项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 邓泉

学 号： 1953871

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](#_Toc10411)

[1.1 题目背景](#_Toc29649)

[1.2背景分析](#_Toc8129)

[1.3 功能分析](#_Toc18301)

[2 设计](#_Toc4755)

[2.1 数据结构设计](#_Toc23457)

[2.2 类结构设计](#_Toc28151)

[2.3 成员与操作设计](#_Toc26920)

[2.4 系统设计](#_Toc12523)

[3.功能实现](#_Toc32206)

[3.1总流程实现](#_Toc31247)

[3.2 建立家庭功能](#_Toc25018)

[3.3 添加儿女功能](#_Toc13909)

[3.4 解散家庭功能](#_Toc3554)

[3.5 修改信息7功能](#_Toc24527)

3.6 查找功能................................................................................................................

[4 测试](#_Toc5690)

[4.1 功能测试](#_Toc3220)

[4.1.1 正常测试](#_Toc16935)

[4.2 边界测试](#_Toc26427)

[4.2.1 新建子女数为0](#_Toc14209)

[4.3 错误测试](#_Toc5467)

### 4.3.1 新建子女数为负数.....................................................................................

[4.3.2 搜索不存在的结点](#_Toc10298)

[4.3.3 未知指令](#_Toc24837)

# 1分析

## 1.1 题目背景

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。

## 1.2背景分析

传统的家谱以纸质为主，具有不易备份、难以修改和容易丢失、损坏的特点。在计算机技术高度发达的今天，一套完整的家谱管理计算机程序能够让家谱更加安全、丰富且易于修改。该程序的设计具有深刻的现实意义和广泛的实用价值。

## 1.3 功能分析

本程序要求对家谱管理进行简单的管理，在对家族成员信息进行记录的基础上，需要实现家族成员信息的查找功能、添加功能和删除功能。实质是完成兑家谱成员信息的建立，查找，插入，修改，删除等功能，可以首先定义家族成员数据结构，然后将每个功能作为一个成员函数来完成对数据的操作，最后完成主函数以验证各个函数功能并得到运行结果。

# 

# 2 设计

## 2.1 数据结构设计

在现实中，每个人只有一个父亲和一个母亲，同时一个人可以有多个子女，这种数字关系与树存储结构不谋而合。因此，树数据结构在本题中是一个满足要求的选择。

树是一种常用的[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450" \t "_blank)，它是由n（n>=1）个有限结点组成一个具有层次关系的[集合](https://baike.baidu.com/item/%E9%9B%86%E5%90%88" \t "_blank)。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：

每个结点有零个或多个子结点；没有父结点的结点称为根结点；每一个非根结点有且只有一个父结点；除了根结点外，每个子结点可以分为多个不相交的子树。

树有许多种存储方式。在本题中，受现实生活中家庭成员关系的启发，我选择子女-兄弟链表表示法来存储树数据结构。子女-兄弟链表表示法又称长子-兄弟表示法，是最节省存储空间的树的存储方式。它的每个结点有两个链域，第一个为兄弟链域，指向当前结点的下一个兄弟结点；第二个为长子结点，在不带附加头结点的存储方式中，指向长子结点，而在我的代码实现中，由于我使用了带附加头结点的链表表示，每个结点的长子域指向附加子女链的附加头结点。

值得注意的是，每个附加头结点随双亲结点的产生而产生，并随双亲结点的释放而释放，但附加头结点自身的长子域为NULL。

## 2.2 类结构设计

本题使用的树由二叉结点类和树类两个模板类组成。使用模板类可以扩大代码的适用范围，有利于代码的重用。

## 2.2.1 TreeNode类设计

结点TreeNode含有三个成员，分别为结点所存储的数据value、指向子结点头 结点的指针next和指向兄弟结点的指针link。

## 2.2.2 Tree类设计

Tree是模板类，它存储了一个私有成员变量，即代表家谱祖先的root结点指针， 公有成员函数除了构造函数和析构函数外，主要还有六个，分别是：

void buildRoot(T& Root) 传入的是祖先名字，创建家谱中第一个元素；

bool Insert(T& fatherName, T& childName) 给家谱中的（姓名为fatherName的）某个人 添加（姓名为childName的）一个子女；

void Destroy(T& fatherName) 删除家谱中的某个人，连带子女一起从家谱中删除；

void destroy(TreeNode<T>\* subTree) 删除家谱中的某个人，连带子女和兄弟一起从 家谱中删除；

bool Update(T& curName, T& newName) 更改家谱中某个人的姓名为newName；

void Show(T& fatherName) 展示家谱中某个人的子女（第一代子孙）；

TreeNode<T>\* Find(T& name) 找出在家谱中这个人是否存在，若存在则返回该位置；

此外还有一个private函数（重载函数）：

TreeNode<T>\* Find(TreeNode<T>\* subTree, T& name) 以subTree为起点寻找姓名为name的结点，若找到则返回此结点的位置供其他函数使用。

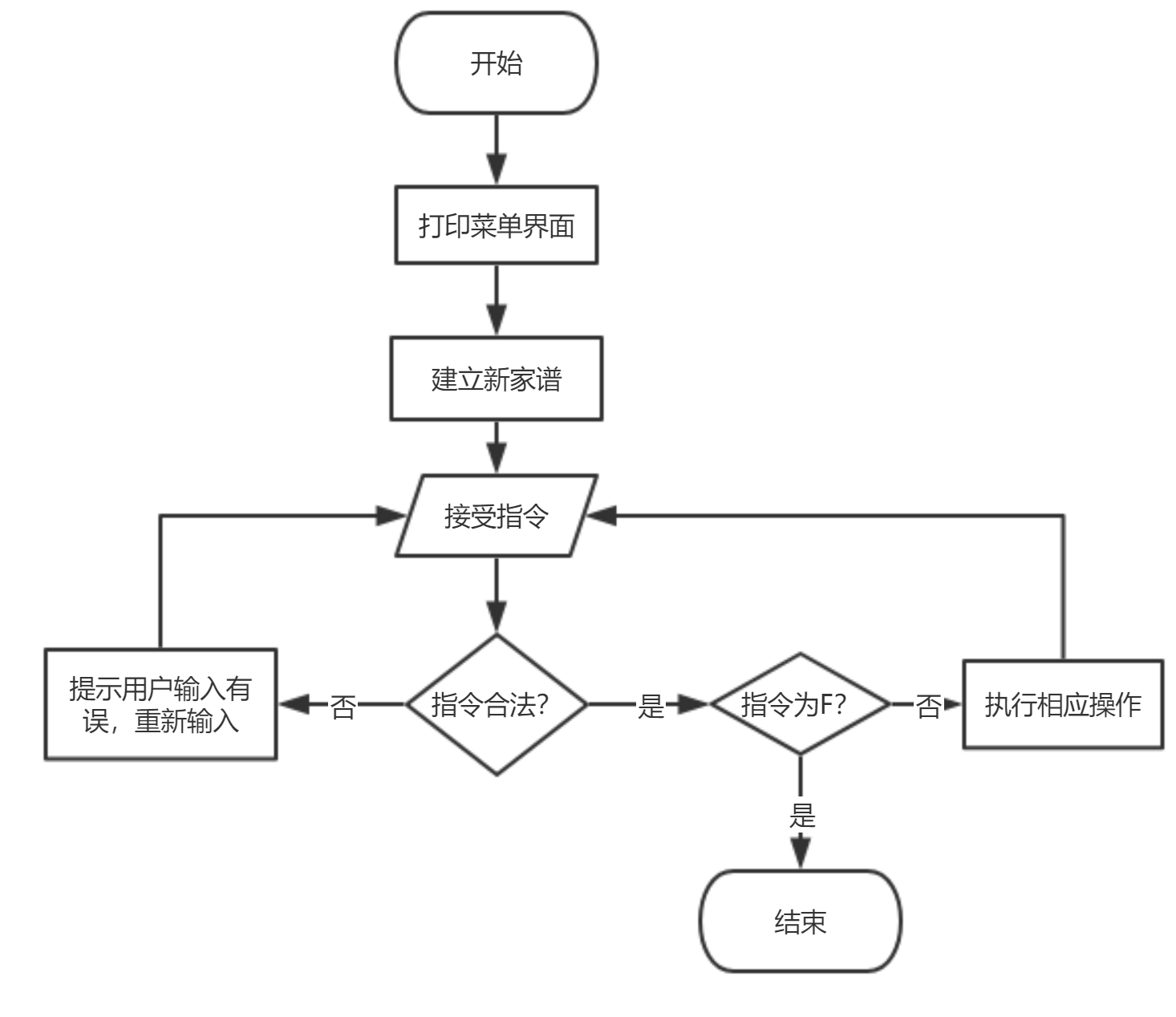
## 2.3 系统设计

系统通过交互式操作，不断根据用户要求对内置的家谱进行修改。首先需要打印出提示信息，让用户知道如何建立家谱树。接着，首先让用户输入家谱中祖先的姓名并展示。之后让用户通过指令自由选择对家谱的操作，完成增删改查等对家谱的基本操作，并且操作完能妥善退出。

# 3.功能实现

## 3.1总流程实现：

流程图：



先输出一些用户提示信息。然后让用户输入家谱中的祖先的名字。随后进入一个直到用户主动退出才结束的while循环，让用户选择相应的操作。当用户选择相应的操作后，给出相应的提示，让用户进行相应的输入，程序调用类的相应增删改查函数操作，并且把结果反馈给用户。

代码实现：

int main()

{

//打印菜单界面

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---查询家谱 \*\*" << endl;

cout << "\*\* F---退出程序 \*\*" << endl;

cout << "=========================================" << endl;

//建立新家谱

Tree<string> Family;

string RootName;

cout << "首先建立一个家谱!" << endl;

cout << "请输入祖先的姓名：";

cin >> RootName;

Family.buildRoot(RootName);

cout << "此家谱的祖先是：" << RootName << endl;

//执行操作

char choice;

bool isContinue = true;

while (isContinue)

{

cout << "\n请选择要执行的操作:";

cin >> choice;

switch (choice)

{

case 'A':

{

cout << "请输入要建立家庭的人的姓名:";

string fatherName;

cin >> fatherName;

cout << "请输入" << fatherName << "的儿女人数：";

int childCount;

cin >> childCount;

if (childCount < 0)

{

cerr << "儿女人数不能为负数！" << endl;

break;

}

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

string childName;

if (Family.Find(fatherName))

{

while (childCount--)

{

cin >> childName;

Family.Insert(fatherName, childName);

}

}

else

{

cerr << "\n查无此人，添加失败！" << endl;

break;

}

cout << fatherName << "的第一代子孙是：";

Family.Show(fatherName);

break;

}

case 'B':

{

cout << "请输入要添加儿子（或女儿）的人的姓名:";

string fatherName;

cin >> fatherName;

if (Family.Find(fatherName))

{

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

string childName;

cin >> childName;

Family.Insert(fatherName, childName);

}

else

{

cerr << "\n查无此人，添加失败！" << endl;

break;

}

cout << fatherName << "的第一代子孙是：";

Family.Show(fatherName);

break;

}

case 'C':

{

cout << "请输入要解散家庭的人的姓名：";

string fatherName;

cin >> fatherName;

//删除操作

cout << "要解散家庭的人是" << fatherName << endl;

cout << fatherName << "的第一代子孙是:";

Family.Show(fatherName);

Family.Destroy(fatherName);

break;

}

case 'D':

{

cout << "请输入更改姓名的人的目前姓名：";

string curName;

cin >> curName;

cout << "请输入更改后的姓名：";

string newName;

cin >> newName;

if (Family.Update(curName, newName) != NULL)

{

cout << curName << "已更名为" << newName << endl;

}

break;

}

case 'E':

{

cout << " 请输入要查询人的姓名：";

string curName;

cin >> curName;

Family.Show(curName);

break;

}

case 'F':

{

isContinue = false;

break;

}

default:cerr << "您所选择的操作不合法！" << endl;

}

}

system("pause");

return 0;

### }

### 3.2 建立家庭功能

该函数往家谱中添加第一个节点，即祖先。

//建立祖先结点

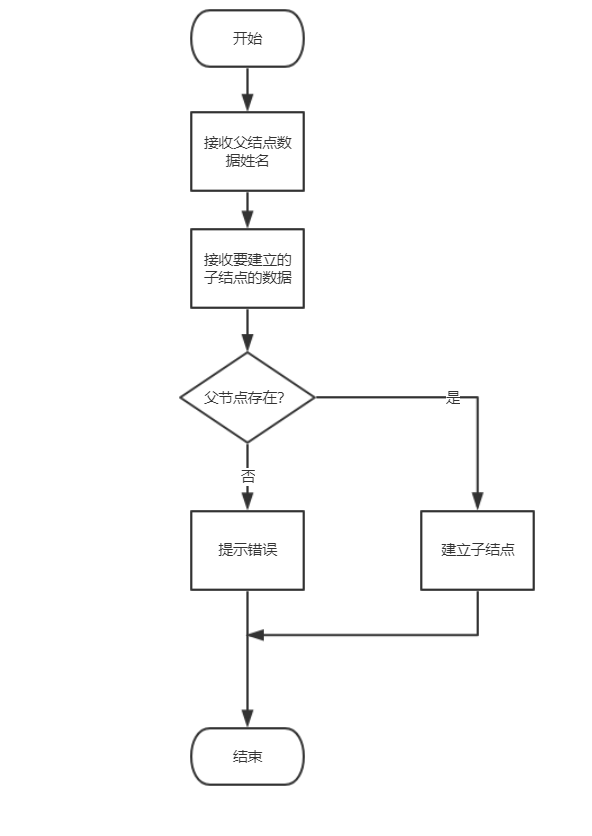
void buildRoot(T& Root)

{

root = new TreeNode<T>(Root);

}

3.3 添加儿女功能



调用Find函数，用指针father记录它的返回值。如果father为NULL（或者值为-1,），就表示家谱中没有这个人，所以插入失败，返回false；否则进行正式的插入操作。在插入时，需做一个判断，即father的子女指针next若指向NULL，表示此人原来没有子女，那么让father的next指针直接指向一个新的而儿女结点；如果next不指向NULL，即代表此人原来就有儿女，则要找到他最后一个儿女的位置，然后再将新儿女结点链接到末尾。

//插入新结点（添加新成员）

bool Insert(T& fatherName, T& childName)

{

TreeNode<T>\* father = Find(fatherName);

if (father == NULL) //插入失败

{

//cerr << "插入失败" << endl;

return false;

}

if (father->next == NULL) //原本无子女，直接插入

{

father->next = new TreeNode<T>(childName);

}

else

{

TreeNode<T>\* p = father->next; //找到要插入的位置

while (p->link != NULL)

{

p = p->link;

}

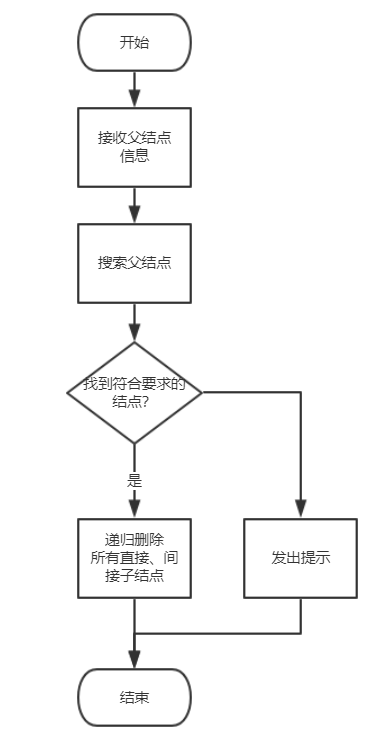
p->link = new TreeNode<T>(childName);

}

return true;

}

3.4 解散家庭功能



Destroy函数作用是将给定人物和他子孙都删除。

需要一个destroy函数，传入一个结点指针，可用于删除此结点和所有的兄弟和子孙。

//删除自己和所有的兄弟和子孙

void destroy(TreeNode<T>\* subTree)

{

if (subTree != NULL)

{

destroy(subTree->next);

destroy(subTree->link);

delete subTree;

}

### }

Destroy函数的实现借助了destroy函数。首先调用Find函数找到要解散的家庭的根结点，如果找不到就提示错误信息并返回，否则把当前结点的next指针传入destroy函数，删除掉各代子孙，最后再删除自身，完成删除工作。

//删除自身和所有子孙

void Destroy(T& fatherName)

{

TreeNode<T>\* father = Find(fatherName);

if (father == NULL)

{

cerr << "查无此人！删除失败！" << endl;

return;

}

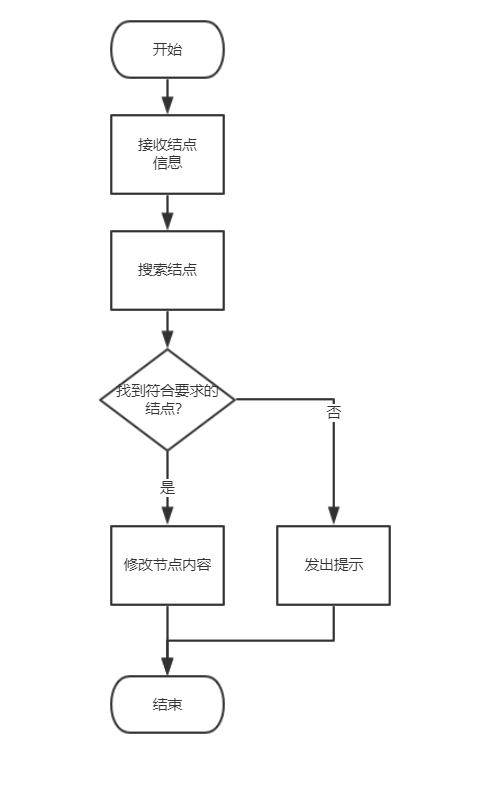
TreeNode<T>\* p = father->next;

destroy(p); //删除所有子孙

delete father; //删除自己

}

3.5 修改信息功能



该函数主要更新家谱中那个人的姓名。先查找到那个人，找不到就提示错误信息并且返回false，找到就把执行改名并返回true。

//更改家庭成员姓名

bool Update(T& curName, T& newName)

{

TreeNode<T>\* p = Find(curName);

if (p == NULL)

{

cerr << "查无此人！更改失败！" << endl;

return false;

}

p->value = newName;

return true;

}

3.6 查找功能

用递归实现函数Find(TreeNode<T>\* subTree, T& name)：以subTree为起点寻找姓名为name的结点，若找到则返回此结点的位置。

TreeNode<T>\* Find(TreeNode<T>\* subTree, T& name)

{

if (subTree == NULL) //未找到，无法继续往下找

{

return NULL;

}

if (subTree->value == name) //找到，返回此结点的位置

{

return subTree;

}

if (Find(subTree->link, name) != NULL)

{

return Find(subTree->link, name);

}

return Find(subTree->next, name);

### }

在此基础上，重载一个Find函数，传入一个name，可实现寻找姓名为name的结点。

TreeNode<T>\* Find(T& name)

{

return Find(root, name);

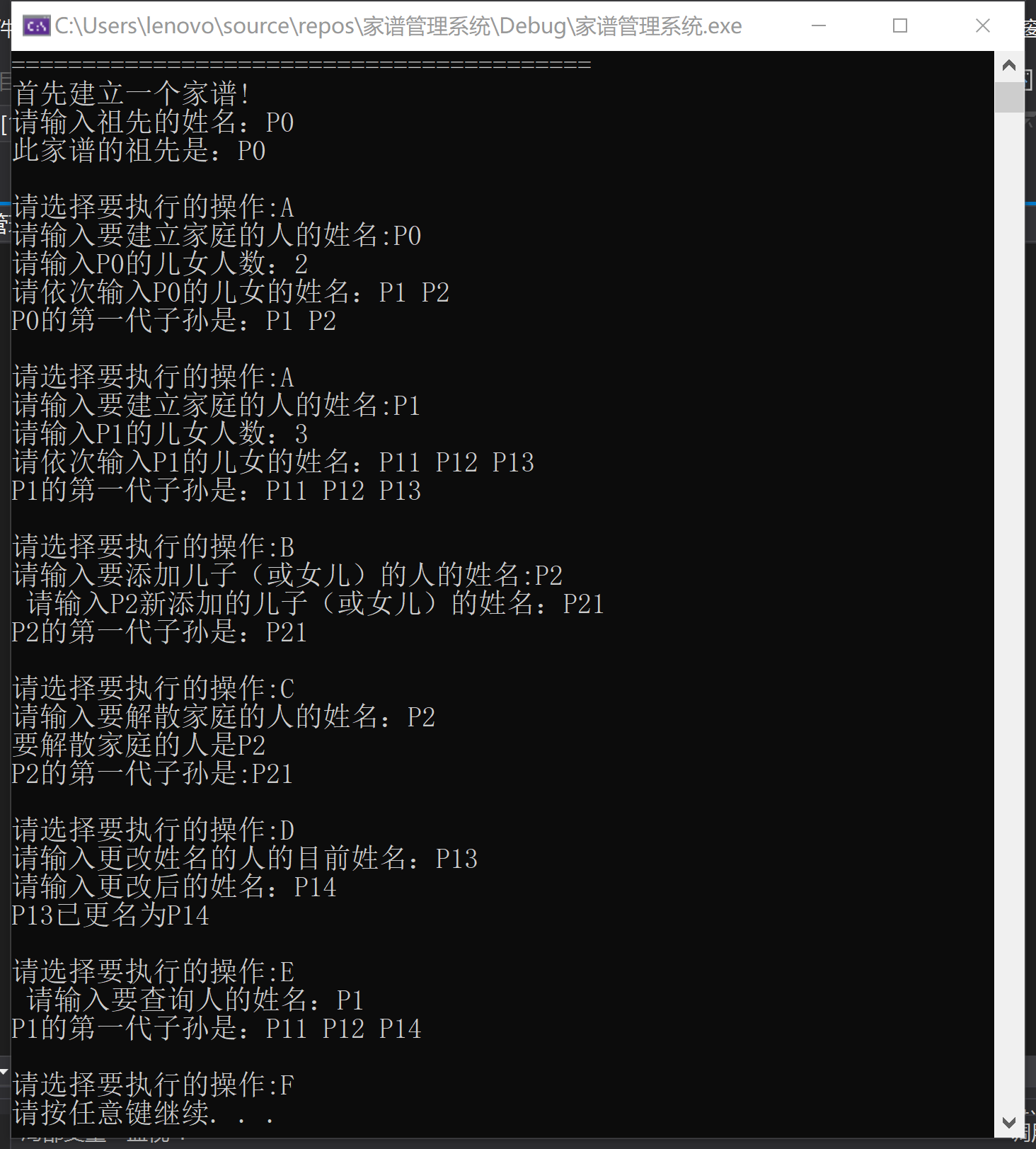
}

# 4 测试

## 4.1 功能测试

### 4.1.1 正常测试

**实验结果：**

****

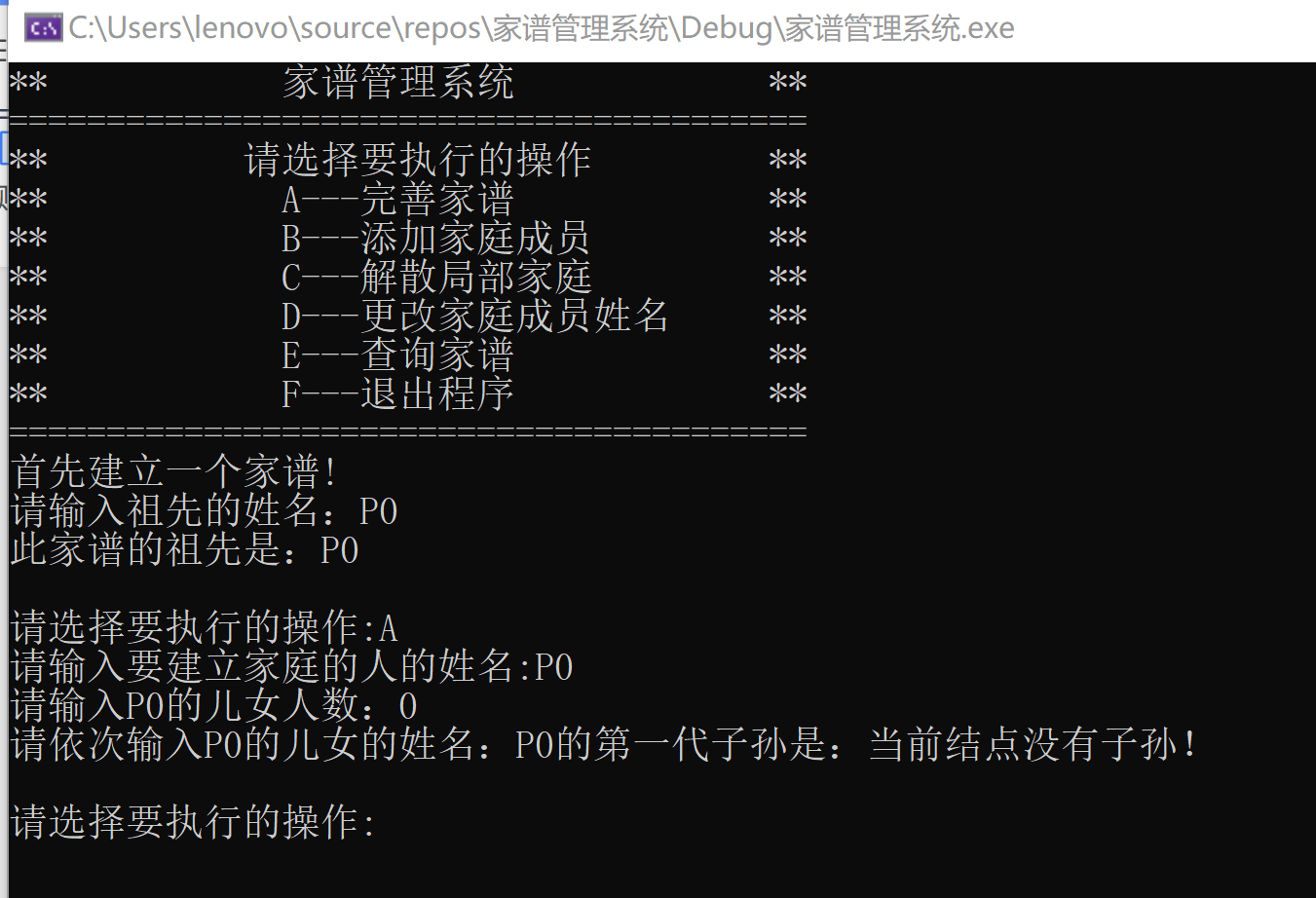
## 4.2 边界测试

### 4.2.1 新建子女数为0

**预期结果：**

程序正常运行不崩溃

**实验结果：**



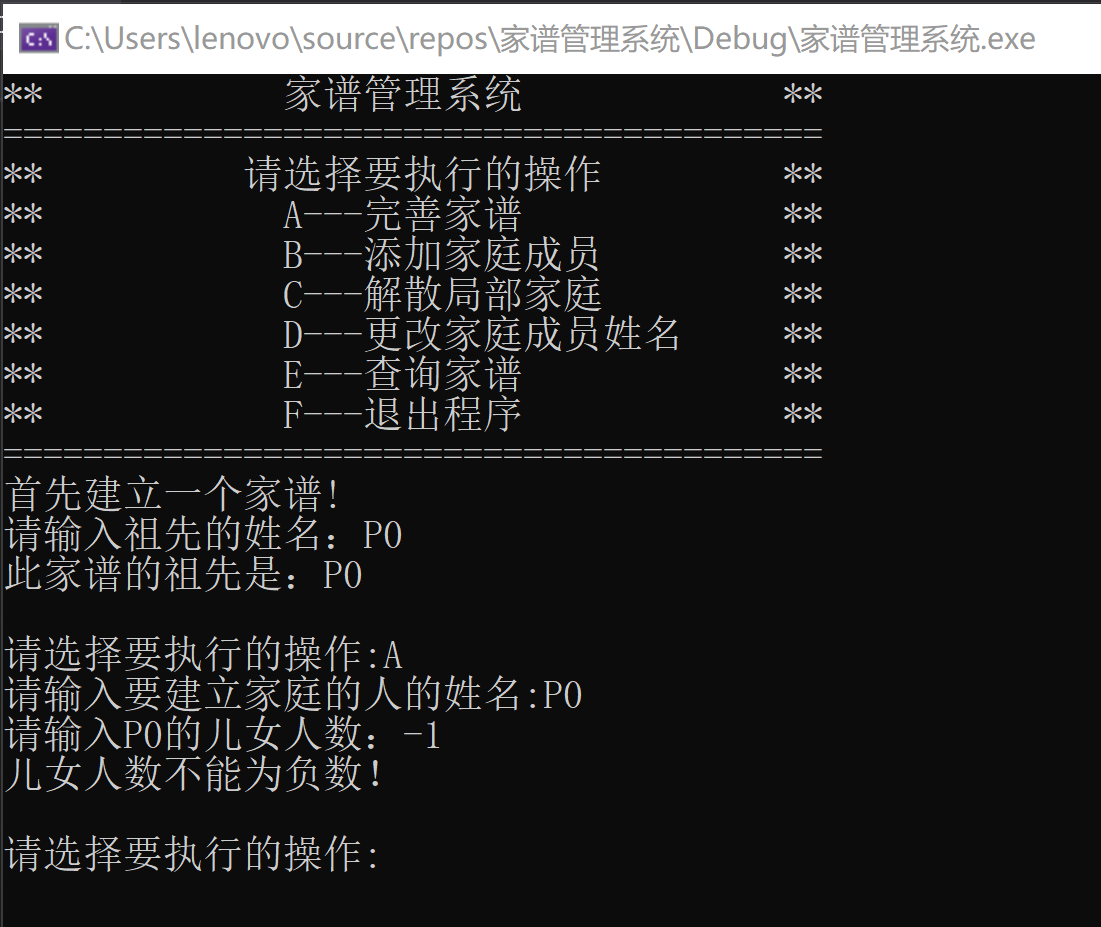
## 4.3 错误测试

### 4.3.1 新建子女数为负数

**预期结果：**

程序正常运行不崩溃，提示用户错误信息。

**实验结果：**

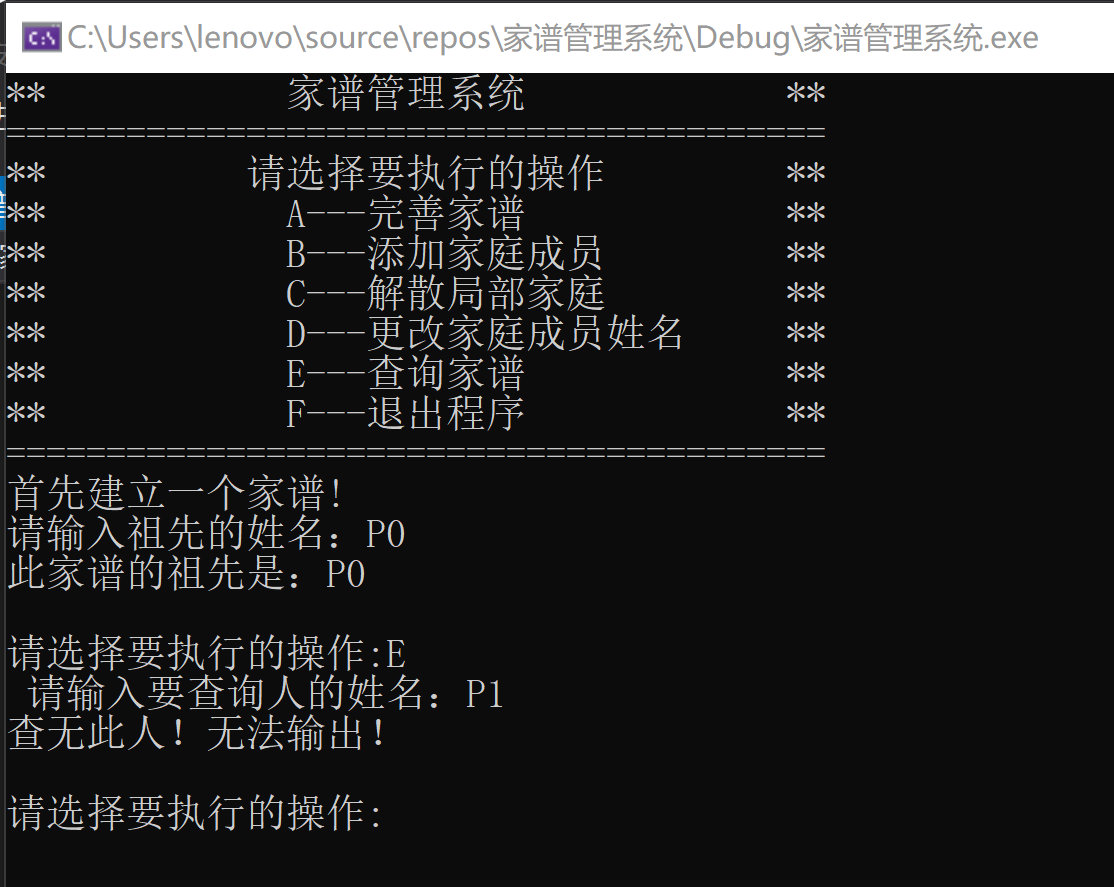


### 4.3.2 搜索不存在的结点

**预期结果：**

程序正常运行不崩溃，提示用户错误信息。

**实验结果：**





### 4.3.3 未知指令

预期结果：

程序正常运行不崩溃，提示用户输入有误。

运行结果：

