# ------基于二叉链表实现二叉树-----

BinTree.h:

#include"BinNode.h"

#include"string"

#include<queue>

template<typename E>

class BinTree//二叉树类

{

private:

BinNode<E>\*root;//根结点

void clear(BinNode<E>\*r)//清空二叉树

{

if (r == NULL)

return;

if (r->left() != NULL)

{

clear(r->left());

}

if (r->right() != NULL)

{

clear(r->right());

}

delete r;

r = NULL;

}

void preOrder(BinNode<E>\*tmp,void(\*visit)(BinNode<E>\*node))//先序遍历，void(\*visit)(BinNode<E>\*node)为一个函数指针参数，用visit代替传进来的函数，在遍历函数中使用传进来的函数功能

{

if (tmp == NULL)

return;

visit(tmp);

preOrder(tmp->left(),visit);

preOrder(tmp->right(),visit);

}

void inOrder( BinNode<E>\*tmp,void(\*visit)(BinNode<E>\*node))//中序遍历，void(\*visit)(BinNode<E>\*node)为一个函数指针参数，用visit代替传进来的函数，在遍历函数中使用传进来的函数功能

{

if (tmp == NULL)

return;

inOrder(tmp->left(),visit);

visit(tmp);

inOrder(tmp->right(),visit);

}

void postOrder(BinNode<E>\*tmp,void(\*visit)(BinNode<E>\*node))//后序遍历，void(\*visit)(BinNode<E>\*node)为一个函数指针参数，用visit代替传进来的函数，在遍历函数中使用传进来的函数功能

{

if (tmp == NULL)

return;

postOrder(tmp->left(),visit);

postOrder(tmp->right(),visit);

visit(tmp);

}

void LevelOrderTranverse( BinNode<E>\*tmp,void(\*visit)(BinNode<E>\*node))//层次遍历，void(\*visit)(BinNode<E>\*node)为一个函数指针参数，用visit代替传进来的函数，在遍历函数中使用传进来的函数功能

{

if (tmp == NULL)

return;

queue<BinNode<E>\*>que;

que.push(tmp);

BinNode<E>\* curr;

int sum = 0;

while (que.empty() != true)

{

curr = que.front();

if (curr->left() != NULL)

{

que.push(curr->left());

}

if (curr->right() != NULL)

{

que.push(curr->right());

}

que.pop();

visit(curr);

}

}

int BinTreeDepth(BinNode<E>\*tmp)//获得二叉树的深度

{

if (tmp == NULL)

return 0;

int ld = 0;

int rd = 0;

if (tmp->left() != NULL)

ld = BinTreeDepth(tmp->left());

if (tmp->right() != NULL)

rd = BinTreeDepth(tmp->right());

if (ld >= rd)

return ld + 1;

else

return rd + 1;

}

int BinTreeNodes(BinNode<E>\*tmp)//获得二叉树的结点数

{

if (tmp == NULL)

return 0;

queue<BinNode<E>\*>que;

que.push(tmp);

BinNode<E>\* curr;

int sum = 0;

while (que.empty() != true)

{

curr = que.front();

if (curr->left() != NULL)

{

que.push(curr->left());

}

if (curr->right() != NULL)

{

que.push(curr->right());

}

que.pop();

sum++;

}

return sum;

}

int BinTreeHeight(BinNode<E>\*tmp)//获得二叉树的高度

{

if (tmp == NULL)

return 0;

return BinTreeDepth(tmp) - 1;

}

int BinTreeLeafs(BinNode<E>\*tmp)//获得二叉树的叶子结点数

{

if (tmp == NULL)

return 0;

queue<BinNode<E>\*>que;

que.push(tmp);

BinNode<E>\* curr;

int sum = 0;

while (que.empty() != true)

{

curr = que.front();

if (curr->left() != NULL)

{

que.push(curr->left());

}

if (curr->right() != NULL)

{

que.push(curr->right());

}

if (curr->right() == NULL&&curr->left() == NULL)

sum++;

que.pop();

}

return sum;

}

bool find(BinNode<E>\*tmp, E e)//查找二叉树中是否含有某个名为e的结点

{

if (tmp == NULL)

return false;

queue<BinNode<E>\*>que;

que.push(tmp);

BinNode<E>\* curr;

int sum = 0;

while (que.empty() != true)

{

curr = que.front();

if (curr->left() != NULL)

{

que.push(curr->left());

}

if (curr->right() != NULL)

{

que.push(curr->right());

}

if (curr->getValue() == e)

return true;

que.pop();

}

return false;

}

public:

BinTree()//默认构造函数

{

root=new BinNode<E>;

}

~BinTree()//析构函数

{

clear(root);

}

bool BinTreeEmpty()//判断二叉树是否为空

{

if (root == NULL)

return true;

else

return false;

}

BinNode<E>\*getRoot()//获得根节点

{

return root;

}

void setRoot(BinNode<E>\*r)//设置根节点

{

root = r;

}

//下面的函数是对外的函数，所以内部还会有一些同名的函数，但是参数列表不一样，实现数据的封装，外部的调用不会涉及到内部的数据对象

void clear()//清空二叉树

{

clear(root);

root = NULL;

}

void preOrder(void(\*visit)(BinNode<E>\*node))//先序遍历，传入相对应的访问函数即可对该当前结点实现不同功能的访问（本程序为输出）

{

preOrder(root,visit);

}

void inOrder(void(\*visit)(BinNode<E>\*node)) //先序遍历，传入相对应的访问函数即可对该当前结点实现不同功能的访问（本程序为输出）

{

inOrder(root,visit);

}

void postOrder(void(\*visit)(BinNode<E>\*node))//先序遍历，传入相对应的访问函数即可对该当前结点实现不同功能的访问（本程序为输出）

{

postOrder(root,visit);

}

void LevelOrderTranverse(void(\*visit)(BinNode<E>\*node))//先序遍历，传入相对应的访问函数即可对该当前结点实现不同功能的访问（本程序为输出）

{

LevelOrderTranverse(root,visit);

}

int BinTreeDepth()//获得二叉树深度

{

return BinTreeDepth(root);

}

int BinTreeNodes()//获得二叉树结点数

{

return BinTreeNodes(root);

}

int BinTreeHeight()//获得二叉树高度

{

return BinTreeHeight(root);

}

int BinTreeLeafs()//获得二叉树叶子结点数

{

return BinTreeLeafs(root);

}

bool find(E e)//查找二叉树中是否存在名为e的结点

{

return find(root, e);

}

};

BinNode.h:

#include<iostream>

using namespace std;

template<typename E>

class BinNode//结点类

{

private:

BinNode\*lc;//左孩子

BinNode\*rc;//右孩子

E elem;

public:

BinNode()//默认构造函数

{

lc = NULL;

rc = NULL;

}

BinNode(E tmp, BinNode\*l=NULL, BinNode\*r=NULL)//带参构造函数

{

elem = tmp;

lc = l;

rc = r;

}

BinNode\*left()//返回左孩子

{

return lc;

}

BinNode\*right()//返回右孩子

{

return rc;

}

void setLeft(BinNode\*l)//设置左孩子

{

lc = l;

}

void setRight(BinNode\*r)//设置右孩子

{

rc = r;

}

void setValue(E tmp)//设置当前结点的值

{

elem = tmp;

}

E getValue()//获得当前结点的值

{

return elem;

}

bool isLeaf()//判断当前结点是否为叶子结点

{

if (lc == NULL&&rc == NULL)

return true;

else

return false;

}

};

main.cpp:

#include"BinTree.h"

template<typename E>

void printNode(BinNode<E>\*tmp)//打印结点的值的函数

{

cout << tmp->getValue() << " ";

}

//update2018.4.23

//先序和中序递归建树

template<typename E>

//pre:先序遍历 preStart先序序列的第一个元素下标，preEnd则为最后一个元素下标

//in:同理

BinNode<E>\* creatBinaryTree(string pre[], int preStart,int preEnd,string in[],int inStart,int inEnd)

{

//边界情况

if(preStart>preEnd || inStart > inEnd)

{

return NULL;

}

BinNode<E>\*node=new BinNode<E>(pre[preStart]);

for(int i=inStart;i<=inEnd;i++)

{

if(in[i]==pre[preStart])//从先序序列找到根节点，再从中序序列划分左右子树

{

node->setLeft(creatBinaryTree<E>(pre,preStart+1,preStart+i-inStart,in,inStart,i-1));

node->setRight(creatBinaryTree<E>(pre,preStart+i-inStart+1,preEnd,in,i+1,inEnd));

break;

}

}

return node;

}

void creatBinaryTree(BinTree<string>\*BT)//构建二叉树的函数，包含了上面的构建二叉树的主函数，仅仅起到了在主函数中简洁一些的作用

{

cout << "现在进入构建二叉树程序......" << endl;

int n=0;

cout<<"请输入二叉树有多少个节点，不计算空节点"<<endl;

cin>>n;

string \*pre=new string[n];

string \*in=new string[n];

cout<<"请输入先序遍历序列，空格隔开"<<endl;

for(int i=0;i<n;i++)

cin>>pre[i];

cout<<"请输入中序遍历序列，空格隔开"<<endl;

for(int i=0;i<n;i++)

cin>>in[i];

BT->setRoot(creatBinaryTree<string>(pre,0,n-1,in,0,n-1));

}

int main()

{

//本程序的二叉树是一个模板类，若想改变为别的类型，可以在相关的地方在“<>”中修改相关参数,本程序默认为最具有普遍性的string

BinTree<string>\*BT = new BinTree<string>;

creatBinaryTree(BT);

//在这里，已经构建好了一棵二叉树

//下面是二叉树的基本函数操作的展示

cout << "there are some functions here" << endl;

cout << "0:判断是否为空树：";

if (BT->BinTreeEmpty() == true)

cout << "是" << endl;

else

cout << "否" << endl;

cout << "1:前序遍历:";

BT->preOrder(printNode);

cout << endl;

cout << "2:中序遍历:";

BT->inOrder(printNode);

cout << endl;

cout << "3:后序遍历:";

BT->postOrder(printNode);

cout << endl;

cout << "4:层次遍历:";

BT->LevelOrderTranverse(printNode);

cout << endl;

cout << "5:记录树的深度:";

cout << BT->BinTreeDepth() << endl;

cout << "6:记录树的高度:";

cout << BT->BinTreeHeight() << endl;

cout << "7:统计结点:";

cout << BT->BinTreeNodes() << endl;

cout << "8:统计叶子结点:";

cout << BT->BinTreeLeafs() << endl;

cout << "9:在二叉树中查找某个值是否存在，默认查找“C”:";

if (BT->find("C") == true)

cout << "存在" << endl;

else

cout << "不存在" << endl;

cout << "10:是否清空:";

BT->clear();

cout << "已清空" << endl;

cout << "5:记录树的深度:";

cout << BT->BinTreeDepth() << endl;

cout << "6:记录树的高度:";

cout << BT->BinTreeHeight() << endl;

cout << "7:统计结点:";

cout << BT->BinTreeNodes() << endl;

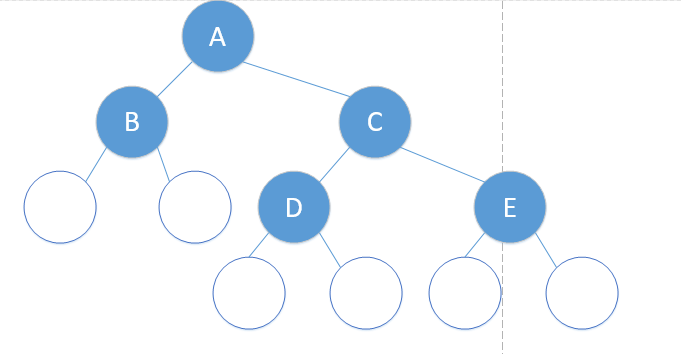
cout << "8:统计叶子结点:";

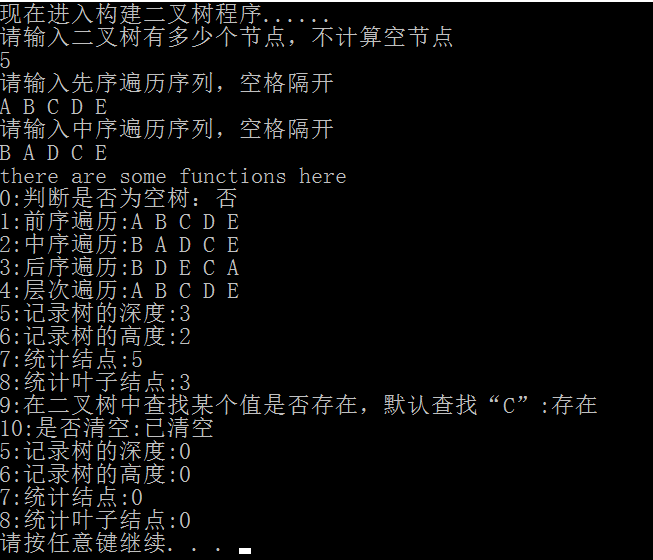
cout << BT->BinTreeLeafs() << endl;

system("pause");

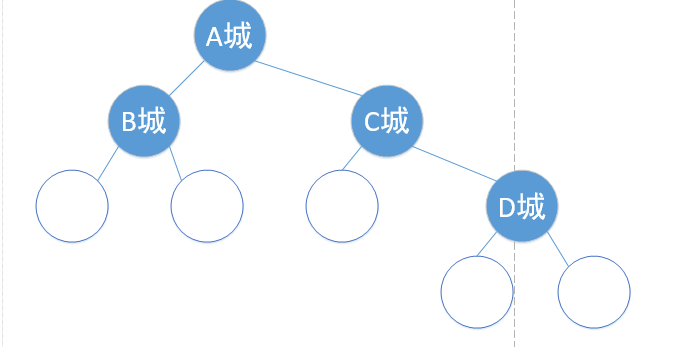
}

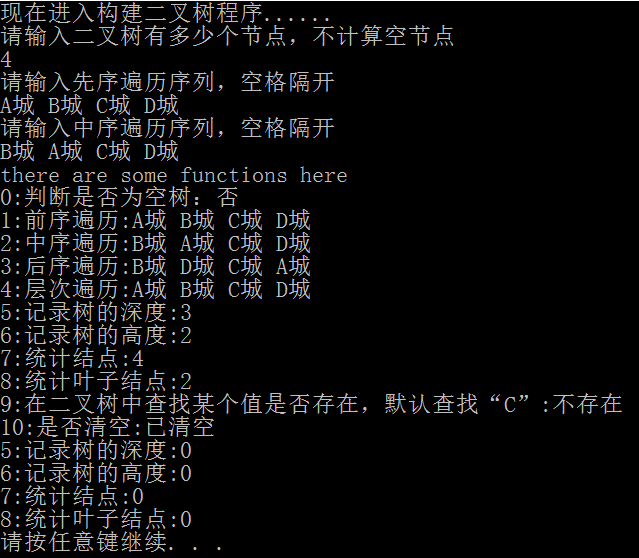
# 样例1：



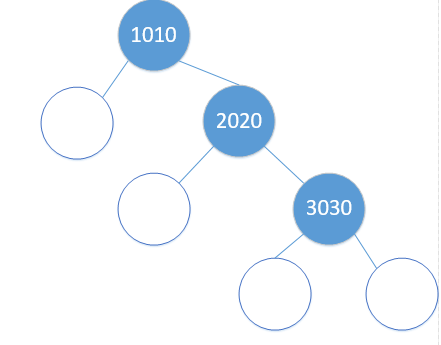


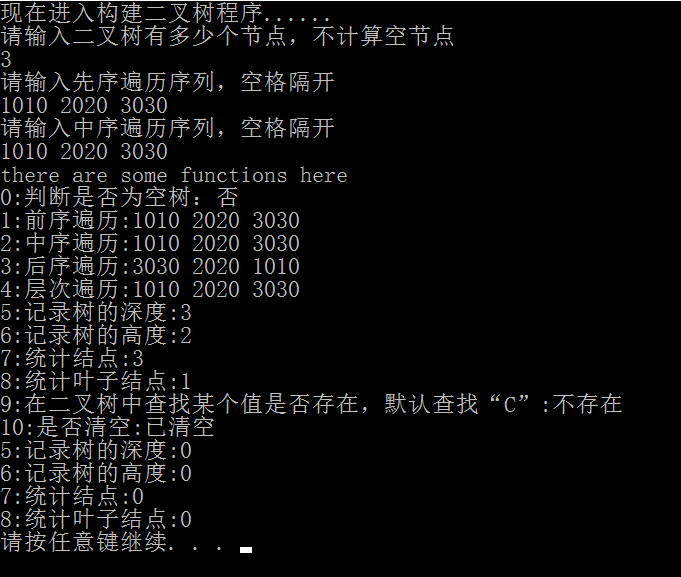
# 样例2





# 样例3：





**【附录】**

版本信息声明：

visual studio 2017

资料整理人：  
计算机科学与技术 1603班 张伟麟

文献参考：

<https://blog.csdn.net/GYQJN/article/details/52709972>

提供了实现根据前序遍历和中序遍历的思路

（实现的时候在从中序序列找左右子树的时候增加了一个break优化了一下算法）