Data Structures and Algorithms LAB – Spring 2022
(BS-IT-F20 Morning & Afternoon)

Lab # 11

Instructions:

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1.1 *(Max Time: 15 Minutes)*

You should have the implementation of a class **BST** to store integers (as we have already done in the lecture). Class declarations will look like as shown below:

```
class BSTNode {
      friend class BST;
  private:
      int data;
      BSTNode *left, *right;
};
```

```
class BST {
  private:
      BSTNode *root;
  public:
      ...
};
```

Apart from the **default constructor** and **destructor**, you should have the implementation of the following functions of the **BST** class:

```
bool search (int val);      // iteratively search for val in the BST
bool insert (int val);      // insert val in the BST
bool remove (int val);      // remove val from the BST
void preOrder ();           // Prints the pre-order traversal of the BST
void inOrder ();            // Prints the in-order traversal of the BST
void postOrder ();          // Prints the post-order traversal of the BST
```

Now, write a **menu-based** driver function to illustrate the working of all functions of the **BST** class. The menu should look like as shown below:

```
1. Insert a value
2. Search a value
3. Remove a value
4. Display values (Pre-order)
5. Display values (In-order)
6. Display values (Post-order)
7. Quit

Enter your choice:
```

Task # 1.2 *(Max Time: 10 Minutes)*

Implement the following functions of the **BST** class to **recursively** determine (and return) the count of the nodes present in the BST:

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

Also modify the **menu**, so that the user can use this function to view the count of nodes present in the BST.

Task # 1.3 *(Max Time: 15 Minutes)*

Implement the following functions of the **BST** class to **recursively** determine (and return) the height of the BST:

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

Also modify the **menu**, so that the user can use this function to view the height of the BST.

Task # 1.4 *(Max Time: 20 Minutes)*

Implement the following **public** member function of the **BST** class

```
void createBalancedTree (int* arr, int start, int end)
```

This function will take an array of integers (**arr**) which is **Sorted (in increasing order)**, and its starting (**start**) and ending (**end**) indices. You can assume that all the integers present in the array are distinct (i.e. there is no repetition).

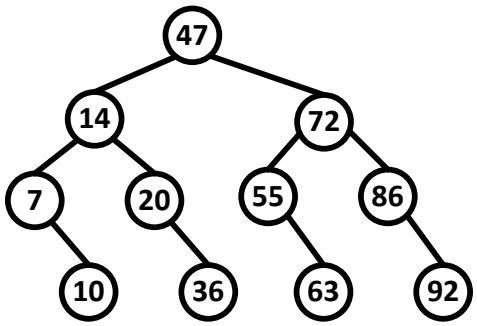
When called on an empty BST, the above function should insert the values of the given array to create a balanced BST. If the BST is non-empty, this function should firstly make the tree empty, and then use the given values to create a balanced BST.

The function **`createBalancedTree(…)`** will call the following **private** member function of the BST class to accomplish its objective:

```
BSTNode* createBalancedTreeHelper (int* arr, int start, int end)
```

This function **`createBalancedTreeHelper(…)`** will be implemented **recursively**. (***Note:*** Do NOT use any global or static variables).

Example: If the array **`arr`** contains the following elements **{ 7 10 14 20 36 47 55 63 72 86 92 }**, the following BST should be created by the **`createBalancedTree(…)`** function:



After you have created the balanced tree, you should display the **pre-order**, **in-order**, and **post-order** traversals of the resulting tree to convince yourself (and us) that you have implemented the algorithm correctly.

Task # 1.5 *(Max Time: 20 Minutes)*

Implement the following two public member functions of the **`BST`** class. Your implementation should be as efficient as possible. Also modify the **menu**, so that the user can use these two functions.

```
int findMin ();      // iteratively find and return the smallest value of the BST
int findMax ();      // iteratively find and return the largest value of the BST
```

Task # 1.6 *(Max Time: 10 Minutes)*

Implement the following functions of the **`BST`** class to **recursively** search for a particular value in the BST:

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

Task # 1.7 *(Max Time: 10 Minutes)*

Implement the following two functions of the **BST** class which should **recursively** find and return the **smallest** value in the given BST. Your implementation should be as efficient as possible.

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

Task # 1.8 *(Max Time: 10 Minutes)*

Implement the following two functions of the **BST** class which should **recursively** find and return the **largest** value in the given BST. Your implementation should be as efficient as possible.

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

Task # 2.1 *(Max Time: 10 Minutes)*

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

```
class StudentBST;

class StudentNode {
      friend class StudentBST;
  private:
      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 BST
  public:
      StudentBST();           // Default constructor
};
```

Task # 2.2 *(Max Time: 20 Minutes)*

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

bool insert (int rn, string n, double c)

This function will insert a new student's record in the **StudentBST**. The 3 arguments of this function are the **Roll number**, **Name**, and **CGPA** of this new student, respectively. The insertion into the tree will be done based upon the roll number of the student. 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 in the tree, then this function should not make any changes in the tree and should return **false**. This function should NOT display anything on screen.

bool search (int rn)

This function will search the **StudentBST** for a student with the given **roll number** (see the parameter). 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**.

Task # 2.3 *(Max Time: 15 Minutes)*

void inOrder ()

This function will perform an **in-order** traversal of the **StudentBST** and display the details (roll number, name, and CGPA) of each student. The list of students displayed by this function should be **sorted in increasing order** of roll numbers. This function will be a public member function of the **StudentBST** class. This function will 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.

Task # 2.4 *(Max Time: 15 Minutes)*

~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.

Task # 2.5 *(Max Time: 20 Minutes)*

Implement the following public member function of the **`StudentBST`** class:

`bool remove (int rn)`

This function will search the BST for a student with the given **roll number** (see parameter). If such a student is found, then this function should remove the record of that student from the BST and should return **true**. If a student with the given roll number is not found, then this function should not make any changes in the tree and should return **false**. This function should not display anything on screen.

Task # 2.6 *(Max Time: 20 Minutes)*

Write a **menu-based** driver function to illustrate the working of all functions of the **`StudentBST`** class. The menu may look like as shown below:

```
1. Insert a new student
2. Search for a student
3. Remove a student
4. See the list of all students (sorted by Roll No.)
5. Quit

Enter your choice:
```

VERY IMPORTANT

In the next lab, you will need some or all of the functions from this Lab. So, make sure that you have the working implementation of **all** the functions of the classes **`BST`** and **`StudentBST`**, when you come to the next lab.