#include<iostream>

#include<stack>

#include<queue>

#include"Header.h"

using namespace std;

int main()

{

BinaryTree<int> \*BT; //creating an object of binary tree

BT=new BinaryTree<int>();

//array to pass,0 means no node exists

int Arr[15]={0,1,2,3,4,5,6,7,8,9,10,0,12,13,14};

BT->BuildTree(Arr,15); //building the tree from the array

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Inorder Traversal(Iterative is: "<<endl;

BT->InOrder();

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Preorder Traversal(Iterative) is: "<<endl;

BT->PreOrder();

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Post order Traversal(Iterative) is: "<<endl;

BT->PostOrder();

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Level order Traversal(Iterative) is: "<<endl;

BT->LevelOrder();

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

system("pause");

return 0;

}

#define BINARYTREE\_H

#include<stack>

template<class DT>

class BNode

{

public:

BNode(){

leftchild=NULL;

rightchild=NULL;

}

void setLeftChild(BNode<DT>\* n){

leftchild=n;

}

BNode<DT>\* getLeftChild(){

return leftchild;

}

void setRightChild(BNode<DT>\* n){

rightchild=n;

}

BNode<DT>\* getRightChild(){

return rightchild;

}

void setData(DT pdate){

data=pdate;

}

DT getData(){

return data;

}

private:

DT data;

BNode\* leftchild;

BNode\* rightchild;

};

template<class DT>

class BinaryTree

{

public:

//constructor already done in Lab7, please reuse that code

BinaryTree ()

{

root=NULL;

}

void BuildTree(DT \* Arr, int Size)

{

if (Size < 2)

return;

BNode <DT>\*\* nodePtrs = new BNode<DT> \* [Size];

for (int i = 1; i < Size; ++i)

{

if (Arr[i] == -1)

{

nodePtrs[i] = nullptr;

}

else

{

nodePtrs[i] = new BNode<DT> ();

nodePtrs[i]->setData(Arr[i]);

}

if (i / 2 >= 1)

{

if (i % 2 == 0)

nodePtrs[i / 2]->setLeftChild(nodePtrs[i]);

else

nodePtrs[i / 2]->setRightChild(nodePtrs[i]);

}

}

root = nodePtrs[1];

delete [] nodePtrs;

}

//part 1: pre order traversal (iterative)

// If a stack is needed please use the one that comes with C++

void PreOrder()

{

stack<BNode<DT>\*> \*s1=new stack<BNode<DT>\*>();

BNode<DT> \*temp=root;

if(temp)

s1->push(temp);

while(!s1->empty())

{

temp=s1->top();

s1->pop();

cout<<temp->getData()<<" ";

if(temp->getRightChild())

s1->push(temp->getRightChild());

if(temp->getLeftChild())

s1->push(temp->getLeftChild());

}

}

//part2: in order traversal (iterative)

// If a stack is needed please use the one that comes with C++

void InOrder()

{

stack<BNode<DT>\*>\* s=new stack<BNode<DT>\*>();

BNode<DT> \*temp = root;

while (!s->empty() || temp)

{

if (temp)

{

s->push(temp);

temp = temp->getLeftChild();

}

else

{

temp = s->top();

s->pop();

cout << temp->getData() << " ";

temp = temp->getRightChild();

}

}

}

//part3: post order traversal (iterative)

// If a stack is needed please use the one that comes with C++

void PostOrder()

{

stack<BNode<DT>\*>\* s1=new stack<BNode<DT>\*>();

stack<BNode<DT>\*>\* s2=new stack<BNode<DT>\*>();

BNode <DT>\* temp;

if (root)

{

s1->push(root);

while (!s1->empty())

{

temp=s1->top();

s1->pop();

s2->push(temp);

if (temp->getLeftChild())

{

s1->push(temp->getLeftChild());

}

if (temp->getRightChild())

{

s1->push(temp->getRightChild());

}

}

while (!s2->empty())

{

temp=s2->top();

s2->pop();

cout<<temp->getData()<<" ";

}

}

}

// part4: level order traversal (iterative)

// If a queue is needed please use the one that comes with C++

void LevelOrder()

{

queue<BNode<DT>\*>\* q=new queue<BNode<DT>\*>();

BNode<DT>\* temp= root;

while(temp)

{

cout<<temp->getData()<<" ";

if(temp->getLeftChild())

{

q->push(temp->getLeftChild());

}

if(temp->getRightChild())

{

q->push(temp->getRightChild());

}

if(!q->empty())

{

temp = q->front();

q->pop();

}

else

{

temp = NULL;

}

}

}

// part5: calculate and return height of the tree iteratively (iterative)

int calculateHeightItr()

{

return 0;

}

private:

BNode<DT>\* root;

};

#endif