07**

Instructions

Work on this lab individually. You can use your books, notes, handouts etc. but you are not allowed to borrow anything from your peer student.

Marking Criteria

Show your work to the instructor before leaving the lab to get some or full credit.

What you must do

Implement a class for **Binary Search Trees (BST)**. Each node of this tree will store the **student id**, **name,** and **fee** of a student exist in a text file named **input.txt**. The data in the **input** file is exactly in the following format: each new line contains **student-id** then a blank space **student-name** then a blank space **student-fee**.

The class definitions will look like:

```
class Student
{
	friend class StudentBST;

private:
	int stdId;			//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
public:
	StudentBST();			//constructor
	~StudentBST();			//destructor
};
```

You are required to implement the following member functions of the **StudentBST** class:

```
bool insert (int stdId, string name, float balance);
```

This function will **insert** a new student's record in the **BST**. The **3 arguments** of this function are the **student-id**, **name**, and **student-fee** of this new student, respectively. This function will check whether a **student** with the same **student-id** already **exists** in the tree. If it does not exist, then this function will insert it into the tree at its appropriate location and return **true**. If a **student** with the same **student-id** already exists, then this function should return **false**.

```
bool search (int stdId);
```

This function will search the **BST** for a member with the given **student-id**. If such a **student** is found, then this function should display the **details (student-id, name, and fee)** of this **student** and return **true**. If such a **student** is not found, then this function should **display a message** indicating this and return **false**.

```
void inOrder ();
```

This function will perform an **in-order** traversal of the **BST** and display the **details (student-id, name, and fee)** of each student. It will be a **public** member function of the **StudentBST** class. This function will call the following helper function to achieve its objective.

```
void inOrder (Student* stree);
```

This will be a **recursive** function which will perform the **in-order** traversal on the sub-tree which is being pointed by **stree**. It will be a **private** member function of the **StudentBST** class.

`void preOrder ();`

This function will perform a **pre-order** traversal of the **BST** and display the **details (student-id, name, and fee)** of each student. It will be a **public** member function of the **StudentBST** class. This function will call the following helper function to achieve its objective.

`void preOrder (Student* stree);`

This will be a **recursive** function which will perform the **pre-order** traversal on the sub-tree which is being pointed by **stree**. It will be a **private** member function of the **StudentBST** class.

`void postOrder ();`

This function will perform a **post-order** traversal of the **BST** and display the **details (student-id, name, and fee)** of each student. It will be a **public** member function of the **StudentBST** class. This function will call the following helper function to achieve its objective.

`void PostOrder (Student* stree);`

This will be a **recursive** function which will perform the **post-order** traversal on the sub-tree which is being pointed by **stree**. It will be a **private** member function of the **StudentBST** class.

`void destroy (Student* stree);`

This will be a **recursive** function which will destroy **(deallocate)** the nodes of the sub-tree pointed by **stree**. This function will destroy the nodes in **post-order** way i.e., the left and right sub-trees of a node are destroyed before de-allocating the node itself. This function will be a **private** member function of the **StudentBST** class.

Implement the **main** function and test the functionality of your created classes by taking data from the **input** file.

☺ ☺ ☺ **BEST OF LUCK** ☺ ☺ ☺