Name: - Atish Kumar

Roll No: - 120CS0173

Lab Sheet: - 09

Q1. Write a program to implement a binary search tree (BST) having following functionalities.

- BSTInsert(): This function adds a given ITEM to the BST. If the ITEM already exists in the BST then it will not insert the ITEM any more.
- BSTInorderStack(): This function finds Inorder traversal sequence of a BST using stack. You are not supposed to use recursive implementation of Inorder traversal.

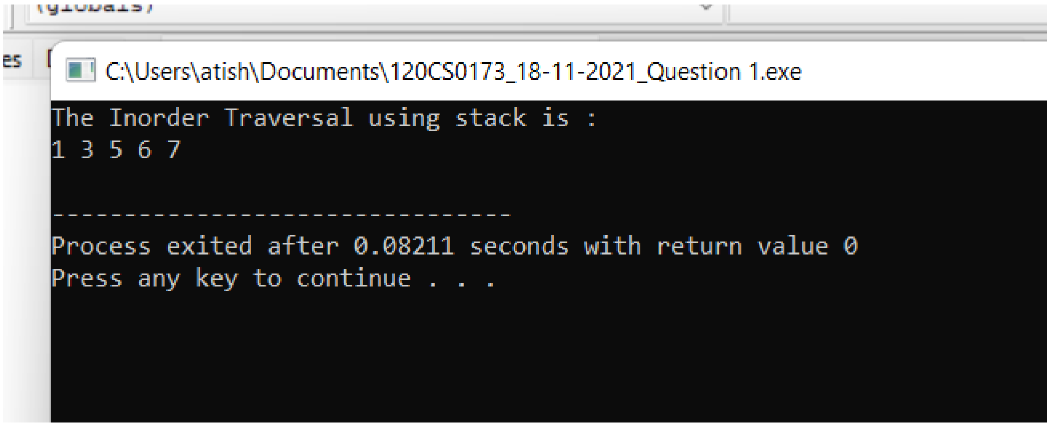
Program:-

```
#include <iostream>
#include <stack>
using namespace std;
struct node {
    int data;
    node *left;
    node *right;
    node(int key)
    {
        data = key;
        left = NULL;
        right = NULL;
    }
};

node *InsertInBST(node *root, int key) {
    node *temp = new node(key);
    if (root == NULL)
    {
        root = temp;
        return root;
    }
```

```
        else if (root->data == key)
        return root;
        else if (root->data > key)
        root->left = InsertInBST(root->left, key);
        else
        root->right = InsertInBST(root->right, key);

        return root;
}

void BSTInorderStack(node *root) {
        stack<node *> s;
        node *curr = root;
        while (curr != NULL || !s.empty())
        {
                while (curr != NULL)
                {
                        s.push(curr);
                        curr = curr->left;
                }
                while (!s.empty())
                {
                        node *temp = s.top();
                        s.pop();
                        cout << temp->data << " ";
                        curr = temp->right;
                }
        }
}

int main()
```

```
{
        node *root = NULL;
        root = InsertInBST(root, 5);
        root = InsertInBST(root, 3);
        root = InsertInBST(root, 7);
        root = InsertInBST(root, 6);
        root = InsertInBST(root, 1);
        cout << "The Inorder Traversal using stack is :" << endl;
        BSTInorderStack(root);
        cout << endl;
        return 0;
}
```

OutPut:-


C:\Users\atish\Documents\120CS0173_18-11-2021_Question 1.exe

```
The Inorder Traversal using stack is :
1 3 5 6 7

--------------------------------
Process exited after 0.08211 seconds with return value 0
Press any key to continue . . .
```

Q2. Write a program to perform deletion operation on BST. While deleting, consider all the deletion cases.

a. Deletion of node with degree 0.

b. Deletion of node with degree 1.

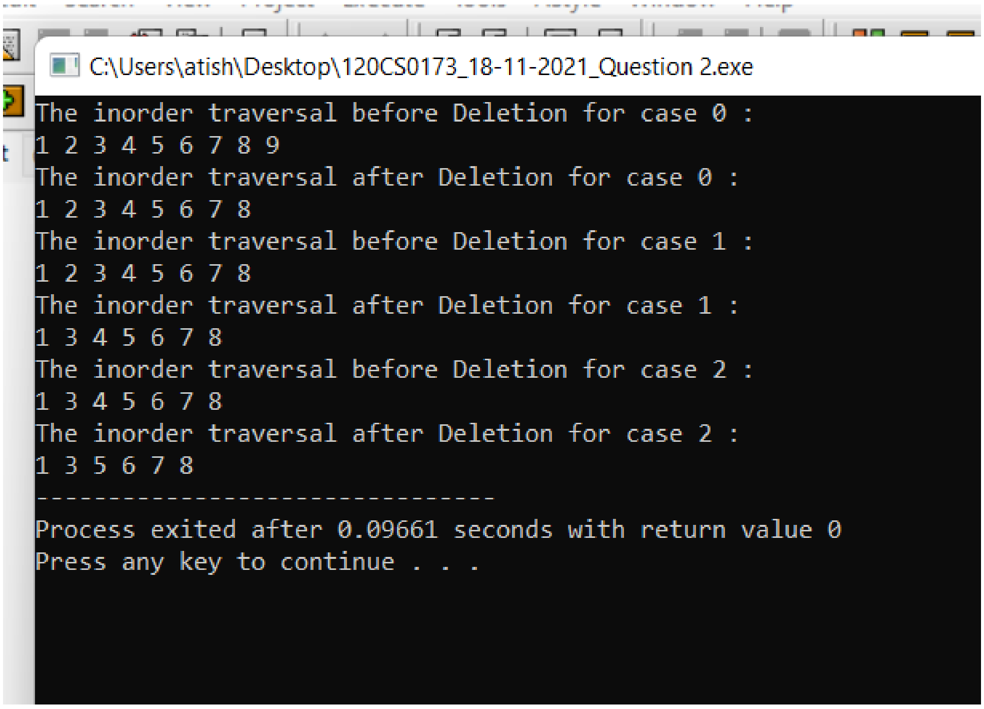
c. Deletion of node with degree 2.

Program:-

```
#include <iostream>
using namespace std;

struct node {
    int data;
    node *left;
    node *right;
    node(int key)
    {
        data = key;
        left = NULL;
        right = NULL;
    }
};

node *InorderPredecessor(node *root) {
    while(root->left != NULL)
    {
        root = root->left;
    }
    return root;
}

node *Deletion(node *root, int key) {
```

```
// Base case;
if(root == NULL)
return root;

if(root->data > key)
root->left = Deletion(root->left, key);
else if (root->data < key)
root->right = Deletion(root->right, key);

// Deletion of node with Degree 0;
else if (root->data == key && root->left == NULL && root->right == NULL)
return NULL;

// Deletion of node with Degree 1;
else if (root->data == key && root->left && root->right == NULL)
{
        root->data = root->left->data;
        root->left = Deletion(root->left, root->left->data);
}
else if (root->data == key && root->right && root->right->right == NULL && root->left->left == NULL)
{
        root->data = root->right->data;
        root->right = Deletion(root->right, root->right->data);
}

// Deletion of node with Degree 2 and >2;
else if (root->data == key)
{
        node *temp = InorderPredecessor(root->right);
        root->data = temp->data;
        root->right = Deletion(root->right, temp->data);
    }
    return root;
}

void Inorder(node *root) {
    if (root == NULL)
    return;

    Inorder(root->left);
    cout << root->data << " ";
    Inorder(root->right);
}

int main() {
    node *root = NULL;
    root = new node(5);
    root->left = new node(3);
    root->right = new node(7);
    root->left->left = new node(2);
    root->left->right = new node(4);
    root->right->left = new node(6);
    root->right->right = new node(8);
    root->left->left->left = new node(1);
    root->right->right->right = new node(9);
    cout << "The inorder traversal before Deletion for case 0 :" << endl;
    Inorder(root);
    cout << endl;
    root = Deletion(root, 9);
    cout << "The inorder traversal after Deletion for case 0 :" << endl;
    Inorder(root);
    cout << endl;
    cout << "The inorder traversal before Deletion for case 1 :" << endl;
    Inorder(root);
    cout << endl;
    root = Deletion(root, 2);
    cout << "The inorder traversal after Deletion for case 1 :" << endl;
    Inorder(root);
    cout << endl;
    cout << "The inorder traversal before Deletion for case 2 :" << endl;
    Inorder(root);
    cout << endl;
    root = Deletion(root, 4);
    cout << "The inorder traversal after Deletion for case 2 :" << endl;
    Inorder(root);
    return 0;
}
```

Program:-



```
The inorder traversal before Deletion for case 0 :
1 2 3 4 5 6 7 8 9
The inorder traversal after Deletion for case 0 :
1 2 3 4 5 6 7 8
The inorder traversal before Deletion for case 1 :
1 2 3 4 5 6 7 8
The inorder traversal after Deletion for case 1 :
1 3 4 5 6 7 8
The inorder traversal before Deletion for case 2 :
1 3 4 5 6 7 8
The inorder traversal after Deletion for case 2 :
1 3 5 6 7 8
--------------------------------
Process exited after 0.09661 seconds with return value 0
Press any key to continue . . .
```