家谱管理系统

作 者 姓 名： 王凌

学 号： 1951504

指 导 教 师： 张颖

学院、 专业： 软件工程

同济大学

Tongji University

目录

[1 分析 1](#_Toc23829)

[1.1 项目简介 1](#_Toc15655)

[1.2 功能要求 1](#_Toc11861)

[2 设计 2](#_Toc16655)

[2.1 Node类设计 2](#_Toc18115)

[2.2 Tree类设计 2](#_Toc27119)

[2.3 Genealogy家谱类设计 3](#_Toc22980)

[2.4主程序设计 3](#_Toc23467)

[3 实现 4](#_Toc2418)

[3.1 Node类实现 4](#_Toc11924)

[3.2 Tree类实现 5](#_Toc6392)

[3.2.1 add(T ancester) 5](#_Toc23776)

[3.2.2 add(T familyName,T childName) 5](#_Toc5788)

[3.2.3 deleteName(T fatherName) 6](#_Toc27899)

[3.2.4 changeName(T preName,T curName) 7](#_Toc25430)

[3.2.5 display(T familyName) 7](#_Toc18258)

[3.2.6 findName(T familyName) 8](#_Toc27621)

[3.2.7 findName(Node<T> \*x,T familyName) 8](#_Toc23432)

[3.3 Genealogy家谱类实现 9](#_Toc18365)

[3.3.1 buildFamily() 9](#_Toc6192)

[3.3.2 addChild() 9](#_Toc21581)

[3.3.3 breakFamily() 10](#_Toc4486)

[3.3.4 editName() 11](#_Toc22367)

[3.3.5 searchName() 11](#_Toc22202)

[3.3.6 process() 12](#_Toc18857)

[3.3 主函数的实现 13](#_Toc13001)

[4 测试 14](#_Toc13007)

[4.1 功能测试 14](#_Toc15616)

[4.1.1 建立功能测试 14](#_Toc2644)

[4.1.2 插入功能测试 16](#_Toc32401)

[4.1.3 删除功能测试 17](#_Toc27994)

[4.1.4 修改功能测试 18](#_Toc24044)

[4.1.5 查找功能测试 19](#_Toc14114)

[4.2 边界测试 20](#_Toc17842)

[4.2.1 没找到对应的familyName对5种操作的影响 20](#_Toc15003)

[4.2.6 操作码为小写字母 25](#_Toc27778)

# 1 分析

## 1.1 项目简介

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。

## 1.2 功能要求

本项目的实质是完成对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 Node类设计

Node类含有三个公有成员变量，包括：一个T类型的权值，一个记录当前结点是否被删除的bool型变量isDelete，两个T类型的指针NextSibling、FirstChild。

我采取了“左子女-右兄弟”的方式来存储树结构，FirstChild指针指向结点的长子，NextSibling指向它的兄弟。

除此之外，Node类还含有一个自定义构造函数。

## 2.2 Tree类设计

Tree类的私有成员变量有两个，分别是是指向根节点的root指针和指向当前结点的currentNode指针。

**私有成员函数：**

void findName(Node<T> \*x,T familyName)，其作用是找到名字叫familyName的人的位置，并将这个位置存在currentNode中

**公有成员函数：**

void add(T ancester)用来存放祖先；

void add(T familyName,T childName)用来给名为familyName的父结点添加子节点；

void deleteName(T fatherName)用来删除父节点名为familyName的一整棵树，即从家谱中移除familyName和他的所有子女;

void changeName(T preName,T curName)更改家谱中名为preName的姓名;

void display(T familyName)展示名叫familyName的所有子女;

bool findName(T familyName)判断名字叫familyName的人是否存在，这个函数调用私用成员函数findName进行判断。

## 2.3 Genealogy家谱类设计

Genealogy家谱类设计的目的是完成对家谱成员信息的建立，查找，插入，修改，删除等功能。

它含有一个私有成员变量Tree<string> familyTree，所有的操作都是对这个家谱树进行的。

**私有成员函数：**

void buildFamily() 家庭的建立

void addChild() 给某个父结点下加入一个子结点

void breakFamily() 拆散一个家庭

void editName() 修改家谱中某个人的姓名

void searchName() 查询一个人的第一代子孙

**公有成员函数：**

void process() 运行家谱程序，通过操作者的指令调用相关的操作

## 2.4主程序设计

定义Genealogy家谱类的一个对象，调用对象的process()函数就可以了。

# 3 实现

## 3.1 Node类实现

我采取了“左子女-右兄弟”的方式来存储树结构，所以结点类中需要两个指针。FirstChild指针指向结点的长子，NextSibling指向它的兄弟。

与此同时，由于“左子女-右兄弟”的存储只有后继结点、没有前驱结点，删除某一节点的之后再连接链表的操作较为繁杂，需要定义额外的指针并且分类讨论，为了避免这种情况，我还在Node类中加了一个isDelete成员变量，用来判断这个结点有没有被删除。

1. **template**<**class** T> **class** Node
2. {
3. **public**:
4. T value;
5. //用 左子女-右兄弟 的方式来存储树结构
6. Node<T> \*NextSibling;
7. Node<T> \*FirstChild;
8. **bool** isDelete;
9. Node(T value):value(value),NextSibling(NULL),FirstChild(NULL),isDelete(0){}
10. };

## 3.2 Tree类实现

### 3.2.1 add(T ancester)

只加一个结点的特殊情况，加的是祖先。

1. **void** add(T ancester)//存放祖先
2. {
3. root=**new** Node<T>(ancester);
4. }

### 3.2.2 add(T familyName,T childName)

给名为familyName的父结点添加子节点。

首先调用findName()函数先在家谱中找有没有叫familyName的人，如果没找到就直接返回报错。如果找到了，那么此时currentNode中存储的就是名字叫familyName的结点的位置了。之后分两种情况讨论。

如果该结点还没有长子，那么直接给他添加一个长子；否则，如果该节点已经有了长子，那么需要遍历该结点的长子的兄弟结点链表，遍历到最后一位结点之后再添加进去。

1. **void** add(T familyName,T childName)
2. //给名为familyName的父结点添加子节点
3. {
4. //置空当前结点位置后，存储名字叫fatherName的结点的位置
5. currentNode=NULL;
6. findName(root,familyName);
7. //找到了父节点位置，并存在currentNode之中
8. **if**(currentNode==NULL || currentNode->isDelete==1) **return**;
9. Node<T> \*curChild=currentNode->FirstChild;
10. //父节点的长子节点
11. **if**(curChild==NULL)
12. //如果没有长子，则加一个名字为childName的长子
13. {
14. curChild=**new** Node<T>(childName);
15. currentNode->FirstChild=curChild;
16. }
17. **else**
18. //如果已经有了长子，则在长子的兄弟结点末尾加一个名为childName的兄弟
19. {
20. Node<T> \*x=curChild;//长子结点
21. **while**(x->NextSibling!=NULL) x=x->NextSibling;
22. //找到长子的兄弟结点链表的末尾，在末尾加新结点
23. x->NextSibling=**new** Node<T>(childName);
24. }
25. }

### 3.2.3 deleteName(T fatherName)

作用是删除父节点名为familyName的一整棵树，即从家谱中移除familyName和他的所有子女。

首先调用findName()函数先在家谱中找有没有叫familyName的人，如果没找到就直接返回报错。如果找到了，那么此时currentNode中存储的就是名字叫familyName的结点的位置了。又因为只有后继指针，没有前驱指针，所以该父节点如果直接删除的话，后面的兄弟结点也会被删除。所以需要用isDelete变量置1记录它已经不在树中，其目的是不影响链表上其它兄弟成员的读取，所以不能直接让它直接变为NULL。而子女结点对兄弟链表没有影响，直接赋值为NULL删除。

1. **void** deleteName(T fatherName)
2. //删除父节点名为familyName的一整棵树
3. //即从家谱中移除familyName和他的所有子女
4. {
5. //置空当前结点位置后，存储名字叫fatherName的结点的位置
6. currentNode=NULL;
7. findName(root,fatherName);
8. **if**(currentNode==NULL || currentNode->isDelete==1) **return**;//没找到则返回
9. currentNode->isDelete=1;
10. //由于只有后继指针，没有前驱指针，所以该父节点如果直接删除的话，后面的兄弟结点也会被删除
11. //所以需要用isDelete变量记录它已经不在树中，其目的是不影响链表上其它兄弟成员的读取，所以不能直接让它直接变为NULL
12. currentNode->FirstChild=NULL;//子女结点对兄弟链表没有影响，直接删除
14. }

### 3.2.4 changeName(T preName,T curName)

该函数的作用是更改家谱中名为preName的姓名。

首先调用findName()函数先在家谱中找有没有叫preName的人，如果没找到就直接返回报错。如果找到了，那么此时currentNode中存储的就是名字叫preName的结点的位置了。将preName结点的value改为curName就行了。

1. **void** changeName(T preName,T curName)//更改家谱中名为preName的姓名
2. {
3. //置空当前结点位置后，存储名字叫fatherName的结点的位置
4. currentNode=NULL;
5. findName(root,preName);
6. **if**(currentNode== NULL || currentNode->isDelete==1) **return**;
7. //没找到则返回
8. Node<T> \*cur=currentNode;
9. cur->value=curName;
10. }

### 3.2.5 display(T familyName)

该函数的作用是展示名叫familyName的所有子女。

首先调用findName()函数先在家谱中找有没有叫familyName的人，如果没找到就直接返回报错。如果找到了，那么此时currentNode中存储的就是名字叫familyName的结点的位置了。之后，遍历他的长子节点的兄弟结点并输出。其中主义要判断“假活”的结点，这些结点已经被删除了，但是它们维系着链表结构，注意把它们排除。

1. **void** display(T familyName)//展示名叫familyName的所有子女
2. {
3. **int** N=0;//子女个数,如果没有输出则输出null
4. //置空当前结点位置后，存储名字叫fatherName的结点的位置
5. currentNode=NULL;
6. findName(root,familyName);
7. **if**(currentNode== NULL || currentNode->isDelete==1) **return**;
8. //没找到则返回
9. **for** (Node<T> \*temp=currentNode->FirstChild;temp!= NULL;temp=temp->NextSibling)
10. {
11. **if**(temp->isDelete!=1)//这个结点没有被删除
12. {
13. ++N;
14. cout<<temp->value<<" ";
15. }
16. }
17. **if**(N==0) cout<<"null"<<endl;
18. }

### 3.2.6 findName(T familyName)

通过调用私有成员函数findName()，如果找到了就返回true，否则返回false。

1. **bool** findName(T familyName)//判断名字叫familyName的人是否存在
2. {
3. //置空当前结点位置后，存储名字叫fatherName的结点的位置
4. currentNode=NULL;
5. findName(root,familyName);
6. **if**(currentNode==NULL || currentNode->isDelete==1)
7. **return** **false**;
8. **else** **return** **true**;
9. }

### 3.2.7 findName(Node<T> \*x,T familyName)

这个函数有两个形参，一个是结点x的指针（也有可能是祖先root），另一个是要找的人的名字familyName。每次先进行判断有没有找到，再用一个循环在结点x的兄弟结点中找，每一个兄弟结点都进行递归（深度优先搜索）。如果找到了，则currentNode记录的就是名字叫familyName的人的位置；否则currentNode不赋值，指向空。

1. **void** findName(Node<T> \*x,T familyName)
2. //找到名字叫familyName的人的位置，并将这个位置存在currentNode中
3. {
4. **if**(x==NULL) **return**;
5. **if**(x->value==familyName)
6. {
7. currentNode=x;
8. **return**;
9. }
10. findName(x->FirstChild,familyName);
11. **for**(Node<T> \*temp=x->NextSibling;temp!=NULL;temp=temp->NextSibling) findName(temp,familyName);
12. }

## 3.3 Genealogy家谱类实现

### 3.3.1 buildFamily()

根据提示建立一个新的家庭。

首先根据提示输入fatherName、childCount、childName，通过findName()函数找到fatherName的位置，并通过add()函数，在找到的位置处插入childCount个子结点，更新后输出fatherName所有的第一代子女名字。

1. **void** buildFamily()
2. {
3. string fatherName;
4. **int** childCount;
5. cout<<"请输入要建立家庭的人的姓名:";
6. cin>>fatherName;
7. **if**(familyTree.findName(fatherName))
8. //必须先找到父节点位置
9. //如果没有找到，则无法加
10. {
11. cout<<"请输入"<<fatherName<<"的儿女人数：";
12. cin>>childCount;
13. cout<<"请依次输入"<<fatherName<<"的儿女的姓名：";
14. string childName;
15. **for**(**int** i=0;i<childCount;i++)
16. {
17. cin>>childName;
18. familyTree.add(fatherName,childName);
19. }
20. cout<<fatherName<<"的第一代子孙是：";
21. familyTree.display(fatherName);
22. }
23. **else** cout<<"该人不存在!";
24. }

### 3.3.2 addChild()

在父节点下添加孩子结点。

首先根据提示输入fatherName、childName，通过findName()函数找到fatherName的位置，并通过add()函数，在找到的位置处插入一个名为childName的结点，更新后输出fatherName所有的第一代子女名字。

1. **void** addChild()
2. {
3. cout<<"请输入要添加儿子（或女儿）的人的姓名:";
4. string fatherName;
5. cin>>fatherName;
6. **if** (familyTree.findName(fatherName))
7. {
8. cout<<"请输入"<<fatherName<<"新添加的儿子（或女儿）的姓名：";
9. string childName;
10. cin>>childName;
11. familyTree.add(fatherName,childName);
12. cout<<fatherName<<"的第一代子孙是：";
13. familyTree.display(fatherName);
14. }
15. **else** cout<<"该人不存在!";
16. }

### 3.3.3 breakFamily()

解散一个家庭。

首先根据提示输入fatherName，通过findName()函数找到fatherName的位置，并通过display()函数先展示第一代子孙的名字，再通过deleteName()函数删除该节点。

1. **void** breakFamily()
2. {
3. cout<<"请输入要解散家庭的人的姓名：";
4. string fatherName;
5. cin>>fatherName;
6. //删除操作
7. cout<<"要解散家庭的人是"<<fatherName<<endl;
8. **if** (familyTree.findName(fatherName))
9. {
10. cout<<fatherName<<"的第一代子孙是:";
11. familyTree.display(fatherName);
12. familyTree.deleteName(fatherName);
13. }
14. **else** cout<<"该人不存在!";
15. }

### 3.3.4 editName()

首先根据提示输入fatherName、changeName，通过findName()函数找到fatherName的位置，并将该位置的结点的value改成changeName。

1. **void** editName()
2. {
3. cout<<"请输入更改姓名的人的目前姓名";
4. string fatherName;
5. cin>>fatherName;
6. **if**(familyTree.findName(fatherName))
7. {
8. cout<<"请输入更改后的姓名";
9. string changeName;
10. cin>>changeName;
11. cout<<fatherName<<"已更名为"<<changeName;
12. familyTree.changeName(fatherName,changeName);
13. }
14. **else** cout<<"该人不存在!";
15. }

### 3.3.5 searchName()

根据提示输入fatherName，通过findName()函数找到fatherName的位置。如果没找到则输出提示。

1. **void** searchName()
2. {
3. string fatherName;
4. cout<<"请输入要查询人的姓名：";
5. cin>>fatherName;
6. **if**(familyTree.findName(fatherName))
7. {
8. cout<<fatherName<<"的第一代子孙是：";
9. familyTree.display(fatherName);
10. }
11. **else** cout<<"该人不存在!";
12. }

### 3.3.6 process()

先输出家谱管理系统的操作面板。

之后让用户输入祖先的名字初始化家谱。随后进入while循环，让用户选择相应的操作或者退出。

用switch判断用户选择的操作（这里模糊了大小写，使得大小写字母均可以被识别），并转向操作编码对应的成员函数之中进行操作。

1. **void** Genealogy::process()
2. {
3. cout<<"\*\*            家谱管理系统              \*\*"<<endl;
4. cout<<"=========================================="<<endl;
5. cout<<"\*\*         请选择要执行的操作           \*\*"<<endl;
6. cout<<"\*\*            A---完善家谱              \*\*"<<endl;
7. cout<<"\*\*            B---添加家庭成员          \*\*"<<endl;
8. cout<<"\*\*            C---解散局部家庭          \*\*"<<endl;
9. cout<<"\*\*            D---更改家庭成员姓名      \*\*"<<endl;
10. cout<<"\*\*            E---查询家谱              \*\*"<<endl;
11. cout<<"\*\*            F---退出程序              \*\*"<<endl;
12. cout<<"=========================================="<<endl;
14. string familyRoot;
15. cout<<"首先建立一个家谱!"<<endl;
16. cout<<"请输入祖先的姓名："<<endl;
17. cin>>familyRoot;
18. familyTree.add(familyRoot);
19. cout<<"此家谱的祖先是："<<familyRoot<<endl;
20. cout<<"请选择要执行的操作:"<<endl;
21. **char** ops;//各项操作对应的编码
22. cin>>ops;
23. **while**(**true**)
24. {
25. **switch**(ops)//大小写都能识别
26. {
27. **case** 'A':
28. **case** 'a':
29. buildFamily();
30. **break**;
31. **case** 'B':
32. **case** 'b':
33. addChild();
34. **break**;
35. **case** 'C':
36. **case** 'c':
37. breakFamily();
38. **break**;
39. **case** 'D':
40. **case** 'd':
41. editName();
42. **break**;
43. **case** 'E':
44. **case** 'e':
45. searchName();
46. **break**;
47. **case** 'F':
48. **case** 'f':
49. **break**;
50. **default**:
51. cout<<"无效操作,请重新输入"<<endl<<endl;
52. **break**;
53. }
54. cout<<endl<<"请选择要执行的操作:"<<endl;
55. cin>>ops;
56. }
57. }

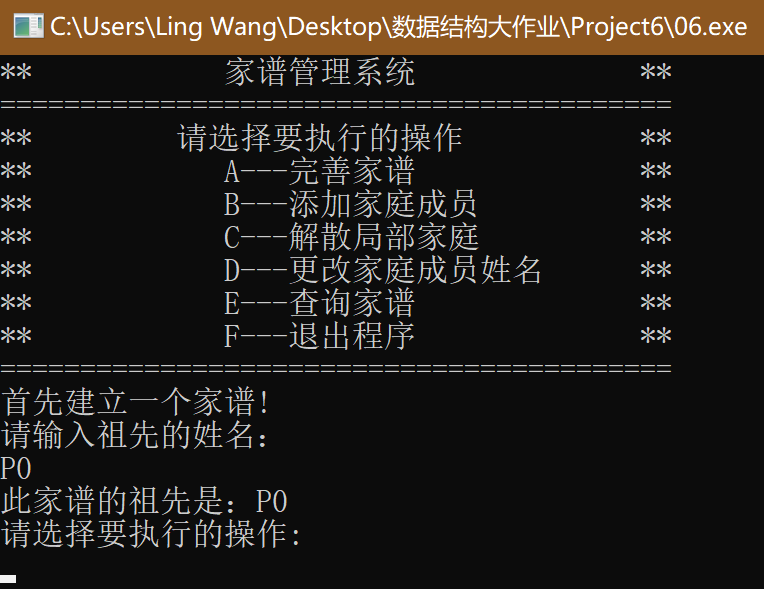
## 3.3 主函数的实现

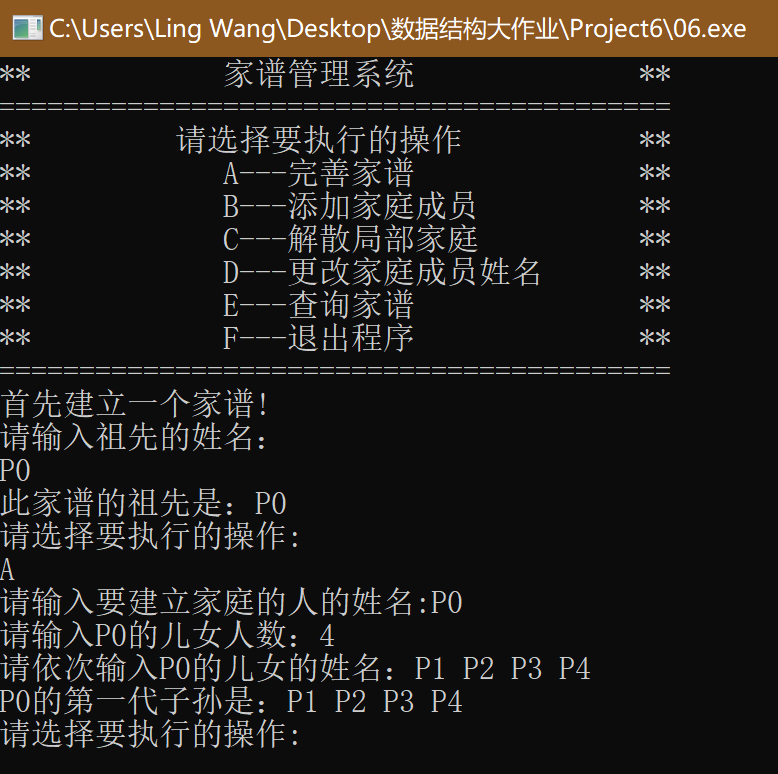
1. **int** main()
2. {
3. Genealogy jiapu;
4. jiapu.process();
5. **return** 0;
6. }

# 4 测试

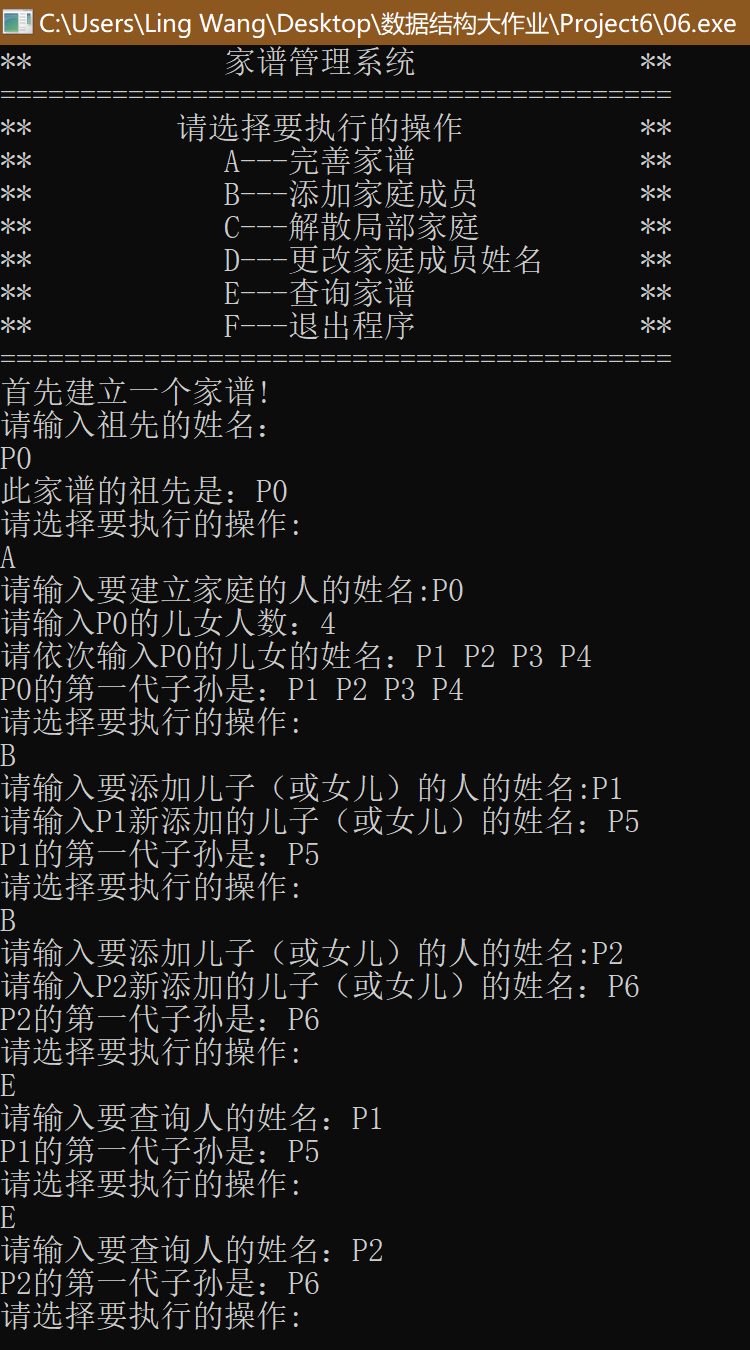
## 4.1 功能测试

### 4.1.1 建立功能测试

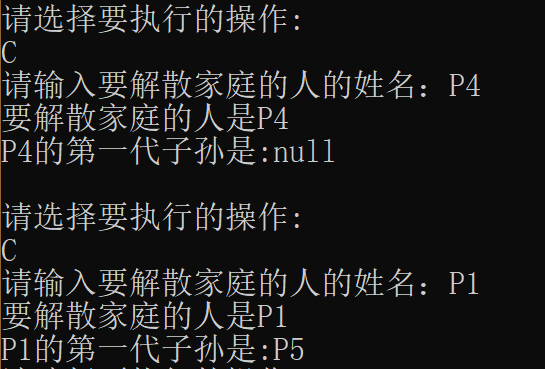




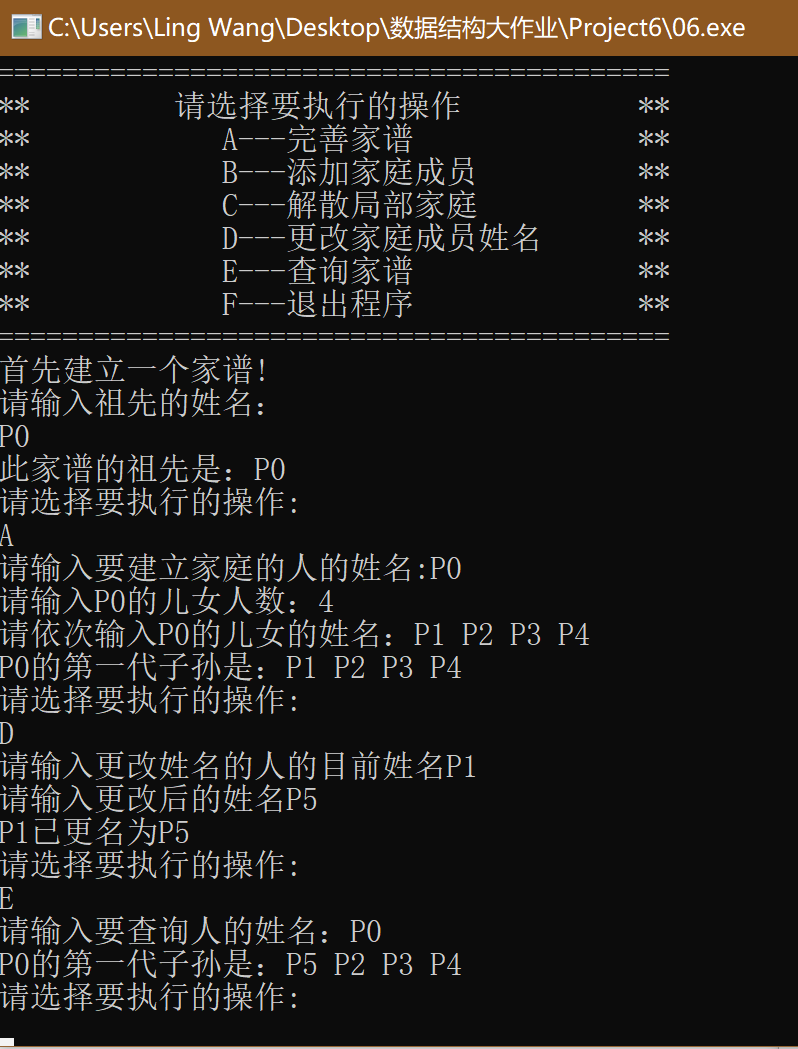
### 4.1.2 插入功能测试



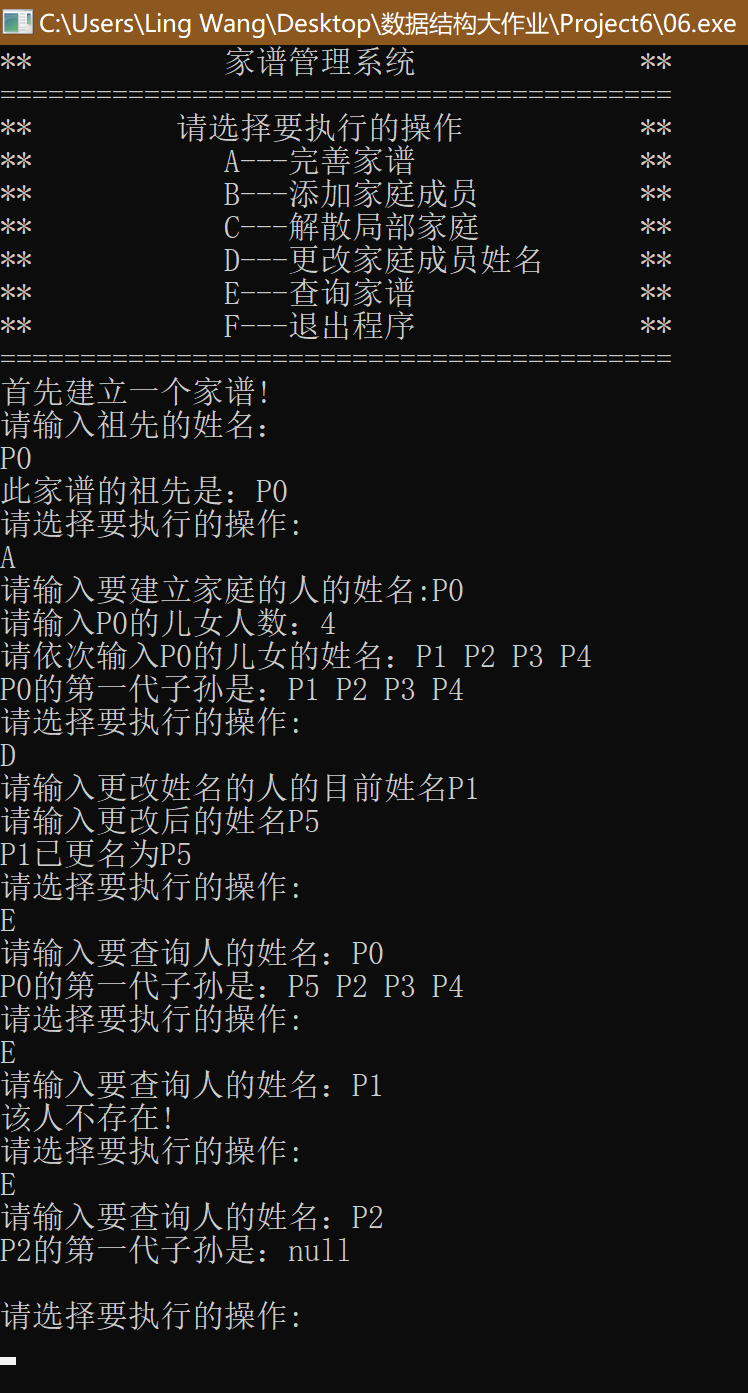
### 4.1.3 删除功能测试



### 4.1.4 修改功能测试



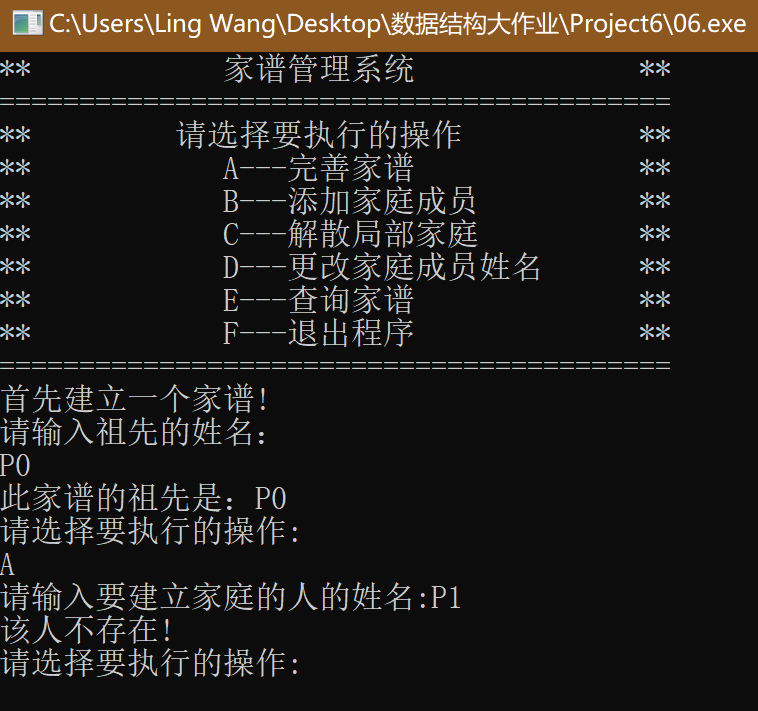
### 4.1.5 查找功能测试



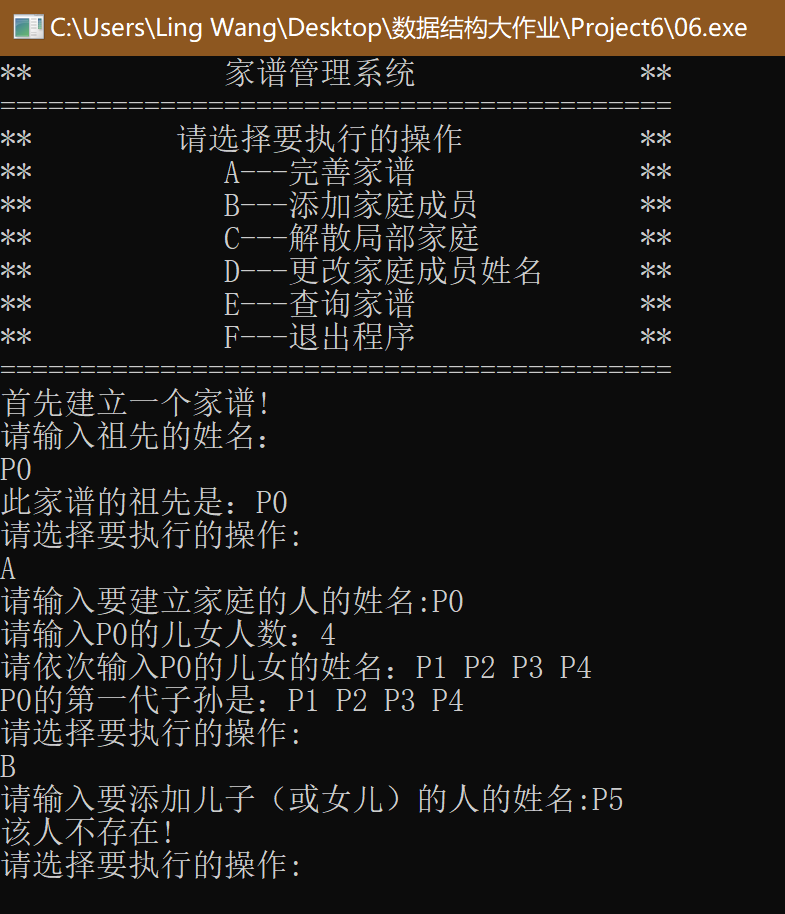
## 4.2 边界测试

### 4.2.1 没找到对应的familyName对5种操作的影响

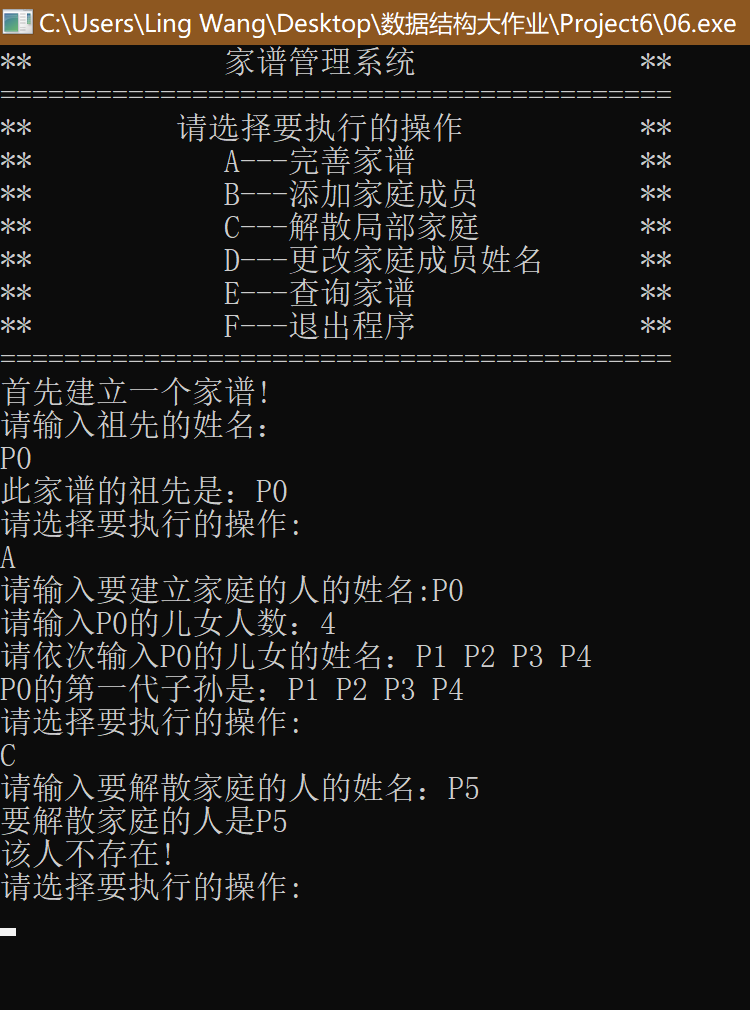
#### 4.2.1.1 A操作



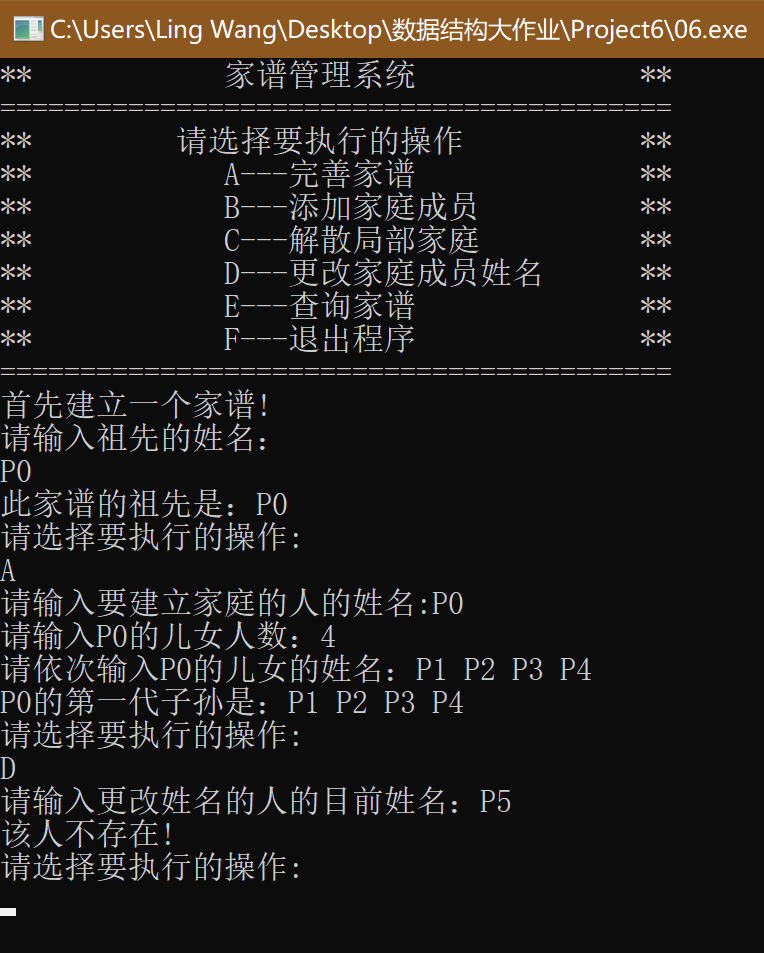
#### 4.2.1.2 B操作



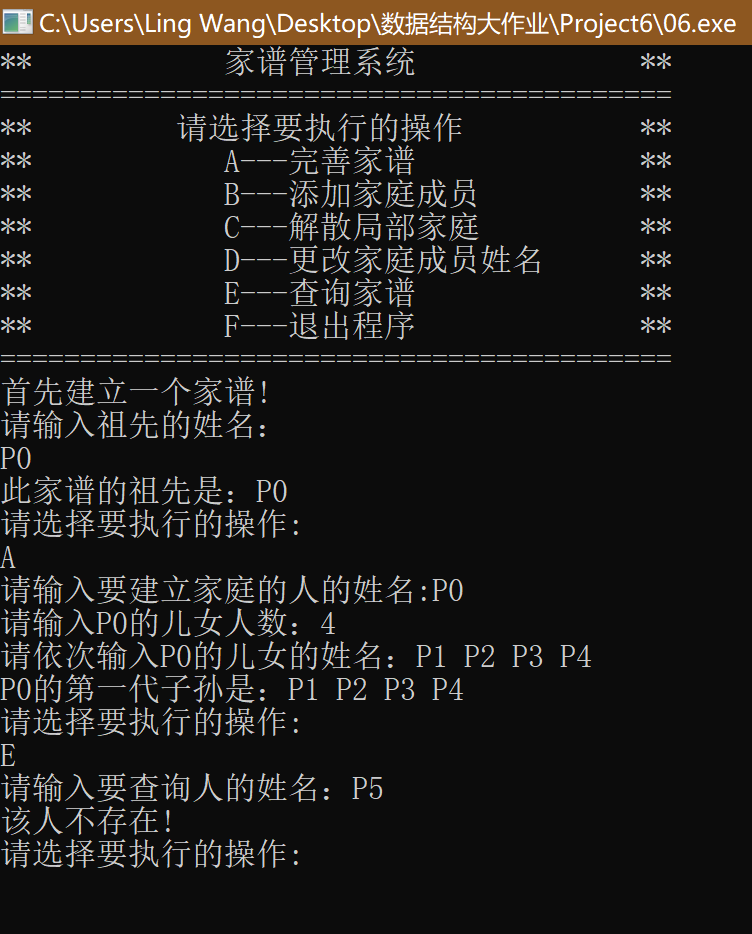
#### 4.2.1.3 C操作



#### 4.2.1.4 D操作



#### 4.2.1.5 E操作



### 4.2.6 操作码为小写字母

