实验三 二叉树的遍历

班级:软件工程一班 姓名:秦源 学号:1525161007

1. 需求分析

问题描述：很多涉及二叉树的操作的算法都是以二叉树的遍历操作为基础的。编写程序，对一棵给定的二叉树进行先、中、后三种次序的遍历。以二叉链表为存储结构，实现二叉树的先、中、后三种次序的递归和非递归遍历。

1. 概要设计

（１）、设二叉树的结点不超过30个，且每个结点的数据均为字符，这样可利用先序遍历序列作为输入顺序创建二叉树链表存储结构。

（２）、也可利用完全二叉树在顺序存储中的特性，创建二叉树的存储结构，此时，二叉树中结点数据的类型不受限制。

三、 详细设计   
//

// main.cpp

// 二叉树的遍历

//

// Created by 秦源.

// Copyright © 2016年 QinYuan. All rights reserved.

//

#include<iostream>

using namespace std;

class BinaryTreeNode{

private:

char m\_data;//储存数据

BinaryTreeNode \* m\_leftChild;//左孩子指针域

BinaryTreeNode \* m\_rightChild;//右孩子指针域

public:

BinaryTreeNode(){m\_leftChild=m\_rightChild=NULL;};

BinaryTreeNode(const char&data,BinaryTreeNode \* leftChild=NULL,BinaryTreeNode \* rightChild=NULL){

m\_data=data;

m\_leftChild=leftChild;

m\_rightChild=rightChild;

};

char & GetData(){return m\_data;};//返回该节点数据

BinaryTreeNode \* GetLeftChild(){return m\_leftChild;};//返回该节点左孩子指针

BinaryTreeNode \* GetRightChild(){return m\_rightChild;};//返回该节点右孩子指针

void SetData(const char&data){m\_data=data;};//设置该节点的数据

void SetLeftChild(BinaryTreeNode \*leftChild){m\_leftChild=leftChild;};//设置该节点左孩子指针域

void SetRightChild(BinaryTreeNode \*rightChild){m\_rightChild=rightChild;};//设置该节点右孩子指针域

};

struct stack{

BinaryTreeNode \* base=NULL;

BinaryTreeNode \* top=NULL;

int size=30;

};

void initStack(stack & e){

e.base=new BinaryTreeNode[30];

if(!e.base){

cout<<"堆栈创建失败"<<endl;

exit(-1);

}

e.top=e.base;

}

void push(stack & e,BinaryTreeNode temp){

//栈满扩充

if(e.top-e.base>=e.size){

long nowSize=e.top-e.base;

BinaryTreeNode \* t=new BinaryTreeNode[e.size+30];

if(!t){

cout<<"堆栈扩容失败"<<endl;

exit(-1);

}

e.size+=30;

for(int i=0;i<nowSize;i++){

\*(t++)=\*(e.base+i);

}

\*(e.top)=\*(t+nowSize);

delete []e.base;

e.base=t;

}

//push

\*(e.top++)=temp;

}

BinaryTreeNode \* pop(stack & e){

if(e.top==e.base){

cout<<"Empty!"<<endl;

exit(-1);

}

return --e.top;

}

BinaryTreeNode \* top(stack & e){

if(e.top==e.base){

cout<<"Empty!"<<endl;

exit(-1);

}

return e.top-1;

}

bool empty(stack & e){

if(e.top==e.base)

return true;

return false;

}

void enterDataByPreOrderTraverse(BinaryTreeNode \* root){

char c;

cin>>c;

if(c=='#'){

root=NULL;

return;

}

root->SetData(c);

BinaryTreeNode \* node1=new BinaryTreeNode();

root->SetLeftChild(node1);

enterDataByPreOrderTraverse(root->GetLeftChild());

BinaryTreeNode \* node2=new BinaryTreeNode();

root->SetRightChild(node2);

enterDataByPreOrderTraverse(root->GetRightChild());

}

BinaryTreeNode \* enterData(){//书上例子

BinaryTreeNode \* root=new BinaryTreeNode();

cout<<"请按照先序递归输入数据，空位置为 # "<<endl;

enterDataByPreOrderTraverse(root);

return root;

}

void PreOrderTraverse(BinaryTreeNode \* root){//先序递归遍历

if(root!=NULL){

cout<<root->GetData()<<" ";

PreOrderTraverse(root->GetLeftChild());

PreOrderTraverse(root->GetRightChild());

}

}

void InOrderTraverse(BinaryTreeNode \* root){//中序递归遍历

if(root!=NULL){

InOrderTraverse(root->GetLeftChild());

cout<<root->GetData()<<" ";

InOrderTraverse(root->GetRightChild());

}

}

void PostOrderTraverse(BinaryTreeNode \* root){//后序递归遍历

if(root!=NULL){

PostOrderTraverse(root->GetLeftChild());

PostOrderTraverse(root->GetRightChild());

cout<<root->GetData()<<" ";

}

}

void PreOrderTraverse\_byStack(BinaryTreeNode \* root){//先序非递归遍历

stack s;

initStack(s);

BinaryTreeNode \*p=root;

while(!empty(s)||p!=NULL){

while(p){

push(s,\*p);

cout<<p->GetData()<<" ";

p=p->GetLeftChild();

}

p=top(s);

pop(s);

p=p->GetRightChild();

}

}

void InOrderTraverse\_byStack(BinaryTreeNode \* root){//中序非递归遍历

stack s;

initStack(s);

BinaryTreeNode \*p=root;

while(!empty(s)||p!=NULL){

while(p){

push(s,\*p);

p=p->GetLeftChild();

}

p=top(s);

cout<<p->GetData()<<" ";

pop(s);

p=p->GetRightChild();

}

}

void PostOrderTraverse\_byStack(BinaryTreeNode \* root){//后序非递归遍历

stack s;

initStack(s);

//用于储存结果的堆栈r

stack r;

initStack(r);

BinaryTreeNode \*p=root;

while(!empty(s)||p!=NULL){

while(p){

push(s,\*p);

push(r,\*p);

p=p->GetRightChild();

}

p=top(s);

pop(s);

p=p->GetLeftChild();

}

while(!empty(r)){

BinaryTreeNode \*t=pop(r);

cout<<t->GetData()<<" ";

}

}

int main(int argc, const char \* argv[]) {

BinaryTreeNode \* root=enterData();

cout<<"先序递归遍历 :"<<endl;

PreOrderTraverse(root);//先序递归遍历

cout<<endl<<"中序递归遍历 :"<<endl;

InOrderTraverse(root);//中序递归遍历

cout<<endl<<"后序递归遍历 :"<<endl;

PostOrderTraverse(root);//后序递归遍历

cout<<endl<<"先序非递归遍历 :"<<endl;

PreOrderTraverse\_byStack(root);//先序非递归遍历

cout<<endl<<"中序非递归遍历 :"<<endl;

InOrderTraverse\_byStack(root);//中序非递归遍历

cout<<endl<<"后序非递归遍历 :"<<endl;

PostOrderTraverse\_byStack(root);//后序非递归遍历

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

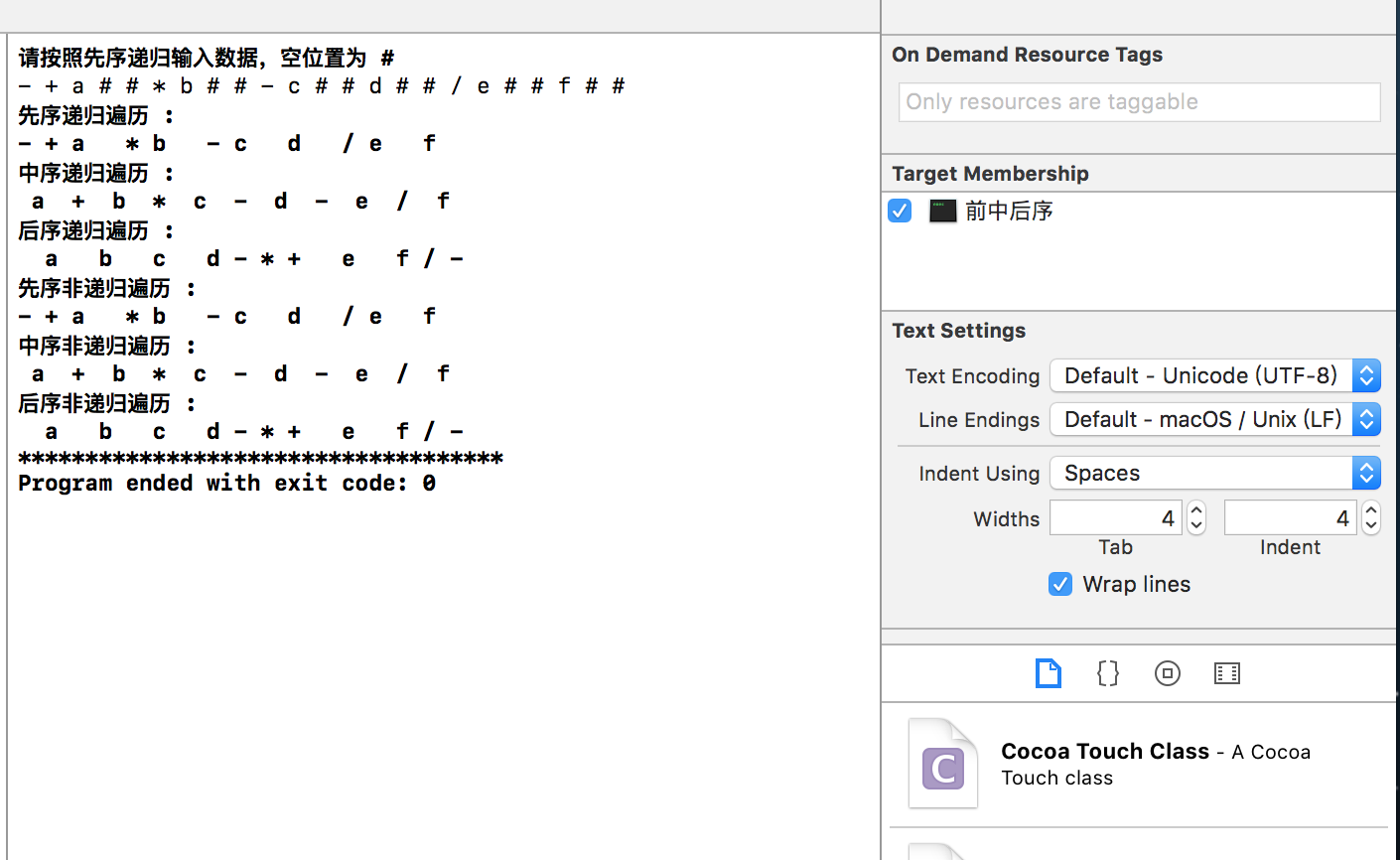
return 0;

}

四、用户使用说明

测试数据：以数据结构教科书P129页图6.9的二叉树为例。

五、 测试结果



六、 附录

非递归方法采用堆栈实现。