# Lab 11
# Data Structures
# BS DS Fall 2024 Morning/Afternoon

## Task # 1

In this lab you are going to start implementing a class for creating and storing **Binary Search Trees (BST)**. Each node of this BST will store the **roll number**, **name** and **CGPA** of a student. The class definitions will look like:

```
class StudentNode {

        int rollNo;              // Student's roll number (must be
                                 unique)
       string name;              // Student's name
       double cgpa;              // Student's CGPA
        StudentNode* left;       // Pointer to the left subtree of a node
        StudentNode* right;      // Pointer to the right subtree of a node
};
class StudentBST {
  private:

   StudentNode* root;            // Pointer to the root node of the tree
public:
     StudentBST();               // Default constructor
};
```

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

`bool insert (int, string, double)`

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

`bool search (int)`

This function will search the BST for a student with the given roll number. If such a student is found, then this function should display the details (roll number, name, and CGPA) of this student and return true. If such a student is not found then this function should display an appropriate message and return false.

`~StudentBST ()`

This is the destructor for the StudentBST class. This function will call the following helper function to achieve its objective.

`void destroy (StudentNode* s)` ***// private member function of StudentBST class***

This will be a **recursive** function which will destroy (deallocate) the nodes of the subtree pointed by **`s`**. This function will be a private member function of the StudentBST class.

void InOrder ()
This function will perform an **in-order** traversal of the BST and display the details (roll number, name, and CGPA) of each student. The list of students produced by this function will be sorted (in increasing order) by roll numbers of students. It will be a public member function of the StudentBST class. This function will actually call the following helper function to achieve its objective.

void InOrder (StudentNode* s) ***// private member function of StudentBST class***
This will be a **recursive** function which will perform the in-order traversal on the subtree which is being pointed by **s**. This function will be a private member function of the StudentBST class.

Write a **menu-based** driver function to illustrate the working of different functions of the StudentBST class. The menu should look like:

```
1. Insert a new student
2. Search for a student
3. See the list of students
4. Quit

Enter your choice:
```

**Task # 1.2**
Add the following public member function to the **StudentBST** class.

bool remove (int rn)
This function will search the BST for a student with the given **roll number**. If such a student is found, then this function should remove the record of that student from the BST and should return true. If such a student is not found then this function should display an appropriate message and return false.

**Task # 1.3**
Now add the following public member function to the **StudentBST** class:

void displayInRange (double cgpaStart, double cgpaEnd)
This function will search the BST for those students whose CGPA is between **cgpaStart** and **cgpaEnd** (both inclusive). The records of all such student should be displayed in **ascending order** of their **Roll Numbers**.
*Hint: You may need to implement one or more helper functions.*

## Task # 2

Implement a BST class to store integers (as we have already done in lectures). Apart from the default constructor and destructor, this class should provide functions to insert, search, and remove elements from the BST.

Now, implement the following member functions of the BST class:

### Task # 2.1

A recursive function to search for a particular element in the BST.

```
bool recSearch (int key);              // public (driver function)
bool recSearch (BSTNode* b, int key);      // private (workhorse function)
```

### Task # 2.2

A recursive function to count the number of nodes in a BST.

```
int countNodes ();                 // public (driver function)
int countNodes (BSTNode* b);   // private (workhorse function)
```

### Task # 2.3

A recursive function to determine the height of a BST.

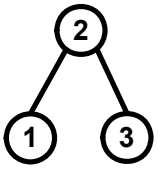
```
int getHeight ();                  // public (driver function)
int getHeight (BSTNode* b);    // private (workhorse function)
```

### Task # 2.4

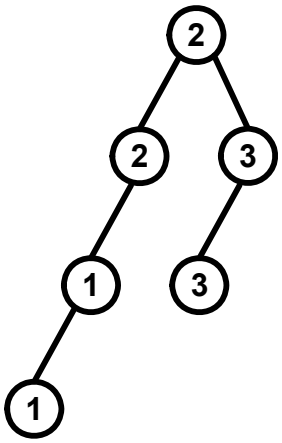
Write a function **`doubleTree`** (determine the exact function prototype yourself) of the BST class, which for each node in the BST, creates a new duplicate node, and inserts the duplicate as the left child of the original node. The resulting tree should still be a binary search tree (left subtree of any node will contain values which smaller than or equal to the value present in the node, while the right subtree will contain values which are strictly larger than the value present in the node).

*Example:*

The following BST:




would be changed to:

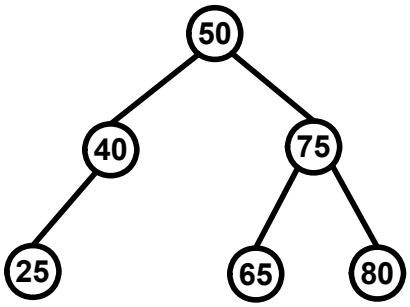



**Task # 2.5**

Implement a member function of the BST class which prints **all the root-to-leaf paths** of a given BST. The prototype of your function will be:

```
void printAllPaths ();
```

For example, if we call this function on the following BST:




It should print the following paths:

```
50 -> 40 -> 25
50 -> 75 -> 65
50 -> 75 -> 80
```

*Hint 1*: You will need to implement a recursive helper function. Think about the prototype of this recursive helper function.
*Hint 2*: You need some way to communicate the list of paths from one function call to another.
*Hint 3*: You can assume that the maximum length of a root-to-leaf path will be 100.
Note: Do NOT use any global or static variables in your implementation.