项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 张喆

学 号： 1754060

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 1](#_Toc532740640)

[1.1 背景分析 1](#_Toc532740641)

[1.2 功能分析 1](#_Toc532740642)

[2 设计 2](#_Toc532740643)

[2.1 数据结构设计 2](#_Toc532740644)

[2.2 类结构设计 2](#_Toc532740645)

[2.3 成员与操作设计 2](#_Toc532740646)

[2.4 系统设计 4](#_Toc532740647)

[3 实现 5](#_Toc532740648)

[3.1 完善家谱功能的实现 5](#_Toc532740649)

[3.1.1 完善家谱功能流程图 5](#_Toc532740650)

[3.1.2 完善家谱功能核心代码 5](#_Toc532740651)

[3.1.1 完善家谱功能截屏示例 6](#_Toc532740652)

[3.2 添加家庭成员功能的实现 7](#_Toc532740653)

[3.2.1 添加家庭成员功能流程图 7](#_Toc532740654)

[3.2.2 添加家庭成员功能核心代码 7](#_Toc532740655)

[3.2.3 添加家庭成员功能截屏示例 7](#_Toc532740656)

[3.3 解散局部家庭功能的实现 8](#_Toc532740657)

[3.3.1 解散局部家庭功能流程图 8](#_Toc532740658)

[3.3.2 解散局部家庭功能核心代码 8](#_Toc532740659)

[3.3.3 解散局部家庭功能截图示例 9](#_Toc532740660)

[3.4 更改家庭成员姓名功能的实现 9](#_Toc532740661)

[3.4.1 更改家庭成员姓名功能流程图 9](#_Toc532740662)

[3.4.2 更改家庭成员姓名功能核心代码 10](#_Toc532740663)

[3.4.3 更改家庭成员姓名功能截屏示例 10](#_Toc532740664)

[3.5 输出家谱功能的实现 10](#_Toc532740665)

[3.5.1 输出家谱功能流程图 10](#_Toc532740666)

[3.5.2 输出家谱功能核心代码 11](#_Toc532740667)

[3.5.3 输出家谱功能截屏示例 12](#_Toc532740668)

[3.6 销毁家谱的实现 13](#_Toc532740669)

[3.6.1 销毁家谱流程图 13](#_Toc532740670)

[3.6.2 销毁家谱核心代码 13](#_Toc532740671)

[3.6.1 销毁家谱截屏示例 14](#_Toc532740672)

[3.7 总体系统的实现 14](#_Toc532740673)

[3.7.1 总体系统流程图 14](#_Toc532740674)

[3.7.2 总体系统核心代码 15](#_Toc532740675)

[3.7.3 总体系统截屏示例 16](#_Toc532740676)

[4 测试 18](#_Toc532740677)

[4.1 功能测试 18](#_Toc532740678)

[4.1.1 完善家谱功能测试 18](#_Toc532740679)

[4.1.2 添加家庭成员功能测试 18](#_Toc532740680)

[4.1.3 解散局部家庭功能测试 19](#_Toc532740681)

[4.1.4 更改家庭成员姓名功能测试 19](#_Toc532740682)

[4.1.5 输出家谱功能测试 19](#_Toc532740683)

[4.1.6 销毁家谱功能测试 20](#_Toc532740684)

[4.2 边界测试 21](#_Toc532740685)

[4.2.1 解散家庭时选择的是整个家谱的根节点 21](#_Toc532740686)

[4.3 出错测试 22](#_Toc532740687)

[4.3.1 建立家庭时儿女人数错误 22](#_Toc532740688)

[4.3.2 操作码错误 22](#_Toc532740689)

[4.3.3 完善家谱时无该祖先 22](#_Toc532740690)

[4.3.4 添加家庭成员时无该祖先 23](#_Toc532740691)

[4.3.5 解散局部家庭时无该家庭 23](#_Toc532740692)

[4.3.6 更改家庭成员姓名时无该成员 24](#_Toc532740693)

# 1 分析

## 1.1 背景分析

家谱管理系统是一个社会不可缺少的部分，它对于家族中的各个人、社会、历史学家等来说都至关重要，所以一个良好的家谱管理系统应该能够为用户提供充足的信息和功能。家谱系统对于家族和历史学家研究家族的发展有极其重要的作用。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，使用计算机对家谱管理进行管理，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族和社会的效率，也是社会走向信息化、科学化、国际化的重要条件。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

作为一个最简易的家谱管理系统，首先应该允许用户完善家谱，并能简单的输出整个家谱。

其次，考试报名系统还应该具有添加成员、解散局部家庭、更改家庭成员姓名功能，以保证家庭成员可以随时更改自己的家庭信息。最后，家谱管理系统软件还应该确保软件可以正常关闭，以及如果用户输入的数据有问题的时候可以做出简单的检查和提示。

综上所述，一个考试报名系统至少应该具有完善家谱、添加成员、解散局部家庭、更改家庭成员姓名、输出家谱、健壮性的检测等功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统具有明显的树形结构，因此考虑使用树作为数据结构。同时，由于每一个父亲的孩子数量是不确定的，因此不能简单的只设置leftChile和rightChild指针，本系统采用链表作为数据结构保存每个父节点的所有孩子的指针。

同时将每个人作为类封装起来，可以方便的修改每个人的属性，例如想增加每个人的年龄这一信息，可以直接在People类中增加一个int类型的变量表示人的年龄。

## 2.2 类结构设计

Pelople类用于储存成员信息

TreeNode类存储树中的结点

Genealogy类作为家谱类，相当于树这个数据结构

## 2.3 成员与操作设计

**成员信息类（People）**

**公有成员：**

string name; //姓名

**考生信息的构造函数：**

People() = default;

People(const string &buf);

People::People(const string & buf)

:name(buf) {}

**重载运算符：**

bool operator ==(const People &buf); //判断两个成员是否是同一个人

bool People::operator==(const People & buf)

{

return (this->name == buf.name);

}

**树中的结点类（TreeNode）**

**公有成员：**

struct People data; //成员信息

list<TreeNode \*> Child; //孩子链表

**友元：**

friend istream& operator>>(istream &is, TreeNode &buf); //重载运算符用于读入树中的结点

friend ostream& operator<<(ostream &os, const TreeNode &buf); //重载运算符用于输出树中的结点

istream & operator>>(istream & is, TreeNode & buf)

{

is >> buf.data.name;

return is;

}

ostream & operator<<(ostream & os, const TreeNode & buf)

{

os << buf.data.name;

return os;

}

**构造函数：**

TreeNode() = default;

TreeNode(const People &buf);

TreeNode::TreeNode(const People & buf)

: data(buf)

{

this->Child.clear();

}

**重载运算符：**

bool operator !=(const TreeNode &buf); //判断两个树中的结点是否不相等

bool TreeNode::operator!=(const TreeNode & buf)

{

return !(this->data == buf.data);

}

**家谱类（Genealogy）**

**私有成员：**

TreeNode \* \_root = NULL; //家谱的根结点

**公有操作：**

Genealogy() = default;

Genealogy(const string &buf);

~Genealogy();

bool isEmpty() { return \_root == NULL; } //判断家谱是否为空

void DestroyGenealogy(TreeNode \*root); //销毁树中所有结点

void showRoot(); //输出根结点

void showFirstGenChild(const TreeNode \*root); //输出第一代子孙

void showTree(); //分层输出树中所有结点

TreeNode \*findPeople(TreeNode \*root, const string &name); //寻找某个人是否在家谱中

TreeNode \*findParent(TreeNode \*root, const TreeNode \*buf); //寻找某个人的父结点

void BuildFamily(); //建立家庭

void addChild(); //添加家庭成员

void removeFamily(); //解散家庭

void changeName(); //更改家庭成员姓名

## 2.4 系统设计

系统首先对屏幕进行初始化，完成对家谱树的创建和输入数据工作，然后根据用户所输入的操作码（OpCode）执行家谱类GenTree对应的成员函数。

# 3 实现

## 3.1 完善家谱功能的实现

### 3.1.1 完善家谱功能流程图

### 3.1.2 完善家谱功能核心代码

void Genealogy::BuildFamily()

{

string name;

cout << "请输入要建立家庭的人的姓名: ";

cin >> name;

TreeNode \*move = findPeople(\_root, name);

if (move)

//如果存在该 待创建家庭的结点

{

int size;

cout << "请输入" << \*move << "的儿女人数: ";

cin >> size;

cout << "请依次输入" << \*move << "的儿女的姓名: ";

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

{

cin >> name;

move->Child.push\_back(new TreeNode(People(name))); //依次添加在孩子数组后面

assert(move->Child.back() != NULL);

}

showFirstGenChild(move); //输出创建家庭后 该结点的第一代子孙

}

else

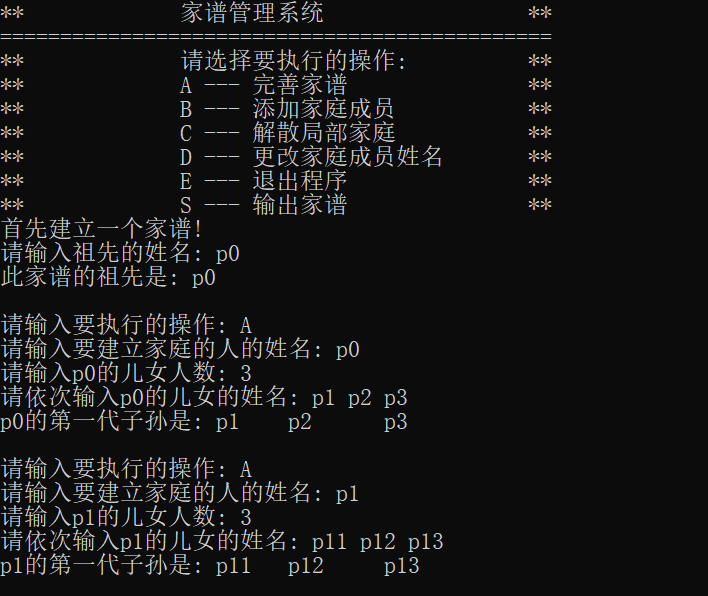
{

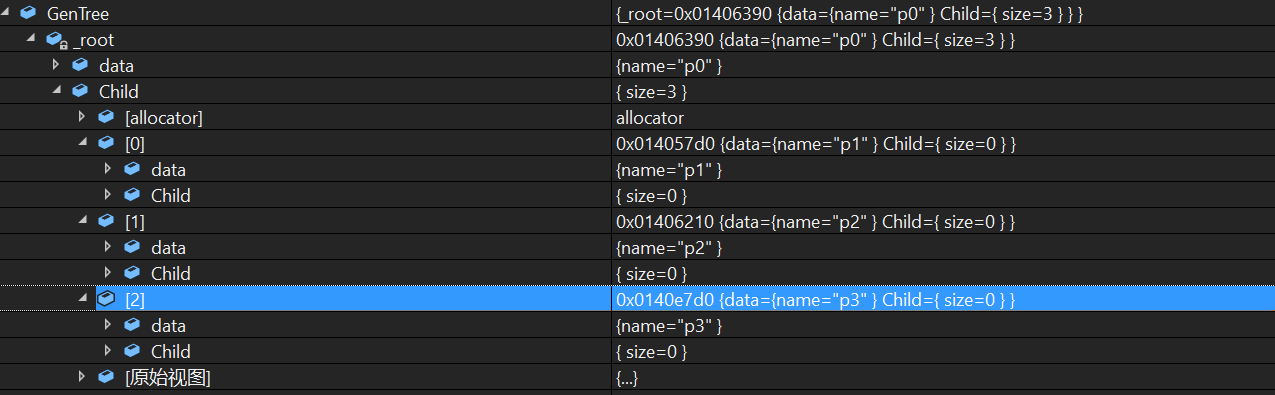
cout << "家谱中没有该家庭成员!" << endl;

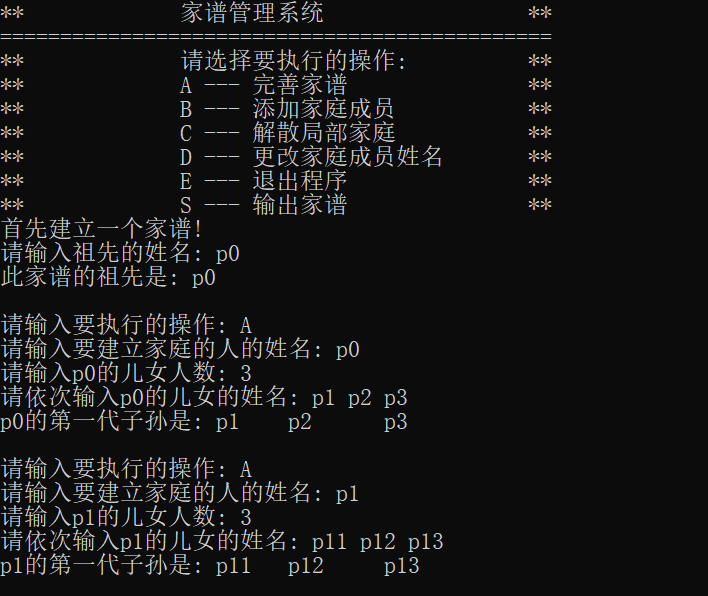
}

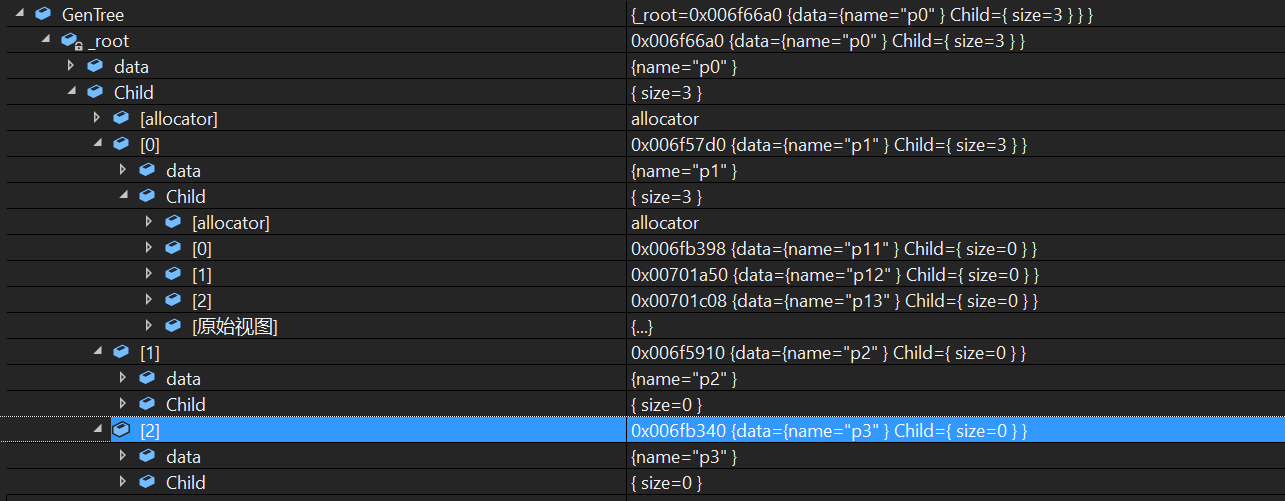
}

### 3.1.1 完善家谱功能截屏示例









## 3.2 添加家庭成员功能的实现

### 3.2.1 添加家庭成员功能流程图

### 3.2.2 添加家庭成员功能核心代码

void Genealogy::addChild()

{

string name;

cout << "请输入要添加儿子(或女儿)的人的姓名: ";

cin >> name;

TreeNode \*parent = findPeople(\_root, name);

if (parent)

//如果存在该 待添加儿子的结点

{

TreeNode \*fresh = new TreeNode; //为添加的孩子新开辟空间

assert(fresh != NULL);

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

cin >> \*fresh;

parent->Child.push\_back(fresh); //添加到当前结点的孩子数组后面

this->showFirstGenChild(parent); //输出添加儿子后 该结点的第一代子孙

}

else

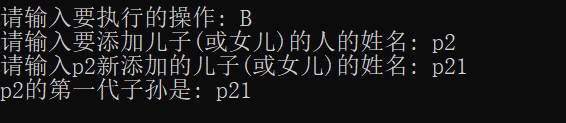
{

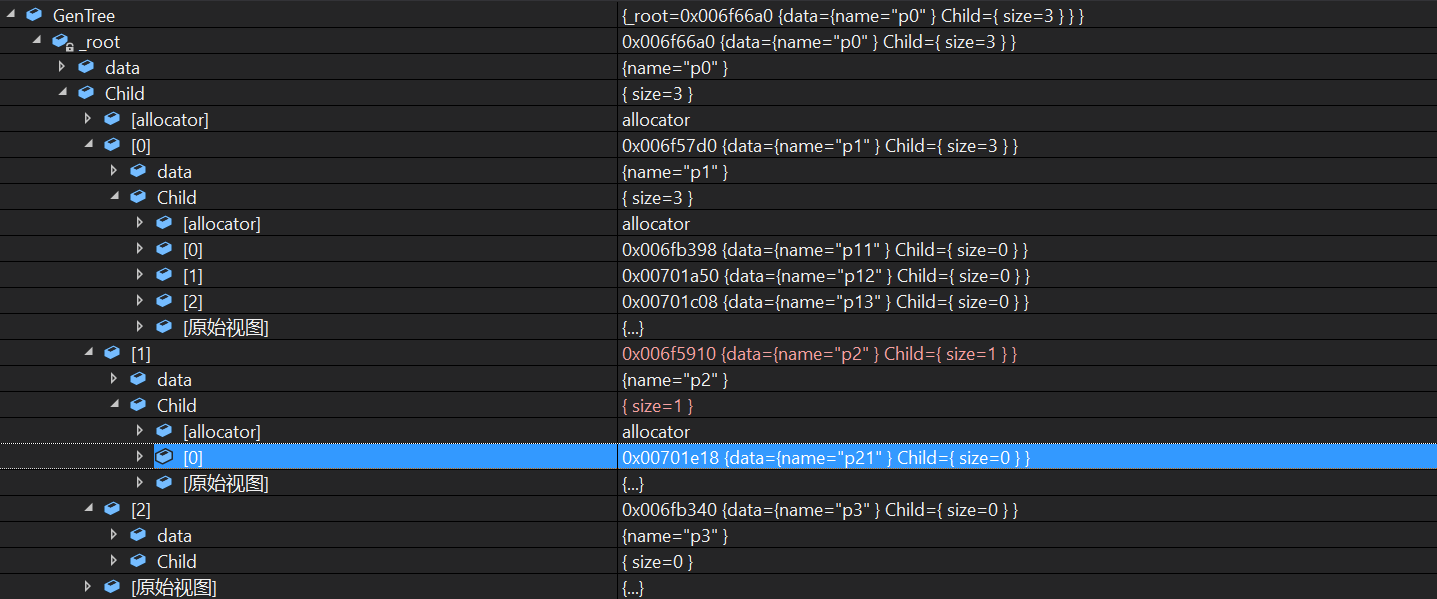
cout << "家谱中没有该家庭成员!" << endl;

}

}

### 3.2.3 添加家庭成员功能截屏示例





## 3.3 解散局部家庭功能的实现

### 3.3.1 解散局部家庭功能流程图

### 3.3.2 解散局部家庭功能核心代码

void Genealogy::removeFamily()

{

string name;

cout << "请输入要解散家庭的人的姓名: ";

cin >> name;

TreeNode \*move = findPeople(\_root, name);

if (move == \_root)

//如果解散的根结点特殊处理

{

cout << "要解散家庭的人是: " << \*move << endl;

this->showFirstGenChild(move); //输出下一代子孙

DestroyGenealogy(\_root);

delete \_root;

\_root = NULL;

}

else if (move)

//如果存在该 待解散家庭的结点

{

cout << "要解散家庭的人是: " << \*move << endl;

this->showFirstGenChild(move); //输出下一代子孙

this->DestroyGenealogy(move); //递归destroy函数, 递归的删除该结点作为根结点的子树

//先在parent的孩子数组中把他一出

TreeNode \*parent = findParent(\_root, move);

parent->Child.erase(find(parent->Child.begin(), parent->Child.end(), move));

//再释放该结点

delete move;

move = NULL;

}

else

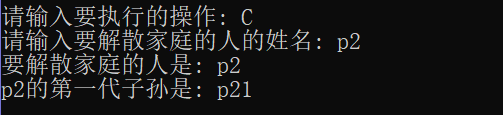
{

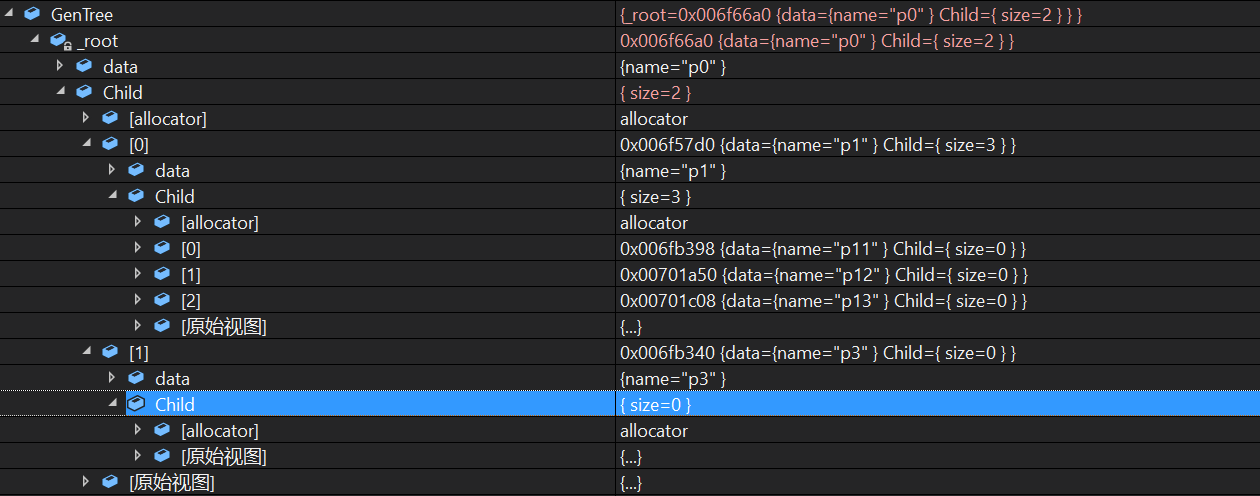
cout << "家谱中没有该家庭成员!" << endl;

}

}

### 3.3.3 解散局部家庭功能截图示例





## 3.4 更改家庭成员姓名功能的实现

### 3.4.1 更改家庭成员姓名功能流程图

### 3.4.2 更改家庭成员姓名功能核心代码

void Genealogy::changeName()

{

string name;

cout << "请输入要更改姓名的人的目前姓名: ";

cin >> name;

TreeNode \*move = findPeople(\_root, name);

if (move)

//如果存在该 待更改姓名的结点

{

cout << "请输入更改后的姓名: ";

cin >> \*move;

cout << name << "已更名为" << \*move << endl;

}

else

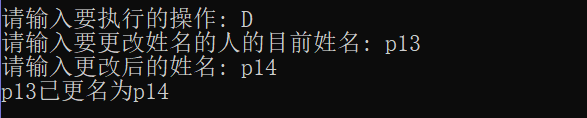
{

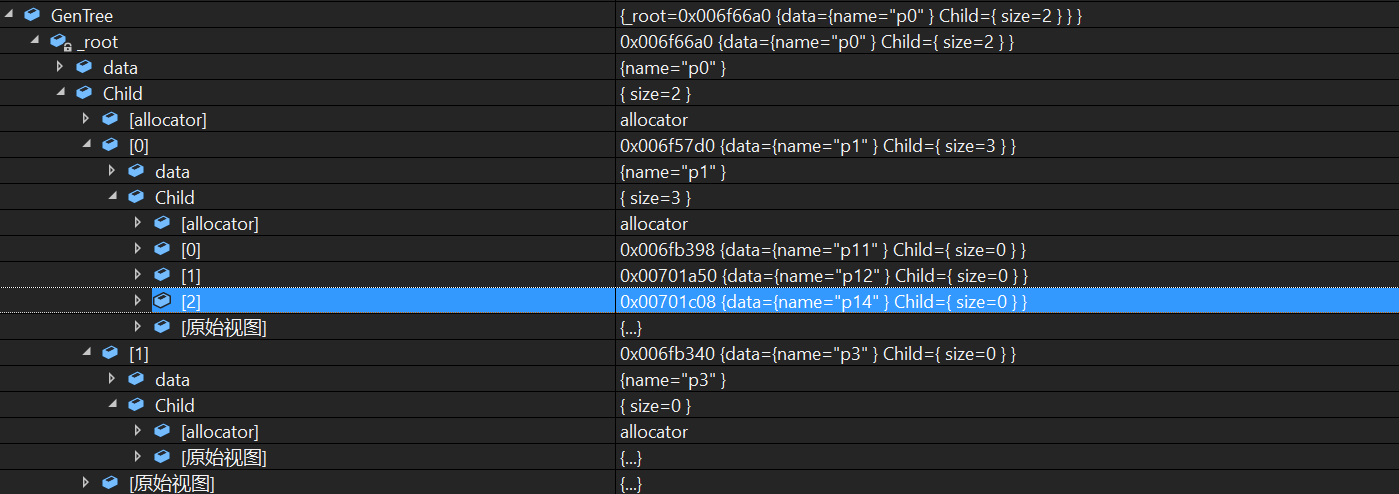
cout << "家谱中没有该家庭成员!" << endl;

}

}

### 3.4.3 更改家庭成员姓名功能截屏示例





## 3.5 输出家谱功能的实现

### 3.5.1 输出家谱功能流程图

### 3.5.2 输出家谱功能核心代码

输出某节点的下一代

void Genealogy::showFirstGenChild(const TreeNode \*root)

{

cout << root->data.name << "的第一代子孙是: ";

for (const auto& buf : root->Child)

//循环当前结点的孩子数组

{

cout << \*buf << " ";

}

cout << endl;

}

输出家谱的整棵树

void Genealogy::showTree()

{

if (isEmpty())

//家谱为空时特殊处理

{

cout << "家谱为空!" << endl;

return;

}

queue<TreeNode \*> Q; //层次化访问工作队列

Q.push(this->\_root); //压入根结点

TreeNode buf(People(ERRORNAME)); //定义分隔结点

TreeNode move;

while (!Q.empty())

{

Q.push(&buf); //压入用于分隔的分隔结点

move = \*(Q.front()); //取出队列头结点

while (move != buf) //当头上结点不是分隔结点时循环输出

{

Q.pop();

for (const auto& elem : move.Child)

//将当前结点的所有孩子压入队列

{

Q.push(elem);

}

if (move != buf)

{

cout << move << " ";

}

move = \*(Q.front());

}

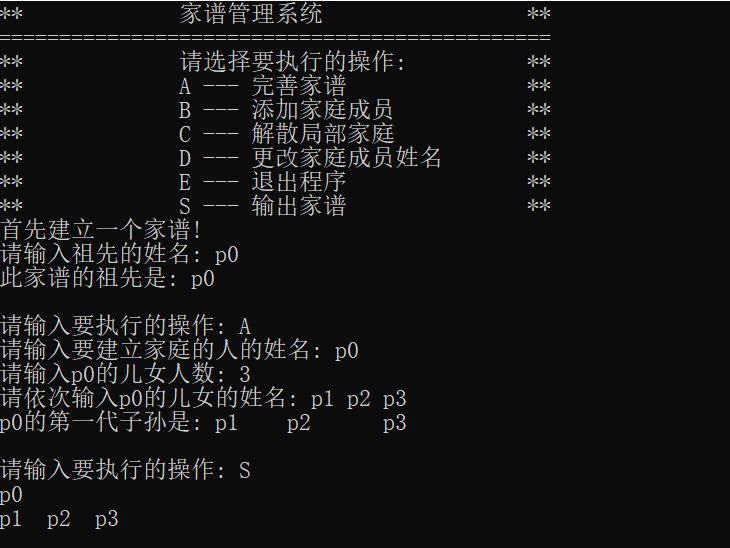
Q.pop(); //将当前层次的分隔结点取出

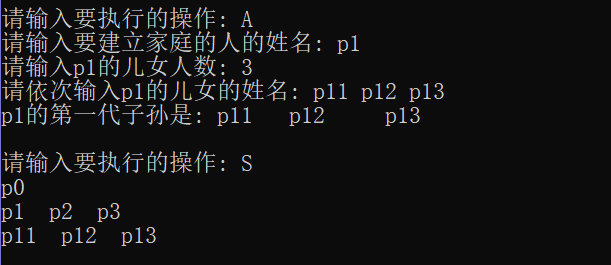
cout << endl;

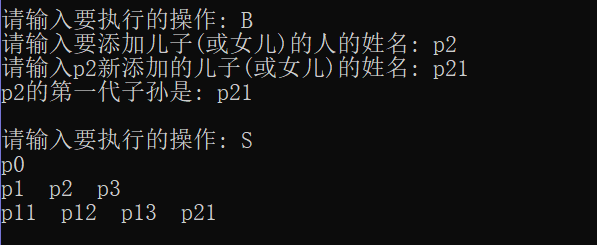
}

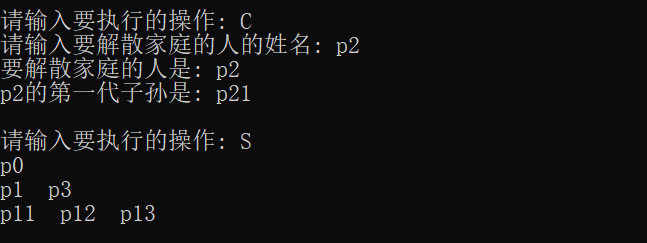
}

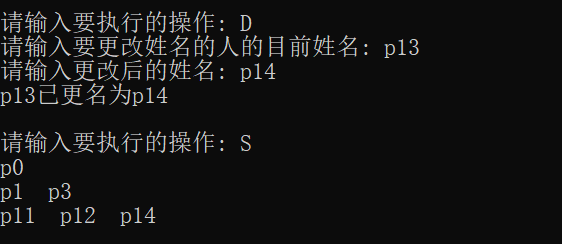
### 3.5.3 输出家谱功能截屏示例











## 3.6 销毁家谱的实现

### 3.6.1 销毁家谱流程图

### 3.6.2 销毁家谱核心代码

Genealogy::~Genealogy()

{

DestroyGenealogy(\_root); //调用destroy函数, 一次释放树中所有结点

delete \_root; //释放根结点

\_root = NULL;

}

void Genealogy::DestroyGenealogy(TreeNode \*root)

{

if (root->Child.empty())

//如果该结点为叶子结点 -> 孩子数组为空

{

delete root; //释放该结点

root = NULL;

}

else

{

for (auto &bufNode : root->Child)

{

DestroyGenealogy(bufNode);

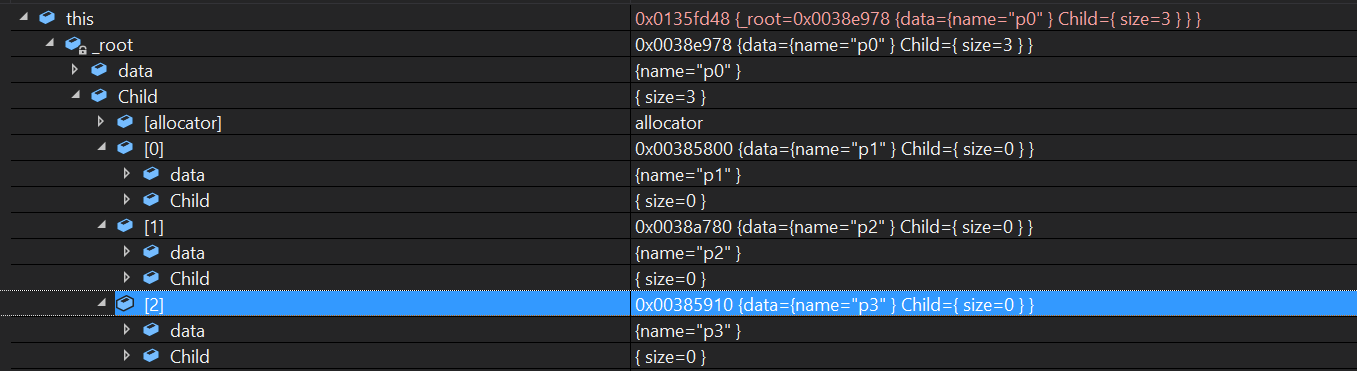
//递归调用destroy函数, 对孩子数组中的每一个元素执行销毁操作

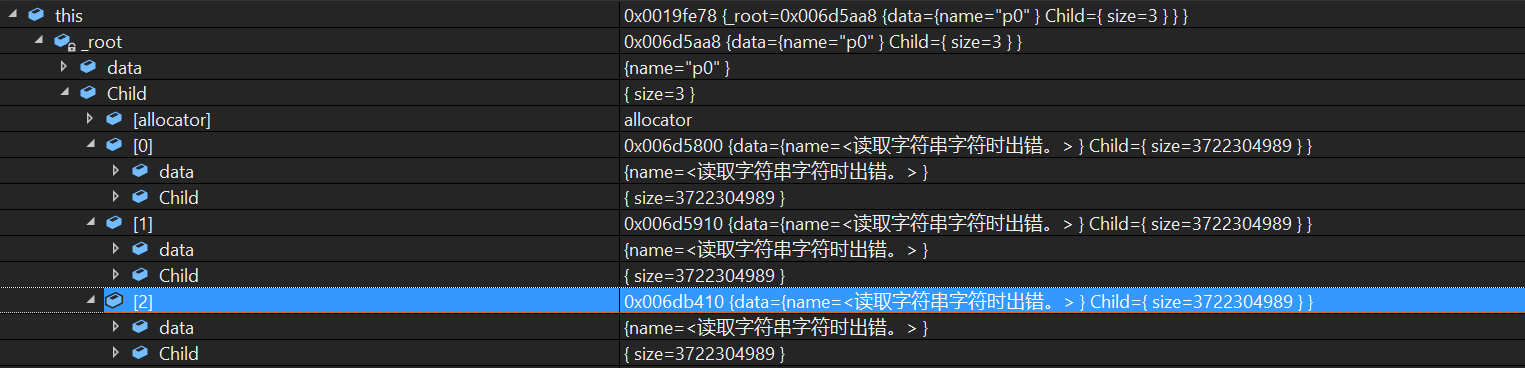
}

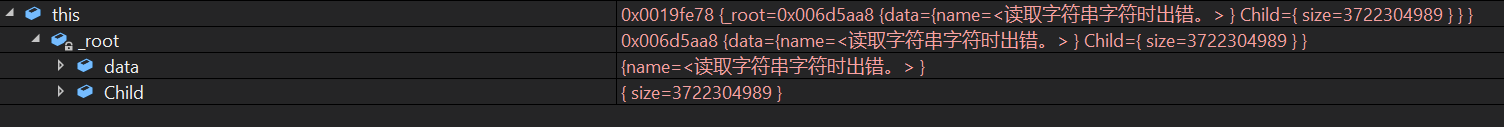
}

}

### 3.6.1 销毁家谱截屏示例







## 3.7 总体系统的实现

### 3.7.1 总体系统流程图

### 3.7.2 总体系统核心代码

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 --- 退出程序" << " \*\*" << endl;

cout << "\*\* " << "S --- 输出家谱" << " \*\*" << endl;

/\*建立家谱\*/

cout << "首先建立一个家谱!" << endl;

string buf;

cout << "请输入祖先的姓名: ";

cin >> buf;

Genealogy GenTree(buf);

GenTree.showRoot();

/\*用户操作\*/

char OpCode;

while (1)

{

cout << endl << "请输入要执行的操作: ";

cin >> OpCode;

if (OpCode == 'E') { break; }

switch (OpCode)

{

case 'A':

{

GenTree.BuildFamily();

break;

}

case 'B':

{

GenTree.addChild();

break;

}

case 'C':

{

GenTree.removeFamily();

break;

}

case 'D':

{

GenTree.changeName();

break;

}

case 'S':

{

GenTree.showTree();

break;

}

default:

{

cout << "请输入大写字母A~E!" << endl;

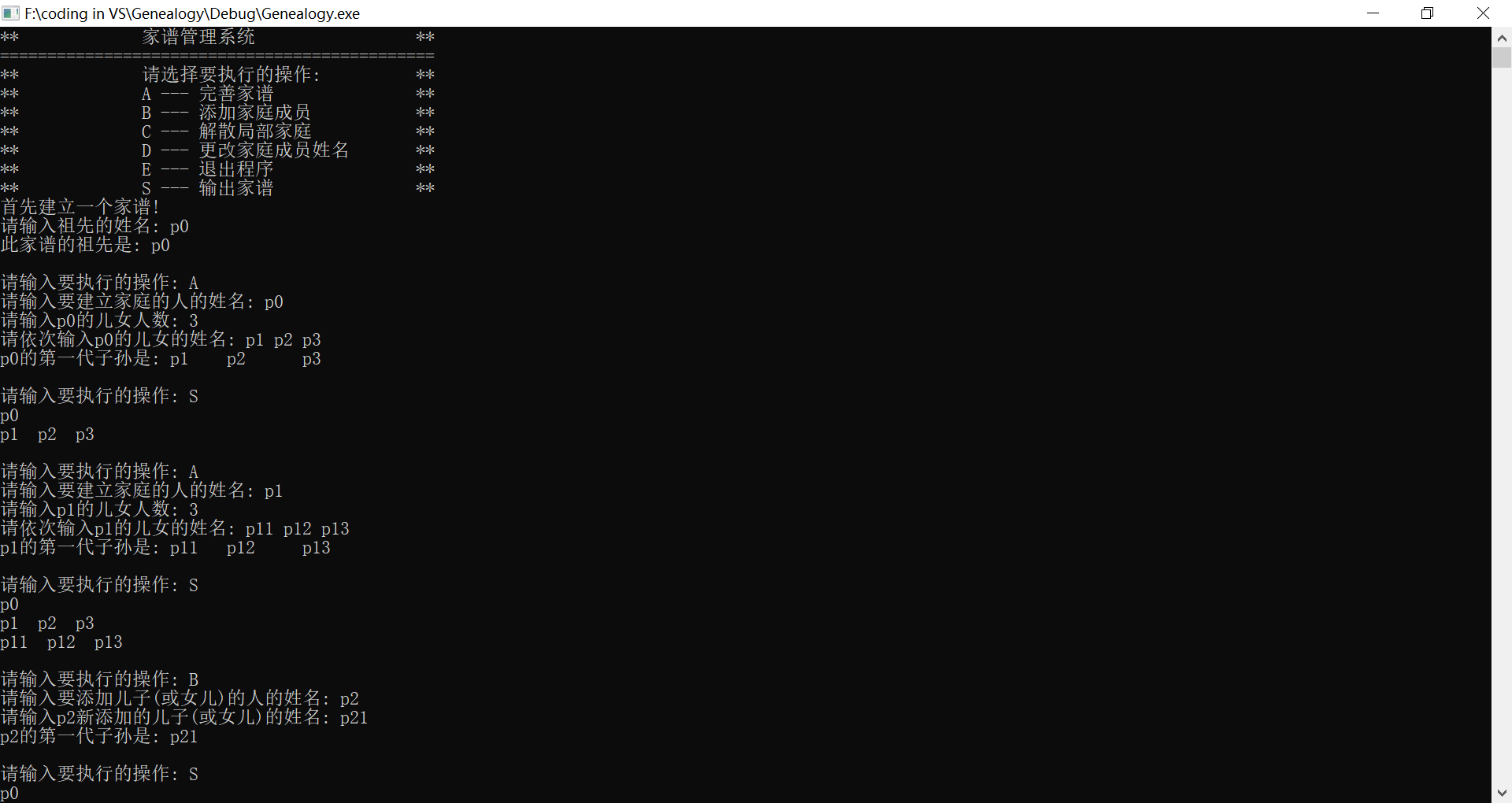
break;

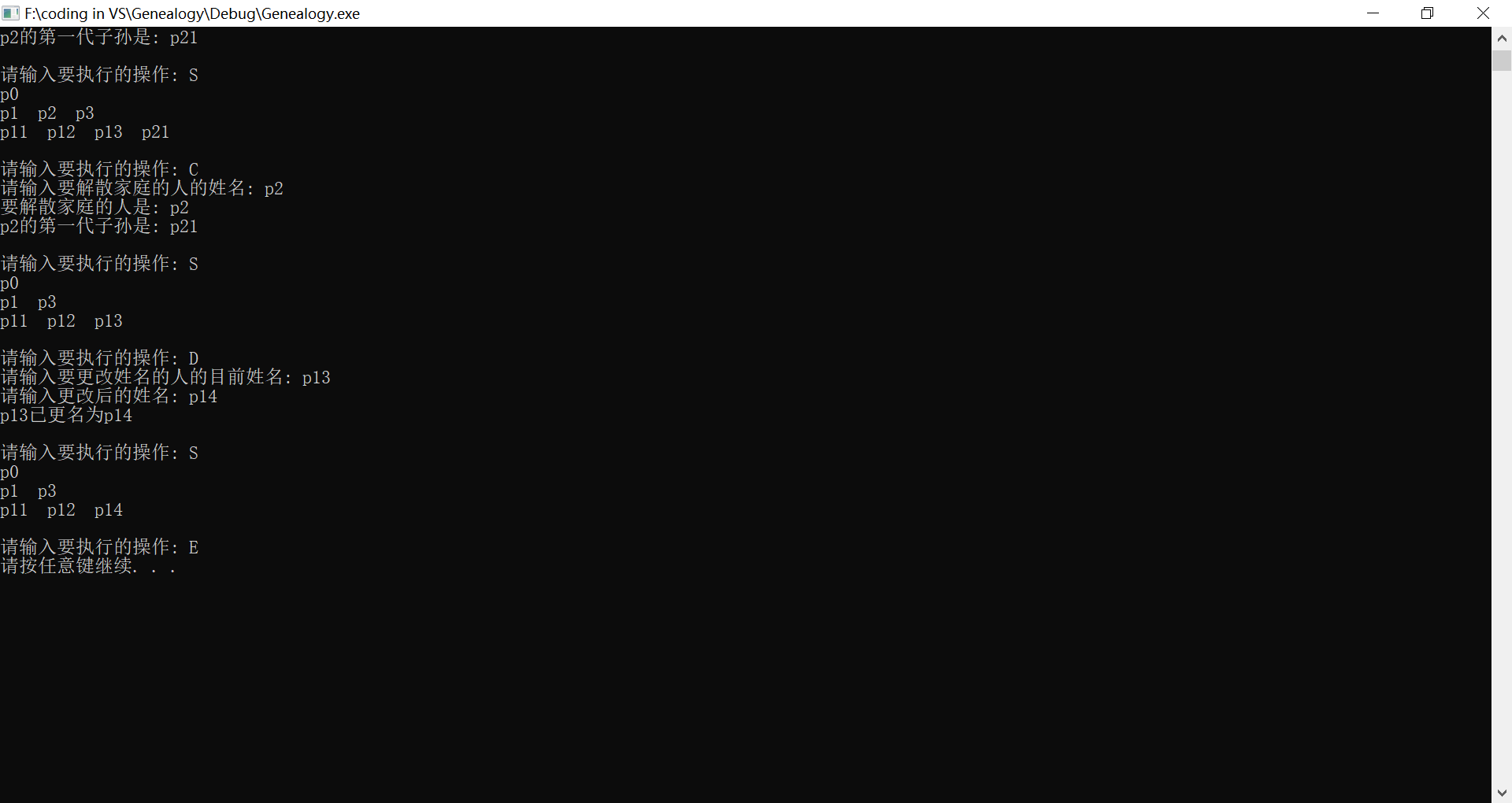
}

}

}

### 3.7.3 总体系统截屏示例





# 4 测试

## 4.1 功能测试

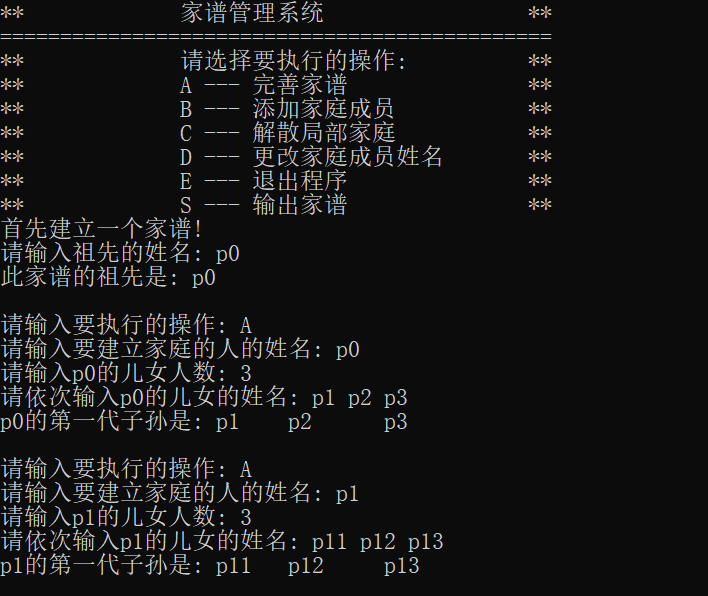
### 4.1.1 完善家谱功能测试

**测试用例**：首先输入祖先姓名p0

对p0完善家庭，建立三个子女p1 p2 p3

**预期结果**：p0的第一代子孙是 p1 p2 p3

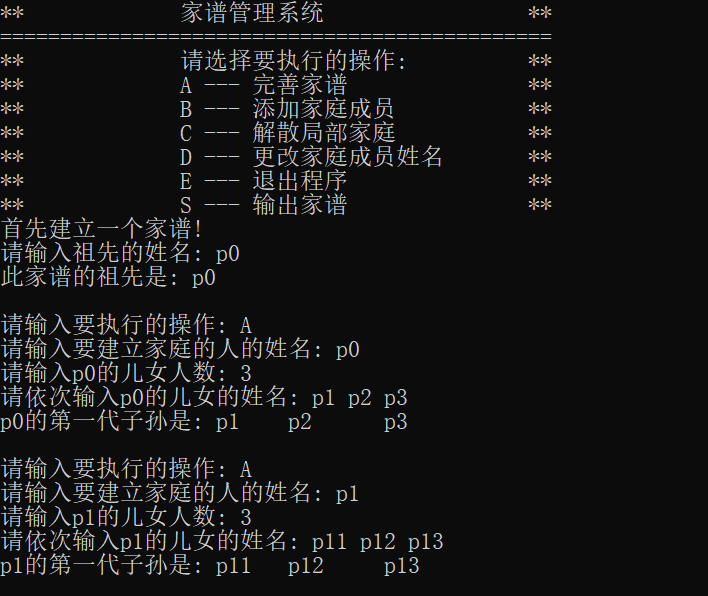
**实验结果：**



**测试用例**：对p1完善家庭，建立三个子女p11 p12 p13

**预期结果**：p1的第一代子孙是p11 p12 p13

**实验结果:**



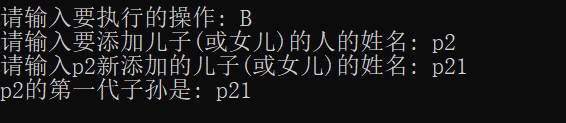
### 4.1.2 添加家庭成员功能测试

**测试用例：**对成员p2添加子女p21

**预期结果：**

p2的第一代子孙是p21

**实验结果：**



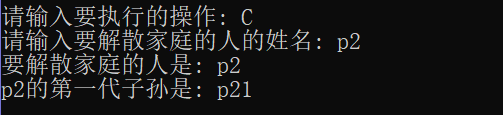
### 4.1.3 解散局部家庭功能测试

**测试用例：**解散p2的家庭

**预期结果：**

p2的第一代子孙是p21

**实验结果：**



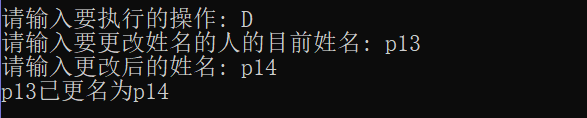
### 4.1.4 更改家庭成员姓名功能测试

**测试用例：**将p13的姓名改为p14

**预期结果：**

p13已更名为p14

**实验结果：**

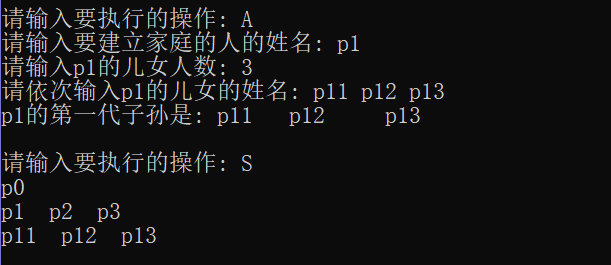


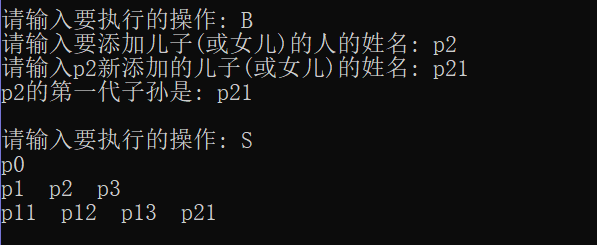
### 4.1.5 输出家谱功能测试

**测试用例：**前面的用例综合

**预期结果：**正确输出家谱树

**实验结果：**



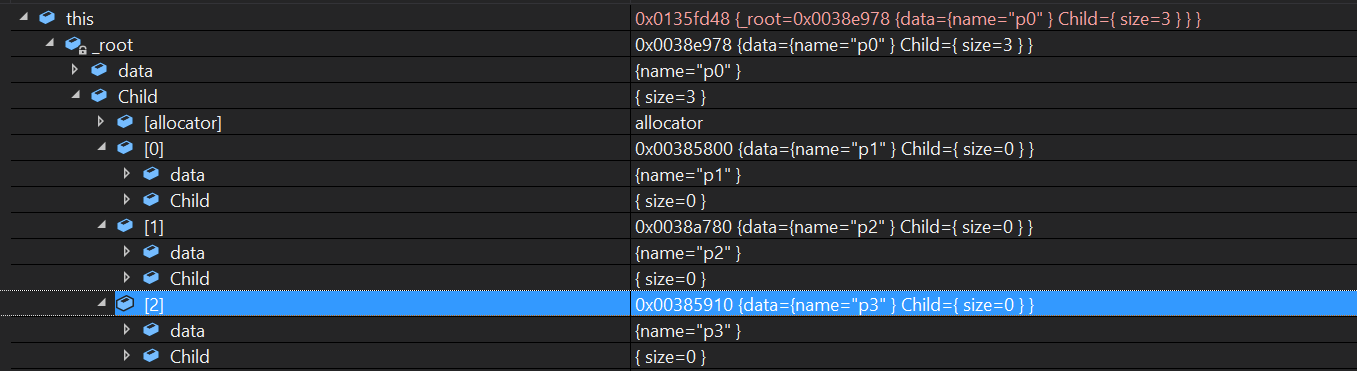


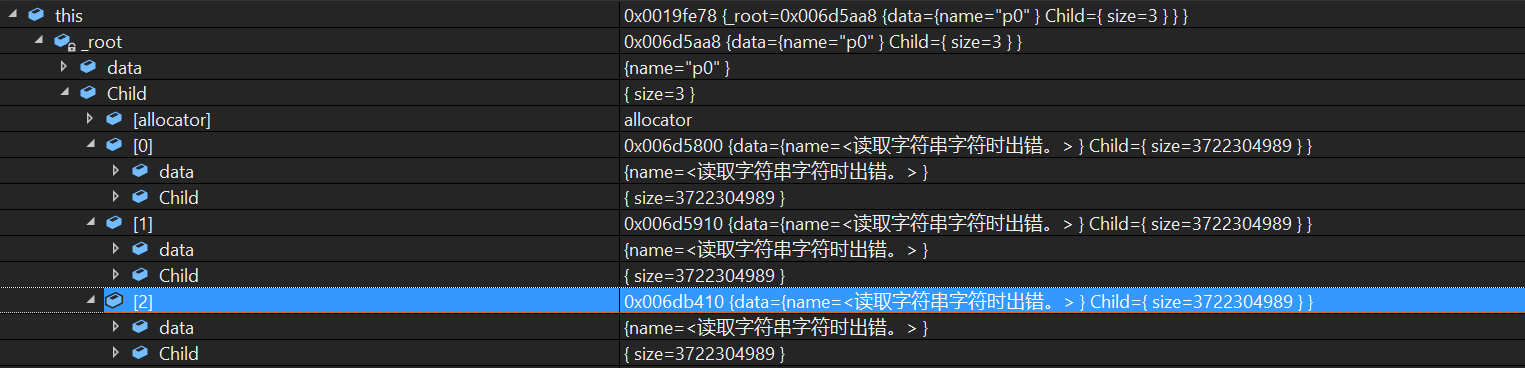
### 4.1.6 销毁家谱功能测试

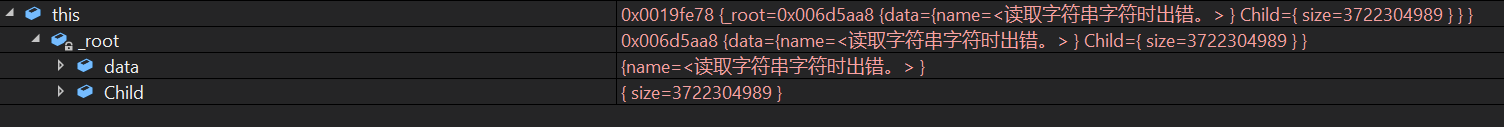
**测试用例：无**

**预期结果：**正确销毁家谱树并释放结点的空间

**实验结果：**







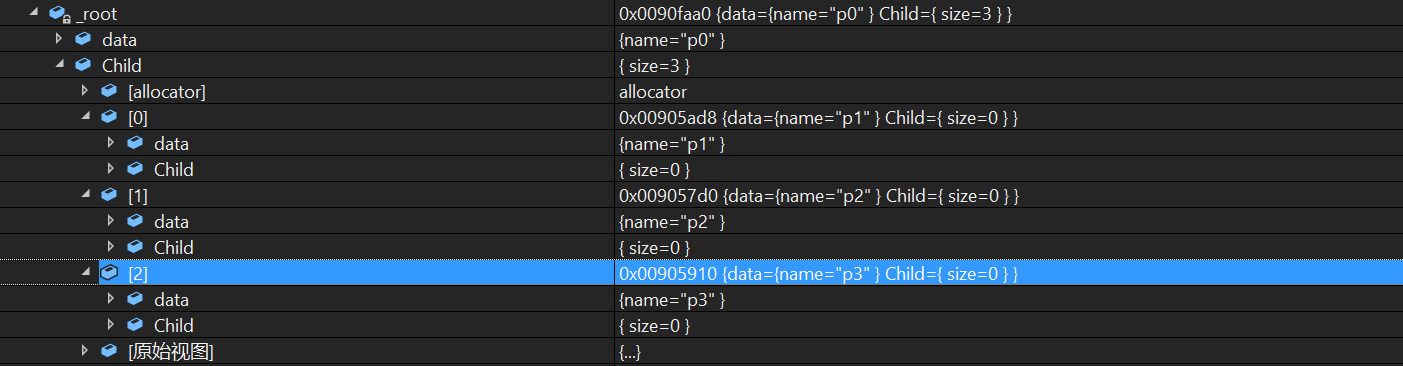
## 4.2 边界测试

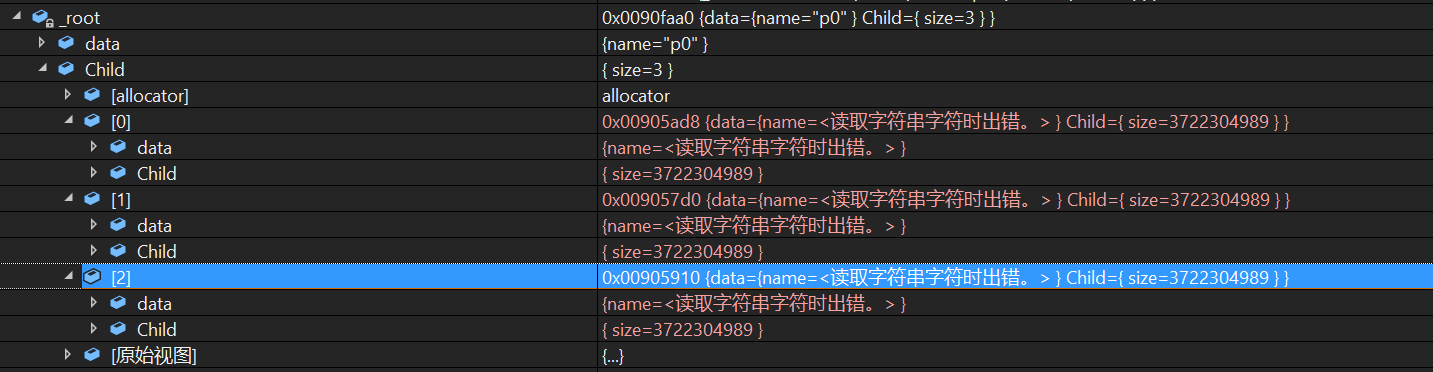
### 4.2.1 解散家庭时选择的是整个家谱的根节点

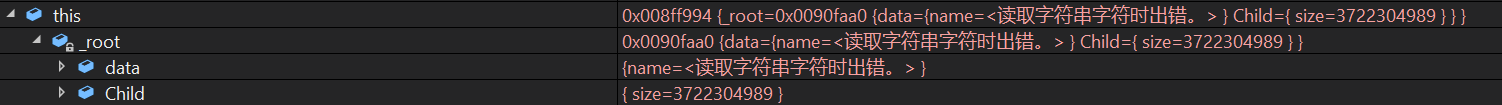
**测试用例：**先以p0作为家谱的根节点创建合适的家谱，再调用解散局部家庭操作解散p0的家庭

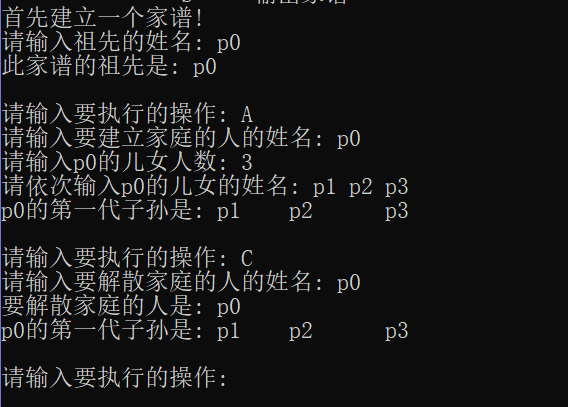
**预期结果：**程序能正常释放整棵家谱树的所有结点空间

**实验结果：**









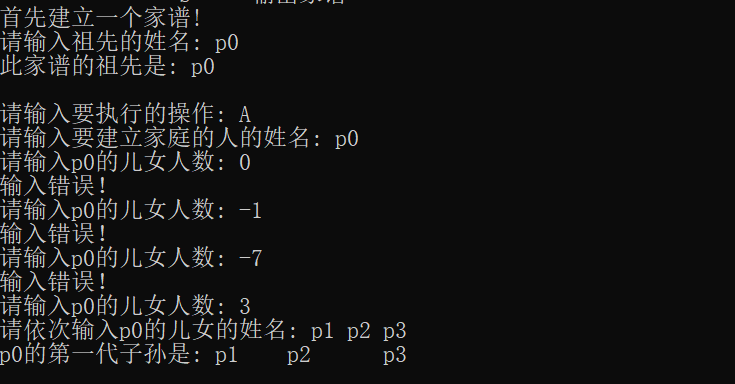
## 4.3 出错测试

### 4.3.1 建立家庭时儿女人数错误

**测试用例：**输入儿女人数为负数

**预期结果：**程序给出提示信息，程序正常运行不崩溃，并让用户继续输入。

**实验结果：**

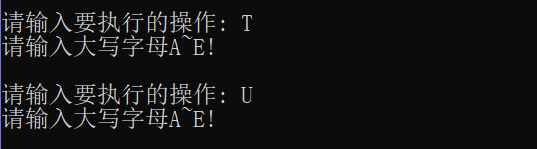


### 4.3.2 操作码错误

**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，并允许用户重新输入。

**实验结果：**

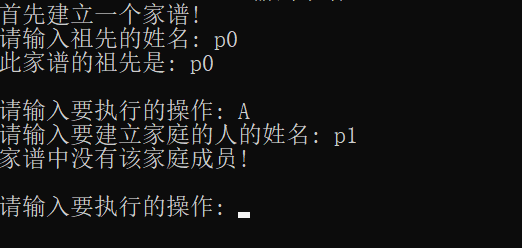


### 4.3.3 完善家谱时无该祖先

**测试用例：**给定家谱的祖先为p0，但是建立p1的家庭成员

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**

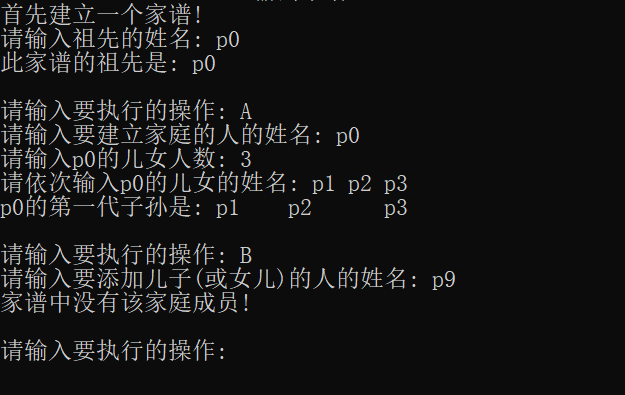


### 4.3.4 添加家庭成员时无该祖先

**测试用例：**先建立好p0的家庭，然后试图对不存在的子孙p9添加家庭成员

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**

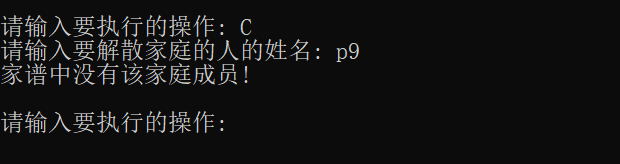


### 4.3.5 解散局部家庭时无该家庭

**测试用例：**在前方程序的基础上试图解散不存在的子孙p9的家庭

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**



### 4.3.6 更改家庭成员姓名时无该成员

**测试用例：**在前方程序的基础上试图修改不存在的子孙p9的姓名

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**

