==

Roll Number: SYCOC303 Division: C

PRN Number: 122B2B303 Batch: C4

Name: VINAYAK MADAN SHETE

==

Problem Statement:

⇨ **Write a C++ program to implement a threaded binary tree and its traversal.**

==

INPUT:

```
/*
 *     ======================================
 *            Program Name: TBT.cpp
 *     Created on: December 05, 2022
 *      Author: Vinayak Shete
 *  ======================================
 */

#include <iostream>
#define MAX_VALUE 65536

using namespace std;

class Node
{
       public:
        int key;
        Node *left, *right;
        bool leftThread, rightThread;
};

class ThreadedBinarySearchTree
{
       private:
        Node *root;
    public:
        ThreadedBinarySearchTree()
        {
            root = new Node();
            root->right = root->left = root;
            root->leftThread = true;
            root->key = MAX_VALUE;
        }

//       Function to delete all elements from tree
        void makeEmpty()
        {
            root = new Node();
            root->right = root->left = root;
            root->leftThread = true;
            root->key = MAX_VALUE;
        }

//        Function to insert a key
        void insert(int key)
        {
            Node *p = root;
            for (;;)
            {
                if (p->key < key)
                {
                    if (p->rightThread)
                        break;
                    p = p->right;
                }
                else if (p->key > key)
                {
                    if (p->leftThread)
                        break;
                    p = p->left;
                }
                else
                {
                    return;
                }
            }
            Node *tmp = new Node();
            tmp->key = key;
            tmp->rightThread = tmp->leftThread = true;
            if (p->key < key)
            {
//                insert to right side
                tmp->right = p->right;
                tmp->left = p;
                p->right = tmp;
                p->rightThread = false;
            }
            else
            {
                tmp->right = p;
                tmp->left = p->left;
                p->left = tmp;
                p->leftThread = false;
            }
        }

//         Function to search for an element
        bool search(int key)
        {
            Node *tmp = root->left;
            for (;;)
            {
                if (tmp->key < key)
                {
                    if (tmp->rightThread)
                        return false;
                    tmp = tmp->right;
                }
                else if (tmp->key > key)
                {
                    if (tmp->leftThread)
                        return false;
                    tmp = tmp->left;
                }
                else
                {
                    return true;
                }
            }
        }

//        Fuction to delete an element
        void Delete(int key)
        {
            Node *dest = root->left, *p = root;
            for (;;)
            {
                if (dest->key < key)
                {
//                  not found
                    if (dest->rightThread)
                        return;
                    p = dest;
                    dest = dest->right;
                }
                else if (dest->key > key)
                {
//                  not found
                    if (dest->leftThread)
                        return;
                    p = dest;
                    dest = dest->left;
                }
                else
                {
//                  found
                    break;
                }
            }
            Node *target = dest;
            if (!dest->rightThread && !dest->leftThread)
            {
//                 dest has two children
                p = dest;
//                 find largest node at left child
                target = dest->left;
                while (!target->rightThread)
                {
                    p = target;
                    target = target->right;
                }
//                 using replace mode
                dest->key = target->key;
            }
            if (p->key >= target->key)
            {
                if (target->rightThread && target->leftThread)
                {
                    p->left = target->left;
                    p->leftThread = true;
                }
                else if (target->rightThread)
                {
                    Node *largest = target->left;
                    while (!largest->rightThread)
                    {
                        largest = largest->right;
                    }
                    largest->right = p;
                    p->left = target->left;
                }
```

```
    else
    {
        Node *smallest = target->right;
        while (!smallest->leftThread)
        {
            smallest = smallest->left;
        }
        smallest->left = target->left;
        p->left = target->right;
    }
}
else
{
    if (target->rightThread && target->leftThread)
    {
        p->right = target->right;
        p->rightThread = true;
    }
    else if (target->rightThread)
    {
        Node *largest = target->left;
        while (!largest->rightThread)
        {
            largest = largest->right;
        }
        largest->right =  target->right;
        p->right = target->left;
    }
    else
    {
        Node *smallest = target->right;
        while (!smallest->leftThread)
        {
            smallest = smallest->left;
        }
        smallest->left = p;
        p->right = target->right;
```

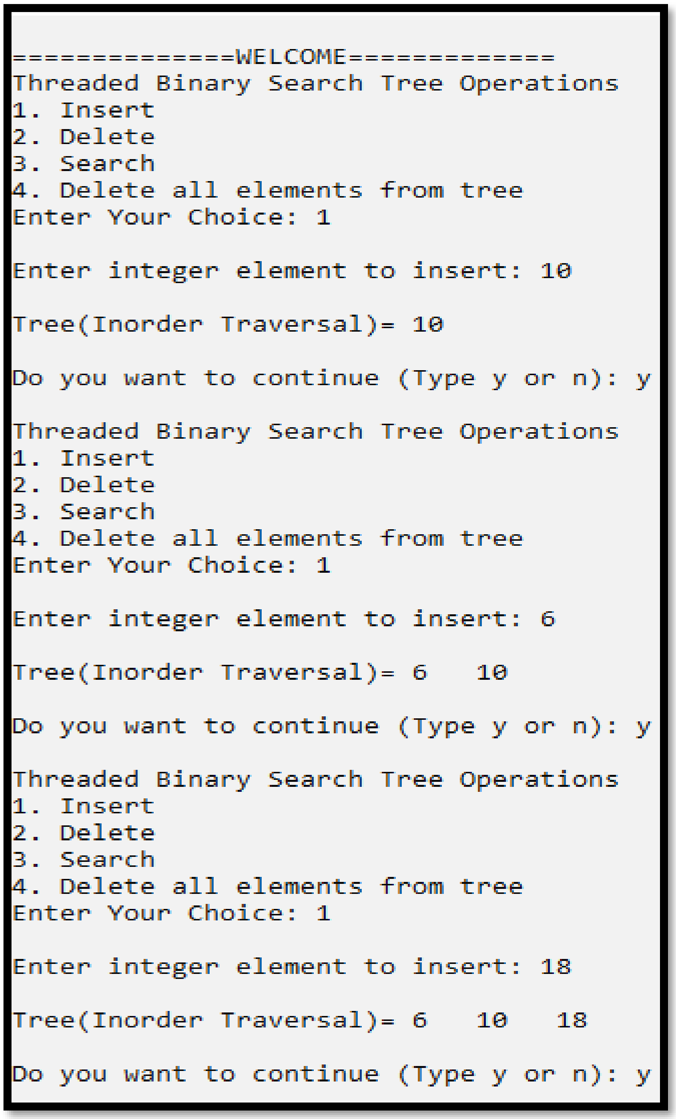
```
                }
            }
        }

        //printing the tree using inorder traversal
        void printTree()
        {
            Node *tmp = root, *p;
            for (;;)
            {
                p = tmp;
                tmp = tmp->right;
                if (!p->rightThread)
                {
                    while (!tmp->leftThread)
                    {
                        tmp = tmp->left;
                    }
                }
                if (tmp == root)
                    break;
                cout<<tmp->key<<"   ";
            }
            cout<<endl;
        }
};

int main()
{
    ThreadedBinarySearchTree tbst;
    char ch;
    int choice, val;
    cout<<"\n==============WELCOME=============";
    do
    {
        cout<<"\nThreaded Binary Search Tree Operations\n";
        cout<<"1. Insert "<<endl;
```

```
cout<<"2. Delete"<<endl;
cout<<"3. Search"<<endl;
cout<<"4. Delete all elements from tree"<<endl;
cout<<"Enter Your Choice: ";
cin>>choice;
switch (choice)
{
case 1 :
    cout<<"\nEnter integer element to insert: ";
    cin>>val;
    tbst.insert(val);
    break;
case 2 :
    cout<<"\nEnter integer element to delete: ";
    cin>>val;
    tbst.Delete(val);
    break;
case 3 :
    cout<<"\nEnter integer element to search: ";
    cin>>val;
    if (tbst.search(val) == true)
        cout<<"\nElement "<<val<<" found in the tree!"<<endl;
    else
        cout<<"\nElement "<<val<<" not found in the tree!"<<endl;
    break;
case 4 :
    cout<<"\nAll the elements from the tree have been deleted\n";
    tbst.makeEmpty();
    break;
default :
    cout<<"\nYou have entered wrong choice!\n ";
    break;
}
/*  Display tree  */
cout<<"\nTree(Inorder Traversal)= ";
tbst.printTree();
cout<<"\nDo you want to continue (Type y or n): ";
```

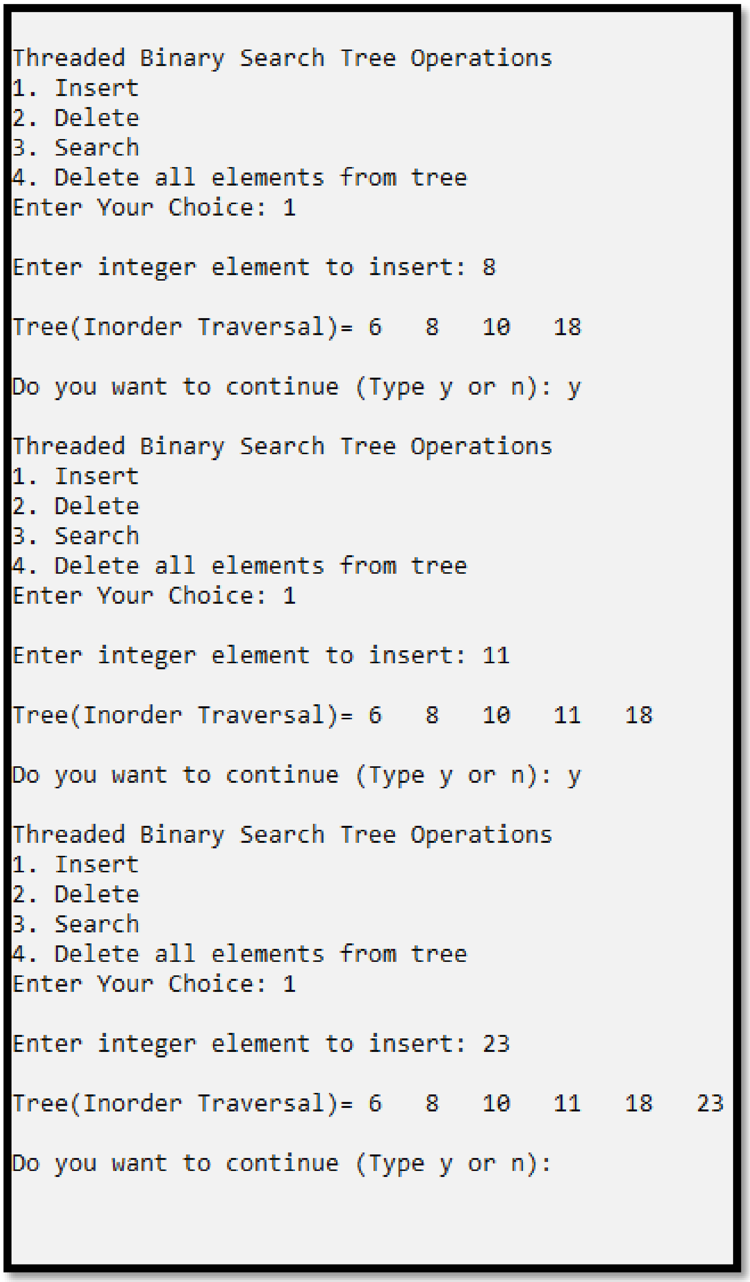
```
            cin>>ch;
        }
        while (ch == 'Y'|| ch == 'y');
          cout<<"\n=============THANK YOU=================";
        return 0;
}
```

OUTPUT:

Inserting elements into the Tree:

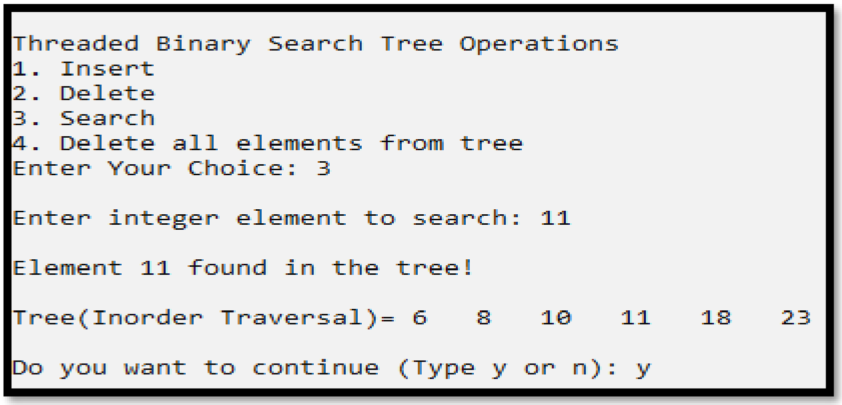


```
==============WELCOME=============
Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 10

Tree(Inorder Traversal)= 10

Do you want to continue (Type y or n): y

Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 6

Tree(Inorder Traversal)= 6   10

Do you want to continue (Type y or n): y

Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 18

Tree(Inorder Traversal)= 6   10   18

Do you want to continue (Type y or n): y
```



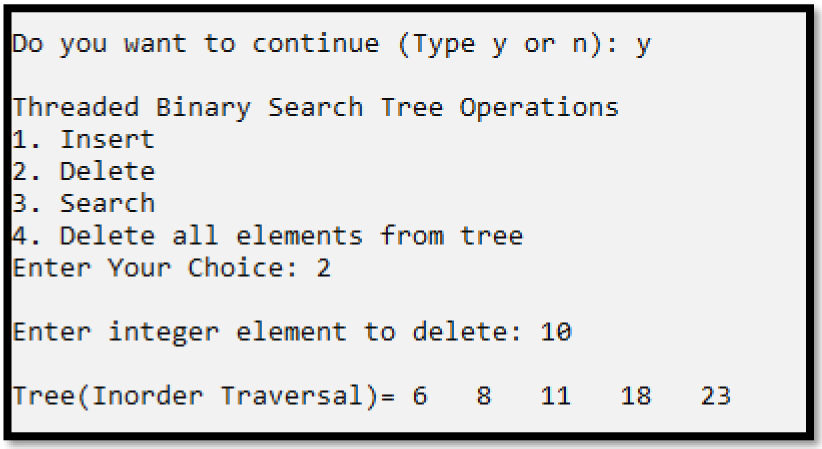
```
Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 8

Tree(Inorder Traversal)= 6   8   10   18

Do you want to continue (Type y or n): y

Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 11

Tree(Inorder Traversal)= 6   8   10   11   18

Do you want to continue (Type y or n): y

Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 1

Enter integer element to insert: 23

Tree(Inorder Traversal)= 6   8   10   11   18   23

Do you want to continue (Type y or n):
```

Searching elements into the Tree:



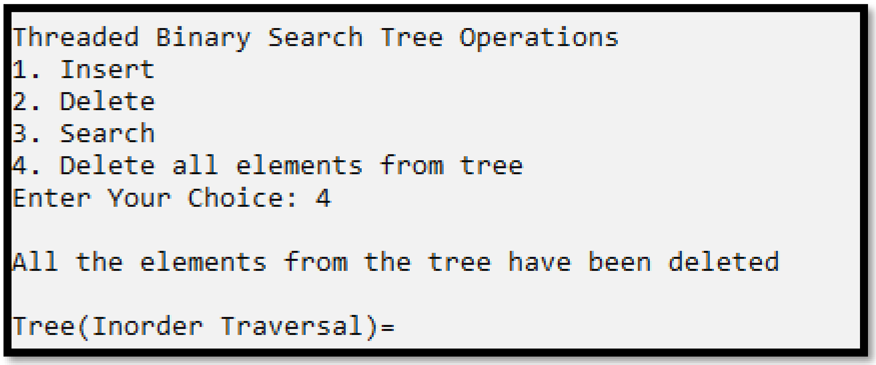
```
Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 3

Enter integer element to search: 11

Element 11 found in the tree!

Tree(Inorder Traversal)= 6    8    10    11    18    23

Do you want to continue (Type y or n): y
```

Deleting elements from the Tree:



```
Do you want to continue (Type y or n): y

Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 2

Enter integer element to delete: 10

Tree(Inorder Traversal)= 6    8    11    18    23
```

Deleting all elements from the Tree:



```
Threaded Binary Search Tree Operations
1. Insert
2. Delete
3. Search
4. Delete all elements from tree
Enter Your Choice: 4

All the elements from the tree have been deleted

Tree(Inorder Traversal)=
```

==