Implementation of Binary Search Tree in C+

Binary Search Tree is an ordered binary tree in which each node has no more than two child nodes. Each child of the tree must be a leaf node or the root of another binary search tree. The left sub-tree of BST has the nodes with key values less than the parent node and the right sub-tree has the nodes with key values greater than parent node.

We implemented the same concept of BST in C++. Each nodes of the tree is represented by node of struct type which stores the element and also contains two child node. binarySearchTree() in the class represents the constructor and it is necessary to initialize root to NULL in it. The add(int) function adds the element to the tree by creating a new node containing the element to be added. If the root node is NULL, the new element is inserted to the root node. Else, it moves down the tree, left for the lower value to be inserted and right for the greater value to be inserted. The remove(int) function removes any user defined element from the tree if it is present in it. If the root node is NULL, a message is shown saying the tree is empty. Else, it moves down the tree just like in add function , left for the lower value to be removed and right for the greater value. If the element is found, the node containing the element is removed and the nodes below the removed node are balanced. If the element is not present in the tree, an error message saying that the new element is not found is displayed. The search(int) function searches for an element in the tree. If the element is present in the tree, the function returns true or else, it returns false.The function inorderTraversal(node* ptr) traverses the left subtree first, then it visits the root and then traverses the right subtree. The function printInOrder() prints the In Order Traversal of the the tree. The function preorderTraversal(node* ptr) visits the root, traverses the left subtree and then traverses the right subtree. The function printPreOrder() prints the Pre Order Traversal of the the tree. The function postorderTraversal(node* ptr) traverses the left subtree, traverses the right subtree and then visits the root. The function printPostOrder() prints the Post Order Traversal of the the tree. The function levelorderTraversal(node* ptr) visits every node of a level before going to the lower level. It is done with the help of queue. The root node is added to the queue and the following is repeated if the queue is non empty. The node is dequeued from the front of the queue and is visited. The node’s children is enqueued from left to right. The function printLevelOrder() prints the Level Order Traversal of the the tree. The function destroy(node* ptr) deletes the binary search tree with help of the function destroyCall(). The main function prompts user to choose operation represented by numbers 1 to 9. The program doesn’t terminate until user chooses number 9 as input. If the numbers other than the numbers from 1 to 9 is entered, it asks user to enter valid choice. If user chooses 1, it asks user for number to be inserted to the tree and inserts it into binary search tree. If the user chooses 2, it prints out the In Order Traversal of the tree. If the user chooses 3, it prints out the Pre Order Traversal of the tree. If the user chooses 4, it prints out the Post Order Traversal of the tree. If the user chooses 5, it prints out the In Level Order Traversal of the tree. If the user chooses 6, it asks user for number to be removed from the tree and deletes it from the binary search tree. If the user chooses 7, it asks user for a number to be searched in the tree and if the number is present, the program prompts found as output. If the user chooses 8, it destroys the whole binary search tree.