Data Structures & Algorithms LAB – Fall 2015

(BS-SE-F14 Morning & Afternoon)

Lab # 11

Instructions:

- **Attempt the following tasks exactly in the given order.**
- Make sure that there are no ***dangling pointers*** or ***memory leaks*** in your programs.
- Indent your code properly.
- Use meaningful variable and function names. Follow the naming conventions.
- Use meaningful prompt lines/labels for all input/output.

Task # 1 *(Max Time: 25 Minutes)*

Add the following public member function to the **`StudentBST`** class (which you implemented in **Lab # 10**):

`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 *(Max Time: 15 Minutes)*

Implement the following functions of the **`BST`** class (which you implemented in **Lab # 10**) to **recursively** determine (and return) the height of the BST:

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

Task # 3 *(Max Time: 25 Minutes)*

Add the following **public member function** to the **`BST`** class (which you implemented in **Lab # 10**):

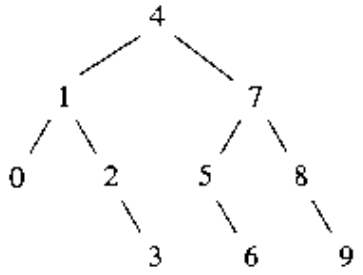
`void createBalancedTree (int* arr, int start, int end)`

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 (using the recursive **`destroy(…)`** function), and then insert the given values to create a balanced BST. Note that:

- The function **`createBalancedTree(…)`** will be implemented **recursively**.
- This function will use the **`insert(…)`** function to insert a value into the BST.

For example, if the array contains the following elements **{0 1 2 3 4 5 6 7 8 9}**, the following BST should be created by the above function:



In your driver program, after you have created the balanced tree, you should display the **pre-order**, **in-order**, and **post-order** traversals of the tree to convince yourself (and the TA's) that you have implemented the algorithm correctly.

Task # 4.1 *(Max Time: 15 Minutes)*

In this task, you are going to implement a class for Max Heap. Each node of this Max Heap will contain the roll number, and CGPA of a student. **The heap will be organized on the basis of students' CGPAs** i.e. the student having the maximum CGPA will be at the root of the heap. The class definitions will look like:

```
class StudentMaxHeap;
class Student {
      friend class StudentMaxHeap;
  private:
      double cgpa;       // Student's CGPA
      int rollNo;        // Student's roll number
};
class StudentMaxHeap {
  private:
      Student* st;       // Array of students which will be arranged like a Max Heap
      int currSize;      // Current number of students present in the heap
      int maxSize;       // Maximum number of students that can be present in the heap
  public:
      StudentMaxHeap (int size);     // Constructor
      ~StudentMaxHeap();             // Destructor
      bool isEmpty();                // Checks whether the heap is empty or not
      bool isFull();                 // Checks whether the heap is full or not
};
```

First of all, implement the **constructor**, **destructor**, **`isEmpty`** and **`isFull`** functions shown above in the class declaration.

Task # 4.2 *(Max Time: 15 Minutes)*

Add a member function to the **`StudentMaxHeap`** class which inserts the record of a new student (with the given roll number and CGPA) in the Max Heap. The prototype of your function should be:

```
bool insert (int rollNo, double cgpa);
```

This function should return true if the record was successfully inserted in the heap and it should return false otherwise. The worst case time complexity of this function should be ***O*(lg *n*)**. If two students have the same CGPA then their records should be stored in a way such that at the time of removal if two (or more) students have the **same highest CGPA** then the student with smaller roll number should be removed before the students with larger roll numbers.

Task # 4.3 *(Max Time: 15 Minutes)*

Now add a member function to remove that student's record from the Max Heap which has the highest CGPA. The prototype of your function should be:

```
bool removeBestStudent (int& rollNo, double& cgpa);
```

Before removing the student's record, this function will store the roll number and CGPA of the removed student in its two arguments. It should return **true** if the removal was successful and it should return **false** otherwise. The worst case time complexity of this function should also be ***O*(lg *n*)**.

Task # 4.4 *(Max Time: 10 Minutes)*

Now add the following two member functions to the **`StudentMaxHeap`** class:

`void levelOrder ();`
This function will perform a level order traversal of the **`StudentMaxHeap`** and display the roll numbers and CGPAs of the students.

`int height ();`
This function will determine and return the height of the **`StudentMaxHeap`**. The worst case time complexity of this function should be ***O*(1)**.

Task # 4.5 *(Max Time: 10 Minutes)*

Finally, write a menu-based driver function to illustrate the working of different functions of the **`StudentMaxHeap`** class. The menu should look like:

```
1. Insert a new student
2. Remove (and display) the student with the Max CGPA
3. Display the list of students (Level-order traversal)
4. Display the height of the heap
5. Exit

Enter your choice:
```