Yash Sarang.

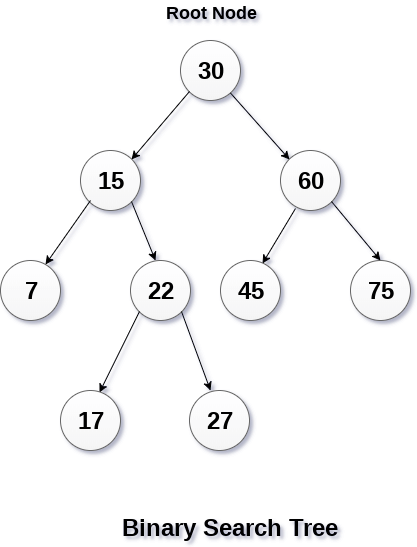
Roll No: 47, Class : D6AD.

Data Structures. Experiment-13.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AIM**: To write a program to implement binary search trees.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**THEORY:**   
A binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called an ordered binary tree. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root. Similarly, the value of all the nodes in the right sub-tree is greater than or equal to the value of the root.  


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OPERATIONS ON BST:

* Searching: Finding the location of some specific element in a binary search tree.
* Insertion: Adding a new element to the binary search tree at the appropriate location so that the property of BST does not violate.
* Deletion: Deleting some specific node from a binary search tree. However, there can be various cases of deletion depending upon the number of children, the node has.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ALGORITHM:

**1. Insert:**

**Step 1** START

**Step 2** Store the key to be inserted (x)

**Step 3** Check element present in the tree if not goto step 4 else step 5

**Step 4** Make inserted key Root Node

**Step 5** Compare x with root node if smaller goto step 6 else goto step 7 or no root node find goto step 9.

**Step 6** Element reaches the left subtree repeat Step 5

**Step 7** Element reaches the right subtree repeat Step 5

**Step 8** Insert the key

**Step 9** STOP

**2.Deletion:**

Step 1: IF TREE = NULL

Write "item not found in the tree" ELSE IF ITEM < TREE -> DATA

Delete(TREE->LEFT, ITEM)

ELSE IF ITEM > TREE -> DATA

Delete(TREE -> RIGHT, ITEM)

ELSE IF TREE -> LEFT AND TREE -> RIGHT

SET TEMP = findLargestNode(TREE -> LEFT)

SET TREE -> DATA = TEMP -> DATA

Delete(TREE -> LEFT, TEMP -> DATA)

ELSE

SET TEMP = TREE

IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL

SET TREE = NULL

ELSE IF TREE -> LEFT != NULL

SET TREE = TREE -> LEFT

ELSE

SET TREE = TREE -> RIGHT

[END OF IF]

FREE TEMP

[END OF IF]

Step 2: END

**3. Search:**

Step1: START

Step2: store the element to be searched

Step3: check if the element to be searched is greater or lower than the root value

Step4: If the element value is less than the root value then search in the left subtree, if greater than the root value then search in the right subtree.

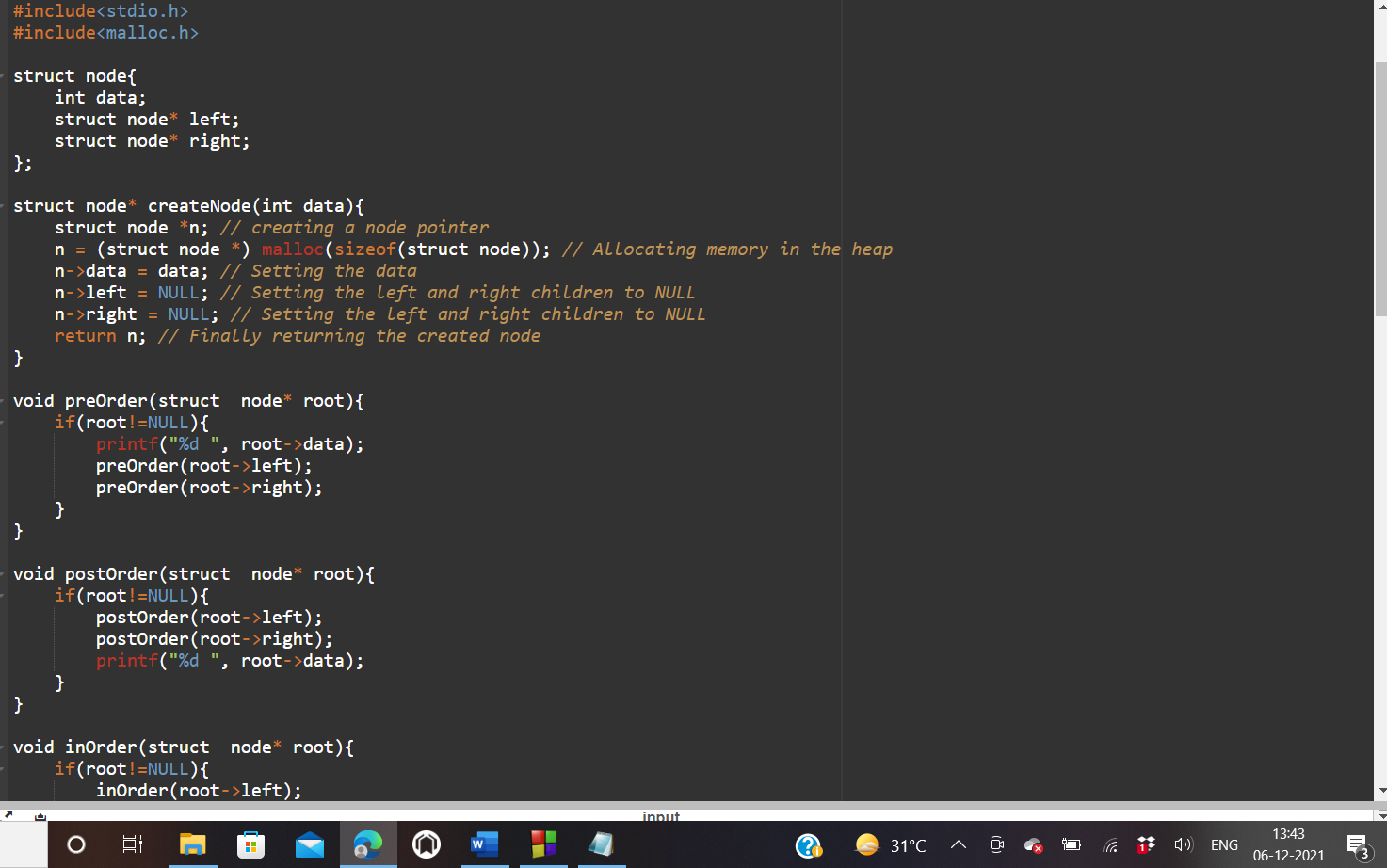
Step5: continue this process for the further nodes till you get the element.

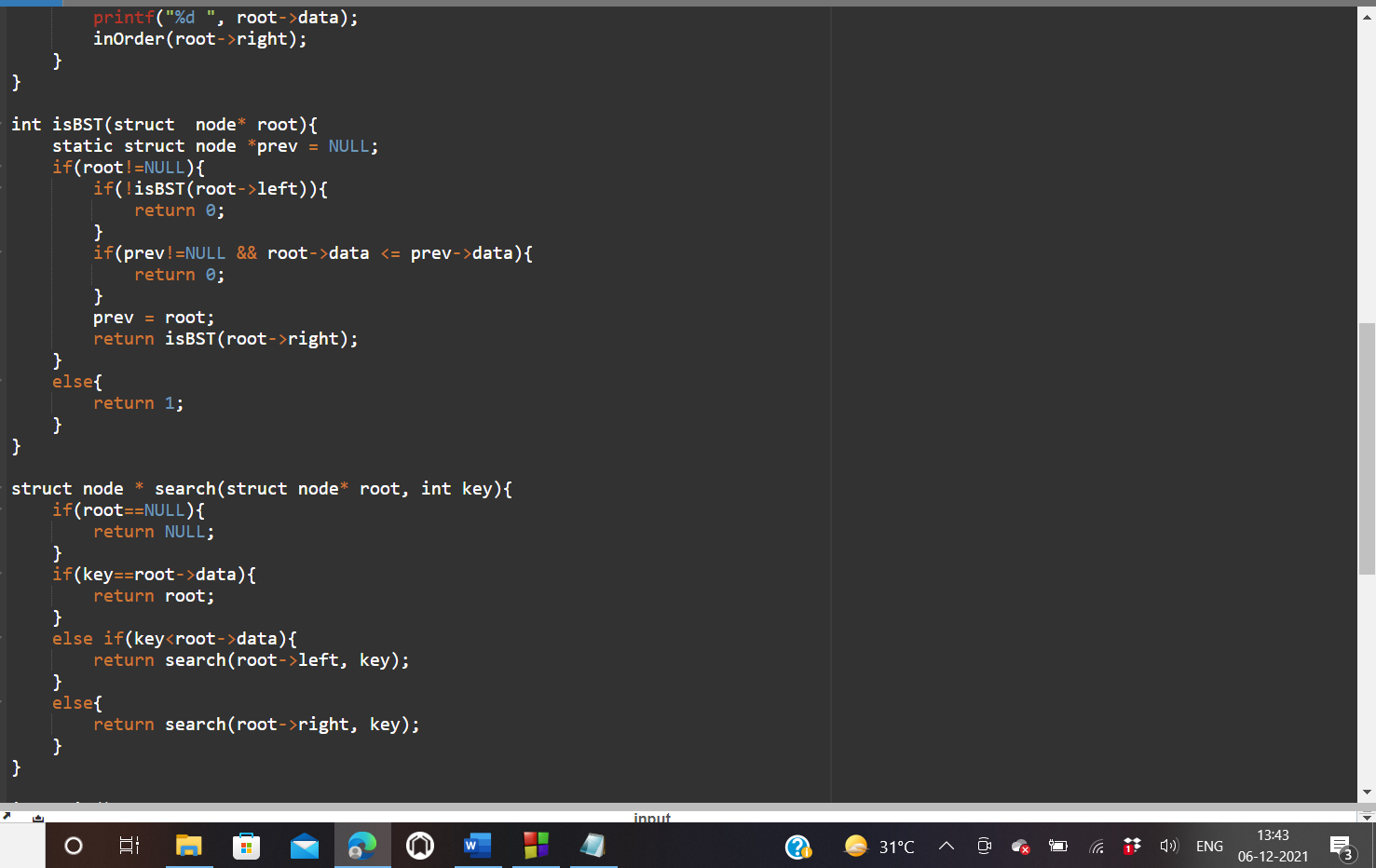
Step6: END

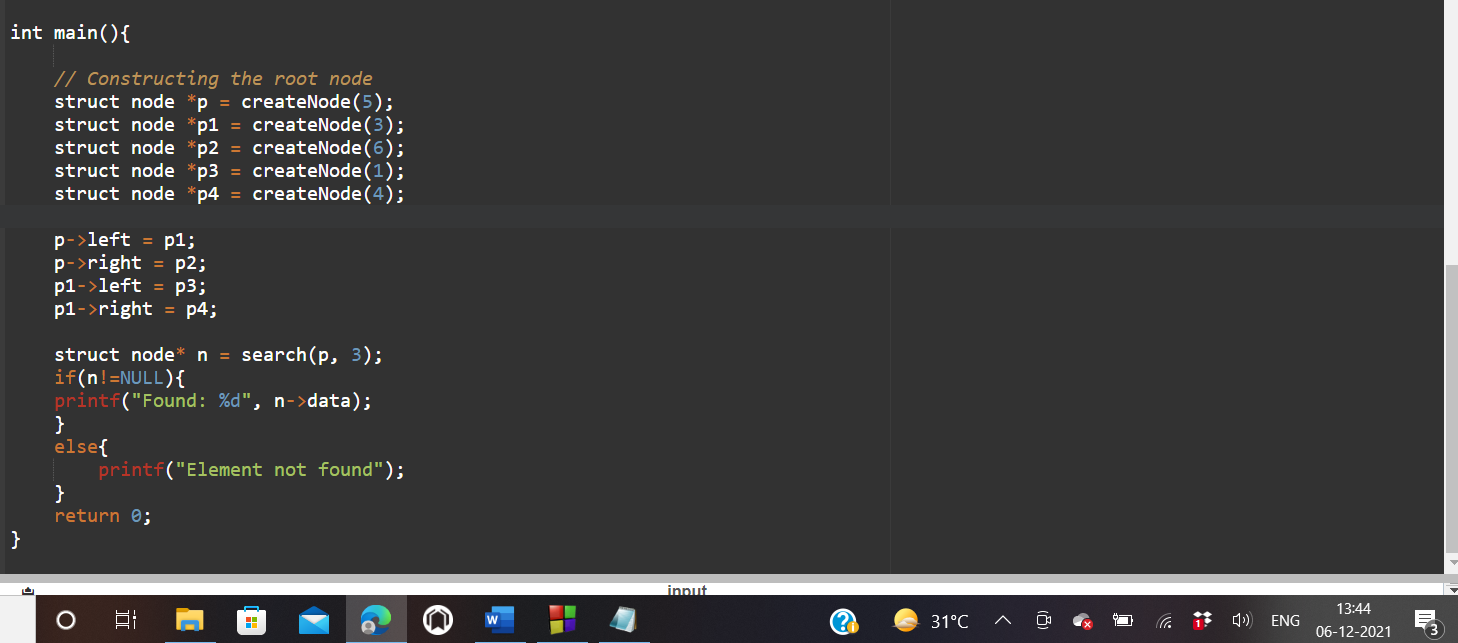
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROGRAM:

1. search:

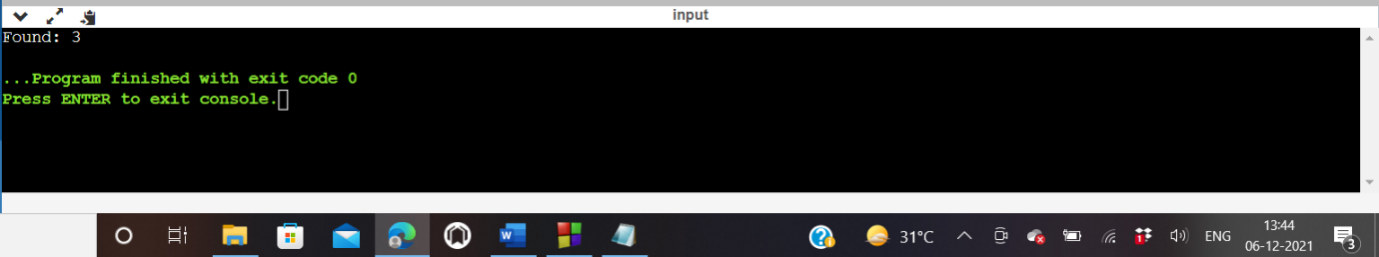






\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

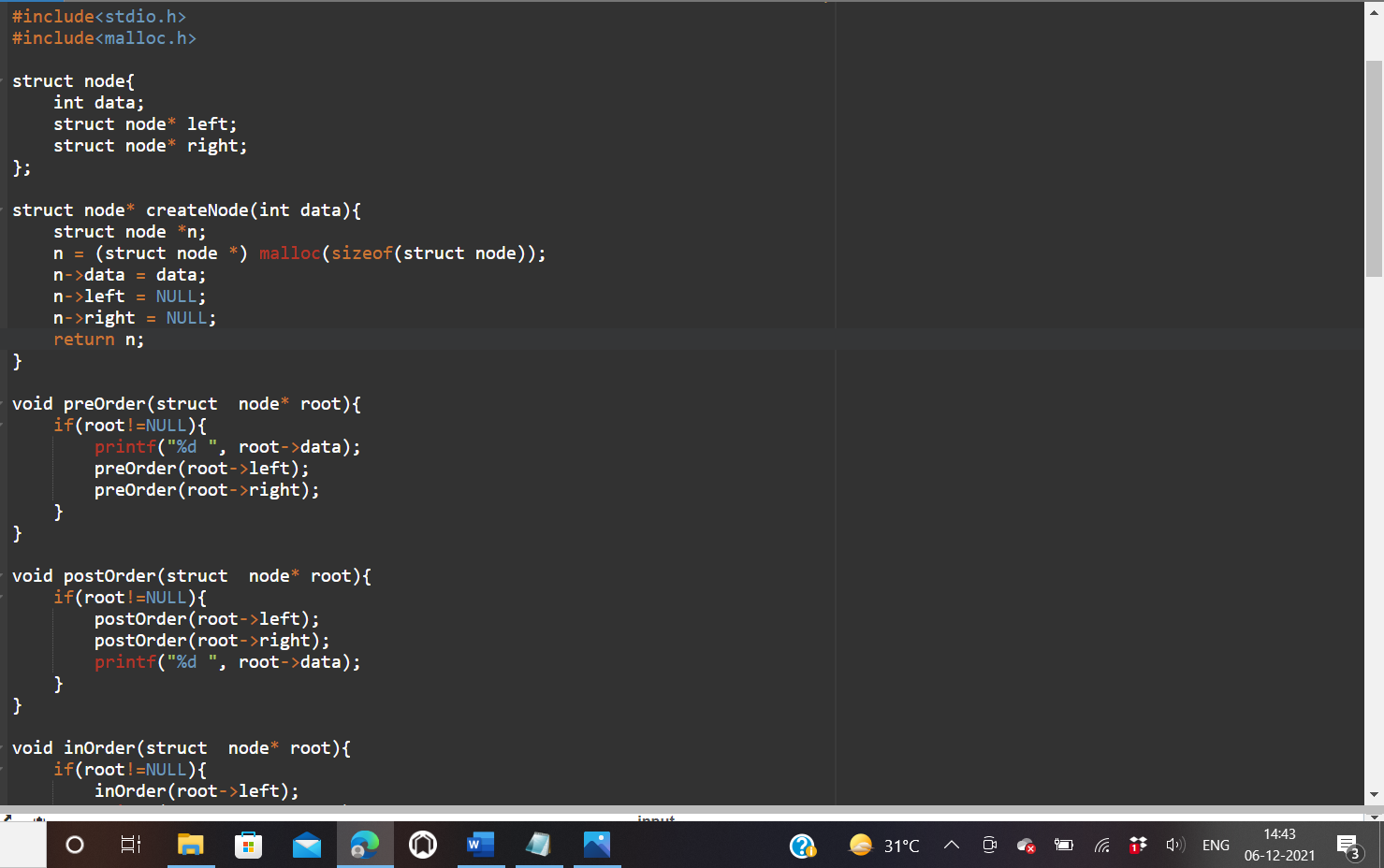
OUTPUT:

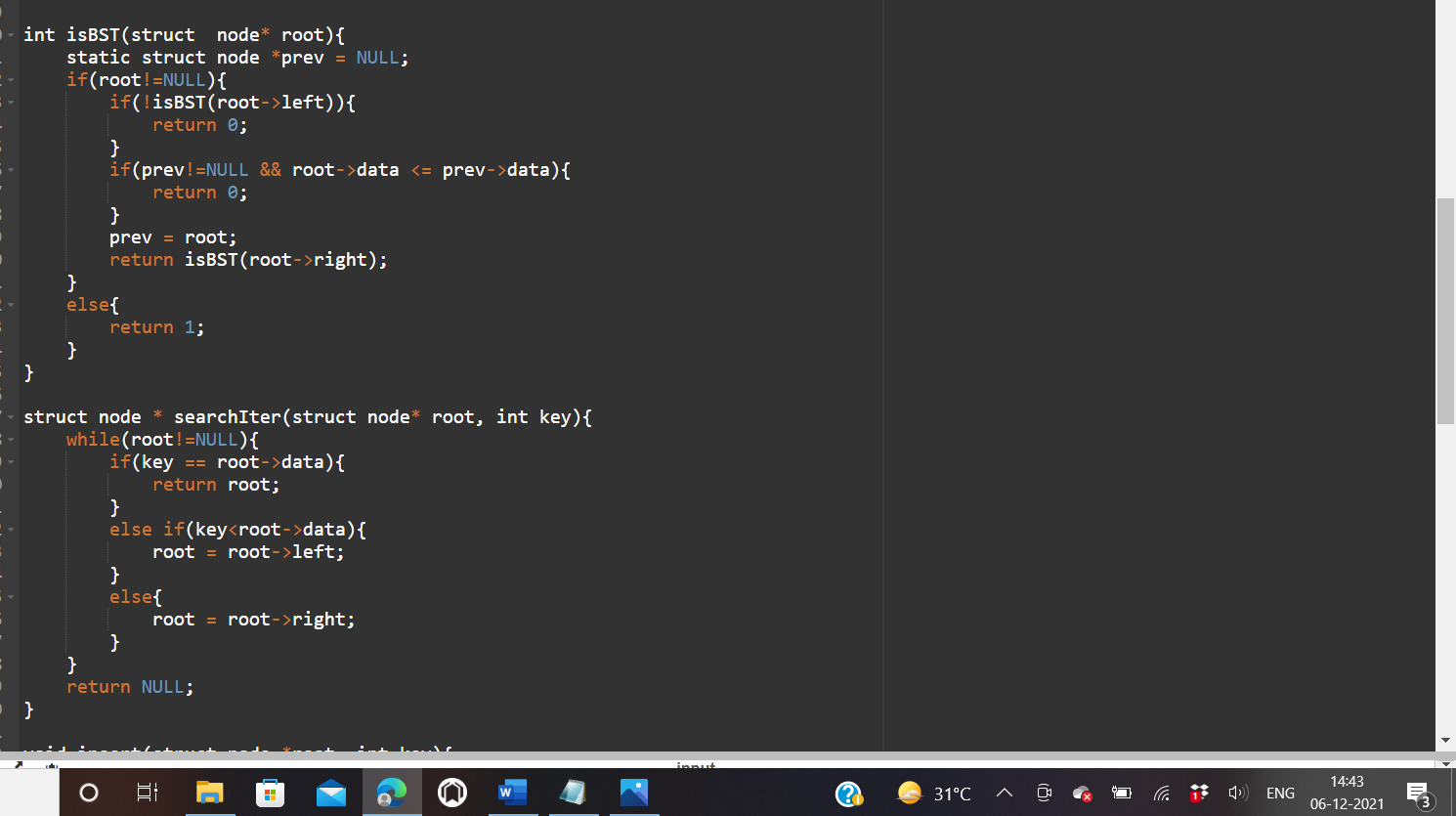


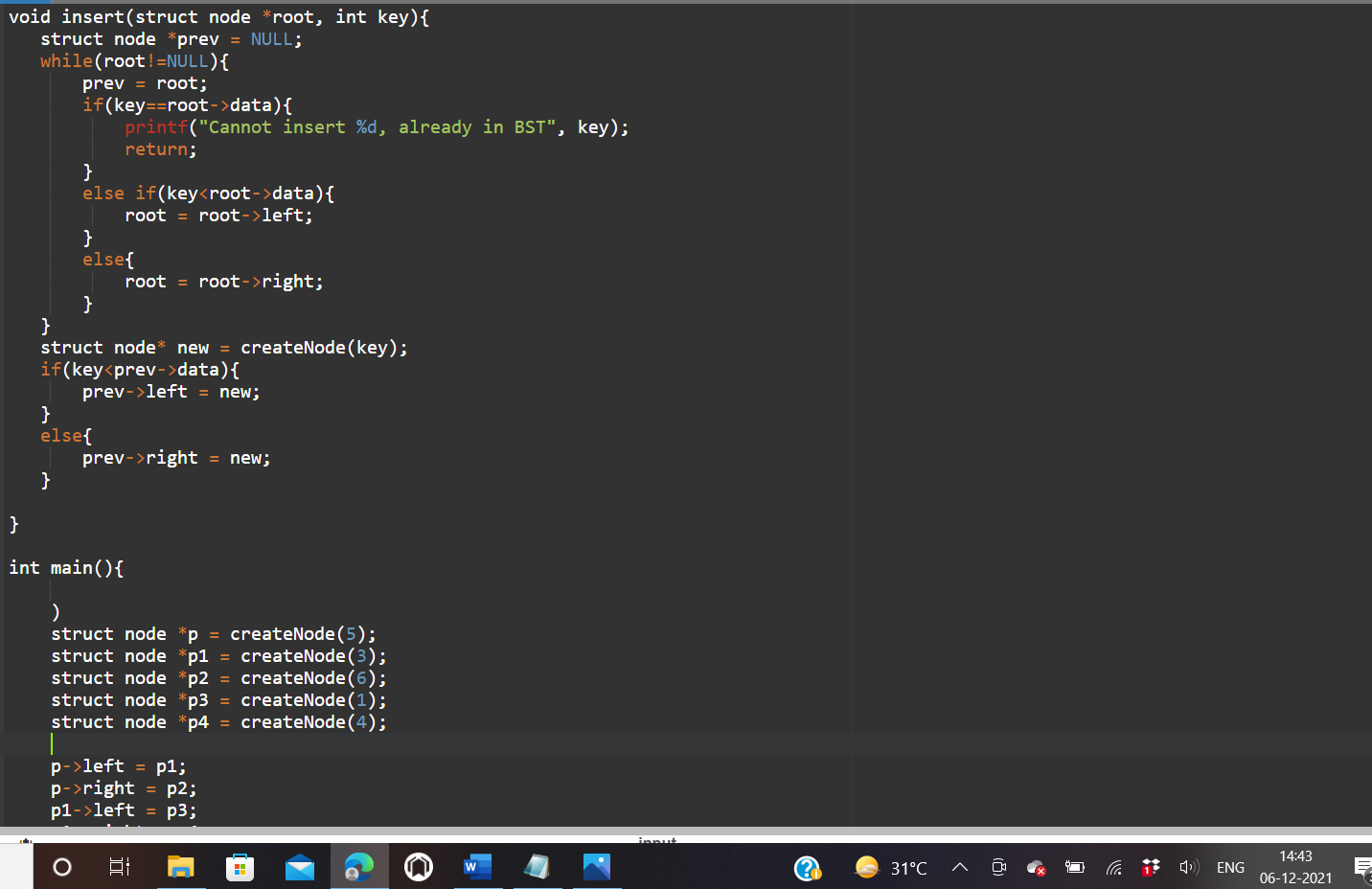
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

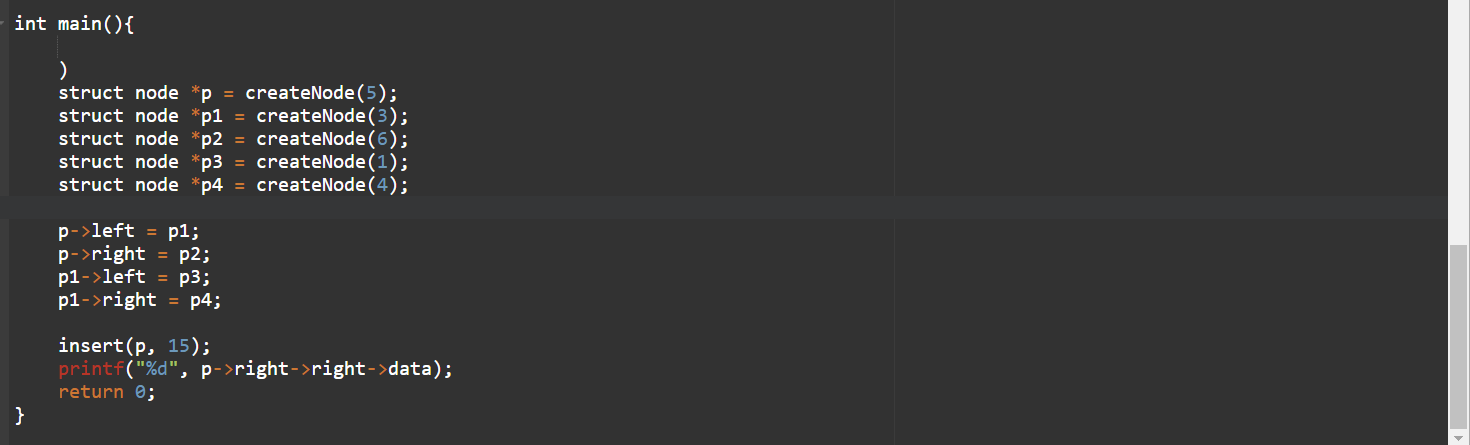
1. Insertion:

Program:



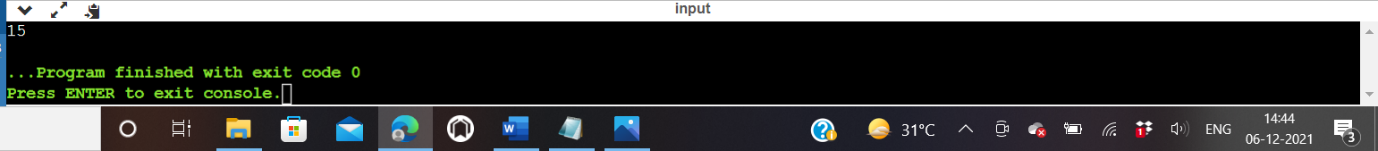






\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

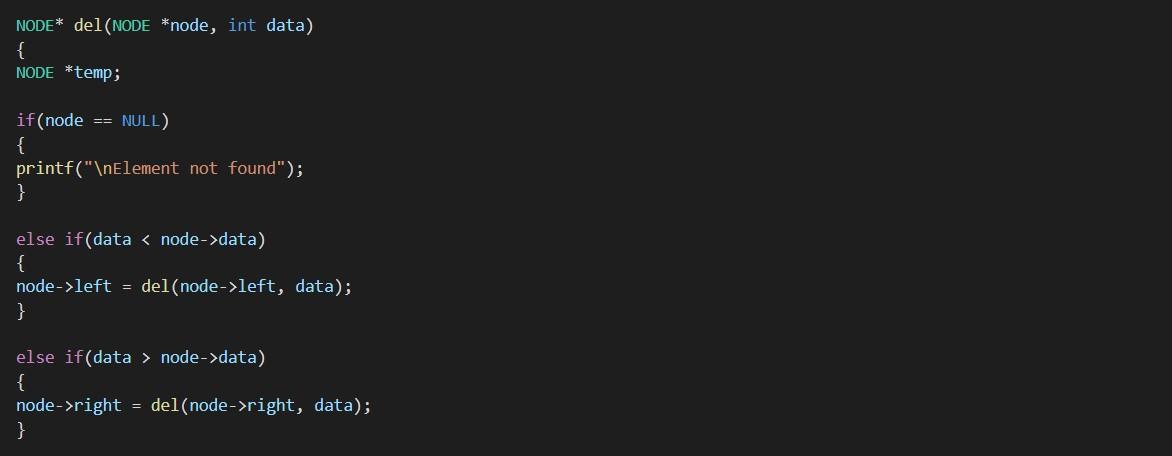
OUTPUT:

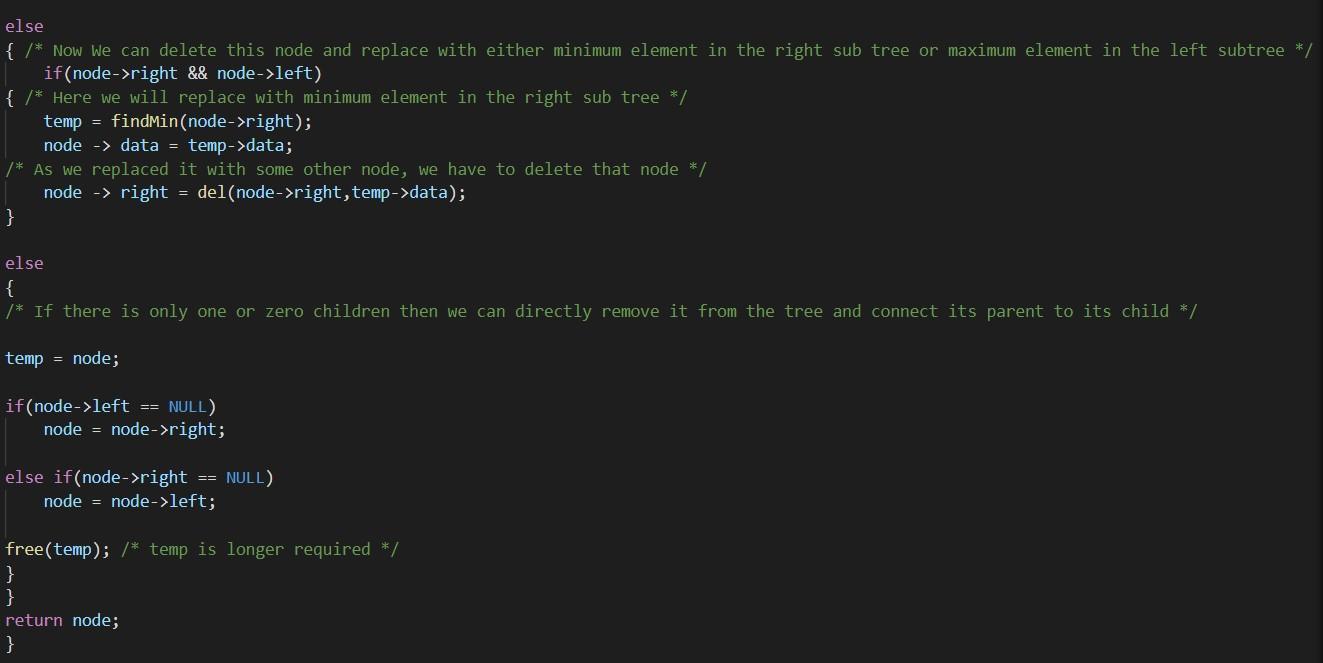


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Deletion

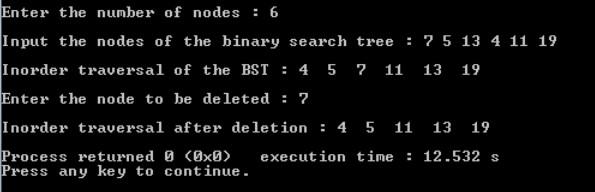
Program:





\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conclusion : We have successfully implemented a Binary Search Tree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_