**实验6**

**6.1 实验目的**

掌握二叉树的动态链表存储结构及表示。

掌握二叉树的三种遍历算法（递归和非递归两类）。

运用二叉树三种遍历的方法求解有关问题。

**6.2 实验要求**

按照C++面向对象方法编写二叉树类；二叉树的测试数据可用多种方式进行输入，如键盘输入、静态写入、文件读入等。

设计二叉树的二叉链表存储结构，编写算法实现下列问题的求解。

<1>打印出二叉树的三种遍历序列。

<2>设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数。

<3>求二叉树的高度。

<4>求二叉树的结点数。

<5>求二叉树的叶子结点数。

<6>求二叉树的度为2的结点数。

<7>键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子。

<8>键盘输入一个元素x，求其在树中的层次，不存在时给出相应提示信息。

<9>将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）。

<10>交换二叉树中每个结点的左右孩子指针的值。（即：左子树变为右子树，右子树变为左子树）。

（下面为选做实验，有兴趣的同学完成）

<11>复制一棵二叉树T到T1。

<12>输出二叉树从每个叶子结点到根结点的路径（经历的结点）。

<13>对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次打印。（提示：需要使用队列）

<14>对二叉链表表示的二叉树，求2个结点最近的共同祖先。

实验测试数据基本要求：

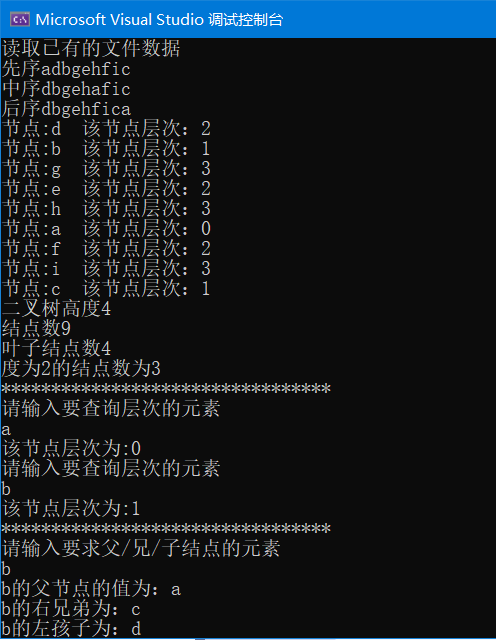
<15>求二叉树中一条最长的路径长度（边数），并输出路径上的个结点值。

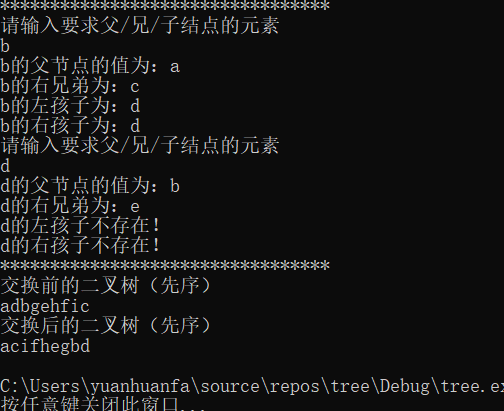
实验测试数据基本要求：

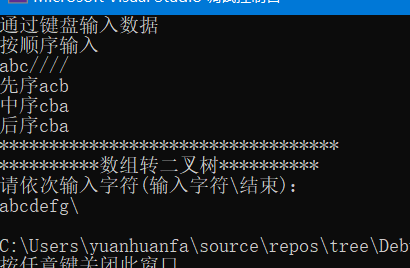
**6.3 实验数据要求**

自我编写测试样例，要求每个功能函数的测试样例不少于两组

**6.4 运行结果截图及说明**







**6.5 附源代码**

#pragma once

#include <iostream>

#include <stdio.h>

using namespace *std*;

typedef char elementType;

typedef struct lBNode

{

elementType data;

struct lBNode\* lChild, \* rChild;

} BiNode,\* BiTree;

class linkedtree

{

public:

linkedtree();

~linkedtree();

void createtree(BiNode\* q, int i);//创建子树

void createbitree(BiNode \*&BT);//创建二叉树

bool readfile(char filename[], char strline[100][3], int& arrlen);

bool fileTotree(BiNode \*&T, char strLine[100][3], int nLen, int& nRow);

void firsttravel(BiNode\* T);//先序遍历

void midtravel(BiNode\* T);//中序遍历

void lasttravel(BiNode\* T);//后序遍历

void getnum\_level(BiNode\* T,int level);//按中序次序输出结点和层次数

int get2num(BiNode\* T, int\* num);//度为2的结点数

void getnodenum(BiNode\* T,int \*num);//结点数

int getdepth(BiNode\* T);//树的高度

int get0num(BiNode\* T, int \*num);//叶子结点数

void getfa\_bro\_ch(BiNode\*T,elementType x);//获取父/兄/子结点

void numgetlevel(BiNode\* T, elementType x, int level,int &lag);//元素获取层次

void arrtreeTolink(BiNode\* T, elementType arr[],int,int);//数组二叉树转二叉链表

void pointRtoL(BiNode\* T);//交换左右孩子指针

void deltree(BiNode\*T);//销毁二叉树

private:

};

#include "linkedtree.h"

linkedtree::linkedtree()

{

}

linkedtree::~linkedtree()

{

}

//由于对io流读取文件信息不熟悉，所以此代码参照于课本

//在c++中需要修改工程属性否则会报错

//等查阅学会c++相关的读取操作再进行修改

bool linkedtree::readfile(char filename[], char strline[100][3], int& arrlen)

{

*FILE*\* pFile; //定义二叉树文件指针

char str[1000]; //存放读取一行文本的字符串

pFile=*fopen*(filename,"r");

if(!pFile)

{

*cout*<<"文件打开失败！"<<*endl*;

return false;

} //读取文件第一行，判断二叉树标识BinaryTree是否正确

if(*fgets*(str,1000,pFile)!=*NULL*)

{

if(*strcmp*(str,"BinaryTree\n")!=0)

{

*cout*<<"打开的文件格式错误！"<<*endl*;

*fclose*(pFile);

return false;

}

}

arrlen=1;

while(*fscanf*(pFile,"%c %c %c\n",&strline[arrlen][0],&strline[arrlen][1],&strline[arrlen][2])!=*EOF*)

{

arrlen++;

}

*fclose*(pFile);

return true;

}

bool linkedtree::fileTotree(BiNode\* &T, char strLine[100][3], int nLen, int& nRow)

{

if ((nRow >= nLen) || (nRow == 0))

{

return false;

//数据已处理完毕，或者没有数据

}

//根据数组数据递归创建二叉树

T=new BiNode ;

T->*data*=strLine[nRow][0];

T->lChild=*NULL*; //初始化左孩子为空

T->rChild=*NULL*; //初始化右孩子为空

int nRowNext=nRow; //保留本次递归的行号

if(strLine[nRowNext][1]=='1')

{

nRow++; //行号加一

fileTotree(T->lChild,strLine,nLen,nRow); //递归创建左子树

}

if(strLine[nRowNext][2]=='1')

{

nRow++; //行号加一

fileTotree(T->rChild,strLine,nLen,nRow); //递归创建右子树

}

return true;

}

void linkedtree::createtree(BiNode\* q, int i)//子树函数

{

BiNode\* u;

elementType x;

*cin* >> x;

if (x != '/')

{

u = new BiNode;

u->*data* = x;

u->lChild = *NULL*;

u->rChild = *NULL*;

if (i==1)

{

q->lChild = u;

}

if (i==2)

{

q->rChild = u;

}

createtree(u, 1);//递归创建子树

createtree(u, 2);

}

}

void linkedtree::createbitree(BiNode\* &BT)//二叉树函数

{

BiNode\* p;

char x;

*cout*<<"按顺序输入"<<*endl*;

*cin* >> x;

if (x == '/')

return;

BT = new BiNode;

BT->*data* = x;

BT->rChild = *NULL*;

BT->lChild = *NULL*;

p = BT;

createtree(p, 1);//创建根节点子树

createtree(p, 2);

}

void linkedtree::firsttravel(BiNode\* T)//先序遍历

{

if (T)

{

*cout* << T->*data*;

midtravel(T->lChild);

midtravel(T->rChild);

}

}

void linkedtree::midtravel(BiNode\* T)//中序

{

if (T)

{

midtravel(T->lChild);

*cout* << T->*data*;

midtravel(T->rChild);

}

}

void linkedtree::lasttravel(BiNode\* T)//后序

{

if (T)

{

midtravel(T->lChild);

midtravel(T->rChild);

*cout* << T->*data*;

}

}

void linkedtree::getnum\_level(BiNode\* T,int level)

{

{

//递归打印结点层次

if (T != *NULL*)

{

getnum\_level(T->lChild, level + 1);

*cout* << "节点:" << T->*data* << " 该节点层次：" << level<< *endl*;;

getnum\_level(T->rChild, level + 1);

}

}

}

void linkedtree::getnodenum(BiNode\* T, int\* num)

{

//全局变量num计数

if (T)

{

(\*num)++;

getnodenum(T->lChild,num);

getnodenum(T->rChild,num);

}

}

int linkedtree::get2num(BiNode\* T, int\* num)

{

if (T)

{

get2num(T->lChild,num);

if (T->lChild != *NULL* && T->rChild != *NULL*)

{

(\*num)++;

}

get2num(T->rChild,num);

}

return \*num;

}

int linkedtree::get0num(BiNode\* T,int \*num)//叶子结点

{

if (T)

{

get0num(T->lChild, num);

if (T->lChild == *NULL* && T->rChild == *NULL*)

{

(\*num)++;

}

get0num(T->rChild, num);

}

return \*num;

}

int linkedtree::getdepth(BiNode\* T)

{

int m, n;

if (T==*NULL*)

{

return 0;

}

else

{

//获取左右子树的高度

m = getdepth(T->lChild);

n = getdepth(T->rChild);

if (m > n)

{

return m + 1;

}

else

return n + 1;

}

}

void linkedtree::getfa\_bro\_ch(BiNode\* T, elementType x)

{

if (T)

{

if (T->lChild != *NULL*)//进行左子树判断

{

if (T->lChild->data == x)

{

*cout* << x<<"的父节点的值为：" << T->*data* << *endl*; //输出父节点的值

if(T->rChild==*NULL*)

{

*cout*<<x<<"的右兄弟不存在！"<<*endl*; //判断右兄弟是否存在，存在输出其值

}

else

{

*cout*<<x<<"的右兄弟为："<<T->rChild->data<<*endl*;

}

if(T->lChild->lChild==*NULL*)

{

*cout*<<x<<"的左孩子不存在！"<<*endl*;

}

else

{

*cout* << x <<"的左孩子为："<<T->lChild->lChild->data<<*endl*;

}

if(T->lChild->rChild==*NULL*)

{

*cout* << x <<"的右孩子不存在！"<<*endl*;

}

else

{

*cout* << x <<"的右孩子为："<<T->lChild->lChild->data<<*endl*;

}

}

}

if(T->rChild!=*NULL*) //重复类似上面左子树的判断

{

if(T->rChild->data==x)

{

*cout* << x <<"的父节点的值为："<<T->*data*<<*endl*; //输出父节点的值

if(T->lChild==*NULL*)

{

*cout* << x <<"的左兄弟不存在！"<<*endl*; //判断左兄弟是否存在，存在输出其值

}

else

{

*cout* << x <<"的左兄弟为："<<T->lChild->data<<*endl*;

}

if(T->rChild->lChild==*NULL*)

{

*cout* << x <<"的左孩子不存在！"<<*endl*;

}

else

{

*cout* << x <<"的左孩子为："<<T->rChild->lChild->data<<*endl*;

}

if(T->rChild->rChild==*NULL*)

{

*cout* << x <<"的右孩子不存在！"<<*endl*;

}

else

{

*cout* << x <<"的右孩子为："<<T->rChild->rChild->data<<*endl*;

}

}

}

getfa\_bro\_ch(T->lChild,x); //递归调用

getfa\_bro\_ch(T->rChild,x);

}

}

void linkedtree::numgetlevel(BiNode\* T, elementType x, int level,int &flag)

{

//引入flag标志，当找到结果时停止递归的调用

if (T && flag == 0)

{

if (T->*data* == x)

{

*cout* << "该节点层次为:" << level << *endl*;

flag = 1;

}

numgetlevel(T->lChild, x, level + 1, flag);

numgetlevel(T->rChild, x, level + 1, flag);

}

}

void linkedtree::arrtreeTolink(BiNode\* T, elementType arr[], int i,int num)

{

if (i <= num && arr[i])

{

T = new BiNode;

if (arr[i] != '0')

T->*data* = arr[i];

T->lChild = *NULL*;

T->rChild = *NULL*;

//递归创建二叉链表

arrtreeTolink(T->lChild, arr, 2 \* i, num);

arrtreeTolink(T->rChild, arr, 2 \* i + 1, num);

}

}

void linkedtree::pointRtoL(BiNode\* T)

{

BiNode\* temp;

if (T != *NULL*)

{

//临时存储结点

temp = T->lChild;

T->lChild = T->rChild;

T->rChild = temp;

pointRtoL(T->lChild);

pointRtoL(T->rChild);

}

}

void linkedtree::deltree(BiNode\* T)

{

if (T)

{

deltree(T->lChild);

deltree(T->rChild);

delete T;

}

}

#include <iostream>

using namespace *std*;

#include "linkedtree.h"

int main()

{

*cout*<<"通过键盘输入数据"<<*endl*;

linkedtree A;

BiNode\* a = new BiNode;

A.createbitree(a);

*cout* << "先序"; A.firsttravel(a); *cout* << *endl*;

*cout* << "中序"; A.midtravel(a); *cout* << *endl*;

*cout* << "后序"; A.lasttravel(a); *cout* << *endl*;

A.deltree(a);

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

*cout* << "\*\*\*\*\*\*\*\*\*\*数组转二叉树\*\*\*\*\*\*\*\*\*\*" << *endl*;

linkedtree D;

BiNode\* d = new BiNode;

*cout* << "请依次输入字符(输入字符\\结束)：" << *endl*;

char mem; int m = 1, n = 0; char arr[100] = {0};

for (int i = 1; i < 100; i++)

{

*cin* >> mem;

if (mem == '\\') //注意转义字符的使用

break;

arr[i] = mem;

n++;

}

D.arrtreeTolink(d, arr, m, n);

*cout* << "读取已有的文件数据" << *endl*;

char strLine[100][3]; //保存二叉树文件数据

char filename[100] = "linktree.txt";//文件名

int nRow = 1;

int nLen;

linkedtree B;

BiNode\* b = *NULL*;

B.readfile(filename, strLine, nLen);

B.fileTotree(b, strLine, nLen, nRow);

*cout* << "先序"; B.firsttravel(b); *cout* << *endl*;

*cout* << "中序"; B.midtravel(b); *cout* << *endl*;

*cout* << "后序"; B.lasttravel(b); *cout* << *endl*;

int level = 0;

B.getnum\_level(b, level);

int depth = 0;

depth=B.getdepth(b);

*cout* << "二叉树高度" << depth << *endl*;

int nodenum = 0;

B.getnodenum(b,&nodenum);

*cout* << "结点数" << nodenum << *endl*;

int num0=0;

B.get0num(b, &num0);

*cout* << "叶子结点数" << num0 << *endl*;

int num2 = 0;

B.get2num(b, &num2);

*cout* << "度为2的结点数为" << num2 << *endl*;

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

*cout*<<"请输入要查询层次的元素"<<*endl*;

char tst3; int level1 = 0; int flag1 = 0;

*cin* >> tst3;

B.numgetlevel(b, tst3, level1,flag1);

if (flag1==0)

{

*cout*<<"元素不存在此树中"<<*endl*;

}

*cout* << "请输入要查询层次的元素" << *endl*;

char tst4; int level2 = 0; int flag2 = 0;

*cin* >> tst4;

B.numgetlevel(b, tst4, level2, flag2);

if (flag2 == 0)

{

*cout* << "元素不存在此树中" << *endl*;

}

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

*cout*<<"请输入要求父/兄/子结点的元素"<<*endl*;

char tst1;

*cin* >> tst1;

B.getfa\_bro\_ch(b, tst1);

*cout* << "请输入要求父/兄/子结点的元素" << *endl*;

char tst2;

*cin* >> tst2;

B.getfa\_bro\_ch(b, tst2);

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

*cout* << "交换前的二叉树（先序）" << *endl*;

B.firsttravel(b); *cout* << *endl*;

B.pointRtoL(b);

*cout* << "交换后的二叉树（先序）" << *endl*;

B.firsttravel(b); *cout* << *endl*;

B.deltree(b);

return 0;

}

**6.6 调试过程中出现的bug总结**

程序的主体结构参考了上次的链表实验，一些函数里直接进行递归调用减少了很多的代码量，过程中碰到了几个问题。一个是在创建二叉树函数时，传入参数时，一开始是调用的binode\*T,后来发现无法构造二叉树，确认函数内代码没问题后经过查阅资料发现应该改为binode\*&T调用实参，才能对T的内存进行修改。

第二个地方是进行计数时，传入的参数为计数变量的地址，在进行自增运算时要带上（），（\*num）++才是num+1；\*num++是指针后移一位。第三个地方是查询到元素层次后返回信息，由于函数主体用了递归，所以找到所需的元素仍然会继续，需要引入标识再找到元素后改变标识使得递归终止。第四个地方是文件的读入和处理，留到了最后才解决，单纯的数组转换为二叉树知道如何下手，但是对于文件的读入以及存入数组，我有些遗忘了，只能参考课本来进行处理，有时间复习之前io流文件处理之后再想办法进一步优化。