项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 何慧琳

学 号： 2152343

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

家谱是一种以表谱形式，记载一个以血缘关系为主体的家族世袭繁衍和重要任务事迹的特殊图书体裁。家谱是中国特有的文化遗产，是中华民族的三大文献（国史，地志，族谱）之一，属于珍贵的人文资料，对于历史学，民俗学，人口学，社会学和经济学的深入研究，均有其不可替代的独特功能。本项目兑对家谱管理进行简单的模拟，以实现查看祖先和子孙个人信息，插入家族成员，删除家族成员的功能。随着家族人数的日益庞大，如何管理如此庞大的数据显得极为复杂，传统的手工管理工作量大且容易出错。

随着计算机科学技术的不断成熟，使用计算机对家谱进行管理，具有手工管理所无法比拟的优势。这些优点能够极大地提高家谱管理的效率，也是家谱这一传统文化与信息时代结合的重要条件。因此，开发一套家谱管理系统具有十分重要的意义。

## 1.2 功能分析

家谱由家族成员的关系和信息组成，一个家族随时有成员的加入、退出或者成员信息的更改。所以作为一个最简易的家谱管理系统，应当能够完成对家谱成员信息的建立、查找、插入、修改、删除等功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求能够表示各成员之间的辈分关系并且涉及大量的插入、修改、删除操作，因此考虑采用树的数据结构。又由于一个父亲可能有多个儿女，于是采用左长子右兄弟的表示方法。家谱中的每个成员即是一个树节点，因此还定义了一个树节点类，其中将树类声明为友元。在添加、删除成员信息时首先要查找成员的插入或删除位置，查找时采用前序遍历，前序遍历过程中要将经历过但未访问的结点入栈，因此还需定义一个栈类。

## 2.2 类结构设计

**栈类（Stack）**

为了方便增删和开辟空间，采用链式栈，其基本组成结构是一个个栈结点，因此在栈类中还定义了一个栈的结点类（class stacknode），即采用了嵌套结构。栈的结点应包含结点数据、链接指针。栈由于其是先进后出式结构，只需要一个stacknode类型的栈顶指针即可。由于其要实现插入元素、删除元素等功能，定义了push()、pop()等函数，具体见下。

**私有成员：**

class stacknode {//栈的结点类

public:

Type data;//结点数据

stacknode\* link;

stacknode(Type d=NULL,stacknode\* l=NULL) :data(d),link(l) {};//栈的结点构造函数

};

stacknode\* top;//栈顶指针

**公有操作：**

stack() :top(NULL) {};//无参构造函数

stack(Type d) { top = new stacknode(d); }//带参构造函数

~stack();//析构函数，与make\_empty操作同

void push(Type d) { top = new stacknode(d, top); }//将以d为值的结点入栈

Type pop();//删除栈顶结点

Type gettop() {return top->data;}//取栈顶结点

void make\_empty();//清空栈

int is\_empty() { return top == NULL; }//判断栈是否为空

**家谱树结点类（family\_treenode）**

一个家族成员首要的信息就是自己的名字，所以类中定义了私有成员char\* name，除此之外还应包含与家人的关系信息，又采用的是左长子右兄弟的表示方法，所以定义了两个family\_treenode类型的指针分别指向左长子，右兄弟。又因为解散成员家庭时要将他的所有子女逐个从树中删除，这就要求从叶结点开始删除，故采用后序遍历。后序遍历时所经过的根节点会两次出栈，第一次出栈进入它的右子树，第二次出栈才访问它。为了区别当次出栈应做什么操作，在结点类中定义了一个变量int visit以标记结点被经过的次数。

**私有成员及操作：**

friend class family\_tree;

char\* name;//成员名

family\_treenode\* first\_child, \* next\_sibling;//指向左长子，右兄弟

int visit=0;//删除成员后序遍历时经过次数

family\_treenode(char\* n=NULL) :name(n),first\_child(NULL),next\_sibling(NULL) {};//构造函数

family\_treenode(family\_treenode\* ftn) :name(ftn->name),first\_child(ftn->first\_child),next\_sibling(ftn->next\_sibling) {};//复制构造函数

**家谱树类（family\_tree）**

要找到一棵树给出其根节点位置即可所以定义了family\_tree类型的指针ancesto指向根节点，此外为了方便成员的插入、查找定义了一个游标指针current。如前面功能分析家谱管理系统应实现添加成员、成员信息的更改、解散成员家庭的功能，及插入、修改、删除功能，于是有如下成员及函数设计。

**私有成员：**

family\_treenode\* ancestor,\*current;//定义指向根节点的指针和指向当前节点的指针

family\_treenode\* preorder(char \*n);//前序遍历查找名为\*n的结点位置

**公有操作：**

family\_tree() :ancestor(NULL), current(NULL) {};//无参构造函数

family\_tree(char \*n);//以祖先名为参数的构造函数

~family\_tree();//析构函数，操作同make\_empty()

void add\_member(int build\_or\_add);//为成员建立家庭或添加子女

void dissolve\_family(int all=0);//解散某人的家庭,若all为1，则解散祖先的家庭

void change\_name();//更改某人的姓名

void make\_empty();//清空家谱

## 2.4 系统设计

系统首先调用main()函数实现对屏幕的初始化，并完成操作的选择和跳转，跳转到的家谱管理功能通过调用family\_tree类中的函数来实现。

# 3 实现

## 3.1 插入功能的实现

### 3.1.1插入功能核心代码

void family\_tree::add\_member(int build\_or\_add)

{//build\_or\_and为0，建立家庭；为1，为某个人在家谱中添加儿女

char\* parent=new char[20];//要建立家庭的人的姓名

int num=1;//要添加的子女数

family\_treenode\* temp=current;//一开始指向长子，方便最后遍历输出parent的第一代子孙

if(build\_or\_add==0)

cout << "请输入要建立家庭的人的姓名：";

else

cout << "请输入要添加儿子或女儿的人的姓名：";

cin >> parent;

if (build\_or\_add == 0) {

cout << "请输入" << parent << "的儿女人数：";

cin.ignore(20, '\n');

...

}

preorder(parent);//查找parent，使current指向其位置

if (current == NULL) {

cout << parent << "不存在！" << endl;

return;

}

if (build\_or\_add == 0)

cout << "请依次输入" << parent << " 的儿女的姓名：";

else

cout << "请输入" << parent << "新添加的儿子（或女儿）的姓名：";

for (int i = 0; i < num; i++) {//逐个完成缓冲区儿女姓名的读取和在家谱中的插入

char\* child = new char[20];//子女的姓名

cin >> child;

if (i == 0) {//插入第一个孩子

if (build\_or\_add == 1) {

if (current->first\_child!=NULL) {

current = current->first\_child;

temp = current;//temp指向长子

while (current->next\_sibling!=NULL) //在最后一个孩子后插入新孩子

current = current->next\_sibling;

current->next\_sibling = new family\_treenode(child);//完成插入新孩子

}

else {

current->first\_child = new family\_treenode(child);//完成插入新孩子

temp = current->first\_child;

}

}

else {

current->first\_child = new family\_treenode(child);//插入parent的长子

temp = current->first\_child;//指向长子

current = current->first\_child;//current指向刚被插入的孩子，为插入下一个孩子做准备

}

}

else {

current->next\_sibling = new family\_treenode(child);//插入parent非长子的孩子即插入长子的右兄弟

current = current->next\_sibling;//current指向刚被插入的孩子，为插入下一个孩子做准备

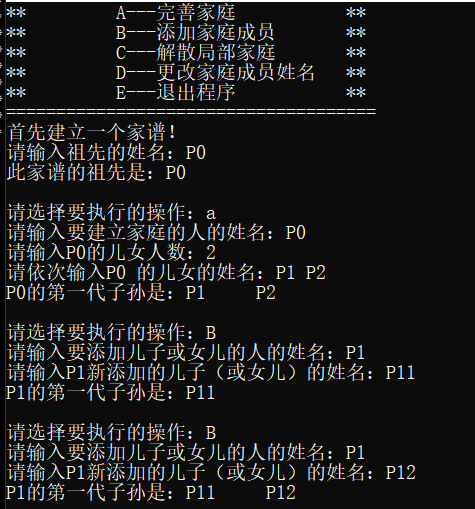
}

}

...

}

### 3.1.2 插入功能截屏示例



## 3.2 删除功能的实现

### 3.2.1 删除功能核心代码

void family\_tree::dissolve\_family(int all)

{//解散以\*parent为名的人的家庭,当all==1时相当于清空整个家谱树（除祖先）

char\* parent = new char[20];//要解散家庭的人的姓名

...

family\_treenode \*p=preorder(parent);//查找parent，使p指向其位置

if (current == NULL) {

cout << parent << "不存在！" << endl;

return;

}

family\_treenode\* temp;

if (!all) {//不是用于清空功能，需要输出被解散家庭的人的第一代子孙

temp = current->first\_child;//temp指向长子

for (int i = 0; temp; i++) {//遍历输出parent的第一代子孙

cout << temp->name << " ";//输出当前孩子的姓名

temp = temp->next\_sibling;//temp指向下一个孩子

}

cout << endl;

}

current= current->first\_child;//current指向长子

stack<family\_treenode\*> st;//存储遍历途中经过但未被访问的结点

while (current || !st.is\_empty()) {//删除工作

while (current) {//一左到底

if (current->visit >= 2)//右子树也遍历完毕该节点成为叶结点

break;

st.push(current);//经历过的结点都入栈

current = current->first\_child;//进入左子树

}

current = st.gettop();

current->visit++;

if (current->next\_sibling==NULL) {//当前节点既无左子树也无右子树，是叶节点，若不是parent则删除

family\_treenode\* temp1;

temp1 = current;

if(!st.is\_empty())

st.pop();//结点出栈

if (st.is\_empty())//栈空则所有元素已处理完，退出循环

break;

current = st.gettop();//current指向下一个未被访问的结点

current->next\_sibling = temp1->next\_sibling;//current的兄弟置空

p->first\_child= temp1->next\_sibling;//使parent的长子为空 if(current)

current->visit++;

delete temp1;//释放temp结点;??为啥删完这个结点后其前一个结点的next\_sibling显示读取异常

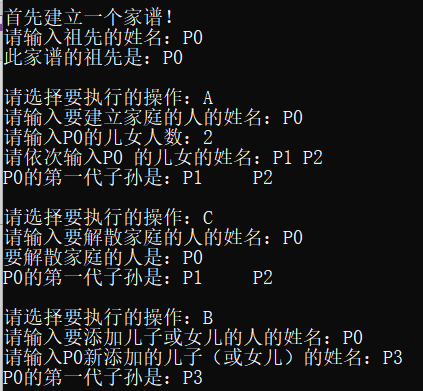
}

else//当前结点存在右孩子，则进入右子树

current = current->next\_sibling;//current指向右孩子

}

### }3.2.2 删除功能截屏示例



## 3.3 修改功能的实现

### 3.3.1 修改功能核心代码

void family\_tree::change\_name()

{//更改当前姓名为\*cur\_name的人的姓名

char\* name = new char[20];//要修改的人的姓名；

char\* toname = new char[20];//更改后的姓名

cout << "请输入要更改姓名的人的目前姓名：";

cin >> name;

preorder(name);////前序遍历查找名为\*name的结点位置

if (current == NULL) {

cout << name << "不存在！" << endl;

return;

}

cout << "请输入更改后的姓名：";

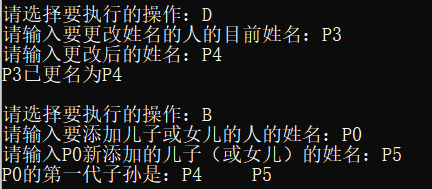
cin >> toname;

current->name = toname;

cout << name << "已更名为" << toname<<endl;

}

### 3.3.2 修改功能截图示例



## 3.4 总体系统的实现

### 3.4.1总体系统核心代码

switch (menu\_chocie) {

case 'A':

ft.add\_member(0);

cout << endl;

break;

case 'B':

ft.add\_member(1);

cout << endl;

break;

case 'C':

ft.dissolve\_family();

cout << endl;

break;

case 'D':

ft.change\_name();

cout << endl;

break;

case 'E':

cout << "程序结束！";

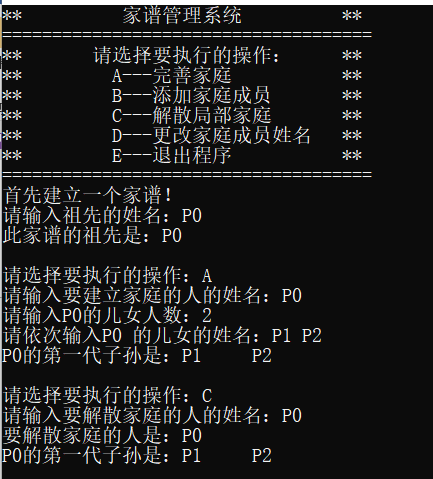
return 0;

default:

break;

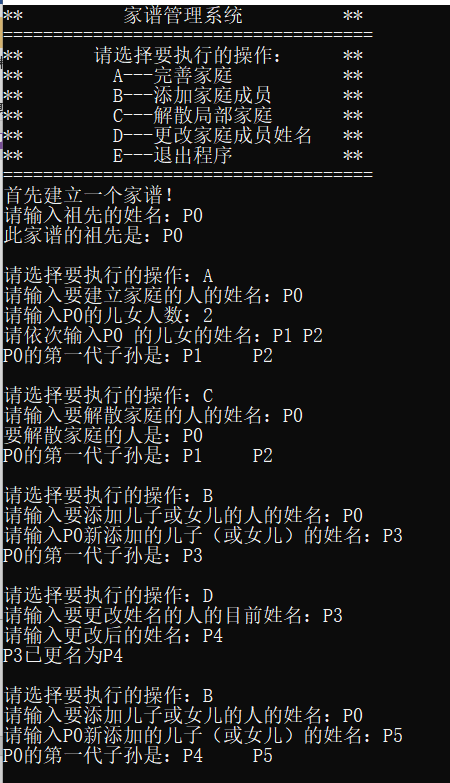
}

### 3.4.2 总体系统截屏示例



# 4 测试

## 4.1 功能测试



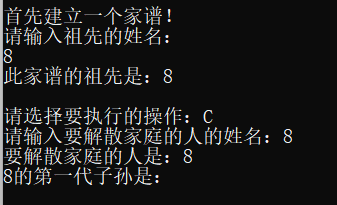
## 4.2 边界测试

### 4.2.1 解散无子女的人的家庭

**测试用例：**选择操作码C，输入无子女的人的名字

**预期结果：**程序运行正常不崩溃。

**实验结果：**

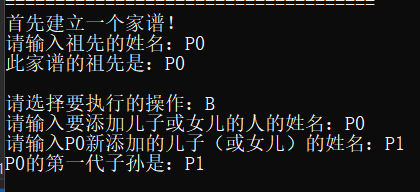


### 4.2.2 为某成员添加子女时该成员尚无子女

**测试用例：**选择操作B,选择一个无子女的人添加一个子女

**预期结果：**程序正常运行。

**实验结果：**



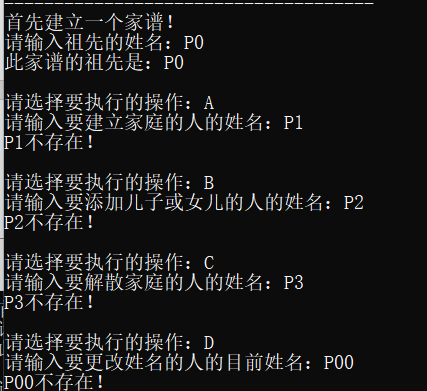
## 4.3 出错测试

### 4.3.1 输入的成员在家谱中不存在

**测试用例：**选择操作A/B/C/D,输入不在家谱中的人名

**预期结果：**程序给出提示，不崩溃。

**实验结果：**

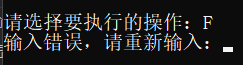
****

### 4.3.2 操作码错误

**测试用例：**输入操作码错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****