|  |  |  |
| --- | --- | --- |
| **Lab 10** | |  |
| **Topic** | * Binary Search Tree * Binary Search Tree Application |
| **Objective** | * The basic purpose of this lab is to implement ADT of Binary Search Tree and test its applications. |

**Instructions:**

* Indent your code.
* Comment your code.
* Use meaningful variable names.
* Plan your code carefully on a piece of paper before you implement it.
* Name of the program should be same as the task name. i.e. the first program should be Task\_1.cpp

# void main() is not allowed. Use int main()

* **You have to work in multiple files. i.e separate .h and .cpp files**

# You are not allowed to use system("pause")

* **You are not allowed to use any built-in functions**

# You are required to follow the naming conventions as follow:

* + **Variables:** firstName; (no underscores allowed)
  + **Function:** getName(); (no underscores allowed)
  + **ClassName:** BankAccount (no underscores allowed)

# Students are required to complete the following tasks in lab timings.

**Binary Search Tree**

*A Binary Search Tree is a Binary Tree where every node's left child has a lower value, and every node's right child has a higher value. Hence in a binary tree, every node in the left sub-tree is* ***less*** *than the root node, and every node in the right sub-tree is of a value* ***greater*** *than the root node. The properties of a binary search tree are recursive: if we consider any node as a “root,” these properties will remain true.*

struct Node

{

int data;

Node\* left;

Node\* right;

};

Create a C++ generic abstract class named as **BST**, with the following:

**Attributes:**

1. Node \*root;

**Functions:**

**virtual Node\* insert(Node\*, int) = 0;**

* + Should add a new node in the BST.

**virtual Node\* delete(Node\*, int) = 0;**

* + Should delete a node in the BST.

BST();

* **Now use the above class to make another derived class named as MyBST having the following additional functionalities:**
* **Implement both pure virtual functions ‘insert(Node \*,int)’ and ‘delete(Node\*, int)’ declared in base class, in MyBST.**

**Task 1**

1. Implement a ‘**printInOrder’** function to print the elements of the Binary Search Tree by performing InOrder traversal.

**void printInOrder(struct node \*root)**

1. Implement a ‘**printPreOrder’** function to print the elements of the Binary Search Tree by performing PreOrder traversal.

**void printPreOrder(struct node \*root)**

1. Implement a ‘**printPostOrder’** function to print the elements of the Binary Search Tree by performing PostOrder traversal.

**void printPostOrder(struct node \*root)**

**Task 2**

Implement a function **‘CountEdges’** to count the number of edges from root node to the given node in the BST.

**int** **CountEdges(struct node \*root, int val)**

**Task 3**

1. Implement a function **‘findMax’** to find the largest node in the BST.

**Node\*** **findMax (struct node \*root)**

1. Implement a function **‘findMin’** to find the smallest node in the BST.

**Node\*** **findMin(struct node \*root)**

**Now create menu based program to perform the following operations**

1. Press 1 to add node in BST.
2. Press 2 to delete node in BST.
3. Press 3 to print BST in InOrder.
4. Press 4 to print BST in PreOrder.
5. Press 5 to print BST in PostOrder.
6. Press 6 to count the number of edges from root node to given node in BST.
7. Press 7 to the find largest node in BST.
8. Press 8 to the find smallest node in BST.
9. Press 0 to exit.