//============================================================================

// Name        : ThreadedBinaryTree.cpp

// Author      :  Yash Sonar

// Title : Convert given binary tree into inordered and preordered threaded binary tree.

//          Analyze time and space complexity of the algorithm.

//============================================================================

#include <iostream>

using namespace std;

class TBT;

class node

{

    node \*left,\*right;

    int data;

    bool rbit,lbit;

public:

    node()

{

        left=NULL;

        right=NULL;

        rbit=lbit=0;

}

    node(int d)

    {

        left=NULL;

        right=NULL;

        rbit=lbit=0;

        data=d;

    }

    friend class TBT;

};

class TBT

{

    node \*root; //acts as a dummy node

public:

    TBT() //dummy node initialization

{

        root=new node(9999);

        root->left=root;

        root->rbit=1;

        root->lbit=0;

        root->right=root;

}

    void create();

    void insert(int data);

    node \*inorder\_suc(node \*);

    void inorder\_traversal();

    node \* preorder\_suc(node \*c);

    void preorder\_traversal();

};

//--------------------------------------------

void TBT::preorder\_traversal()

{

    node \*c=root->left;

    while(c!=root)

    {

        cout<<" "<<c->data;

        c=preorder\_suc(c);

    }

}

void TBT::inorder\_traversal()

{

    node \*c=root->left;

    while(c->lbit==1)

        c=c->left;

    while(c!=root)

    {

        cout<<" "<<c->data;

        c=inorder\_suc(c);

    }

}

node\* TBT::inorder\_suc(node \*c)

{

    if(c->rbit==0)

        return c->right;

    else

        c=c->right;

    while(c->lbit==1)

    {

        c=c->left;

    }

    return c;

}

node \*TBT::preorder\_suc(node \*c)

{

    if(c->lbit==1)

    {

        return c->left;

    }

    while(c->rbit==0)

    {

        c=c->right;

    }

    return c->right;

}

//-------- Create Method

void TBT::create()

{

    int n;

    if(root->left==root&&root->right==root)

    {

    cout<<"\nEnter number of nodes:";

    cin>>n;

    for(int i=0;i<n;i++)

    {

        int info;

        cout<<"\nEnter data: ";

        cin>>info;

        this->insert(info);

    }

    }

    else

    {

        cout<<"\nTree is Already created.\n";

    }

}

void TBT::insert(int data)

{

    if(root->left==root&&root->right==root) //no node in tree

    {

        node \*p=new node(data);

        p->left=root->left;

        p->lbit=root->lbit; //0

        p->rbit=0;

        p->right=root->right;

        root->left=p;

        root->lbit=1;

        cout<<"\nInserted start"<<data;

        return;

    }

    node \*cur=new node;

    cur=root->left;

    while(1)

    {

        if(cur->data<data)   //insert right

        {

            node \*p=new node(data);

            if(cur->rbit==0)

            {

                p->right=cur->right;

                p->rbit=cur->rbit;

                p->lbit=0;

                p->left=cur;

                cur->rbit=1;

                cur->right=p;

                cout<<"\nInserted right "<<data;

                return;

            }

            else

                cur=cur->right;

        }

        if(cur->data>data) //insert left

        {

            node \*p=new node(data);

            if(cur->lbit==0)

            {

                p->left=cur->left;

                p->lbit=cur->lbit;

                p->rbit=0;

                p->right=cur; //successor

                cur->lbit=1;

                cur->left=p;

                cout<<"\nInserted left"<<data;

                return;

            }

            else

                cur=cur->left;

        }

    }

}

int main() {

    TBT t1;

    int value;

    int choice;

    do

    {

        cout<<"\n1.Create Tree\n2.Insert into tree\n3.Preorder\n4.Inorder\n0.Exit\nEnter your choice: ";

        cin>>choice;

        switch(choice)

        {

        case 1:

            t1.create();

            break;

        case 2:

            cout<<"\nEnter Number(data): ";

            cin>>value;

            t1.insert(value);

            break;

        case 3:

            cout<<"\nPreorder traversal of TBT\n";

            t1.preorder\_traversal();

            break;

        case 4:

            cout<<"\nInoder Traversal of TBT\n";

            t1.inorder\_traversal();

            break;

        default:

            cout<<"\nWrong choice";

        }

    }while(choice!=0);

    return 0;

}