**数据结构课程设计**

**项目说明文档**

**家谱管理系统**

作 者 姓 名： 苏家铭

学 号： 2151299

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 项目分析 1](#_Toc4818)

[1.1 项目背景 1](#_Toc9826)

[1.2项目要求 1](#_Toc29765)

[1.2.1 功能要求 1](#_Toc1876)

[本项目的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。 1](#_Toc21683)

[1.2.2 项目实例 1](#_Toc8936)

[2 项目设计及实现 2](#_Toc30590)

[2.1 数据结构设计思路 2](#_Toc21893)

[2.2 类设计 2](#_Toc7110)

[2.2.1 链表 2](#_Toc3319)

[2.2.2 队列 4](#_Toc32443)

[2.2.3 多叉树 4](#_Toc26069)

[2.3 项目算法 7](#_Toc5634)

[2.3.1 实现思路 7](#_Toc17659)

[2.3.2 代码实现 8](#_Toc27076)

[3 项目测试 10](#_Toc21946)

[3.1 建立家谱 10](#_Toc29848)

[3.2 完善只有单个子女的家庭 10](#_Toc23219)

[3.3 完善有多个子女的家庭 11](#_Toc24431)

[3.4 添加家庭成员测试 11](#_Toc28290)

[3.5 解散家庭成员 11](#_Toc25126)

[3.7 查看家庭成员的第一代子孙 12](#_Toc31378)

[3.8 选择操作时输入非法字符的情况（健壮性） 12](#_Toc5489)

[3.9 完善家谱时未找到成员（边界测试） 13](#_Toc7646)

[3.10 添加家庭成员时未找到成员（边界测试） 14](#_Toc10542)

[3.11 解散家庭成员时未找到成员（边界测试） 14](#_Toc19046)

[3.11 更改家庭成员姓名时未找到成员（边界测试） 15](#_Toc19059)

[3.12 查看儿女时未找到成员（边界测试） 15](#_Toc22432)

[4 算法性能分析 16](#_Toc7880)

[4.1 正确性 16](#_Toc31031)

[4.2 可使用性 16](#_Toc9833)

[4.3 可读性 16](#_Toc17604)

[4.4 效率 16](#_Toc9619)

[4.5 健壮性 16](#_Toc1650)

# 1 项目分析

## 项目背景

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。

经历了历朝历代的连年战乱和社会动荡，历史上传世的家谱几乎丧失[殆尽](https://baike.baidu.com/item/%E6%AE%86%E5%B0%BD/9481607?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%AE%B6%E8%B0%B1/_blank)，许多家族的世系也因此断了线、失了传。家谱的失传不仅仅是一个家族的损失，更是整个中华民族的损失。为了能让后人不再面临家谱丢失的情况，延续中华文化的传承，我们应当借助现代计算机帮助存储家谱中每代人的信息。

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项目要求

### 1.2.1 功能要求

### 本项目的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。

### 1.2.2 项目实例



# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

家谱中的每一代人数都是不固定的，并且在同辈中存在兄弟姐妹关系，鉴于此情况本例应当使用多叉树作为存储结构，并且应当使用左子女-右兄弟表示法表示多叉树。使用此方法可以更好地保留兄弟之间的信息，并且不像广义表表示法那样需要使用递归，算法不至于如此复杂。

## 2.2 **类设计**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

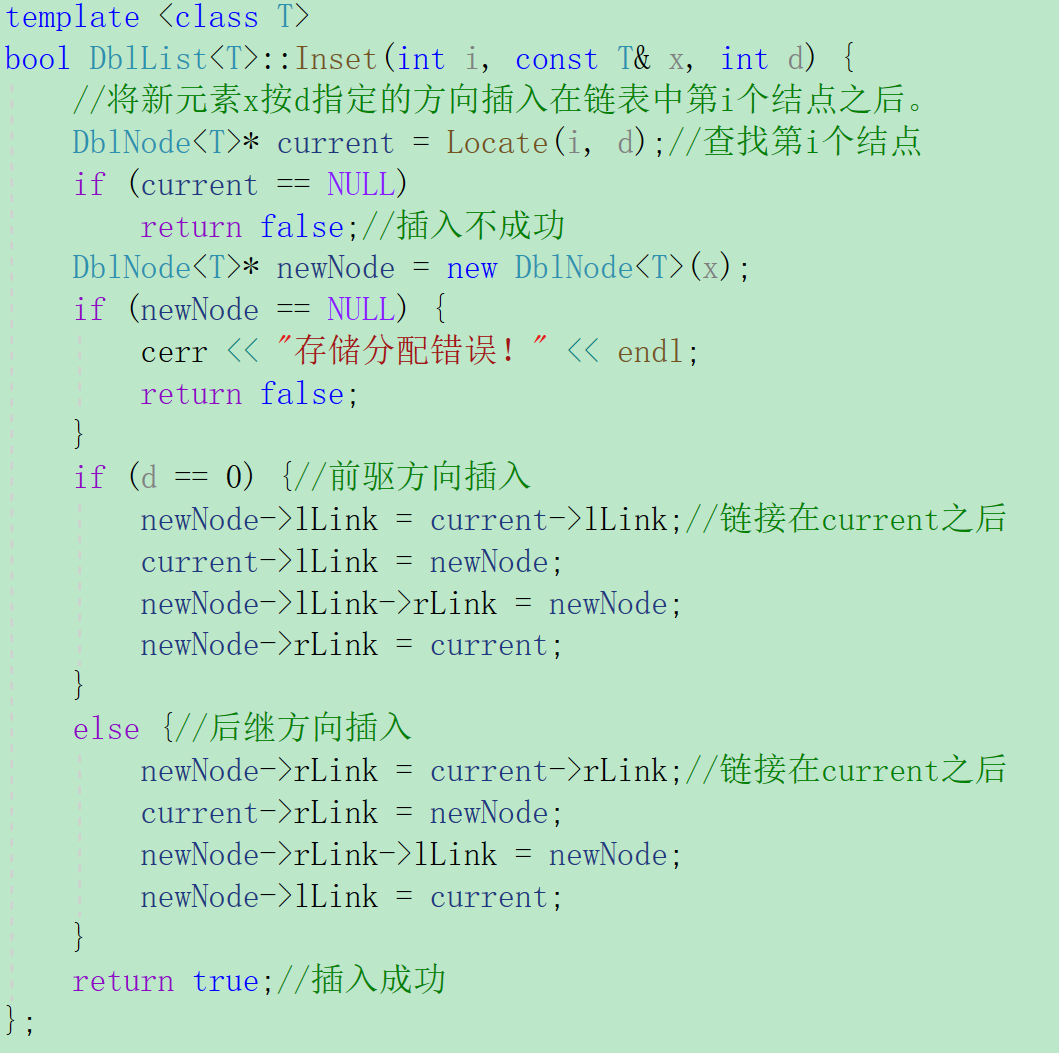
DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

插入算法的代码实现：



主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

### 2.2.2 队列

队列(Queue)是只允许在一端进行插入，而在另一端进行删除的运算受限的线性表，允许插入的一端叫做队尾(back)，允许删除的一端叫做队头(front)。队列也叫做先进先出的线性表，即FIFO(First In/First Out).

而本项目基于已经实现了的双向循环链表，实现了链式队列。每次入队时在链表的末尾插入一个元素，出队时则删除链表的第一个元素。出队入队都可以通过链表的first指针的左右指针找到队首和队尾，则无需再定义last指针，也无需遍历一遍链表找队尾元素，即出队、入队的时间复杂度可以降为O(1).

队列类数据成员：

template<class T>

DblList<T> list;

该链式队列已经实现6种功能：入队、出队、置空、判空、返回队首元素、返回队列长度。

具体函数声明为：

* void push(const T& val);//入队
* void pop();//出队
* void makeEmpty();//置空
* bool empty()const;//判断空否
* int size()const;//返回队列大小
* const T& front();//返回队首元素

主要算法的时间复杂度：

* 入队：O(1)
* 出队：O(1)
* 置空：O(n)
* 判断空否：O(1)
* 返回队首元素：O(1)
* 返回队列大小：O(n)

### 2.2.3 多叉树

树形结构是以分支关系定义的层次结构，是一类重要的非线性数据结构，在计算机领域和现实生活中有着广泛的应用。比如家族的家谱、书的章节等。

树是n（n>=0）个结点的有限集。当n = 0时，称为空树。在任意一棵非空树中应满足：

有且仅有一个特定的称为根的结点。

当n>1时，其余节点可分为m（m>0）个互不相交的有限集T1,T2,…,Tm，其中每个集合本身又是一棵树，并且称为根的子树。

显然，树的定义是递归的，即在树的定义中又用到了自身，树是一种递归的数据结构。树作为一种逻辑结构，同时也是一种分层结构，具有以下两个特点：

1：树的根结点没有前驱，除根结点外的所有结点有且只有一个前驱。

2：树中所有结点可以有零个或多个后继。

因此n个结点的树中有n-1条边。

对于一颗一般的树（多叉树）的存储方式，可以分为四种：广义表表示、父指针表示法、子女链表表示法、左子女-右兄弟链表表示法。综合不同存储方法的优缺点，本项目基于左子女-右兄弟链表表示法实现了一颗多叉树。此方法是一种二叉树表示法，它的每个结点的度为2，是最节省空间的树的存储表示。每一个结点由3个域组成：

|  |  |  |
| --- | --- | --- |
| T data | TreeNode<T>\* firstChild | TreeNode<T>\* nextSibling; |

定义如下：

template <class T>

struct TreeNode {

T data;//多叉树树结点数据

TreeNode<T>\* firstChild, \* nextSibling;//左子女及右兄弟指针

};

左指针表示孩子，右指针指向同辈兄弟。

多叉树类数据成员：

template <class T>

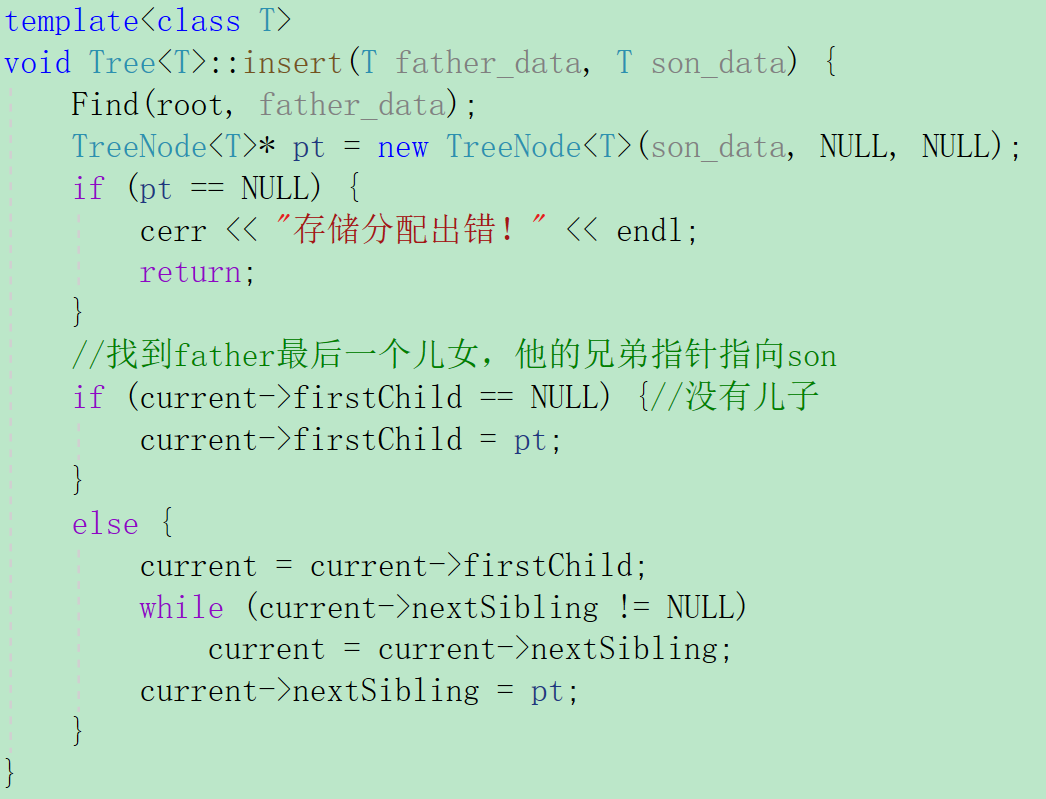
TreeNode<T>\* root, \* current;//根指针及当前指针

为了满足多个项目的要求，增加代码的复用性，在今后的程序中也能使用该多叉树类，本多叉树类实现包括插入、删除、搜索、返回各类指针、设置各类指针等，共17种功能。

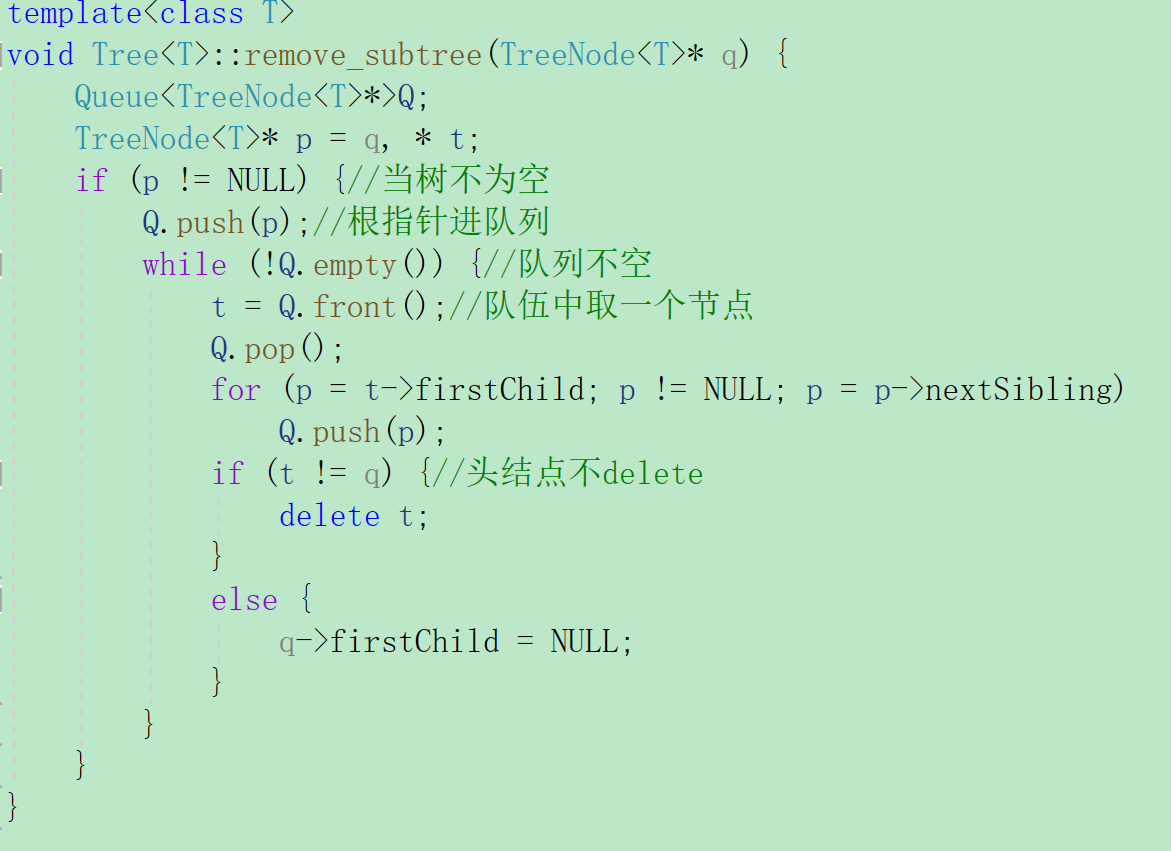
具体函数声明为：

* void set\_root(T r);//设置根结点
* bool Root();//current变为根节点
* bool IsEmpty() ;//判断该树是否为空，空则true，不空false
* TreeNode<T>\* get\_root();//返回根节点的地址
* TreeNode<T>\* get\_current();//返回current结点地址
* bool FirstChild();//current变为当前结点的第一个儿子
* bool NextSibling();//寻找current当前结点的下一个兄弟，使之成为current当前结点
* bool Parent();//寻找当前结点的父亲，使之成为当前结点
* bool Find(T target);//在树中搜索含有target的结点，使之成为current当前结点
* bool Find(TreeNode<T>\* p, T value);//在以p为根的树中搜索值value
* void change\_data(T old\_data, T new\_data);//更改old\_data的值变成new\_data
* void makeEmpty();//把树置空
* void insert(T father\_data, T son\_data);//在以father\_data为父结点的子节点插入一个son\_data结点
* void output\_children(T father\_data);//输出孩子们的data，current指向父亲
* void remove\_subtree(TreeNode<T>\* q);//删除以q为根的子树
* bool FindParent(TreeNode<T>\* t, TreeNode<T>\* p);//找到p结点的父亲结点
* TreeNode<T>\* find\_value(T val);//返回值为val结点的地址

插入算法代码实现：



删除算法代码实现：



主要算法的时间复杂度：

* 搜索：O(n)
* 插入：O(n)
* 删除：O(n)

## 2.3 **项目算法**

### 2.3.1 实现思路

1. 完善家谱：

基于多叉树的插入函数，完善家谱即在根结点中循环插入若干子孙。

1. 添加家庭成员：

通过多叉树的查找函数找到父结点，再使用多叉树的插入函数，参数为父结点和子节点。

1. 解散局部家庭：

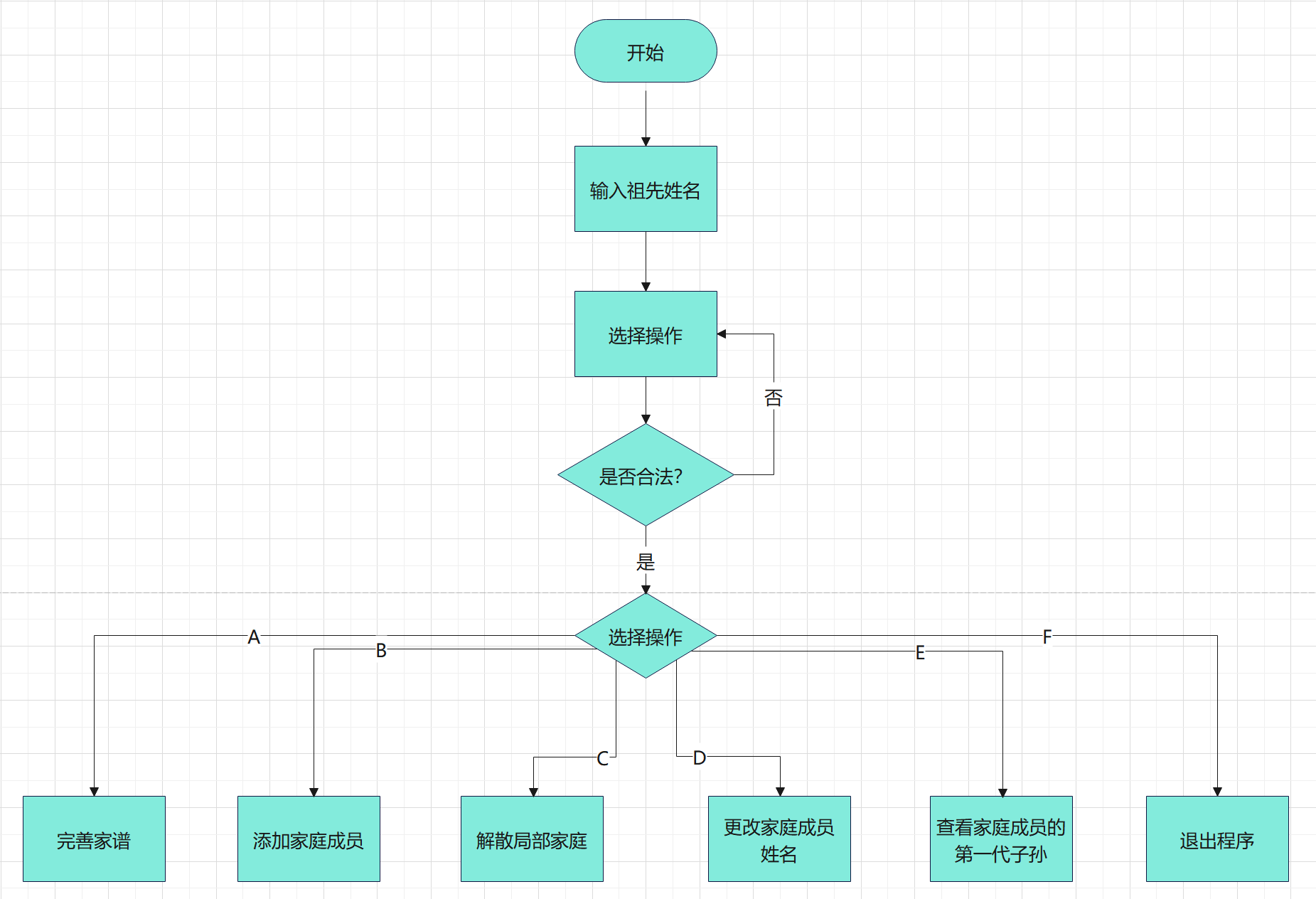
通过多叉树的查找函数找到父结点，再使用多叉树的删除函数，参数为父结点，删除父节点下以左结点为根的子树。

1. 更改家庭成员姓名：

通过多叉树的查找函数找到该结点，再使用多叉树的更改函数，更改该节点的信息。

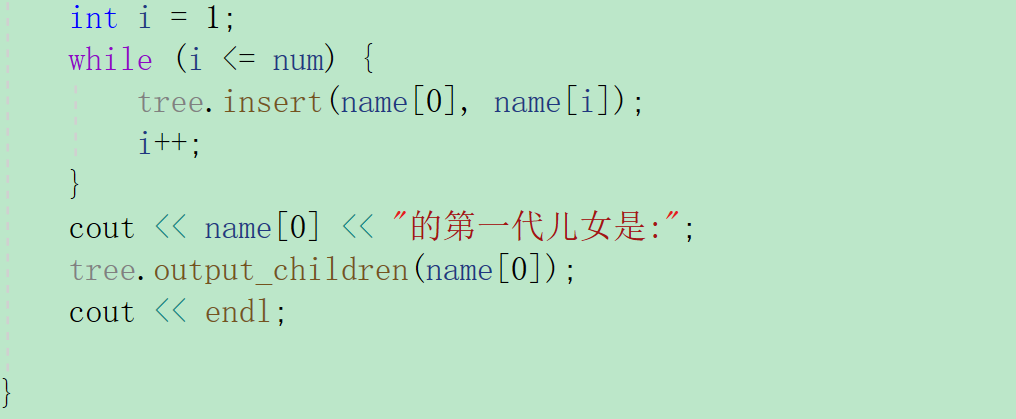
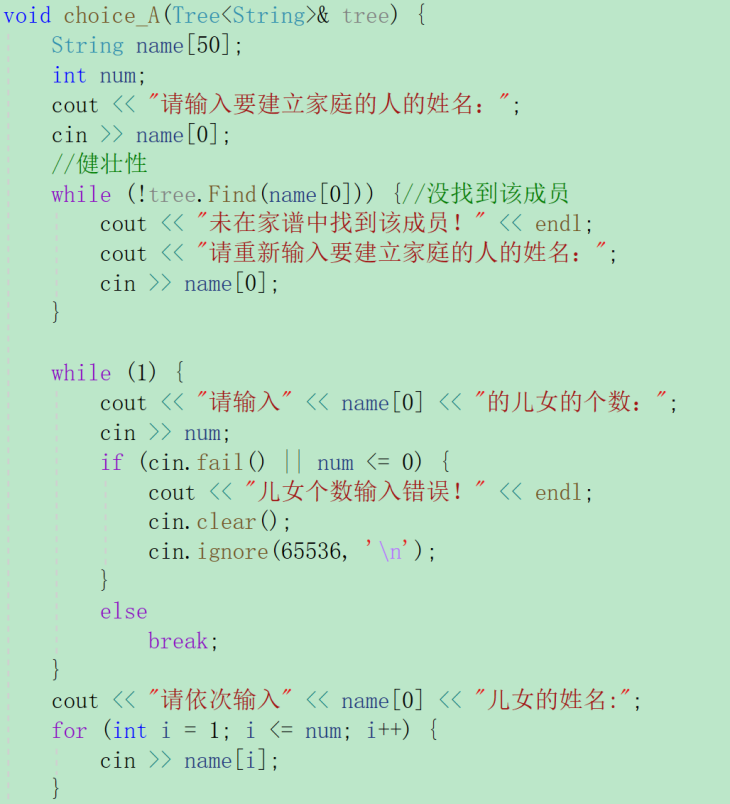
1. 查看家庭成员的第一代子孙：

通过多叉树的查找函数找到该结点，再输出该节点的左子女信息，再依次输出该左子女的右指针信息，直到右指针为NULL。

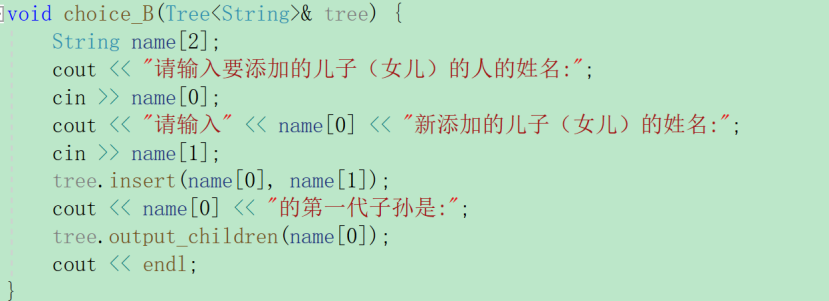
程序总体流程图如下：

### 2.3.2 代**码实**现

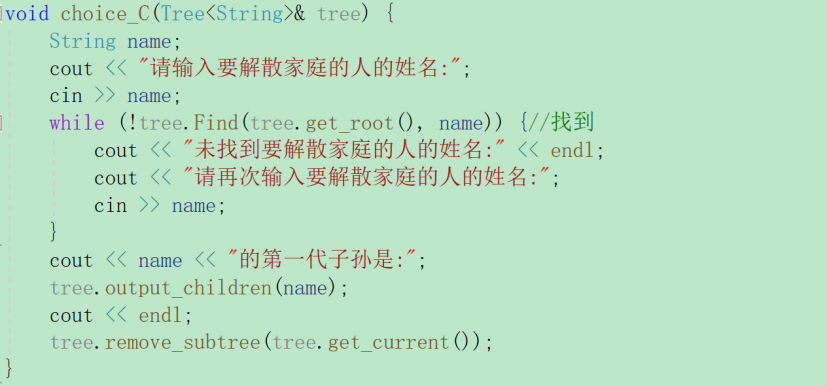
完善家谱：



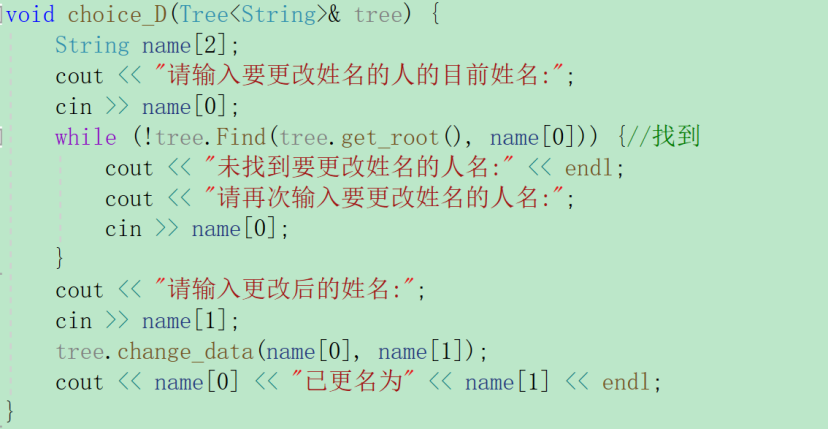
添加家庭成员：



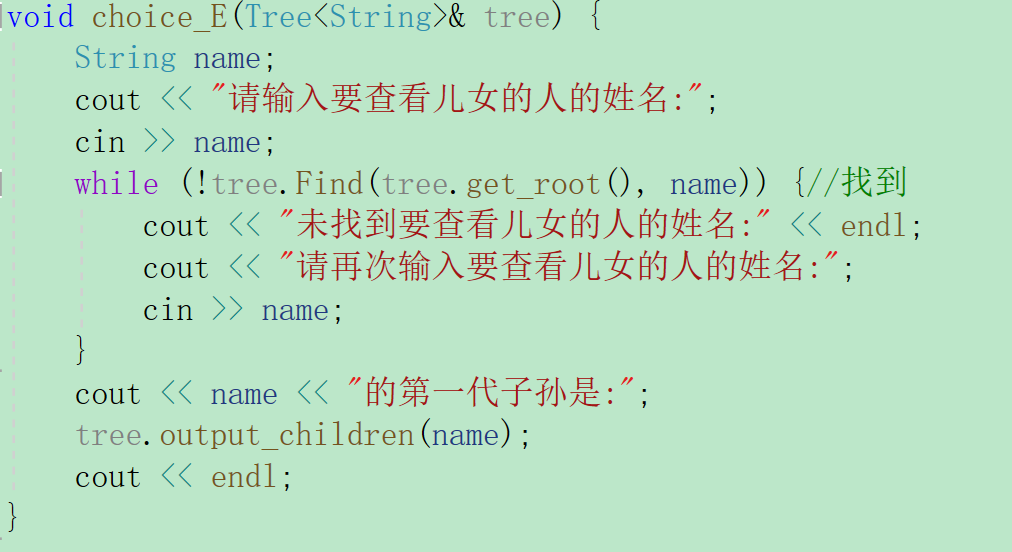
解散局部家庭：



更改家庭成员姓名：



查看家庭成员的第一代子孙：



# 3 项目测试

## 3.1 建立家谱

测试输入：

P0

实验结果：



## 3.2 完善只有单个子女的家庭

测试输入：

A P0 2 P1 P2

预期输出：

P0的第一代儿女是:P1 P2

实验结果：



## 3.3 完善有多个子女的家庭

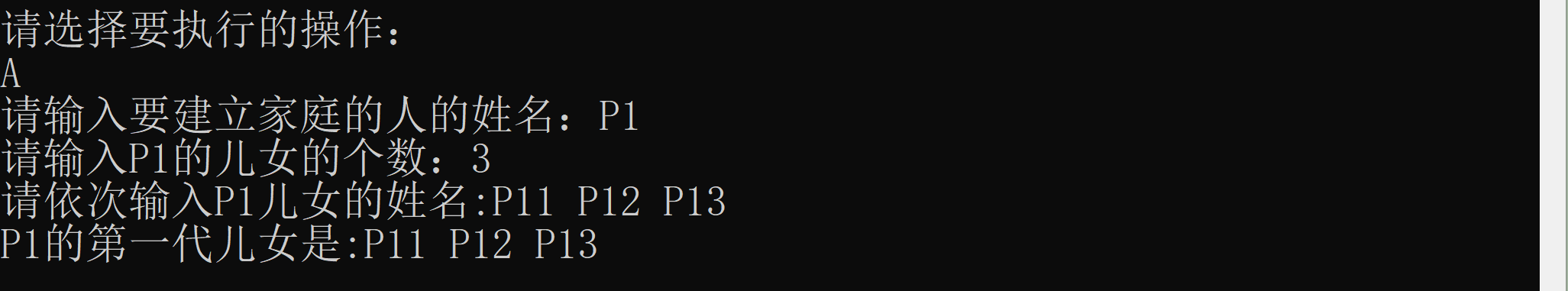
测试输入：

A P1 3 P11 P12 P13

预期输出：

P1的第一代儿女是:P11 P12 P13

实验结果：



## 3.4 添加家庭成员测试

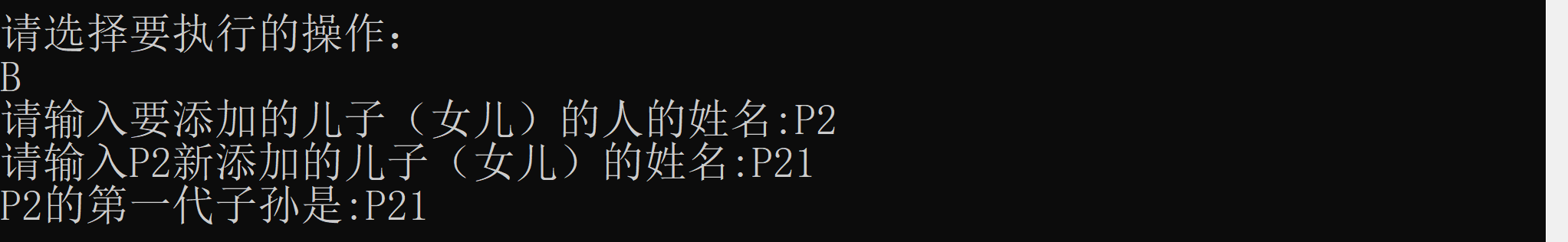
测试输入：

B P2 P21

预期输出：

P2的第一代子孙是:P21

实验结果：



## 3.5 解散家庭成员

测试输入：

D P13 P14

预期输出：

P13已更名为P14

实验结果：



## 3.7 查看家庭成员的第一代子孙

测试输入：

E P1

预期输出：

P1的第一代子孙是:P11 P12 P14

实验结果：



测试输入：

E P0

预期输出：

P0的第一代子孙是:P1 P2

实验结果：



## 3.8 选择操作时输入非法字符的情况（健壮性）

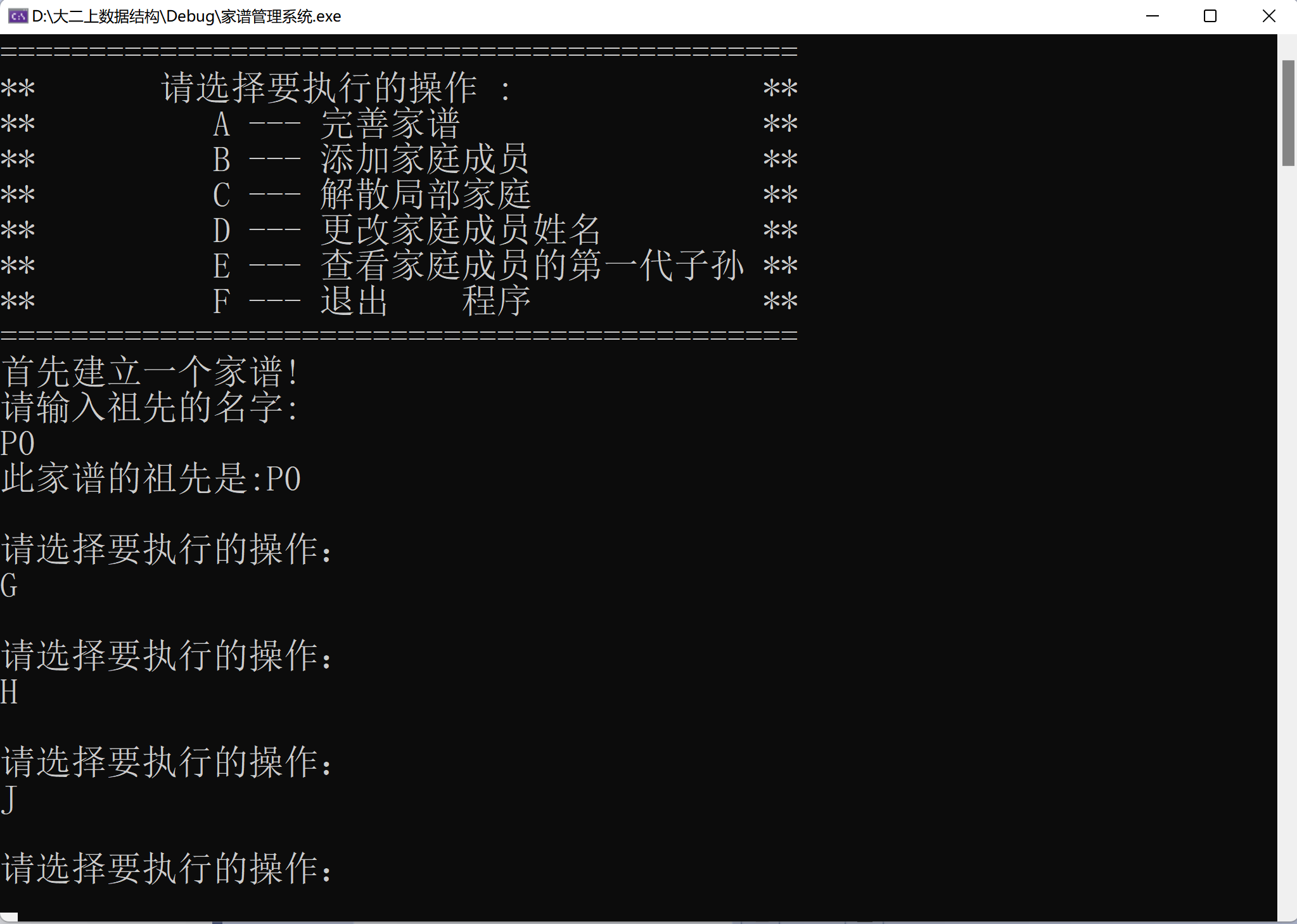
测试输入：

G H J

预期输出：

可重新输入

实验结果：



## 3.9 完善家谱时未找到成员（边界测试）

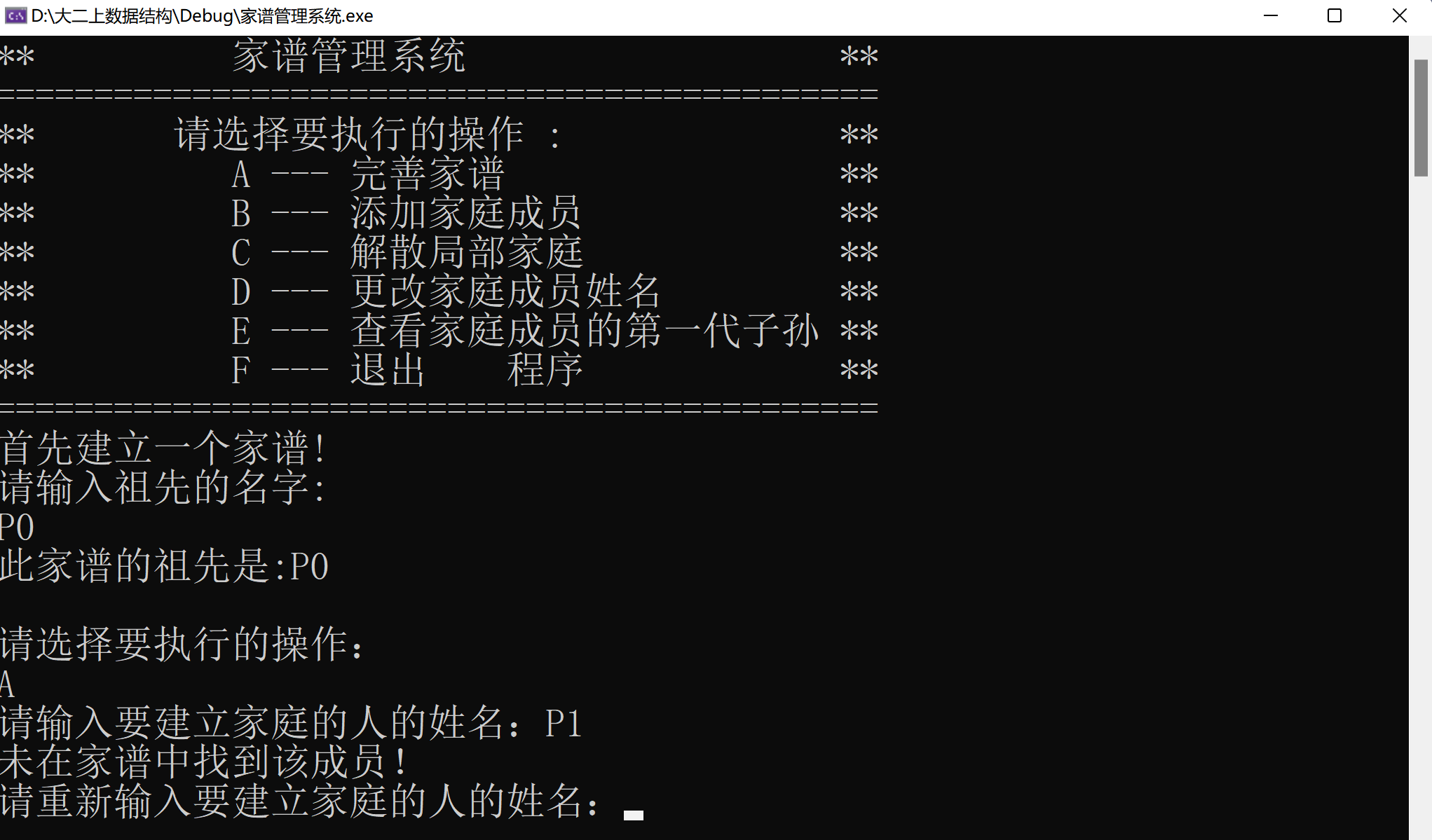
测试输入：

P0 A P1

预期输出：

未在家谱中找到该成员！

实验结果：



## 3.10 添加家庭成员时未找到成员（边界测试）

此时祖先：P0

P0子女：P1 P2

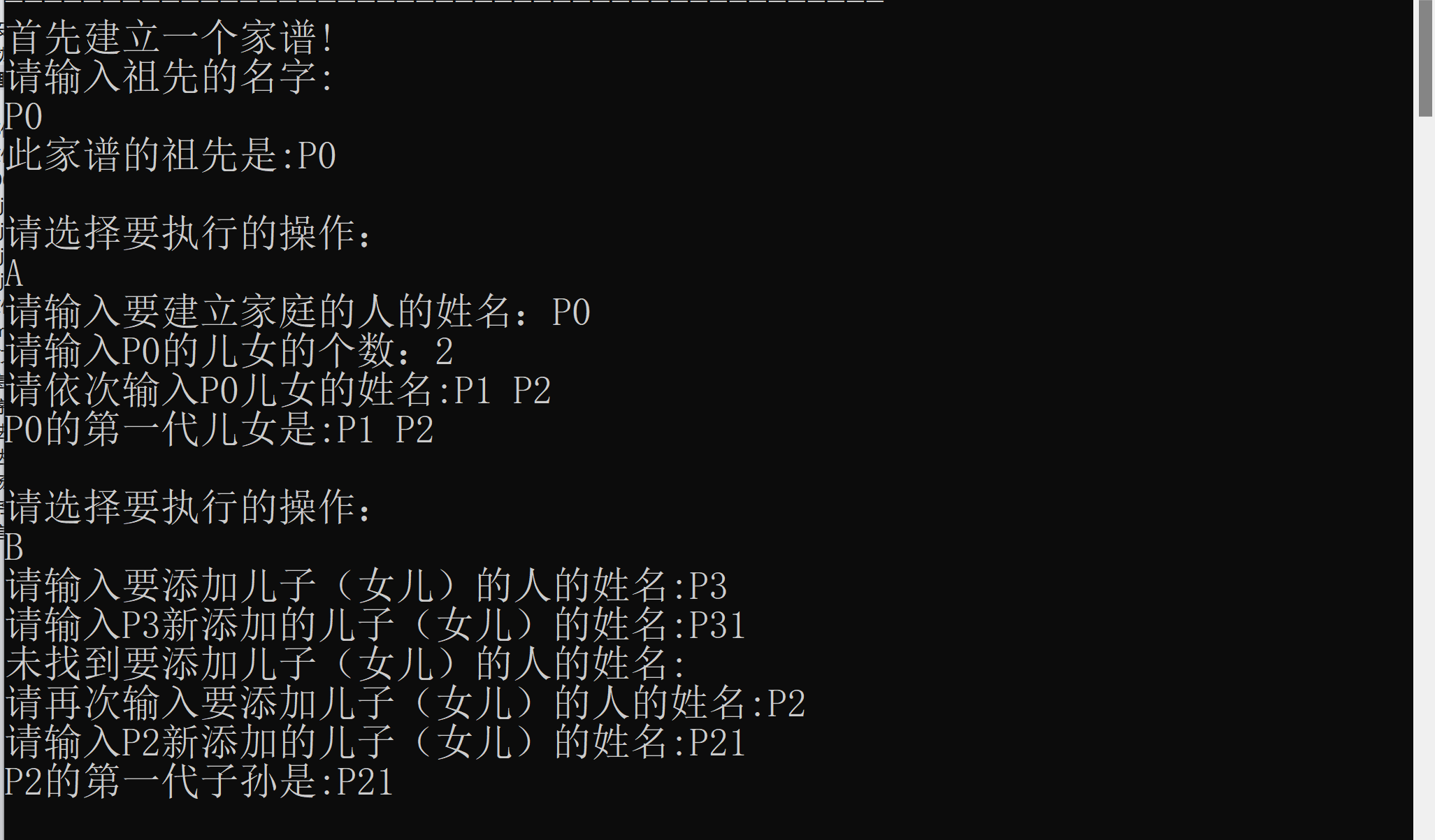
测试输入：

B P3 P31

预期输出：

错误提示+再次输入

实验结果：



## 3.11 解散家庭成员时未找到成员（边界测试）

此时祖先：P0

P0子女：P1 P2

P2子女：P21

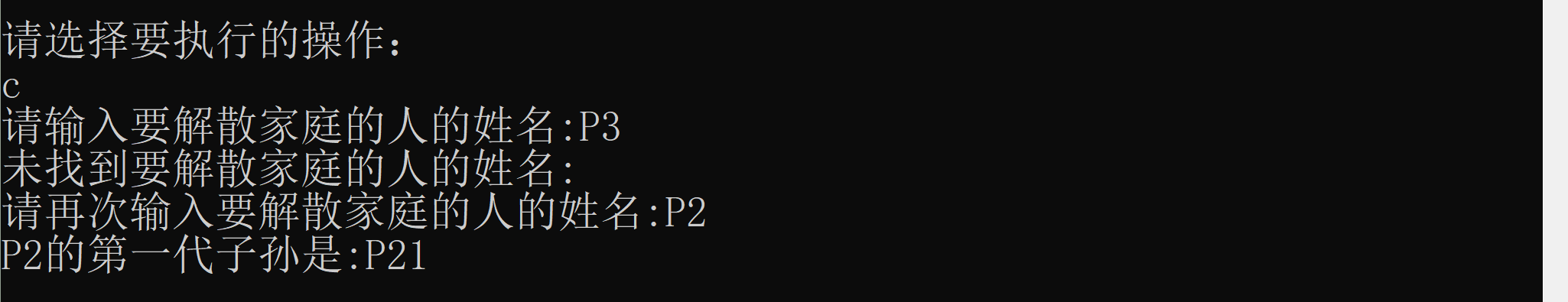
测试输入：

C P3 P2

预期输出：

错误提示+再次输入

实验结果：



## 3.11 更改家庭成员姓名时未找到成员（边界测试）

此时祖先：P0

P0子女：P1 P2

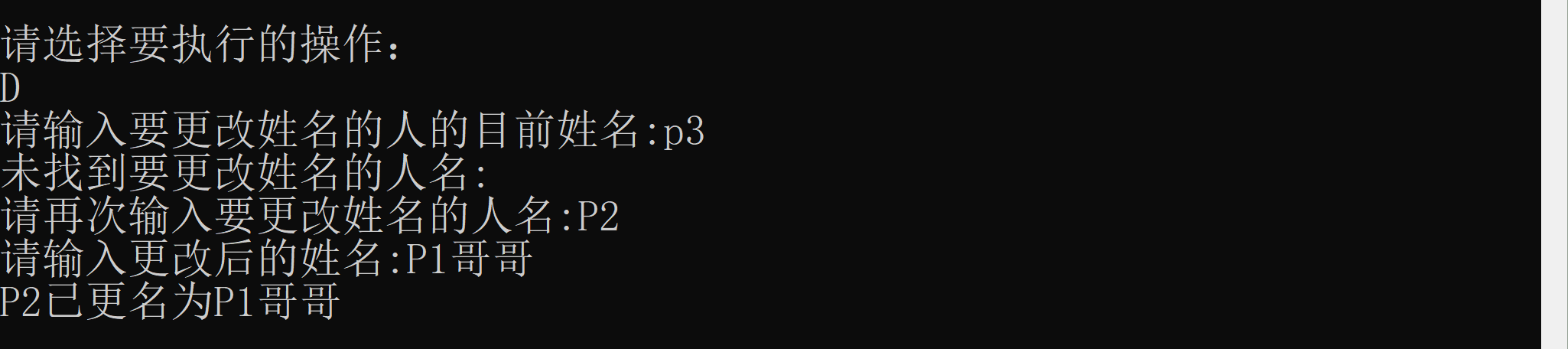
测试输入：

D p3 P1哥哥

预期输出：

错误提示+再次输入

实验结果：



## 3.12 查看儿女时未找到成员（边界测试）

此时祖先：P0

P0子女：P1 P1哥哥

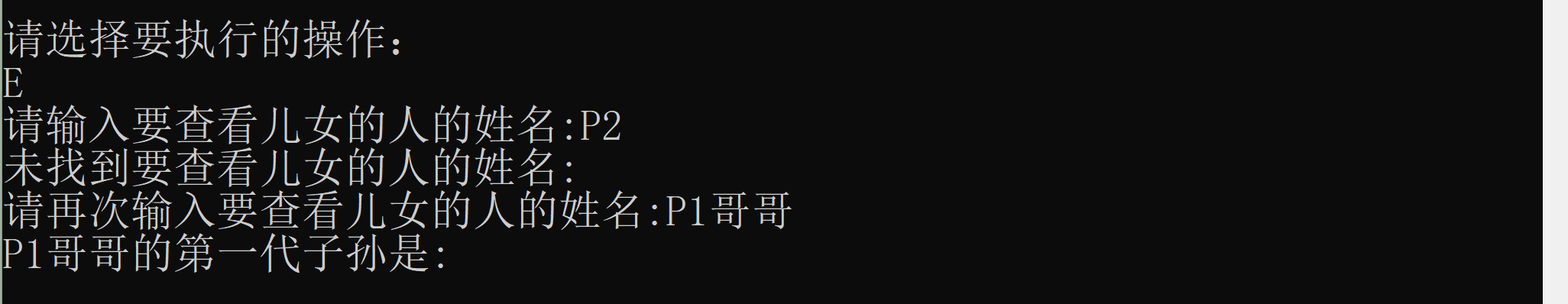
测试输入：

E P2 P1哥哥

预期输出：

错误提示+再次输入

实验结果：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，实例验证说明本项目能完成完善家谱、添加家庭成员、解散局部家庭、更改家庭成员姓名、查看家庭成员的第一代子孙5项基本功能。

## 4.2 可使用性

本算法可以很方便地使用，五种功能已封装在一个函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

每个对于树的操作都离不开查找算法，而对于一颗多叉树的查找则采用逐层遍历的广度优先算法，使得查找成功的时间复杂度为O(n)。

## 4.5 健壮性

对于选择操作时输入非法字符的情况，本项目可以完成错误提示和再次输入。并且本算法对于边界条件，诸如：建立家谱、删除成员、添加成员时找不到成员的情况都有相应的判断，并且对于链表中新结点的申请失败、输入错误的情况也有相应的错误提示。