**8 二叉树的操作**

**【实验简介】**二叉树是树形结构的一种重要类型。通过本次实验，熟悉二叉树结点的结构，掌握二叉树的基本操作以及具体实现，学会利用递归方法编写对二叉树这种递归数据结构进行处理的算法。

**【实验内容】**

编写程序，实现对二叉树的以下操作：

1. 建立二叉树。
2. 按任一种遍历次序输出二叉树中的所有结点。
3. 求二叉树的深度。
4. 求二叉树中的所有节点数。
5. 求二叉树中的所有叶子节点数。
6. 清除二叉树，使之编程一只空树。

**【主要代码】**

头文件：

#pragma once

#ifndef head\_h

#define head\_h

#include<iostream>

#include <fstream>

using namespace std;

template <class t>

struct bintree\_node

{

t data;//数值域

bintree\_node<t>\* left\_child, \* right\_child;//链域

bintree\_node() //不带默认参数的构造函数

{

left\_child = NULL;

right\_child = NULL;

}

bintree\_node(t x, bintree\_node<t>\* l = NULL, bintree\_node<t>\* r = NULL)

{//带默认参数的构造函数

data = x;

left\_child = l;

right\_child = r;

}

};

template<class t>

class bintree

{

public:

bintree() { root = NULL; }//不带默认参数的构造函数

bintree(t value) { ref\_value = value;root = NULL; }//带默认参数的构造函数

~bintree() { destroy(root); };//析构函数

bintree\_node<t>\* get\_root(){ return root; }//取根

int Hight() { return height(root); }//返回高度

void bintree\_count(bintree\_node<t>\* subtree,int& m1,int& m2);//m1是总结点数,m2是叶子结点数

void print\_bintree( bintree\_node<t>\* tree);//前序遍历输出二叉树

void set\_ref(t ref) { ref\_value = ref; }//结束符设置

private:

t ref\_value;//数据停止标志

bintree\_node<t>\* root;//根节点

static void create(istream& in, bintree\_node<t>\*& subtree, t ref); //静态成员函数，从文件读入建树，递归方式

friend istream& operator>>(istream& in, bintree<t>& tree) {//>>重载

create(in, tree.root,tree.ref\_value);

return in;

}

void destroy(bintree\_node<t>\*& subtree);//删除

int height(bintree\_node<t>\* subtree); //返回树高度

};

template<class t>

void bintree<t>::destroy(bintree\_node<t>\*& subtree)

{//后序遍历，递归删除

if (subtree != NULL)

{

destroy(subtree->left\_child);

destroy(subtree->right\_child);

delete subtree;

}

}

template<class t>

int bintree<t>::height(bintree\_node<t>\* subtree)

{//后序遍历，递归访问

if (subtree == NULL)

return 0;

else

{

int i = height(subtree->left\_child);

int j = height(subtree->right\_child);

return (i > j) ? i + 1 : j + 1;

}

}

template<class t>

void bintree<t>::create(istream& in, bintree\_node<t>\*& subtree,t ref)

{//前序遍历，读入文件建树

t item;

if (!in)

{

cout << "文件不能打开！" << endl;

}

else

{

in >> item;

if (item != ref)

{

subtree = new bintree\_node<t>(item);

if (subtree == NULL)

{

cout << "内存分配错误！" << endl; exit(1);

}

create(in,subtree->left\_child,ref);

create(in,subtree->right\_child,ref);

}

else

subtree = NULL;

}

}

template<class t>

void bintree<t>::print\_bintree(bintree\_node<t>\* tree)

{//前序遍历，输出

if (tree != NULL)

{

cout << tree->data;

if (tree->left\_child != NULL || tree->right\_child != NULL)

{

cout << "(";

print\_bintree(tree->left\_child);

cout << ",";

print\_bintree(tree->right\_child);

cout << ")";

}

}

}

template<class t>

void bintree<t>::bintree\_count(bintree\_node<t>\* subtree,int& m1, int& m2)

{//前序遍历，统计所有结点和叶子结点

if (subtree != NULL)

{

m1++;//统计所有结点

if (subtree->left\_child == NULL && subtree->right\_child == NULL)

m2++;//统计叶子结点

bintree\_count(subtree->left\_child, m1, m2);

bintree\_count(subtree->right\_child, m1, m2);

}

}

#endif head\_h

源文件：

#include <iostream>

#include <fstream>

#include"head.h"

using namespace std;

int main()

{

int a = 0, b = 0;

ifstream open\_num;

open\_num.open("num.txt", ios::in);

bintree<char> num;

num.set\_ref('0');//把结束符设置为‘0’

open\_num >> num;//读入文件建树

open\_num.close();

num.print\_bintree(num.get\_root());//前序遍历输出

cout << endl;

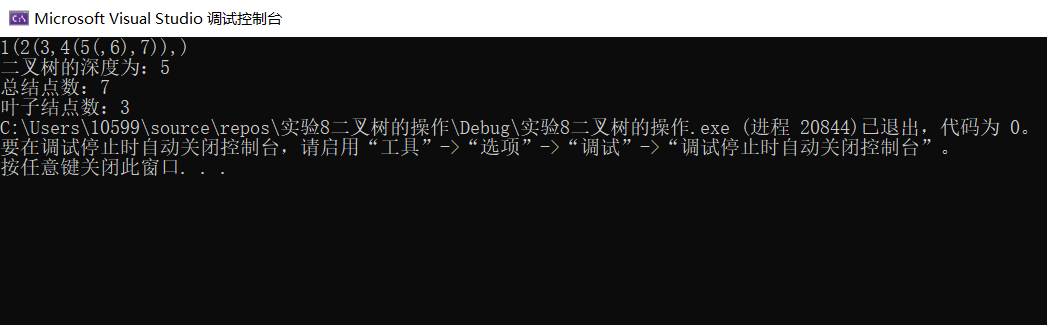
cout << "二叉树的深度为：" << num.Hight() << endl;//后序遍历求深度

num.bintree\_count(num.get\_root(), a, b);

cout << "总结点数：" << a << "\n" << "叶子结点数：" << b;

}

**【实验过程】**



**【实验体会】**

二叉树的许多成员函数都是用递归方式编写的，其中我认为难度最大的是把文件输入流和>>运算符重载结合，通过友元函数重载的同时，还需要把调用的create函数声明为静态，多个对象就可以共用create函数。另外，根据二叉树存储的数据类型不同，需要使用set\_ref函数设置数据停止标志。统计结点使用引用传递方式，把结点数和叶子结点数传给了两个变量，一个函数就可以得到两个数。