同濟大學
TONGJI UNIVERSITY

项目说明文档

家谱管理系统

指导老师：张颖

1851009 沈益立

目录

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。本项目的目的是对家谱管理进行简单的模拟

1.2 功能分析

在一一个简单的家谱管理模型中，应至至少具备创建、完善、添加成员、解散局部家庭、更改姓名和错 误提示几个基本功能，本程序通过输入不同的操作码来实现这些不同的功能。

2.设计

2.1 数据结构设计

查找了很多家谱资料，发现大多数家谱都以树的数据结构形式组织，这种处理方式，一方面来说可以清晰地表达出亲子之间的上下级关系，另一方面可以体现出兄弟之间的分支关系。

另外，由于本系统需要实现的操作包含了大量的插入、删除，因此，用链表的形式，以森林来实现树是良好的选择。

综上，为解决本问题，我采取了长子-兄弟表示法来储存必要的数据。

2.2 类结构设计

由于作业的相关规定，实现了自己定义的用数组模拟的可变长的栈`Stack<T>`，以及表示二元数对的`MyPair<T1, T2>`。为简化IO、增加代码复用性和美观性，同时设计了类`Maze`，用于输入迷宫、解决迷宫问题和输出迷宫中可行的路径。

2.3 成员和操作设计

经典的树结构一般包括两个抽象数据类型(ADT)——树节点类`FamilyMember`和树类`FamilyTree`， 而两个类之间的耦合关系可以采用嵌套、继承等多种关系。本程序中将树节点类作为树类的成员变量，使得家谱树可以访问成员。同时，我设计了一个解决方案类`Solution`来调用树类，实现功能控制。

树节点类`FamilyMember`

公有操作：

```
FamilyMember() {};                          //无参构造函数
FamilyMember(FamilyMember* nextSibling, FamilyMember*
firstChild,
    string name) :                          //三个参数的构造函数
    m_nextSibling(nextSibling), m_firstChild(firstChild),
m_name(name) {};
FamilyMember(string name) :                 //仅包含名字的构造函
数
    m_name(name) {};
```

公有成员:

```
FamilyMember* m_nextSibling = nullptr;  //存储下一个兄弟
FamilyMember* m_firstChild = nullptr;   //存储长子
string m_name;                          //存储姓名
```

树类`FamilyTree`:

公有操作:

```
FamilyTree() :                  //无参构造函数
m_root(nullptr) {};
void initTree();                //树的初始化IO，输入根节点的值
void promoteTree();             //完善树的某节点
FamilyMember* getMemberAdd(FamilyMember* p, string name);
                                //根据姓名递归地得到某子成员地址
FamilyMember* getPreAdd(FamilyMember* p, string name);
                                //根据姓名递归地得到某子成员的前继地
址
void addMember();               //为某双亲增加孩子
void dissolve();                //解散局部家庭
void recursiveDissolve(FamilyMember* root);
                                //递归地删除某个树及其子树
void changeName();              //为某家庭成员改名
```

公有成员:

```
FamilyMember* m_root;           //存储家族树的根节点
```

解决方案类`Solution`

公有操作:

```
void init();              //初始化屏幕
void IOLoop();            //实现功能IO
```

公有成员

```
FamilyTree m_tree;        //私有的家谱树存储
```

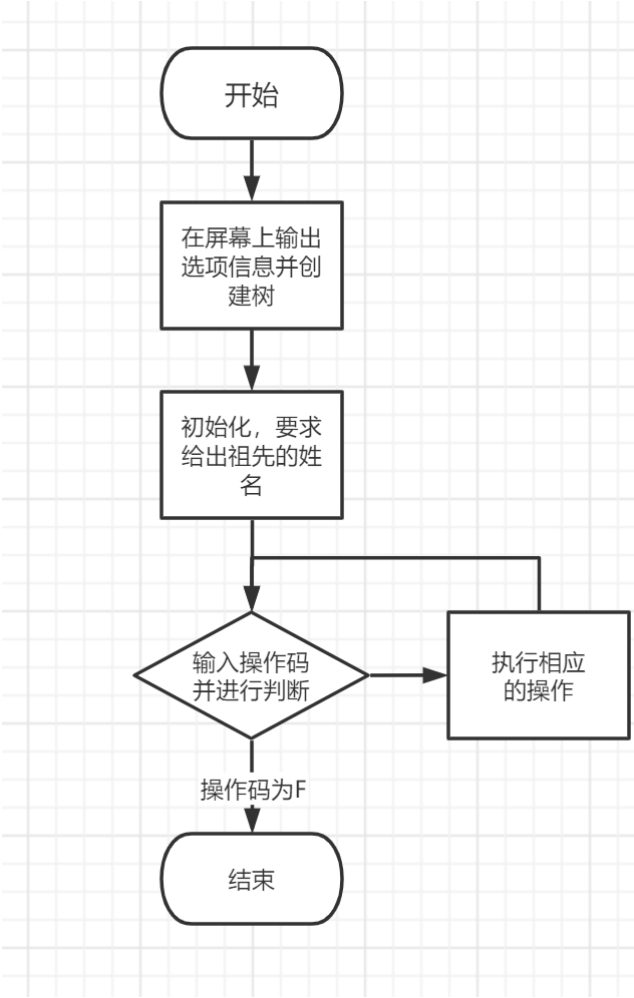
2.4 系统设计

程序运行后，系统会自动创建一个解决方案的实例化对象`sol`，随后调用基本的IO来实现屏幕的初始化和功能输入，建立家谱树的祖先；之后，程序会根据用户的输入进行选择，并根据用户的输入执行不同的操作。

3.实现

3.1 系统总体功能的实现

3.1.1 系统总体功能流程图



3.1.2 系统总体功能核心代码实现

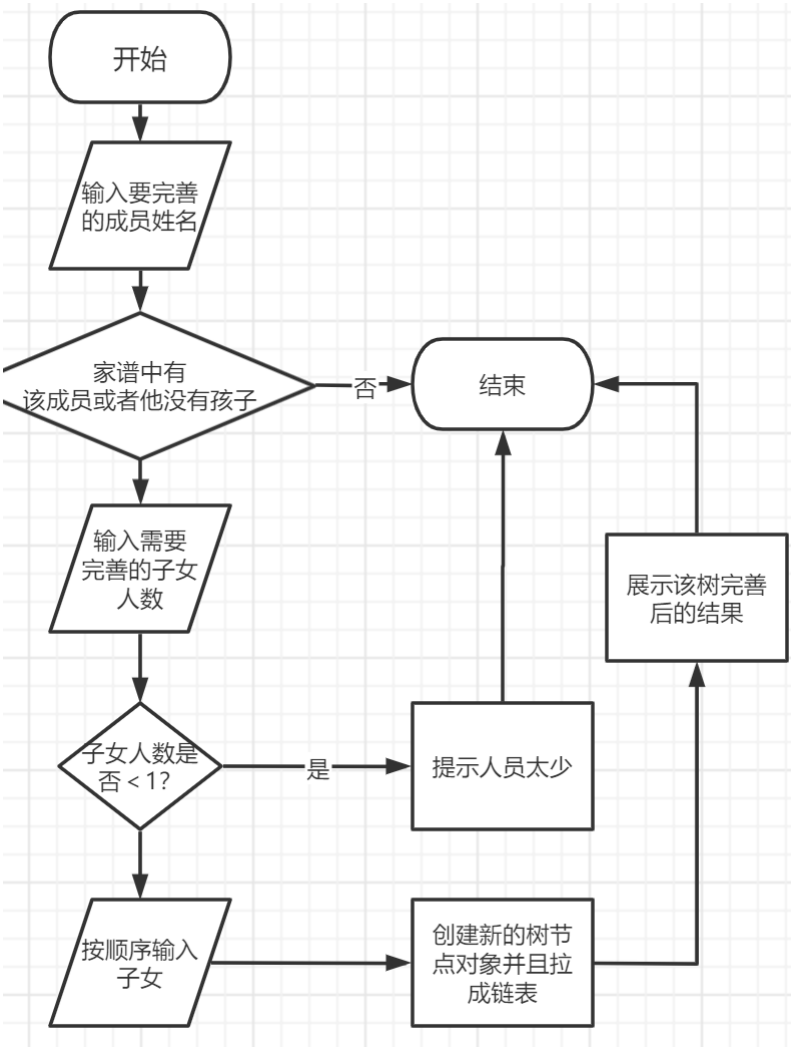
```
void Solution::init() {
    cout << "**                  家谱管理系统                  **" <<
endl;
    cout << "----------------------------------------" <<
endl;
    cout << "**                  请选择要执行的操作            **" <<
endl;
```

```
    cout << "**                A --- 完善家谱                **" <<
endl;
    cout << "**              B --- 添加家庭成员              **" <<
endl;
    cout << "**              C --- 解散局部家庭              **" <<
endl;
    cout << "**           D --- 更改家庭成员姓名             **" <<
endl;
    cout << "**              E --- 查找家庭成员              **" <<
endl;

    cout << "**                F --- 退出程序                **" <<
endl;
    cout << "----------------------------------------" <<
endl;
    //cout << "首先建立一个家谱! " << endl;
}


void Solution::IOLoop() {
    m_tree = FamilyTree();
    m_tree.initTree();
    char optCode;
    while (1) {
        cout << "请输入要执行的操作:";
        cin >> optCode;
        if (optCode == 'F') {
            break;
        }
        else if (optCode == 'A') {
            m_tree.promoteTree();
        }
        else if (optCode == 'B') {
            m_tree.addMember();
        }
        else if (optCode == 'C') {
            m_tree.dissolve();
        }
        else if (optCode == 'D') {
            m_tree.changeName();
        }
```

```
        else if (optCode == 'G') {
            assert(0);
        }
        else if (optCode == 'E') {
            string memberName;
            cout << "请输入要找的家庭成员姓名:";
            cin >> memberName;

            auto address =
m_tree.getMemberAdd(m_tree.m_root, memberName);
            if (!address) {
                cout << "未找到该成员" << endl;
                return;
            }

            auto pCur = address->m_firstChild;
            if (!pCur) {
                cout << memberName << "没有子孙" << endl;
                return;
            }
            cout << memberName << "的第一代子孙是:";
            while (pCur) {
                cout << pCur->m_name << '\t';
                pCur = pCur->m_nextSibling;
            }
        }
        else {
            cerr << "请输入正确的操作码" << endl;
        }
        cout << endl;
    }
}
```

3.2 完善功能的实现

3.2.1 完善功能流程图



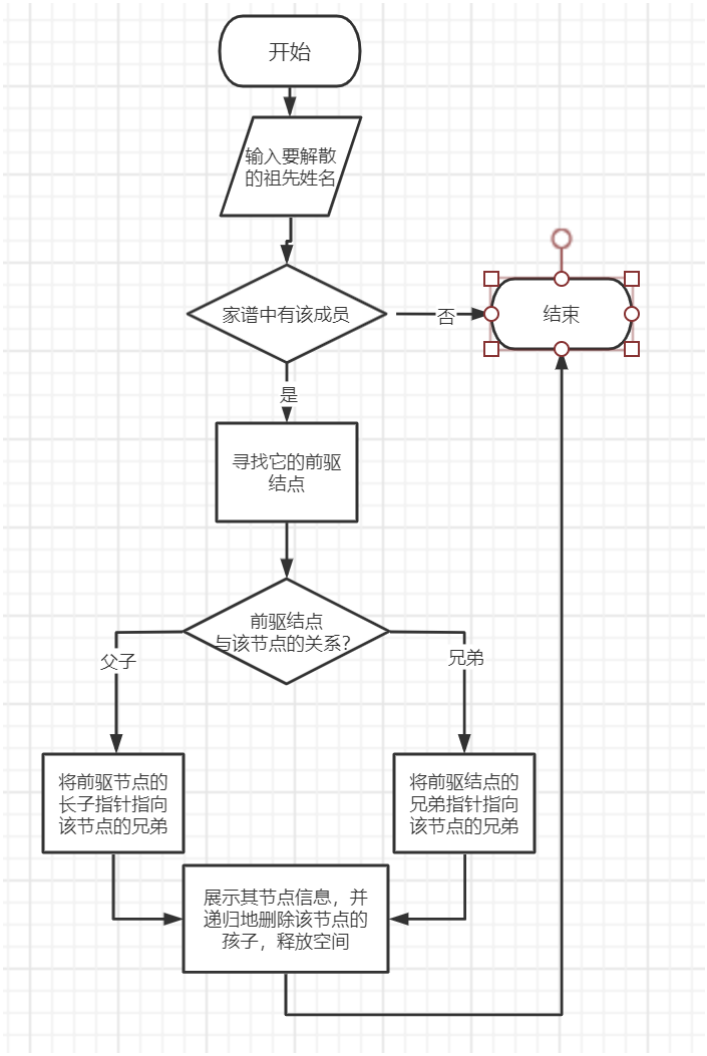
3.2.2 完善功能核心代码实现

```
void FamilyTree::promoteTree() {
    string memberName;
    int siblingNum;
    cout << "请输入要建立家庭的人的姓名:";
    cin >> memberName;
    auto targetMember = getMemberAdd(m_root, memberName);

    if (! targetMember) {
        cerr << "家谱中找不到该成员。" << endl;
        return;
    }
    if (targetMember->m_firstChild) {
        cerr << "该成员已经有子女了" << endl;
        return;
    }
```

```
    cout << "请输入" << memberName << "的儿女人数:";
    cin >> siblingNum;
    if (siblingNum < 1) {
        cout << "子孙节点数太少，无需添加" << endl;
        return;
    }

    cout << "请依次输入" << memberName << "的儿女姓名:";
    cin >> memberName;
    auto firstChild = new FamilyMember(memberName);
    targetMember->m_firstChild = firstChild;
    auto pCur = firstChild;
    for (int i = 0; i < siblingNum - 1; i++) {
        cin >> memberName;
        auto newMember = new FamilyMember(memberName);
        pCur->m_nextSibling = newMember;
        pCur = newMember;
    }
    pCur = firstChild;
    cout << targetMember->m_name << "的第一代子孙是:";
    while (pCur) {
        cout << pCur->m_name << '\t';
        pCur = pCur->m_nextSibling;
    }
    cout << endl;
}
```

3.3 解散家庭功能的实现

3.3.1 解散成员功能流程图



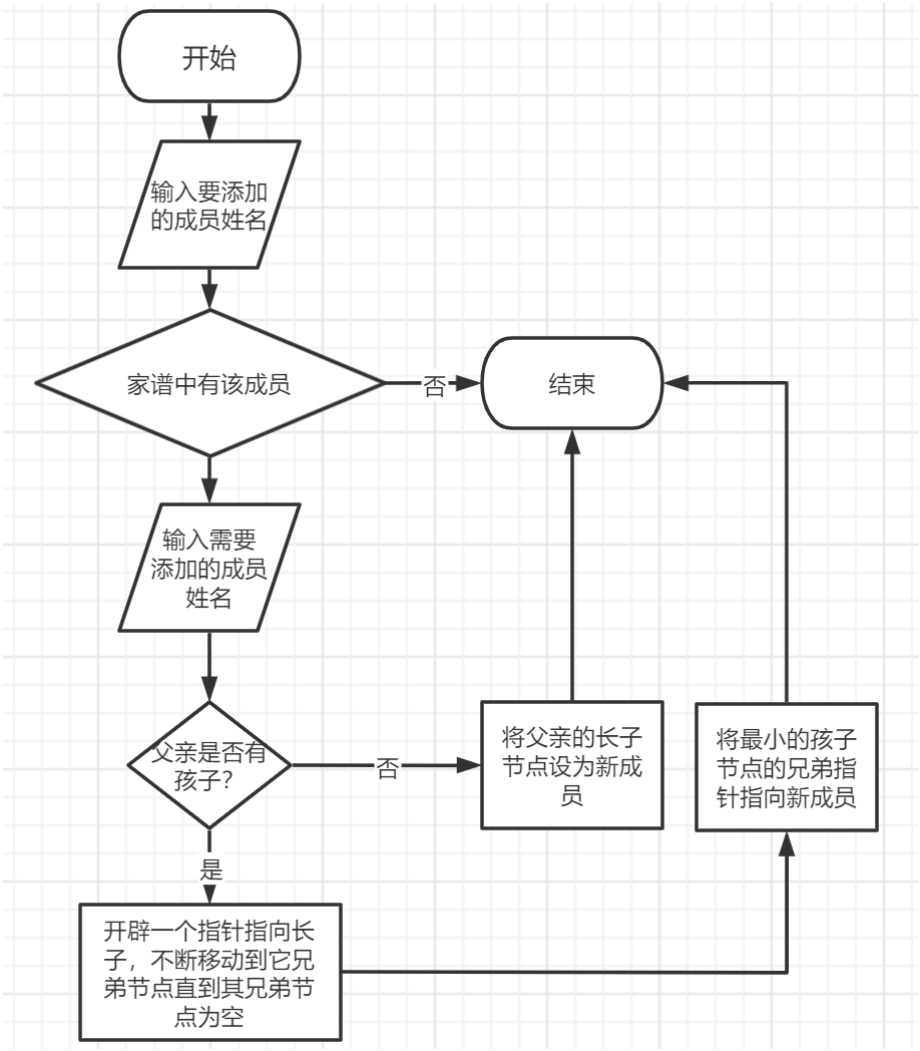
3.3.2 解散家庭核心代码实现

```
void FamilyTree::dissolve() {
    cout << "请输入要解散家庭的人的姓名:";
    string memberName;
    cin >> memberName;
    auto preAdd = getPreAdd(m_root, memberName);
    if (!preAdd) {
        cout << "不存在这样的家庭" << endl;
        return;
    }
    FamilyMember* targetTree = nullptr;
    if (preAdd->m_firstChild->m_name == memberName) {
        targetTree = preAdd->m_firstChild;
        preAdd->m_firstChild = targetTree->m_nextSibling;
    }
    else {
```

```
            targetTree = preAdd->m_nextSibling;
            preAdd->m_nextSibling = targetTree->m_nextSibling;
        }
        auto pCur = targetTree->m_firstChild;
        if (!pCur) {
            cout << memberName << "没有子孙" << endl;
            recursiveDissolve(targetTree);
            return;
        }

        cout << memberName << "的第一代子孙是:";
        while (pCur) {
            cout << pCur->m_name << '\t';
            pCur = pCur->m_nextSibling;
        }
        cout << endl;
        recursiveDissolve(targetTree);
}
```

3.4 添加成员功能的实现

3.4.1 添加成员功能流程图



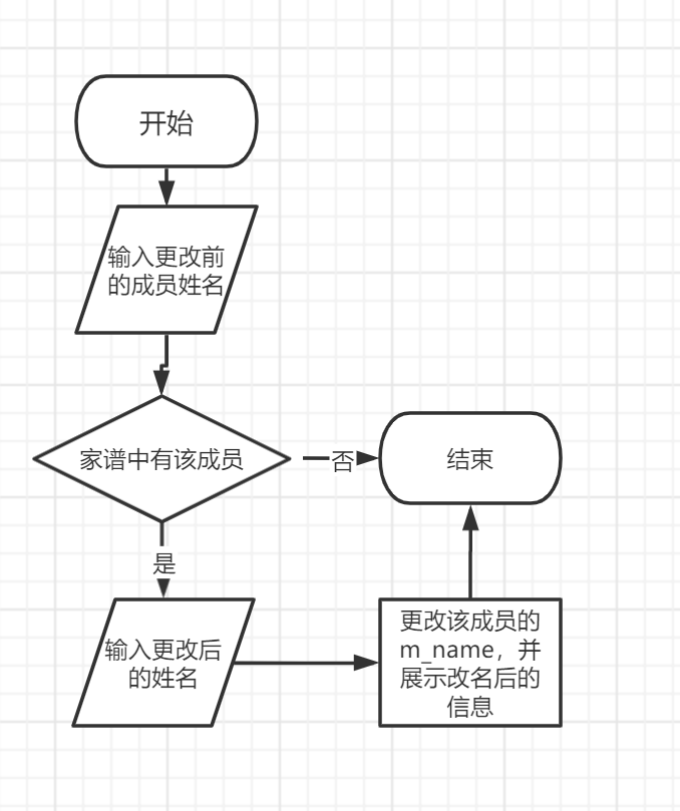
3.4.2 添加成员核心代码实现

```
void FamilyTree::addMember() {
    cout << "请输入要添加子女的人的姓名:";
    string memberName;
    cin >> memberName;
    auto targetMember = getMemberAdd(m_root, memberName);
    if (!targetMember) {
        cerr << "家庭中没有该成员" << endl;
        return;
    }
    auto pCur = targetMember->m_firstChild;
    if (!pCur) {//没有子女
        cout << "请输入" << memberName << "新添加的子女的姓
名:";
        cin >> memberName;
```

```
            targetMember->m_firstChild = new
FamilyMember(memberName);
            return;
        }
        while (pCur->m_nextSibling) {
            pCur = pCur->m_nextSibling;
        }

        cout << "请输入" << memberName << "新添加的子女的姓名:";
        cin >> memberName;
        pCur->m_nextSibling = new FamilyMember(memberName);
}
```

3.5 改名功能的实现

3.5.1 改名功能流程图



3.5.2 改名核心代码实现

```
void FamilyTree::changeName() {
    cout << "请输入要修改的成员目前的姓名:";
    string memberName;
    cin >> memberName;
    auto targetMember = getMemberAdd(m_root, memberName);
    if (!targetMember) {
        cout << "此人不存在" << endl;
        return;
    }
    cout << "请输入更改后的姓名:" ;
    string targetName;
    cin >> targetName;
    targetMember->m_name = targetName;
    cout << memberName << "已更名为" << targetName << endl;
}
```

4.测试

4.1 常规测试

4.1.1 初始化家谱

测试用例:

```
P0
```

预期结果:

```
基本IO, 输出祖先P0
```

实验结果:

4.1.2 完善家庭

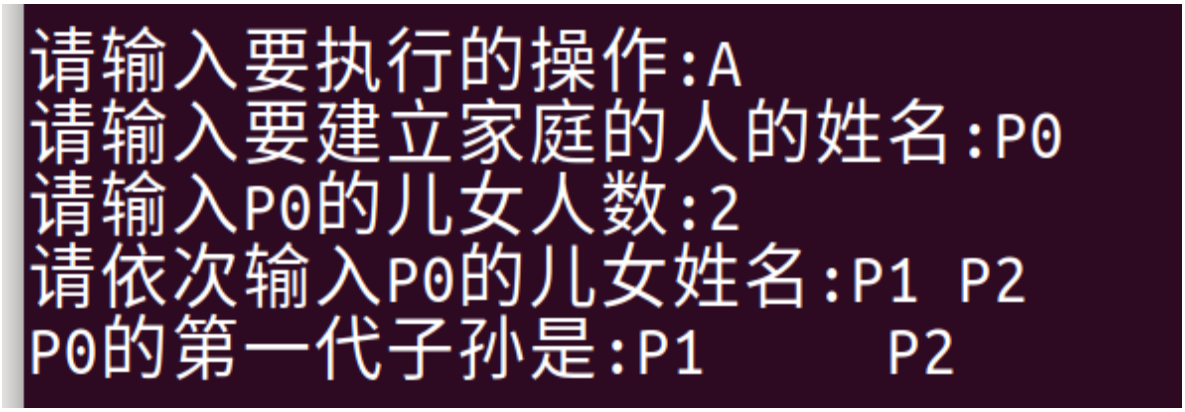
测试用例:

```
A
2
P1 P2
```

预期结果：

```
    P0
   /  \
  P1   P2
```

实验结果：



4.1.3 完善下一代家庭

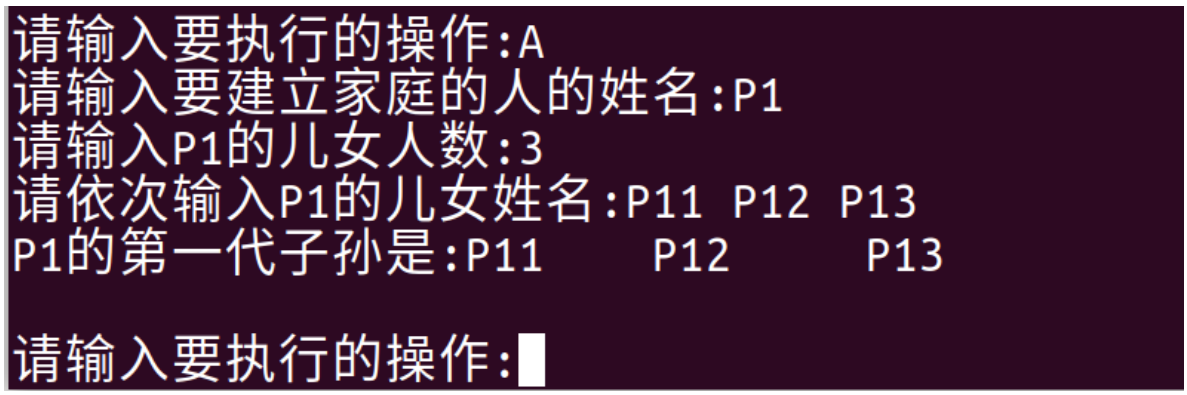
测试用例：

```
A
P1
3
P11 P12 P13
```

预期结果：

```
    P1
   / | \
P11 P12 P13
```

实验结果：



4.1.4 添加子女

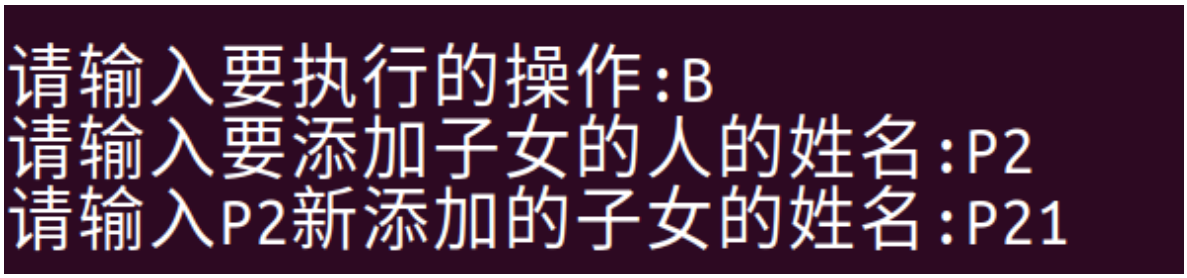
测试用例：

```
B
P2
P21
```

预期结果：

```
P2
|
P21
```

实验结果：



4.1.5 解散家庭

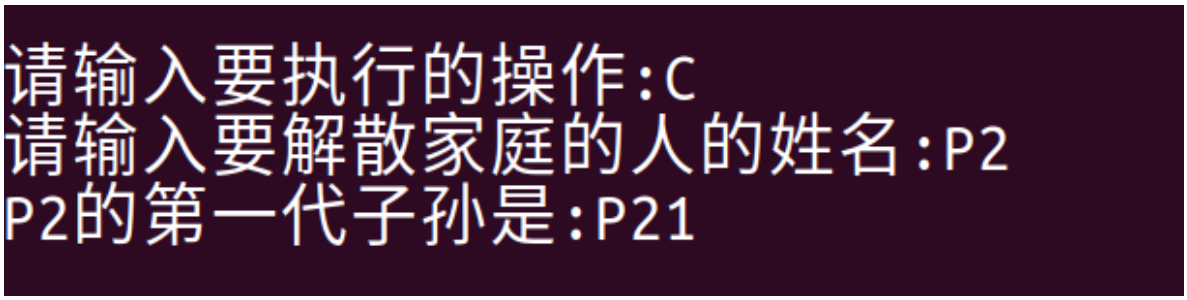
测试用例：

```
C
P2
```

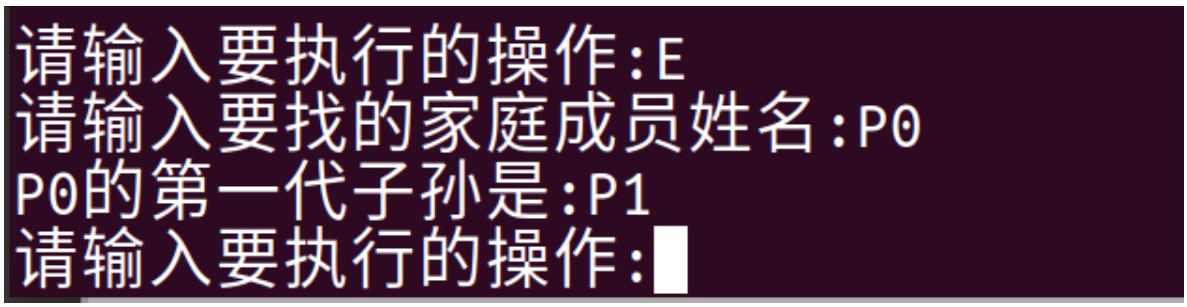
预期结果：

```
P0的子女无P2
```

实验结果：



由于已解散，可以看到`P0`的子女已经无`P2`



4.1.6 改名

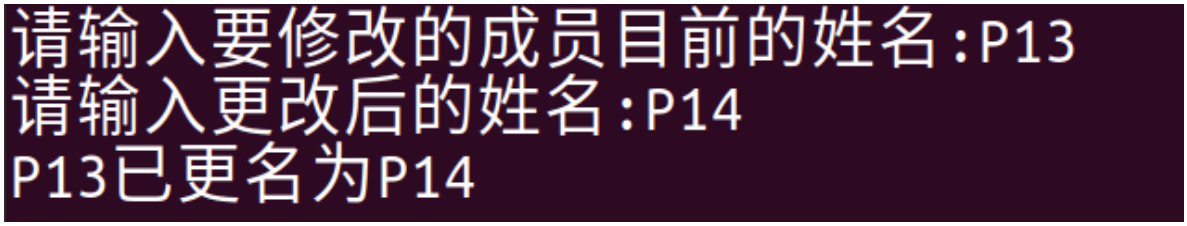
测试用例：

```
D
P13
P14
```

预期结果：

```
P14
```

实验结果：



4.2 错误测试

4.2.1 不存在的家庭成员

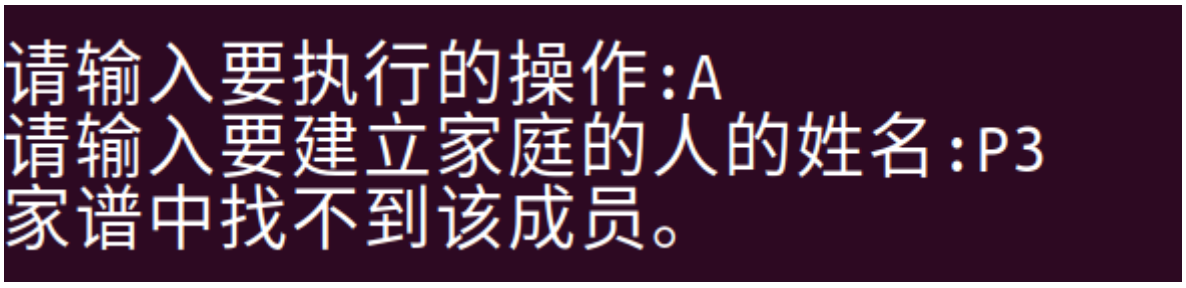
测试用例：

```
A
P3
```

预期结果：

```
提示找不到该成员
```

实验结果：



4.2.2 不能完善的成员

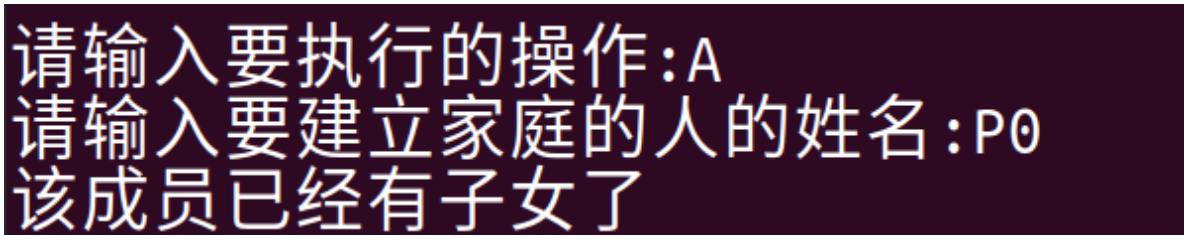
测试用例：

```
A
P0
```

预期结果：

```
提示该成员已经有子女了
```

实验结果：



4.2.3 错误的人数

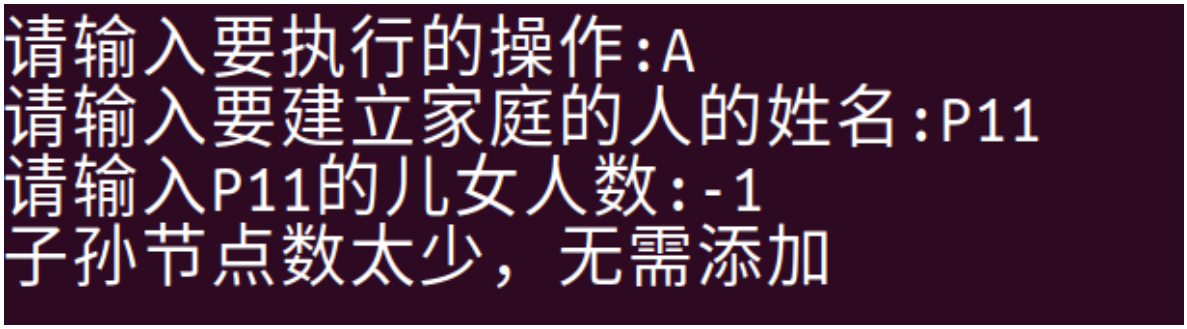
测试用例:

```
A
P11
-1
```

预期结果:

```
提示子孙的节点数太少
```

实验结果:



4.2.4 操作码错误

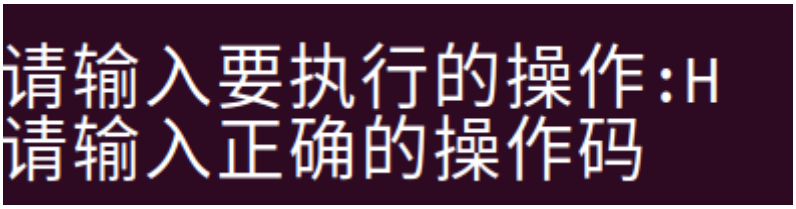
测试用例:

```
A
H
```

预期结果:

```
提示操作码错误
```

实验结果:


请输入要执行的操作:H
请输入正确的操作码