Assignment 03 – Student Binary Search Tree

Average expected estimated time to complete this assignment is 12 hours

- [1 HOUR] software requirement understanding
  - reading of the assignment document
  - understanding of problem statement
  - understanding of software inputs and outputs
- [2 HOURS] software design
  - identification of user-software interaction sequences
    - software menus
    - user inputs and
    - software flows
  - identification of required variables and data structures to be used
    - for input, processing, and output
    - identification of storage class specifications of required variables
  - modularization of software into required functions
    - proper naming of functions
    - identification of tasks to be performed into functions
    - identification of parameters and return types of functions
  - identification of user defined header files
    - names and functions to be placed in
  - flow sequence of main function and call of various user defined functions from it
- [6 HOURS] software coding
  - coding of user defined functions
  - placement of user defined functions into header file/s
  - coding of main function
- [2 HOURS] software testing
  - running software on various inputs
  - identification of software error and bugs
  - removal of software error and bugs
- [1 HOUR] software documentation
  - proper indentation of code
  - use of proper naming conventions
  - commenting of code

Objectives:

- This assignment will provide experience with usage of Binary Search Tree as the storage nonlinear structure for Student Data.

Prerequisite Skills:

- Tree Data Structure
- Binary Search Tree

Problem Statement:

Instructions:

In this Assignment, 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 1:

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 three 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:

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 3:

~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 4:

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 5:

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:

File(s) to be submitted:

The proper working solution of the Student Binary Search Tree. There should not be memory leak, dandling pointers, null pointer assignment errors or an Illegal memory access Exceptions. Try to handle all the exceptions and possible run time errors in proper way. You Driver program should be menu based allowing to call all above methods options and also exit the program option.

Deadline:

December 10, 2023. (11:59pm)