Name: Pratik Rameshwar Shinde

Roll No.: 2274

Class: SE-IT

Div: B-3

Subject: DSA LAB

Practical 6: Threaded Binary tree

Aim: Implement In-order Threaded Binary Tree and traverse it in In-order and Pre-order.

**********PROGRAM**********

```
#include <iostream>
#include <cstdlib>
#define MAX_VALUE 65536 using
namespace std;

/* Class Node */

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

/* Class ThreadedBinarySearchTree */

class ThreadedBinarySearchTree
{
        private:
Node *root;    public:
    /* Constructor */
    ThreadedBinarySearchTree()
    {
```

```
        root = new Node();          root->right = root->left = root;          root->leftThread = true;
        root->key = MAX_VALUE;
    }

    /* Function to clear 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
            {
               /* redundant key */
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;
            }
        }
    }
     /* Function to print tree */
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;
    }
};
 /* Main Contains Menu */ int
main()
{
```

```
   ThreadedBinarySearchTree tbst;
cout<<"\nThreadedBinarySearchTree Test\n";
char ch;    int choice, val;
   /*  Perform tree operations  */    do
   {
      cout<<"\nThreadedBinarySearchTree Operations\n";
cout<<"1. Insert "<<endl;        cout<<"2. Delete"<<endl;
cout<<"3. Search"<<endl;        cout<<"4. Clear"<<endl;
cout<<"Enter Your Choice: ";        cin>>choice;
      switch (choice)
      {
case 1 :
         cout<<"Enter integer element to insert: ";
cin>>val;
         tbst.insert(val);
break;                          case 2 :
         cout<<"Enter integer element to delete: ";
cin>>val;
         tbst.Delete(val);
break;                          case 3 :
         cout<<"Enter integer element to search: ";
cin>>val;
         if (tbst.search(val) == true)
            cout<<"Element "<<val<<" found in the tree"<<endl;           else
            cout<<"Element   "<<val<<"   not   found   in   the   tree"<<endl;
break;             case 4 :
         cout<<"\nTree Cleared\n";
         tbst.makeEmpty();
break;                default :
         cout<<"Wrong Entry \n ";
break;
      }
      /*  Display tree  */        cout<<"\nTree
= ";
      tbst.printTree();
      cout<<"\nDo you want to continue (Type y or n): ";
      cin>>ch;
   }
   while (ch == 'Y'|| ch == 'y');                return 0;
```

```
}
*****************************OUTPUT**************
*****************

[admin@fedora ~]$ g++ hfb6.cpp
[admin@fedora ~]$ ./a.out
ThreadedBinarySearchTree Test

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear
Enter Your Choice: 1
Enter integer element to insert: 21

Tree = 21

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear
Enter Your Choice: 1
Enter integer element to insert: 2

Tree = 2   21

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear
```

Enter Your Choice: 1
Enter integer element to insert: 56
Tree = 2 21 56

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear
Enter Your Choice: 2
Enter integer element to delete: 21

Tree = 2 56

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear
Enter Your Choice: 3
Enter integer element to search:
56
Element 56 found in the tree

Tree = 2 56

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear

Enter Your Choice: 8
Wrong Entry

Tree = 2 56

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

ThreadedBinarySearchTree Operations
1. Insert
2. Delete
3. Search
4. Clear

Enter Your Choice: 4

Tree Cleared

Tree =

Do you want to continue (Type y or n): n
[admin@fedora ~]$