Instructions: Please read carefully

- Please rename this file as only your ID number **(e.g. 18-*****-1.doc or 18-*****-1.pdf).**
- Submit the file by **Friday** in the Portal Lab Performance section labeled **Lab task 10. If you cannot complete the full task, do not worry. Just upload what you have completed.**

Name:- Amit Podder

ID:- 20-42273-1

Section:- [F]

1. **Write a C++ code to implement Binary Search Tree operations (insertion, traversal and searching)**

   Do the following to write program for a BST:

   1. To construct a binary search tree of integers (**insert** one by one).
   2. To **traverse** the tree using all the methods i.e., in order, preorder and post order.
   3. To **search** an element on the BST.
   4. There are three cases when you delete a node.
      - Case 1: Node with zero child (Leaf node)
      - Case 2: Node with one child
      - Case 3: Node with both children

      Implement the logic of 3 cases one by one.

Hint: Your program should ask the user to input the choice what operation the user wants to perform.

1. Insert
2. Travers
3. Search
4. Delete

```
Class Node{
       Data
       lptr , rptr
       Node(){
       Everything null
       }
       Node(int a){
       Data =a
       Everything else null
        }
}
```

```
Class BST{
    root
    BST(){
     Root = NULL
}
insert(x){
  if(Root == null){
    Node * nptr = new Node();
    Nptr->Data = x;
    Root = nptr;
……}// for inserting root node
  else {………..}// for inserting rest of the nodes
}
Preorder(Node * tptr){
    If(tptr !=Null){
        Print(tptr->Data)
        Preorder(tptr->lptr)
        Preorder(tptr->rptr)

}
}
}
```

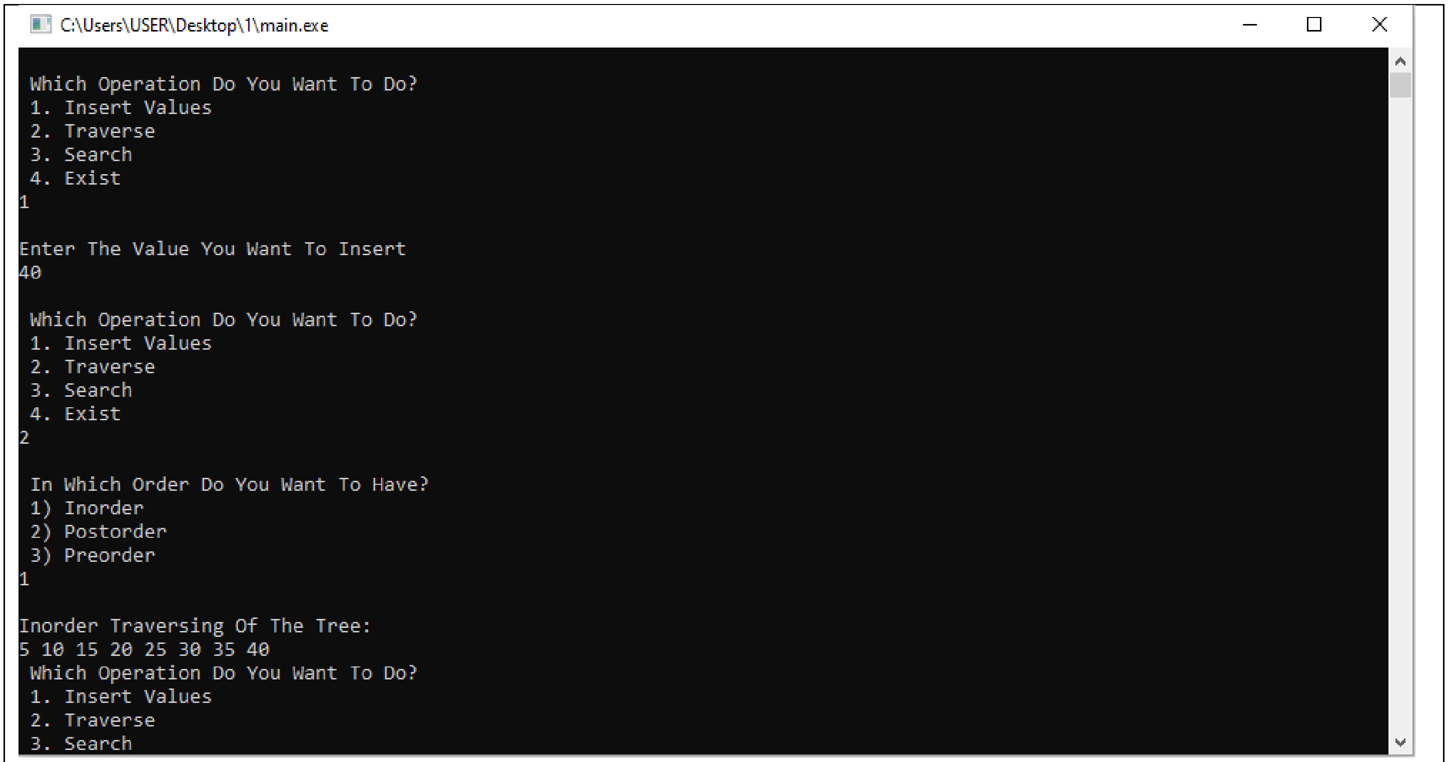
```
main(){
BST b
b.insert(10)
b.insert(20)
b.Preorder(b.root)
}
```

Your code here:-

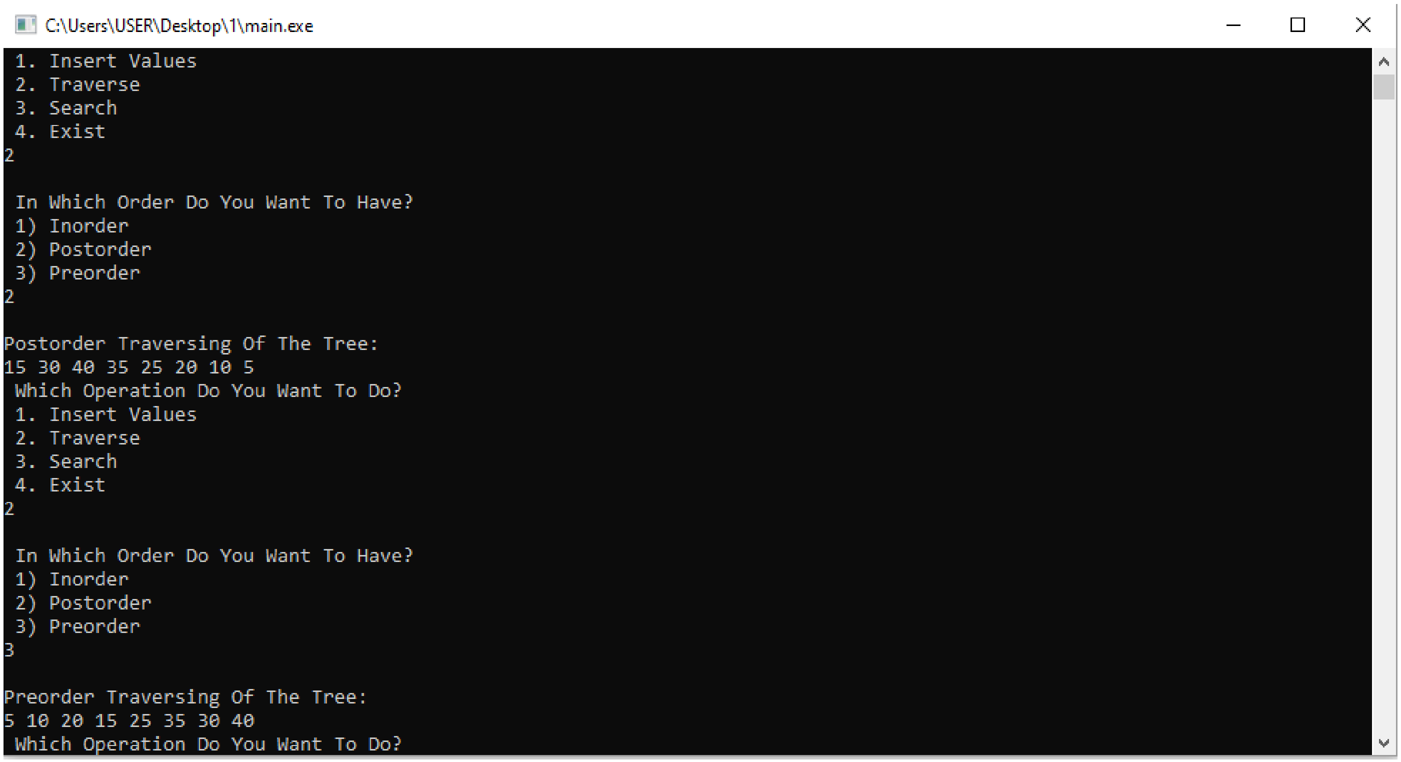
```
#include <iostream>

using namespace std;

struct node
{
   int value;
   node *left;
   node *right;
};

struct node *root=NULL;

class BinarySearchTree
{
public:
   void insert(int x, node *leaf);
   void inorder(node *leaf);
   void postorder(node *leaf);
   void preorder(node *leaf);
};

void BinarySearchTree :: insert(int x, node *leaf)
{
   if(root == NULL)
   {
      root= new node;
      root->value= x;
      root->left= NULL;
      root->right= NULL;
   }
   else
   {
      if(x < leaf->value)
      {
         if(leaf->left != NULL)
         {
            insert(x, leaf->left);
         }
         else
         {
            leaf->left= new node;
            leaf->left->value= x;
            leaf->left->left= NULL;
            leaf->left->right= NULL;
         }
      }
      else if(x > leaf->value)
      {
         if(leaf->right != NULL)
         {
            insert(x, leaf->right);
         }
         else
         {
            leaf->right= new node;
            leaf->right->value= x;
            leaf->right->right= NULL;
            leaf->right->left= NULL;
         }
      }
   }
}

void BinarySearchTree :: inorder(node *leaf)
{
   if(leaf != NULL)
   {
      inorder(leaf->left);
      cout<< leaf->value << " ";
      inorder(leaf->right);
   }
}

void BinarySearchTree :: postorder(node *leaf)
{
   if(leaf != NULL)
   {
      postorder(leaf->left);
      postorder(leaf->right);
      cout<< leaf->value << " ";
   }
}

void BinarySearchTree :: preorder(node *leaf)
{
   if(leaf != NULL)
   {
      cout<< leaf->value << " ";
      preorder(leaf->left);
      preorder(leaf->right);
   }
}

bool Search(node *root, int item)
{
   while(root != NULL)
   {
      if(item > root->value)
         root= root->right;
```

```
     else if(item < root->value)
        root= root->left;
     else
        return true;
   }
   return false;
}

int main()
{
  BinarySearchTree BST;

  BST.insert(5, root);
  BST.insert(10, root);
  BST.insert(20, root);
  BST.insert(25, root);
  BST.insert(15, root);
  BST.insert(35, root);
  BST.insert(30, root);

  int option, value, op;

  repeat:
    cout<<endl;
    cout<<" Which Operation Do You Want To Do?"<<endl;
    cout<<" 1. Insert Values"<<endl;
    cout<<" 2. Traverse"<<endl;
    cout<<" 3. Search"<<endl;
    cout<<" 4. Exist"<<endl;

    cin>> option;
    cout<<endl;

    if(option == 1)
    {
      cout<<"Enter The Value You Want To Insert "<<endl;
      cin>> value;
      BST.insert(value, root);
    }
    else if(option == 2)
    {
      cout<<" In Which Order Do You Want To Have?"<<endl;
      cout<<" 1) Inorder"<<endl;
      cout<<" 2) Postorder"<<endl;
      cout<<" 3) Preorder"<<endl;

      cin>> op;
      cout<<endl;

      if(op == 1)
      {
        cout<<"Inorder Traversing Of The Tree: "<<endl;
        BST.inorder(root);
        }
        else if(op == 2)
        {
            cout<<"Postorder Traversing Of The Tree: "<<endl;
            BST.postorder(root);
        }
        else if(op == 3)
        {
            cout<<"Preorder Traversing Of The Tree: "<<endl;
            BST.preorder(root);
        }
    }
    else if(option == 3)
    {
        cout<<" Enter The Value You Want To Search: "<<endl;
        int n;
        cin>> n;
        {
            if(Search(root, n))
             cout<<"The Value Is Here"<<endl;
            else
            {
                cout<<"The Value Is Not Here"<<endl;
            }
        }
    }
    else if(option == 4)
    {
        return 0;
    }
    else
    {
        cout<<"Wrong Option"<<endl;
        return 0;
    }
    if(option != 4)
    {
        goto repeat;
    }

    return 0;
}
```

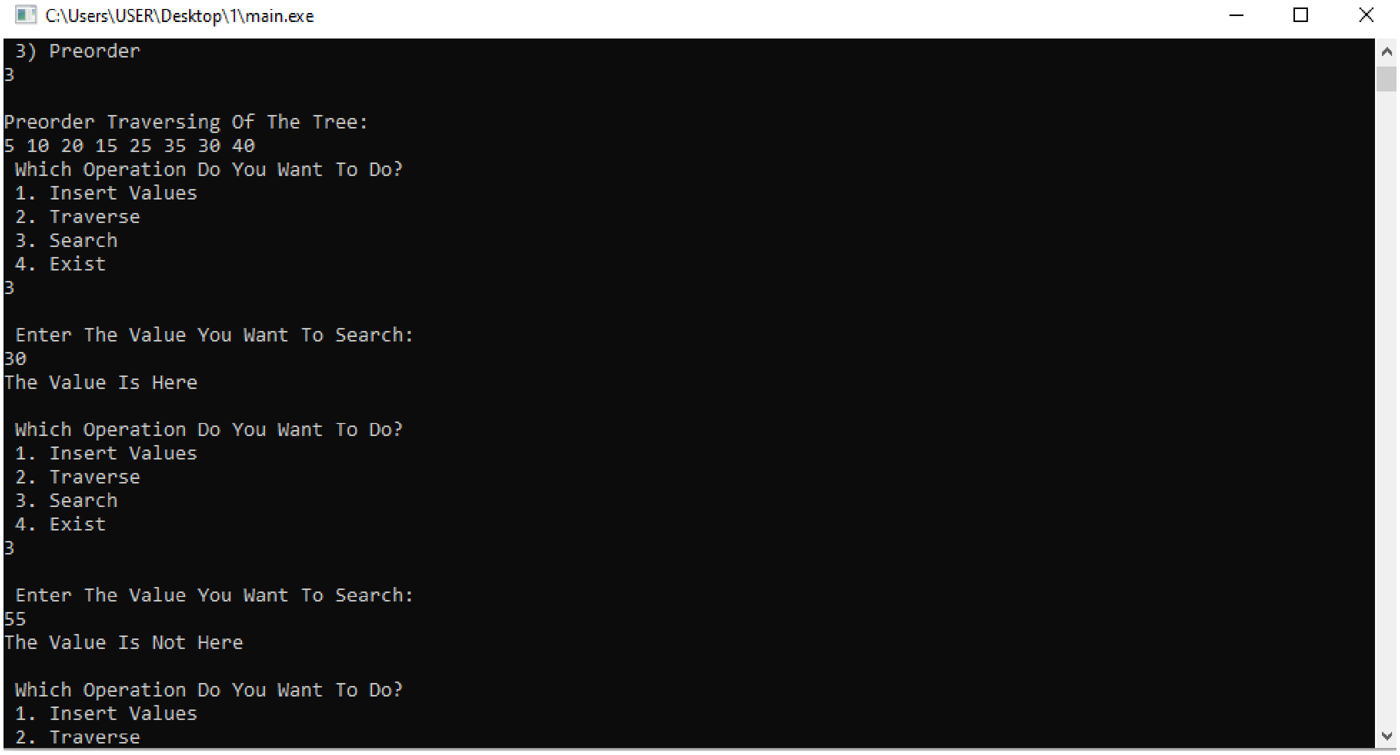
<u>Your whole Screenshot here: (Console Output):-</u>



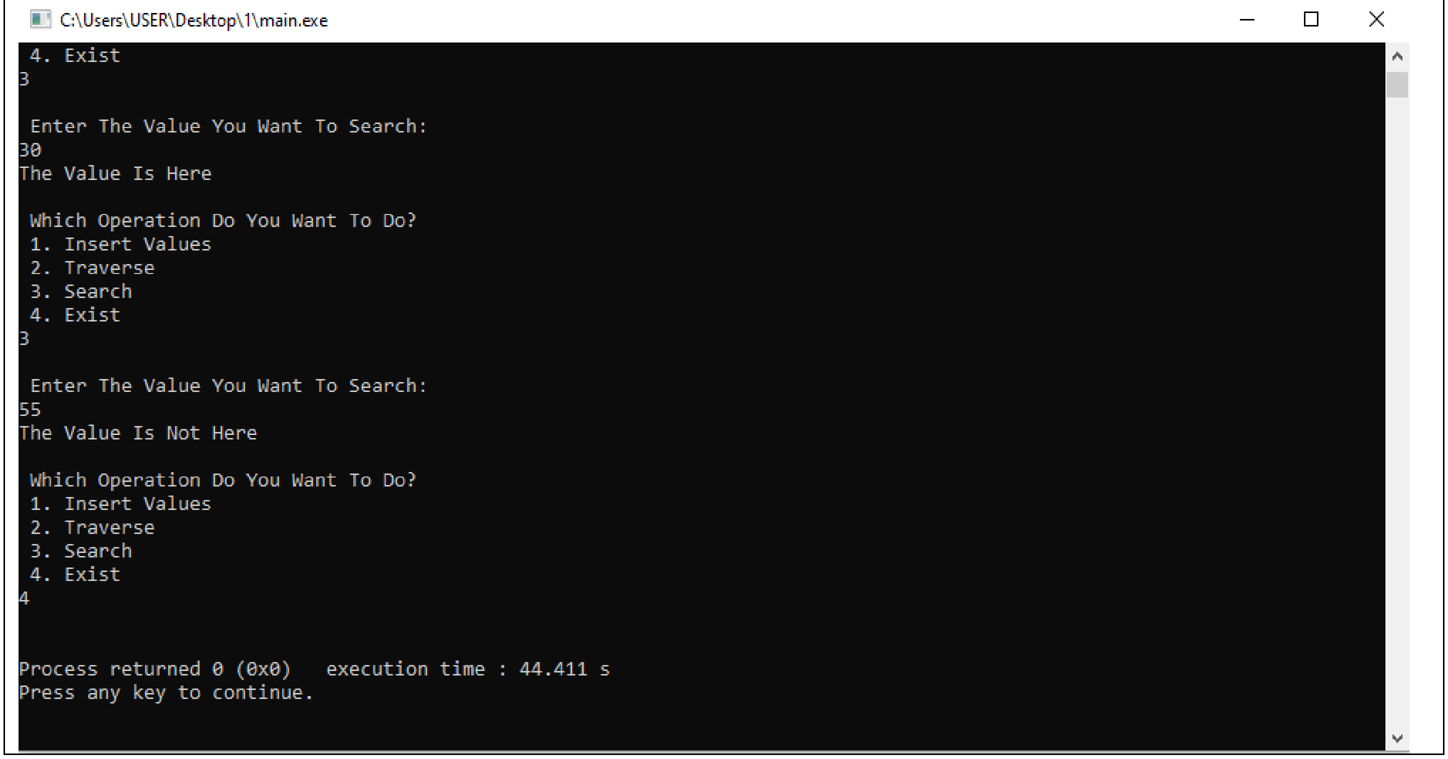
```

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
1

Enter The Value You Want To Insert
40

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
2

 In Which Order Do You Want To Have?
 1) Inorder
 2) Postorder
 3) Preorder
1

Inorder Traversing Of The Tree:
5 10 15 20 25 30 35 40
 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
```



```
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
2

 In Which Order Do You Want To Have?
 1) Inorder
 2) Postorder
 3) Preorder
2

Postorder Traversing Of The Tree:
15 30 40 35 25 20 10 5
 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
2

 In Which Order Do You Want To Have?
 1) Inorder
 2) Postorder
 3) Preorder
3

Preorder Traversing Of The Tree:
5 10 20 15 25 35 30 40
 Which Operation Do You Want To Do?
```



```
 3) Preorder
3

Preorder Traversing Of The Tree:
5 10 20 15 25 35 30 40
 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
3

 Enter The Value You Want To Search:
30
The Value Is Here

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
3

 Enter The Value You Want To Search:
55
The Value Is Not Here

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
```



```
 4. Exist
3

 Enter The Value You Want To Search:
30
The Value Is Here

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
3

 Enter The Value You Want To Search:
55
The Value Is Not Here

 Which Operation Do You Want To Do?
 1. Insert Values
 2. Traverse
 3. Search
 4. Exist
4


Process returned 0 (0x0)   execution time : 44.411 s
Press any key to continue.
```