项目说明文档

数据结构课程设计

——**家谱管理系统**

作 者 姓 名： 肖杨

学 号： 1950430

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

1 分析 3

2 设计 3

3 实现 9

4 测试 17

1 分析

1.1 背景分析

家谱，又称族谱、家乘、祖谱、谱牒、宗谱等，是记载某个姓氏家族子孙世系传承之书，具有区分家族成员血缘关系亲疏远近的作用，是中国封建宗法制度的产物。随着历史的发展，家谱由官修变为私修，所录内容不断丰富，其作用也不断增加和变化。如今，家谱同各姓氏的郡望、堂号一样，不仅为区别姓氏源流，可作为数典认祖、研究历史、地理、社会、民俗等参考资料，它还是姓氏文化的重要组成部分。

1.2 功能分析

作为一个家谱管理系统，其作用是帮助用户对家谱进行管理，以管理家庭中成员的信息。首先应该有的功能就是建立家谱，其次，家谱管理系统还应该具有查找功能，这是家谱其他功能的基础。然后家谱还应该具有插入、修改、删除的功能，以此来实现对家谱中家庭成员信息的管理。

综上所述，该家谱管理系统需要有建立家谱、查找、插入、删除、修改、退出的功能。

2 设计

2.1 数据结构设计

如上功能分析所述，我们需要频繁地进行数据的插入与删除，并且家谱的结构也是一种树的结构，所以本项目采用多叉树的数据结构来存储家谱，因为家谱中每个结点的子女结点的个数可能不尽相同，但为了保障效率和形式统一，子女仍然选择用树结点表示，没有采用链表表示，使用时选择额外开辟连续内存保存指向子女的指针。

同时考虑到遍历树的操作，遍历时使用栈进行前序遍历，因此需要使用栈。

2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目的栈采用模板类，树未进行单独实现，选择直接进行实例化的家谱类，提供了添加、删除子女等操作的接口。查找时采用前序遍历，先访问根节点再访问子节点。

2.3 成员与操作设计

树成员类（结点类）

成员类是FamilyList内部的类

类定义

class Member

{

public:

string name;

int childnum = 0;

Member\*\* children = nullptr;

Member(string val)

{

name = val;

}

Member() = default;

~Member()

{

removeChildren();

delete[] children;

}

void removeChildren()

{

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

{

delete children[i];

}

childnum = 0;

}

void setChildnum(int size)

{

children = new Member\*[size];

}

void addChildren(string newChild)

{

childnum++;

Member\*\* temp = children;

children = new Member\*[childnum];

memcpy(children, temp, (childnum - 1) \* sizeof(Member\*));

children[childnum - 1] = new Member(newChild);

delete[] temp;

}

friend ostream& operator <<(ostream& os, const Member& person)

{

cout << person.name;

if (person.childnum == 0)

{

cout << "无子女" << endl;

}

else

{

cout << "有" << person.childnum << "个子女，分别为：";

for (int i = 0; i < person.childnum; i++)

{

cout << person.children[i]->name << " ";

}

cout << endl;

}

return os;

}

};

成员：

string name;

//成员名称

int childnum = 0;

//成员子女数量

Member\*\* children = nullptr;

//指向成员子女指针的指针

操作：

Member(string val)

//构造函数，用姓名初始化

void removeChildren()

//移除所有子女

void setChildnum(int size)

//设置子女数量

void addChildren(string newChild)

//已经有子女，向其中添加新子女

//或没有子女的时候，进行调用，会添加一个子女

friend ostream& operator <<(ostream& os, const Member& person)

//重载输出

家谱类

类定义

class FamilyList

{

private:

//已省略Member类的内容

Member\* root;

public:

FamilyList();

~FamilyList();

void insert(string parent);

void addChild(string parent, string child);

void removeChildren(string parent);

Member\* find(const string& name);

void changeName(string preName, string postName);

bool getOrder();

bool sameName(const string& name);

void getNumber(int& temp);

};

私有成员：

Member\* root;

//祖先结点，这个节点在运行中始终不会被删除

公有操作：

FamilyList();

//构造函数，用于得到祖先结点

void insert(string parent);

//完善家庭

void addChild(string parent, string child);

//添加子女

void removeChildren(string parent);

//移除一个父母结点的所有子女

Member\* find(const string& name);

//查找是否有该姓名的成员，返回指向该成员的指针或空指针

void changeName(string preName, string postName);

//为成员更名

bool getOrder();

//获取下一条要执行的命令

bool sameName(const string& name);

//判断是否有相同姓名的成员已在家谱内

void getNumber(int& temp);

//辅助函数，用于数字输入进行检查

栈类

类定义：

template <class T>

class stack

{

public:

stack(int size);

stack(const stack<T>& source);

~stack();

bool push(T elem);

void expand();

int size();

T pop();

bool isEmpty();

bool isFull();

T peek();

private:

T\* m\_buffer;

int m\_size;

int m\_element;

};

私有成员：

T\* m\_buffer;

//储存数据元素的地址

int m\_size;

//栈当前大小

int m\_element;

//栈当前元素数量

公有操作：

stack(int size);

//构造函数

bool push(T elem);

//栈顶增加元素

void expand();

//push中自动调用，扩大栈大小

int stacklength();

//返回栈的大小

T pop();

//出栈并返回出栈元素

bool isEmpty();

//判栈是否为空

T peek();

//返回栈顶元素

bool isFull();

//判栈是否满

2.4 系统设计

系统首先初始化家谱，随后持续执行getOrder()函数直到收到结束指令。

3 实现

3.1 建立家谱功能的实现

3.1.1 建立家谱流程

首先输入祖先名称，之后提示建立成功。

3.1.2 核心代码

FamilyList::FamilyList()

{

cout << "请输入祖先名称" << endl;

string name;

cin >> name;

root = new Member(name);

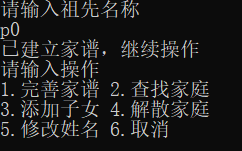
cout << "已建立家谱，继续操作" << endl;

cout << "请输入操作\n" << "1.完善家谱 2.查找家庭\n"

<< "3.添加子女 4.解散家庭\n" << "5.修改姓名 6.取消" << endl;

}

3.1.3 建立家谱功能演示



3.2 完善家庭功能实现

3.2.1 完善家庭流程

首先，输入姓名，找到姓名对应的成员。

若无该成员，则提示完善失败，若该成员已有子女，则提示已完善，应选择添加子女功能，若该成员无子女，则输入子女数量。

随后，向该成员添加指定数量的子女并获取姓名。

期间保持不重名，一旦出现重名则会要求重新输入。

3.2.2 核心代码

void FamilyList::insert(string parent)

{

Member\* p\_parent = find(parent);

if (!p\_parent)

{

cout << "未找到姓名为" << parent << "的成员" << endl;

return;

}

if (p\_parent->childnum > 0)

{

cout << "该成员的家庭已经完善，请使用添加子女功能" << endl;

return;

}

cout << "请输入子女人数：" << endl;

int childnum = 0;

getNumber(childnum);

if (childnum == 0)

{

cout << "完善家庭时子女数目必须合法" << endl;

return;

}

p\_parent->setChildnum(childnum);

cout << "请输入子女姓名：" << endl;

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

{

string child;

cin >> child;

if (sameName(child))

{

cout << "家谱内请不要重名，应添加标记区分重名者，请重新输入" << endl;

i--;

continue;

}

else

{

p\_parent->children[i] = new Member(child);

p\_parent->childnum++;

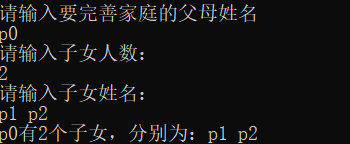
}

}

cout << \*p\_parent;

}

3.2.3 完善家庭功能演示



3.3 查找家庭功能的实现

3.3.1 查找家庭的流程

对家谱树进行一遍遍历，直到找到对应姓名的成员，返回指向它的指针，否则返回空指针。

3.3.2 核心代码

FamilyList::Member\* FamilyList::find(const string& name)

{

stack<Member\*> order(1000);

if (root->name == name)

{

return root;

}

order.push(root);

while (!order.isEmpty())

{

Member\* temp = order.pop();

for (int i = 0; i < temp->childnum; i++)

{

if (temp->children[i]->name == name)

{

return temp->children[i];

}

else

{

order.push(temp->children[i]);

}

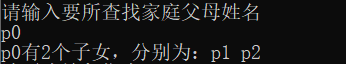
}

}

return nullptr;

}

3.3.3 查找家庭功能演示



3.4 添加子女功能实现

3.4.1 添加子女功能流程

首先，输入父母和子女的姓名。

然后，找到父母姓名对应的成员，若无该成员则输出错误信息并返回上一级，若有该成员，检查子女姓名是否重名，若不重名则为父母节点添加新子女，否则输出错误信息并返回上一级。

本操作可以为没有完善家庭的成员进行添加子女操作，一旦拥有子女便被标记为已被完善，用子女数目进行表示。

3.4.2 核心代码

void FamilyList::addChild(string parent, string child)

{

Member\* p\_parent = find(parent);

if (p\_parent)

{

if (sameName(child))

{

cout << "家谱内请不要重名，应添加标记区分重名者，添加失败" << endl;

return;

}

p\_parent->addChildren(child);

cout << \*p\_parent;

}

else

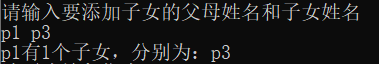
{

cout << "未找到父母，添加子女失败" << endl;

}

}

3.4.3 添加子女功能演示



3.5 解散家庭功能实现

3.5.1 解散家庭功能流程

首先，输入要解散家庭的成员姓名，找到对应成员。

若该成员存在，则调用removeChildren()函数直接移除其所有子女，这个操作保证会移除其子女的所有后代（由析构函数进行）。

若该成员不存在，则输出错误信息并返回上一级。

该操作会保留父母，仅解散其子女和其他后代，操作后会输出被解散的第一代子女。

3.5.2 核心代码

void FamilyList::removeChildren(string parent)

{

Member\* p\_parent = find(parent);

if (!p\_parent)

{

cout << "未找到姓名为" << parent << "的成员" << endl;

return;

}

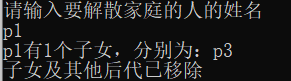
cout << \*p\_parent;

p\_parent->removeChildren();

cout << "子女及其他后代已移除" << endl;

}

3.5.3 解散家庭功能演示



3.6 修改姓名功能实现

3.6.1 修改姓名功能流程

首先，输入改名前、改名后的姓名，然后查找改名前的姓名。

若有对应成员，则检查改名后的姓名，若与原名相同，则保证不输出错误信息，此时提示姓名不变，若出现重名，则输出不允许重名并返回上一级，若无重名，则修改该成员姓名并提示成功。

3.6.2 核心代码

void FamilyList::changeName(string preName, string postName)

{

Member\* person = find(preName);

if (!person)

{

cout << "未找到姓名为" << preName << "的成员" << endl;

return;

}

if (preName == postName)

{

cout << "更名前后名字相同，为" << preName << endl;

return;

}

if (sameName(postName))

{

cout << "家谱内请不要重名，应添加标记区分重名者，更名失败" << endl;

return;

}

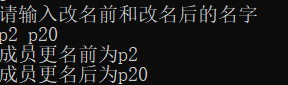
cout << "成员更名前为" << preName << endl;

cout << "成员更名后为" << postName << endl;

person->name = postName;

}

3.6.3 修改姓名功能演示



3.7 辅助功能的实现

辅助函数主要为输入数字时的检查函数与接收并执行指令的函数，实现如下。

void FamilyList::getNumber(int& temp)

{

string checkIn;

cin >> checkIn;

if (checkIn.size() > 10)

{

cout << "输入的数字不合法" << endl;

temp = 0;

return;

}

for (auto elem : checkIn)

{

if (!(elem >= '0'&&elem <= '9'))

{

cout << "输入的数字不合法" << endl;

temp = 0;

return;

}

}

if (atoi(checkIn.c\_str()) < INT\_MAX)

{

temp = stoi(checkIn);

}

else

{

cout << "输入的数字过大" << endl;

temp = 0;

}

}

bool FamilyList::getOrder()

{

int order = 0;

getNumber(order);

string parent, child, prename, postname;

Member\* p\_parent;

switch (order)

{

case 1:

cout << "请输入要完善家庭的父母姓名" << endl;

cin >> parent;

insert(parent);

break;

case 2:

cout << "请输入要所查找家庭父母姓名" << endl;

cin >> parent;

p\_parent = find(parent);

if (p\_parent)

{

cout << \*p\_parent;

}

else

{

cout << "查无此人" << endl;

}

break;

case 3:

cout << "请输入要添加子女的父母姓名和子女姓名" << endl;

cin >> parent >> child;

addChild(parent, child);

break;

case 4:

cout << "请输入要解散家庭的人的姓名" << endl;

cin >> parent;

removeChildren(parent);

break;

case 5:

cout << "请输入改名前和改名后的名字" << endl;

cin >> prename >> postname;

changeName(prename, postname);

break;

case 6:

cout << "访问家谱结束" << endl;

return false;

default:

cout << "请输入正确的指令" << endl;

break;

}

return true;

}

3.8 总体系统实现

总体系统首先建立家谱，随后不断接收指令直到退出家谱系统，核心代码如下：

int main()

{

FamilyList list;

while (list.getOrder())

{

cout << "请再次输入指令" << endl;

}

system("pause");

return 0;

}

4 测试

4.1 功能测试

4.1.1 完善家庭功能测试

**测试用例**：

1

P0

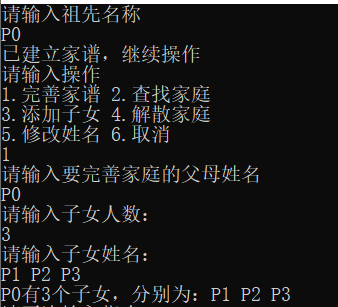
3

P1 P2 P3

**预期结果**：

程序运行正常，输出对应子孙:P1 P2 P3

**实验结果：**



**测试用例**：

1

P0

3

P1 P2 P3

1

P1

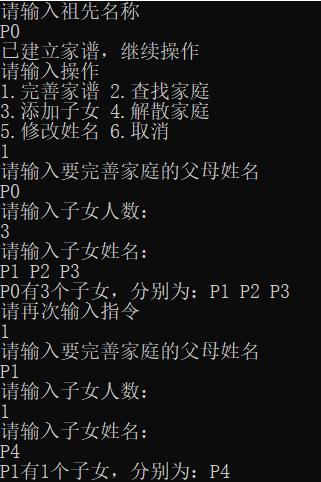
1

P4

**预期结果**：

程序运行正常，输出对应子孙:P4

**实验结果：**



4.1.2 添加子女功能测试

**测试用例：**

1

P0

3

P1 P2 P3

3

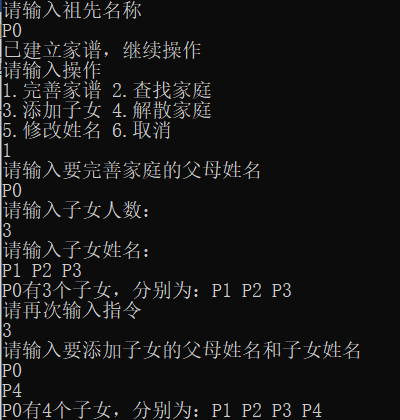
P0

P4

**预期结果：**

程序运行正常，输出对应子孙:P1 P2 P3 P4

**实验结果：**



4.1.3 解散家庭功能测试

**测试用例：**

1

P0

3

P1 P2 P3

1

P1

1

P4

4

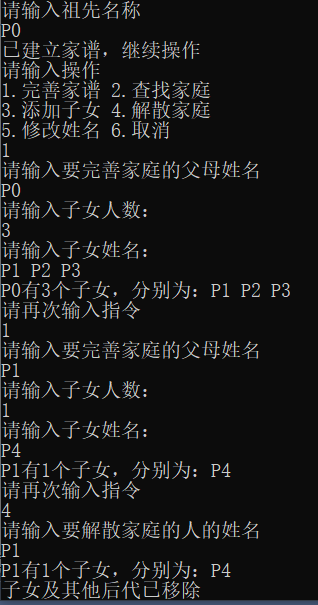
P1

**预期结果：**

程序运行正常，输出对应子孙:P4

P0子孙剩余P1 P2 P3，但P1不再拥有后代。

**实验结果：**



4.1.4 修改姓名功能测试

**测试用例：**

1

P0

3

P1 P2 P3

5

P1

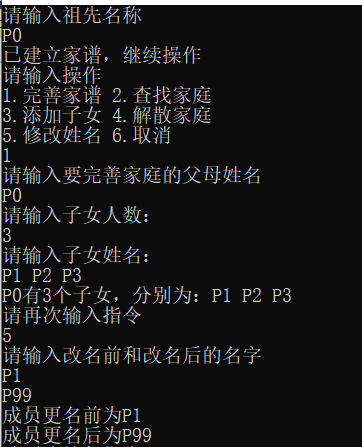
P99

**预期结果：**

程序运行正常，

P0子孙变为P99 P2 P3

**实验结果：**



4.1.5 查询家庭功能测试

**测试用例：**

1

P0

3

P1 P2 P3

2

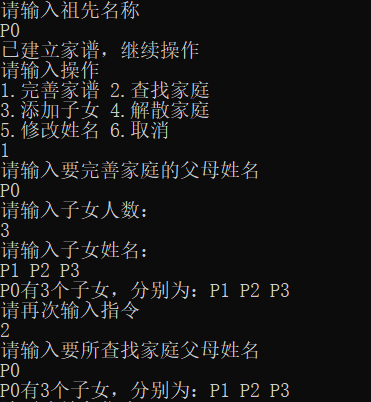
P0

**预期结果：**

程序运行正常，

P0子孙为P1 P2 P3

**实验结果：**



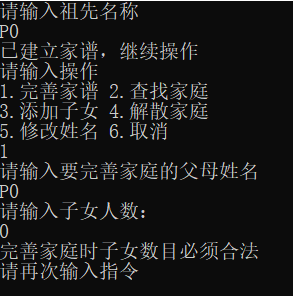
4.2 边界测试

4.2.1 完善家庭时输入成员数量错误

**测试用例：**建立家庭时成员数量错误

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

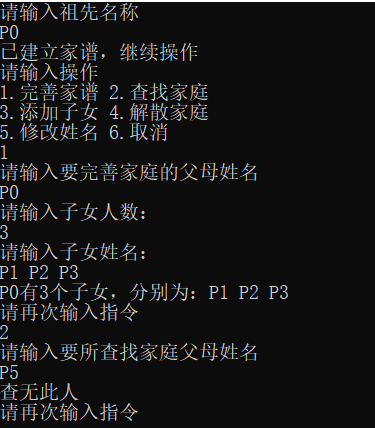


4.2.2 查询的人不存在

**测试用例：**查询的人不存在于家谱中

**预期结果：**程序给出相应提示，程序正常运行。

**实验结果：**

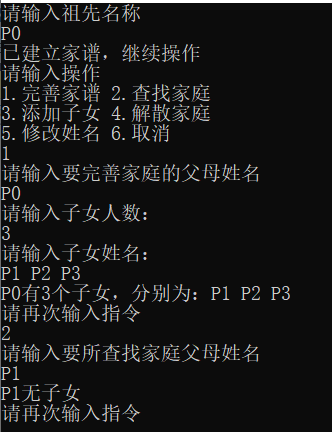


4.2.3 查询的人无子女

**测试用例：**查询的人没有任何子女

**预期结果：**程序给出正确信息，程序正常运行。

**实验结果：**



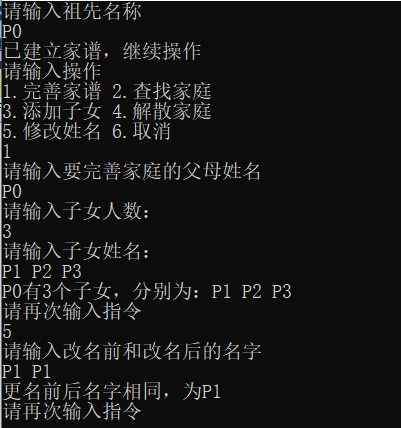
4.2.4 更名前后重名

**测试用例：**更改姓名前后姓名一样

**预期结果：**程序给出正确信息，程序正常运行。

**预期结果：**程序给出正确信息，程序正常运行。

**实验结果：**



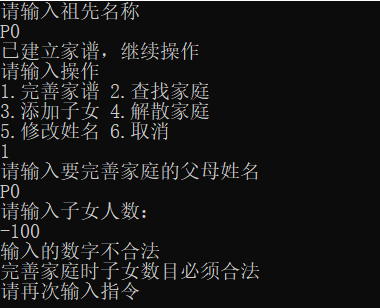
4.3 错误测试

4.3.1 子女人数错误

**测试用例：**完善家庭时输入子女数为负数

**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：**

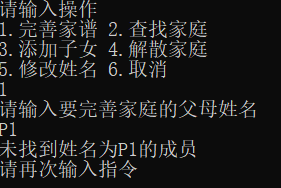


4.3.2 完善家庭时输入姓名不存在

**测试用例：**完善家庭时输入姓名不存在

**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：**

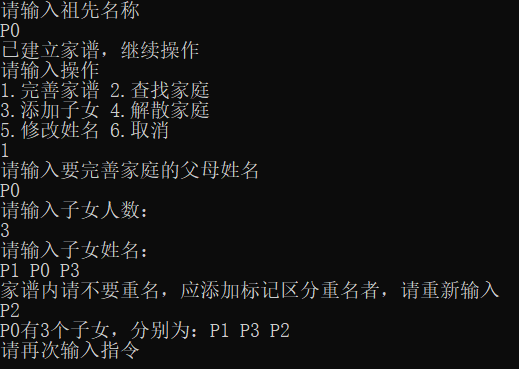


4.3.3 完善家庭时子女姓名于已有家庭成员重复

**测试用例：**完善家庭时子女姓名于已有家庭成员重复

**预期结果：**程序给出错误信息，并要求重新输入重复成员的姓名

**实验结果：**

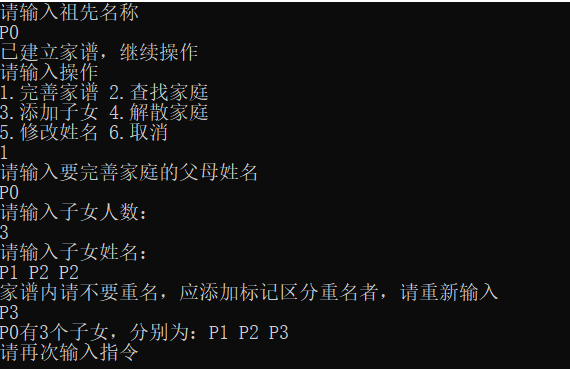


4.3.4 完善家庭时子女重名

**测试用例：**完善家庭时子女姓名有重复项

**预期结果：**程序给出错误信息，并要求重新输入重复成员的姓名。

**实验结果：**

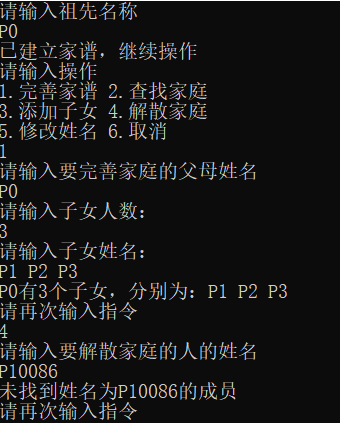


4.3.5 解散家庭的人不存在

**测试用例：**解散家庭的人不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

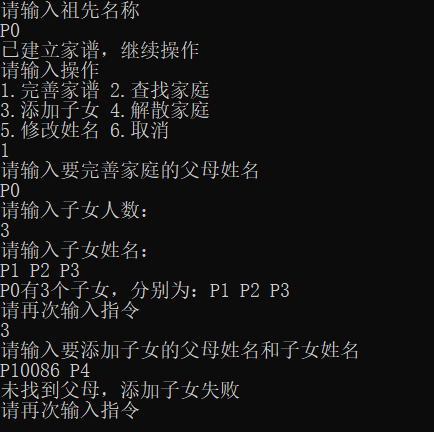


4.3.6 添加子女的人不存在

**测试用例：**添加子女的人不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

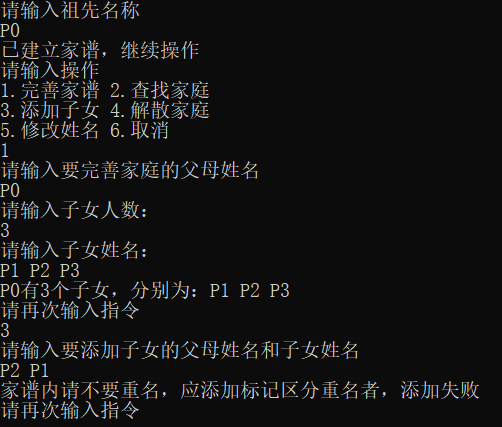


4.3.7 添加的子女姓名与原家谱内成员重复

**测试用例：**添加的子女姓名与原家谱内成员重复

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

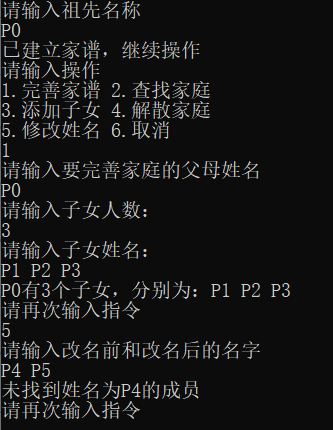


4.3.8 更改姓名的人不存在

**测试用例：**更改姓名的人不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**



4.3.9 更改姓名的新姓名与原家谱中成员重复

**测试用例：**更改姓名的新姓名与原家谱中成员重复

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

