Course Name: **A Level (2nd Sem)**
Topic: **Threaded binary tree**

Subject: **Data Structure using C++**
Date: **20-05-2020**

Threaded binary tree is a binary tree that provides the facility to traverse the tree in a particular order. It makes inorder traversal faster and do it without stack and without recursion. There are two types of threaded binary trees.

Single Threaded Each node is threaded towards either left or right means in-order predecessor or successor. Here, all right null pointers will point to inorder successor or all left null pointers will point to inorder predecessor.

Double threaded Each node is threaded towards either left and right means in-order predecessor and successor. Here, all right null pointers will point to inorder successor and all left null pointers will point to inorder predecessor.

```
#include <iostream>
#include <cstdlib>
#define MAX_VALUE 65536
using namespace std;
class N
{
    public:
    int k;
    N *l, *r;
   bool leftTh, rightTh;
};
class ThreadedBinaryTree
{
  private:
  N *root;
  public:
  ThreadedBinaryTree()
  {
    root= new N();
    root->r= root->l= root;
    root->leftTh = true;
    root->k = MAX_VALUE;
  }
  void makeEmpty()
  {
    root= new N();
    root->r = root->l = root;
    root->leftTh = true;
    root->k = MAX_VALUE;
  }
  void insert(int key)
  {
    N *p = root;
    for (;;) {
      if (p->k< key) { / /move to right thread
        if (p->rightTh)
          break;
        p = p->r;
```

```
        } else if (p->k > key) { // move to left thread
          if (p->leftTh)
            break;
          p = p->l;
        } else {
          return;
        }     }
      N *temp = new N();
      temp->k = key;
      temp->rightTh= temp->leftTh= true;
      if (p->k < key) {
        temp->r = p->r;
        temp->l= p;
        p->r = temp;
        p->rightTh= false;
      } else {
        temp->r = p;
        temp->l = p->l;
        p->l = temp;
        p->leftTh = false;
      }   }
    bool search(int key) {
      N *temp = root->l;
      for (;;) {
      if (temp->k < key) { //search in left thread
      if (temp->rightTh)
          return false;
        temp = temp->r;
      } else if (temp->k > key) { //search in right thread
        if (temp->leftTh)
          return false;
        temp = temp->l;
      } else {
        return true;
      }   }}
  void Delete(int key) {
    N *dest = root->l, *p = root;
    for (;;) { //find Node and its parent.
      if (dest->k < key) {
        if (dest->rightTh)
          return;
        p = dest;
        dest = dest->r;
      } else if (dest->k > key) {
        if (dest->leftTh)
          return;
        p = dest;
        dest = dest->l;
      } else {
        break;
      }
    }
    N *target = dest;
    if (!dest->rightTh && !dest->leftTh) {
      p = dest;  //has two children
      target = dest->l;   //largest node at left child
```

```
    while (!target->rightTh) {
      p = target;
      target = target->r;
    }
    dest->k= target->k; //replace mode
  }
  if (p->k >= target->k) { //only left child
    if (target->rightTh && target->leftTh) {
      p->l = target->l;
      p->leftTh = true;
    } else if (target->rightTh) {
      N*largest = target->l;
      while (!largest->rightTh) {
        largest = largest->r;
      }
      largest->r = p;
      p->l= target->l;
    } else {
      N *smallest = target->r;
      while (!smallest->leftTh) {
        smallest = smallest->l;
      }
      smallest->l = target->l;
      p->l = target->r;
    }
  } else {//only right child
    if (target->rightTh && target->leftTh) {
      p->r= target->r;
      p->rightTh = true;
    } else if (target->rightTh) {
      N *largest = target->l;
      while (!largest->rightTh) {
        largest = largest->r;
      }
      largest->r= target->r;
      p->r = target->l;
    } else {
      N *smallest = target->r;
      while (!smallest->leftTh) {
        smallest = smallest->l;
      }
      smallest->l= p;
      p->r= target->r;
    }}}
void displayTree() { //print the tree
  N *temp = root, *p;
  for (;;) {
    p = temp;
    temp = temp->r;
    if (!p->rightTh) {
      while (!temp->leftTh) {
        temp = temp->l;
      }
    }
    if (temp == root)
      break;
```

```
      cout<<temp->k<<" ";
    }
    cout<<endl;
  }
};
int main() {
  ThreadedBinaryTree tbt;
  cout<<"ThreadedBinaryTree\n";
  char ch;
  int c, v;
  while(1) {
    cout<<"1. Insert "<<endl;
    cout<<"2. Delete"<<endl;
    cout<<"3. Search"<<endl;
    cout<<"4. Clear"<<endl;
    cout<<"5. Display"<<endl;
    cout<<"6. Exit"<<endl;
    cout<<"Enter Your Choice: ";
    cin>>c;
    //perform switch operation
    switch (c) {
      case 1 :
cout<<"Enter integer element to insert: ";
        cin>>v;
        tbt.insert(v);
        break;
      case 2 :
        cout<<"Enter integer element to delete: ";
        cin>>v;
        tbt.Delete(v);
        break;
      case 3 :
        cout<<"Enter integer element to search: ";
        cin>>v;
        if (tbt.search(v) == true)
          cout<<"Element "<<v<<" found in the tree"<<endl;
        else
          cout<<"Element "<<v<<" not found in the tree"<<endl;
        break;
      case 4 :
        cout<<"\nTree Cleared\n";
        tbt.makeEmpty();
        break;
      case 5:
        cout<<"Display tree: \n ";
        tbt.displayTree();
        break;
      case 6:
        exit(1);
      default:
        cout<<"\nInvalid type! \n";
    }   }
  cout<<"\n";
  return 0;
}
```