Project documentation

Data structure course design

——Family tree management system

Author name：Xuedi Liu
Number：1752985
instructor：Ying Zhang
College/Major：School of Software Engineering/Software Engineering

1. Analysis

1.1 Project background analysis

Genealogy is a special book genre in the form of a spectrum that records a family's hereditary reproduction and important missions based on blood relationship. Genealogy is a unique cultural heritage of China. It is one of the three major Chinese literatures (national history, local history, genealogy). It is a valuable humanistic material. It is an in-depth study of history, folklore, demography, sociology and economics. It has its unique and irreplaceable features. This project performs a simple simulation of genealogy management to view the personal information of ancestors and descendants, insert family members, and delete family members.

1.2 Project function requirements

The essence of this project is to complete the establishment, search, insert, modify, delete and other functions of family members. You can first define the family member data structure, and then use each function as a member function to complete the operation of the data. Function to verify the function of each function and get the running result.

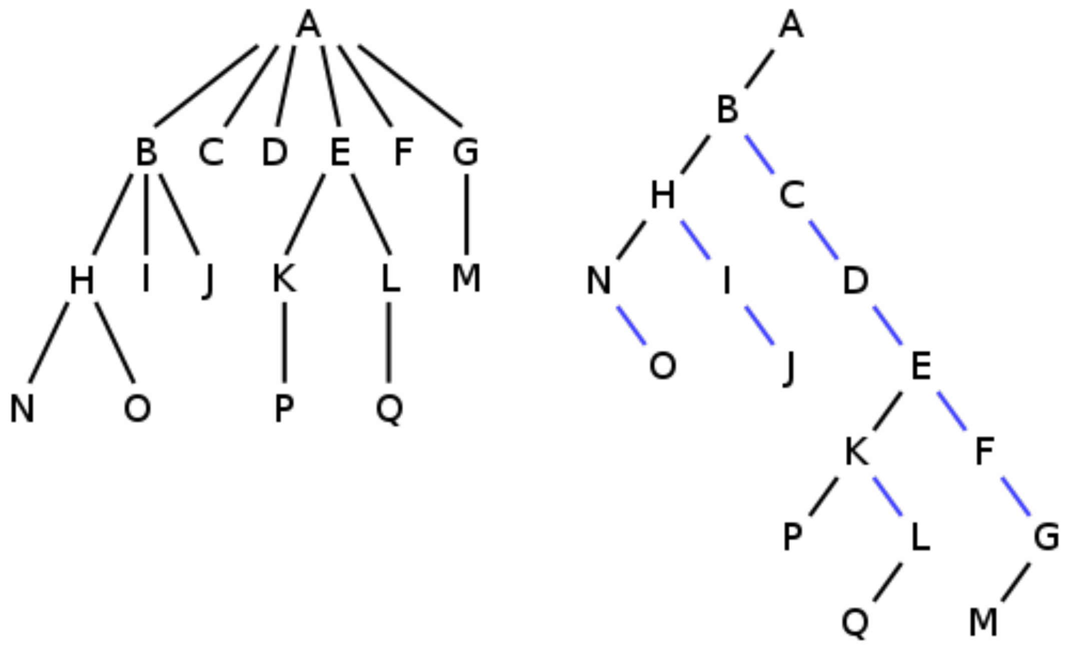
2. Design

2.1 Data structure design

In the family tree, an ancestor will breed multiple children, and his children will continue to breed, generating multiple children and two generations. Among them, the father and offspring of each generation directly form a one-to-many relationship, but one offspring cannot correspond to multiple parents. This non-linear and non-ringing relationship is a typical tree structure. We use a tree data structure to store the family relationships in the family tree.

Considering that a parent may correspond to multiple children, the number of children cannot be completely determined at this time, which will cause the structure of the tree to be diversified and it will be difficult to standardize it in a specific form, so that we are traversing , Insert, delete and other operations will encounter a lot of coding trouble.

We consider the use of the rule of "left eldest son, right brother" for a multi-tree, normalize the multi-tree to a binary tree, and construct an equivalent tree of a family tree: Left-child right-sibling binary tree.



This kind of data structure has such advantages:

1. This representation saves up memory by limiting the maximum number of references required per node to two.
2. It is easier to code.

But it also has disadvantages as follow:

Basic operations like searching/insertion/deletion tend to take a longer time because in order to find the appropriate position we would have to traverse through all the siblings of the node to be searched/inserted/deleted (in the worst case).

2.2 Class structure design

In computer science, a binary tree is a tree structure with up to two subtrees per node. Subtrees are usually called "left subtree" and "right subtree". Binary trees are often used to implement binary search trees and binary heaps. A binary tree with a depth of k and 2 ^ k-1 nodes is called a full binary tree.

As a non-linear chain storage structure that satisfies the one-to-many correspondence relationship, the implementation of the binary tree can use arrays and linked lists. Among them, since the implementation of the family tree management in this question involves many insertions and deletions, we need to perform more frequent insertions, deletions, and changes to the nodes and subtrees of the binary tree. If an array is used, the management of the array will It becomes very cumbersome, and despite the savings in space complexity, it is still very cumbersome to code. So we finally chose the structure of the chain tree to store the family tree.

The data structure of a tree is generally similar to the composition of a linked list, and is composed of a node class and a tree class that stores nodes and relationships between nodes.

2.3 Member and operational design

1. Node

```
template<typename T>
class BinNode {
private:
    T _data;//数据
    int _height;//高度
public:
    BinNode *parent, *lChild, *rChild;//父亲 左孩子，右孩子
    BinNode() : parent(nullptr), lChild(nullptr), rChild(nullptr), _height(0) {}

    BinNode(T data, BinNode<T> *p = nullptr, BinNode<T> *lc = nullptr,
            BinNode<T> *rc = nullptr, int h = 0) :
            _data(data), parent(p), lChild(lc), rChild(rc), _height(h) {}

    int size();//以当前节点为根节点的后代规模
    T data() { return this->_data; }//返回该节点的数据
    void setData(T data1) { this->_data = data1; }

    BinNode *insertAsLC(T t);

    BinNode *insertAsRC(T t);
};
```

Core function:

Insert a new node as left/right child:

```
template<typename T>
BinNode<T> *BinNode<T>::insertAsLC(T t) {
    return this->lChild = new BinNode(t, this);
}

template<typename T>
BinNode<T> *BinNode<T>::insertAsRC(T t) {
    return this->rChild = new BinNode(t, this);
}
```

2. BinTree

Private members:

```
private:
    int _size;//树的规模
    BinNode<T> *_root;//根节点
```

Public operation:

```
public:
    BinTree() : _size(0), _root(nullptr) {}

    ~BinTree() { if (0 < _size) remove(_root); }

    int size() { return this->_size; }

    bool empty() { return !_size; }

    BinNode<T> *root() { return _root; }

    BinNode<T> *insertAsRoot(T t);//作为跟节点插入（初始化
    BinNode<T> *insertAsLC(BinNode<T> *x, T t);//作为x节点的左孩子插入，data=t
    BinNode<T> *insertAsRC(BinNode<T> *x, T t);//作为x节点的右孩子插入，data=t
    BinNode<T> *search(T t);//查找内容为t的某个元素(使用前序遍历)

    std::queue<BinNode<T> *> showChild(BinNode<T> *x);//输出这个节点的所有子一代,并用队
列存储所有子一代
    void remove(BinNode<T> *p);//解散data为t的子🌲及他自身
```

Core function:

Insert subtree as root node:

```
template<typename T>
BinNode<T> *BinTree<T>::insertAsRoot(T t) {
    this->_root = new BinNode<T>(t);
    _size++;
}
```

Insert node as left/right child:

```
template<typename T>
BinNode<T> *BinTree<T>::insertAsLC(BinNode<T> *x, T t) {
    x->insertAsLC(t);
    _size++;
}

template<typename T>
BinNode<T> *BinTree<T>::insertAsRC(BinNode<T> *x, T t) {
    x->insertAsRC(t);
    _size++;
}
```

Find the position of the node in the binary tree:

```
template<typename T>
BinNode<T> *BinTree<T>::search(T t) {
    std::stack<BinNode<T> *> s;
    auto p = this->_root;
    s.push(p);
    while (!s.empty() || p != nullptr) {
        if (p->data() == t) {
            return p;
        } else {
            s.push(p);
            p = p->lChild;
            while (p == nullptr) {
                p = s.top();
                p = p->rChild;
                s.pop();
            }
        }

    }
}
```

Show sub-generation of this node:

```
template<typename T>
std::queue<BinNode<T> *> BinTree<T>::showChild(BinNode<T> *x) {
    std::queue<BinNode<T> *> child;
    if (x->lChild != nullptr) {
        std::cout << x->lChild->data() << "  ";
        child.push(x->lChild);
    } else {
        std::cout << "他没有子代" << std::endl;
    }
    while (x->rChild != nullptr) {
        child.push(x->rChild);
        std::cout << x->rChild->data() << "  ";
```

```
            x = x->rChild;
        }
        std::cout << std::endl;
        return child;
    }
```

Delete the node and the child tree:

```
template<typename T>
void BinTree<T>::remove(BinNode<T> *p) {
    if (p == nullptr) {
        return;
    }
    if (p->lChild != nullptr) {
        remove(p->lChild);
    }
    if (p->rChild != nullptr) {
        remove(p->rChild);
    }
    delete p;
}
```

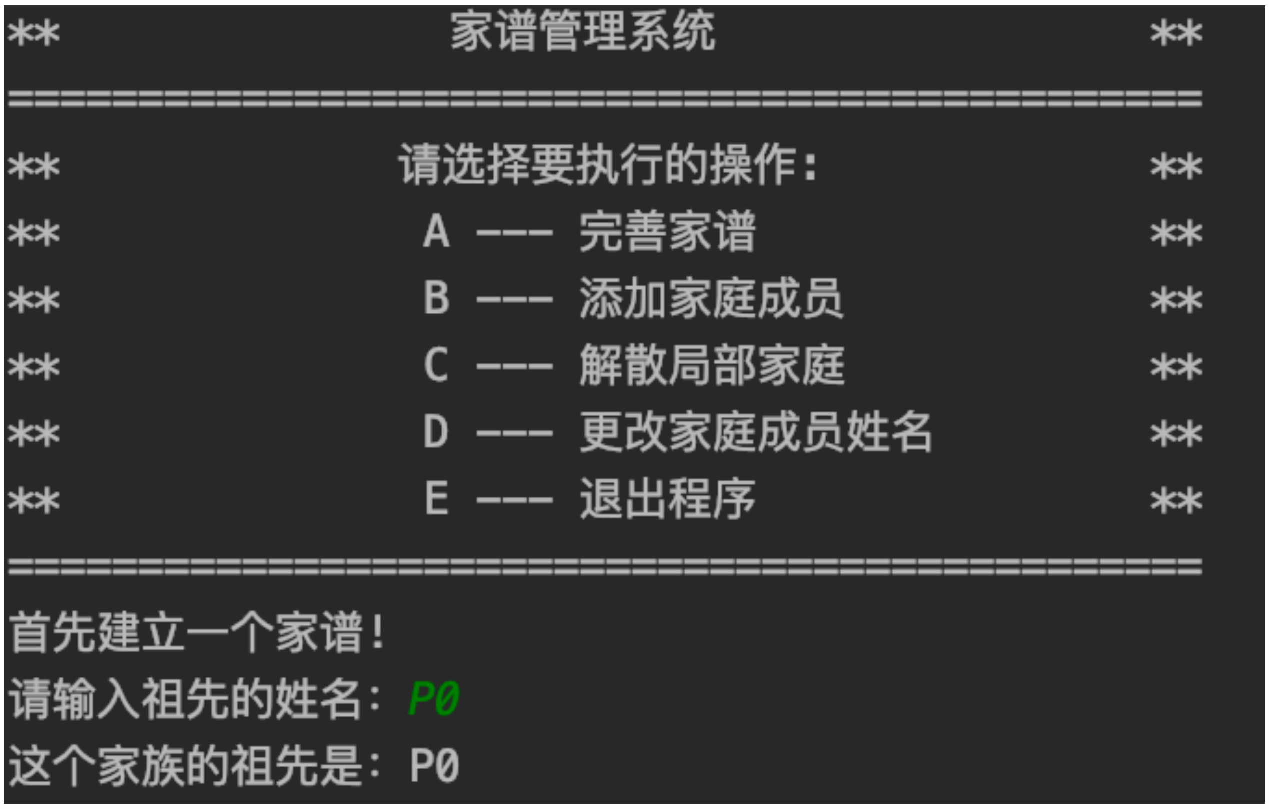
3. Realization

3.1 *Init a family tree*

1. Init core code

```
BinTree<string> *initFamily() {
    cout << "首先建立一个家谱!" << endl;
    cout << "请输入祖先的姓名: ";
    string ancestor;
    cin >> ancestor;
    auto tree = new BinTree<string>;
    tree->insertAsRoot(ancestor);
    cout << "这个家族的祖先是: " << tree->root()->data() << endl;
    return tree;
}
```

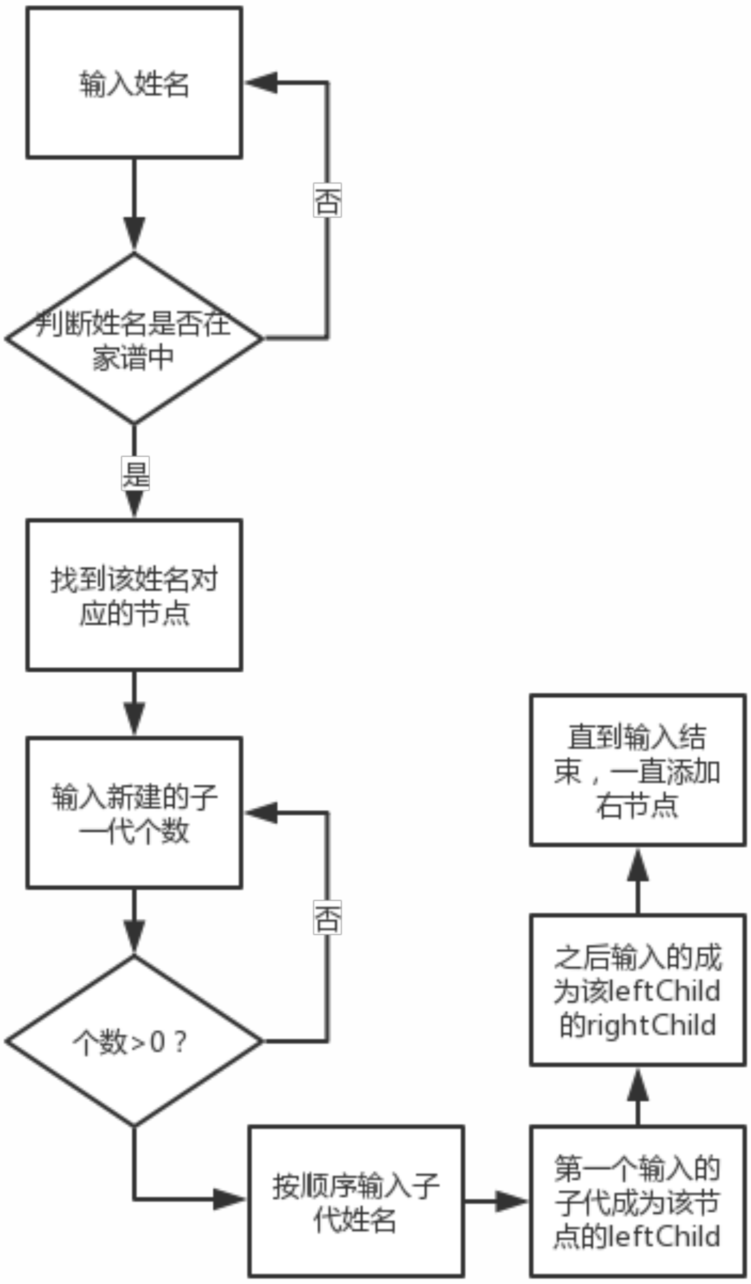
3. Init result demonstration



```
**                  家谱管理系统                  **
==================================================
**               请选择要执行的操作:               **
**                A --- 完善家谱                  **
**                B --- 添加家庭成员              **
**                C --- 解散局部家庭              **
**                D --- 更改家庭成员姓名          **
**                E --- 退出程序                  **
==================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
```

3.2 Complete family tree

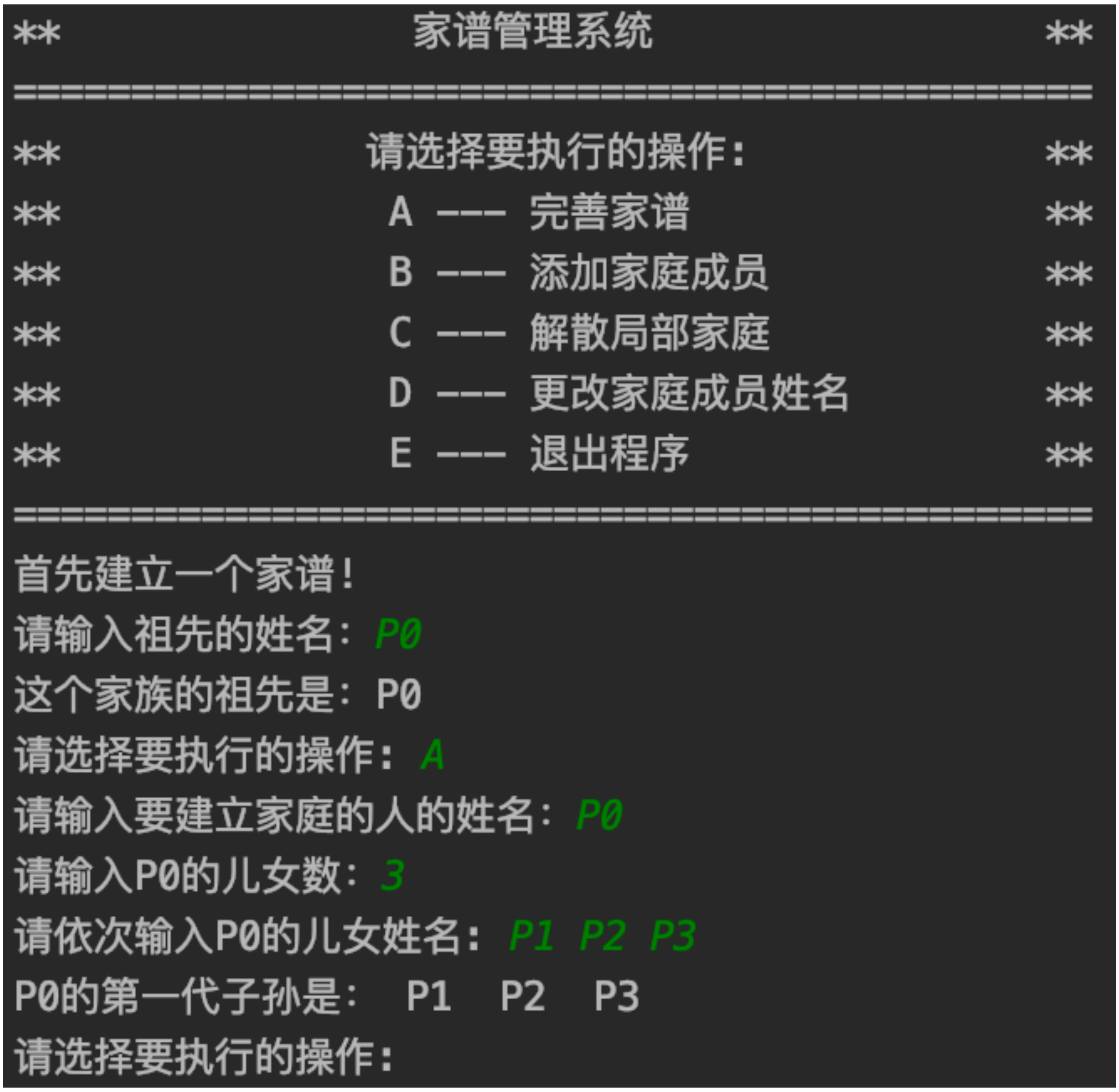
1. Complete flowchart and description


输入姓名
判断姓名是否在家谱中
否
是
找到该姓名对应的节点
输入新建的子一代个数
个数>0？
否
按顺序输入子代姓名
第一个输入的子代成为该节点的leftChild
之后输入的成为该leftChild的rightChild
直到输入结束，一直添加右节点

2. Complete function core code

```
void completeFamily(BinTree<string> *tree) {
    cout << "请输入要建立家庭的人的姓名: ";
    string name;
    cin >> name;
    auto p = tree->search(name);
    while (p == nullptr) {
        cout << "您输入的姓名不在家谱中,请您重新输入: ";
        cin >> name;
        p = tree->search(name);
    }
    cout << "请输入" << name << "的儿女数: ";
    int num;
    cin >> num;
    auto p0 = p;//记录该节点位置
    while (num <= 0) {
        cout << "您输入的数字有误, 请重新输入" << endl;
        cin >> num;
    }
    cout << "请依次输入" << name << "的儿女姓名: ";
    string child;
    cin >> child;
    p->insertAsLC(child);
    num--;
    while (num--) {
        cin >> child;
        p->insertAsRC(child);
        p = p->rChild;
    }
    cout << name << "的第一代子孙是:  ";
    tree->showChild(p0);
}
```

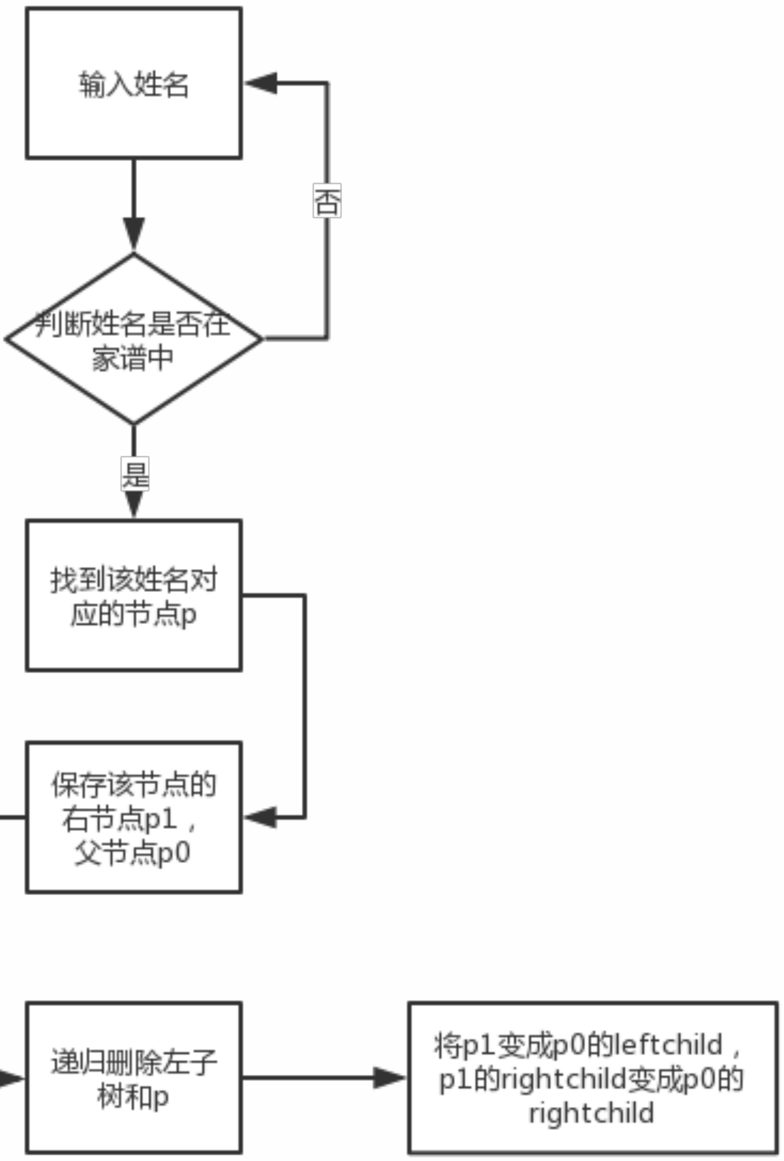
3. Complete operation diagram



```
**                    家谱管理系统                    **
====================================================
**                  请选择要执行的操作:                 **
**                   A --- 完善家谱                    **
**                   B --- 添加家庭成员                  **
**                   C --- 解散局部家庭                  **
**                   D --- 更改家庭成员姓名                **
**                   E --- 退出程序                    **
====================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1  P2  P3
请选择要执行的操作:
```

3.3 Implementation of dissolution of local families

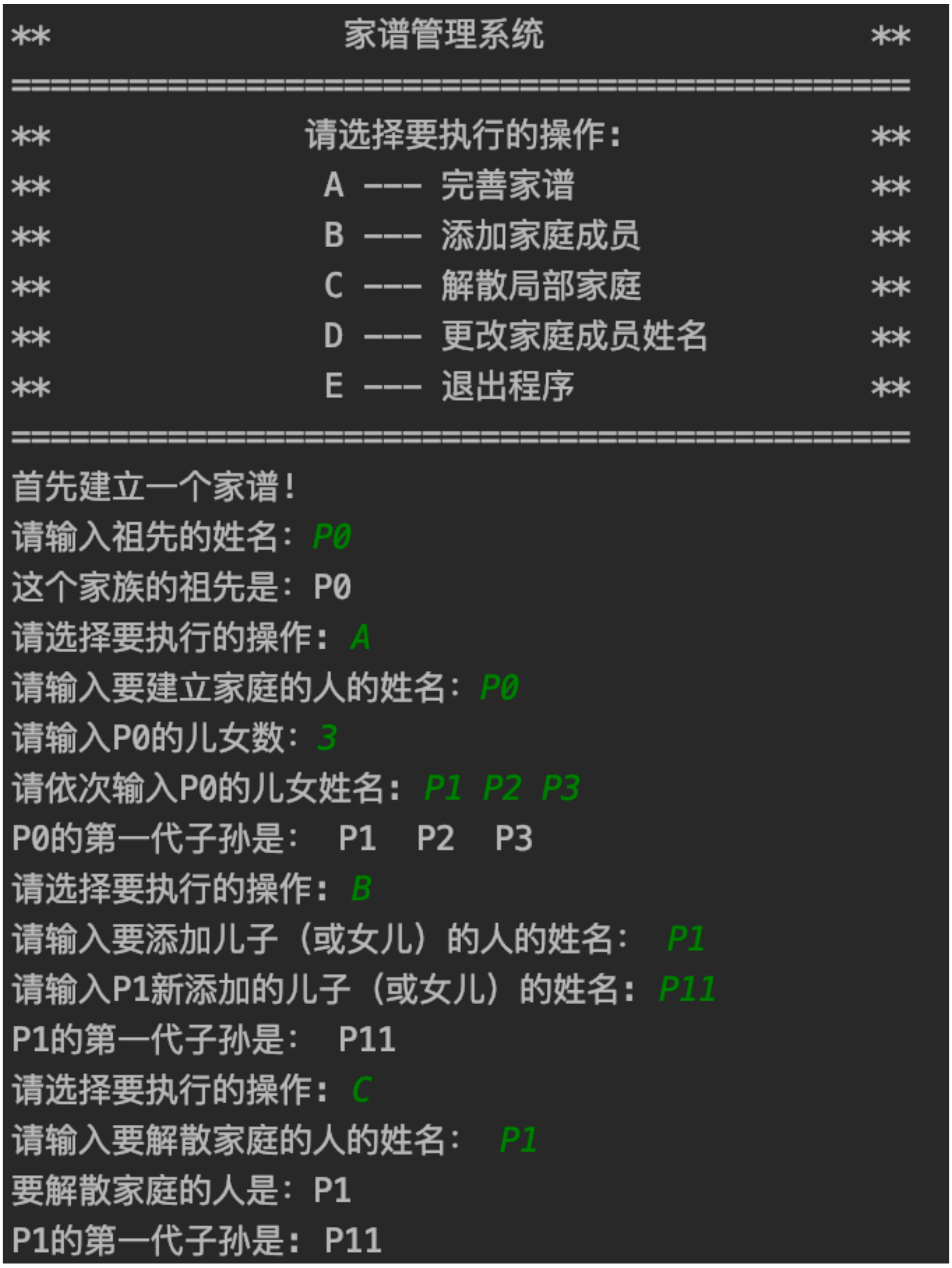
1. Dissolution operation flowchart


输入姓名
否
判断姓名是否在家谱中
是
找到该姓名对应的节点p
保存该节点的右节点p1，父节点p0
递归删除左子树和p
将p1变成p0的leftchild，p1的rightchild变成p0的rightchild

2. Core code of dissolution operation

```
void disbandLocalFamily(BinTree<string> *tree) {
    cout << "请输入要解散家庭的人的姓名：  ";
    string name;
    cin >> name;
    while(tree->search(name) == nullptr){
        cout << "您输入的姓名不在家谱中,请您重新输入: ";
        cin >> name;
    }
    cout << "要解散家庭的人是: " << name << endl;
    cout << name << "的第一代子孙是: ";
    auto p = tree->search(name);
    tree->showChild(p);
    tree->remove(p);
}
```

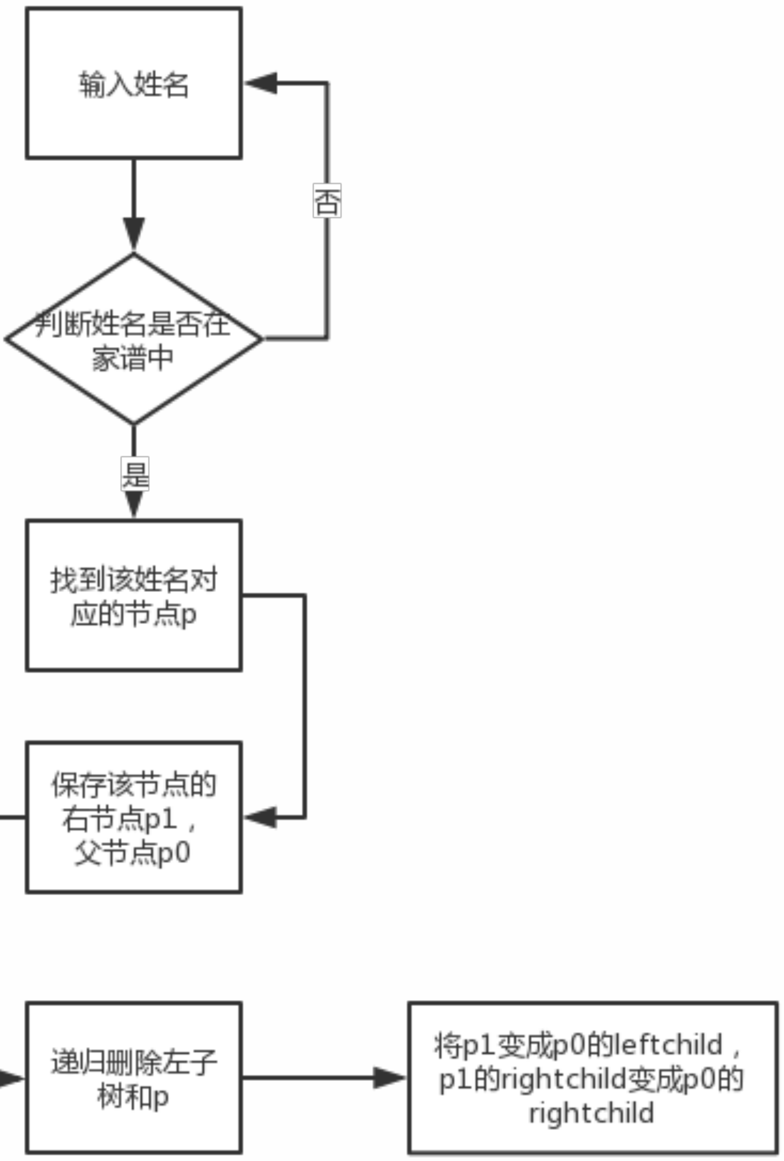
3. Evaluation diagram



```
**                      家谱管理系统                      **
==========================================================
**                  请选择要执行的操作:                   **
**                   A --- 完善家谱                       **
**                   B --- 添加家庭成员                   **
**                   C --- 解散局部家庭                   **
**                   D --- 更改家庭成员姓名               **
**                   E --- 退出程序                       **
==========================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1   P2   P3
请选择要执行的操作: B
请输入要添加儿子(或女儿)的人的姓名:  P1
请输入P1新添加的儿子(或女儿)的姓名: P11
P1的第一代子孙是:  P11
请选择要执行的操作: C
请输入要解散家庭的人的姓名:  P1
要解散家庭的人是: P1
P1的第一代子孙是: P11
```

3.4 Implementation of changing the name of a family member

1. Rename operation flowchart

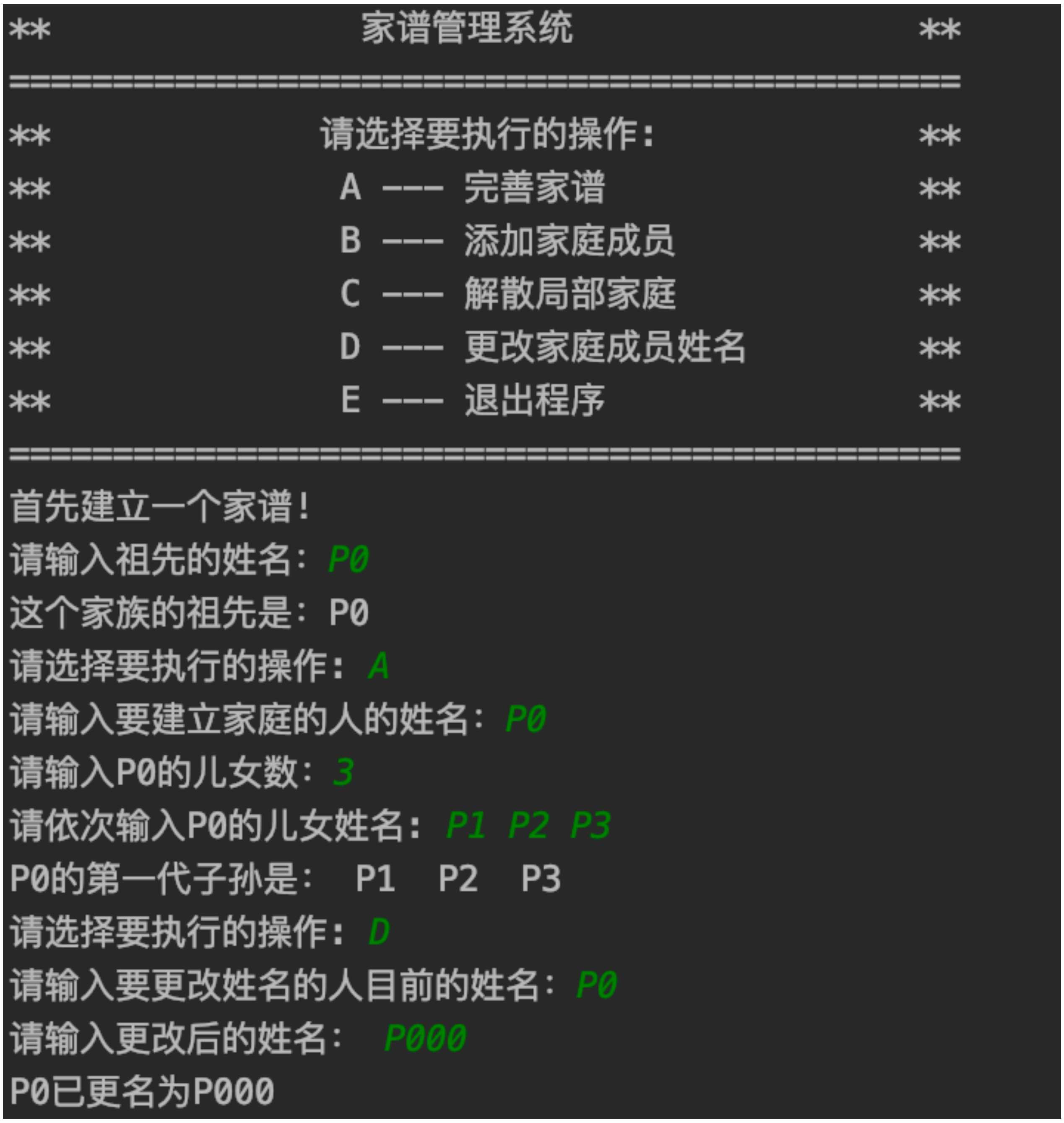

输入姓名
否
判断姓名是否在家谱中
是
找到该姓名对应的节点p
保存该节点的右节点p1，父节点p0
递归删除左子树和p
将p1变成p0的leftchild，p1的rightchild变成p0的rightchild

2. Rename operation core code

```
void changeName(BinTree<string> *tree) {
    cout << "请输入要更改姓名的人目前的姓名：";
```

```
        string formerName, name;
        cin >> formerName;
        auto p = tree->search(formerName);
        while(p == nullptr){
            cout << "您输入的姓名不在家谱中,请您重新输入: ";
            cin >> formerName;
            p = tree->search(formerName);
        }
        cout << "请输入更改后的姓名:  ";
        cin >> name;
        p->setData(name);
        cout << formerName << "已更名为" << name << endl;
    }
```

3. Rename operation diagram



```
**                       家谱管理系统                        **
=========================================================
**                    请选择要执行的操作:                     **
**                     A --- 完善家谱                        **
**                     B --- 添加家庭成员                     **
**                     C --- 解散局部家庭                     **
**                     D --- 更改家庭成员姓名                  **
**                     E --- 退出程序                        **
=========================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1  P2  P3
请选择要执行的操作: D
请输入要更改姓名的人目前的姓名: P0
请输入更改后的姓名:  P000
P0已更名为P000
```

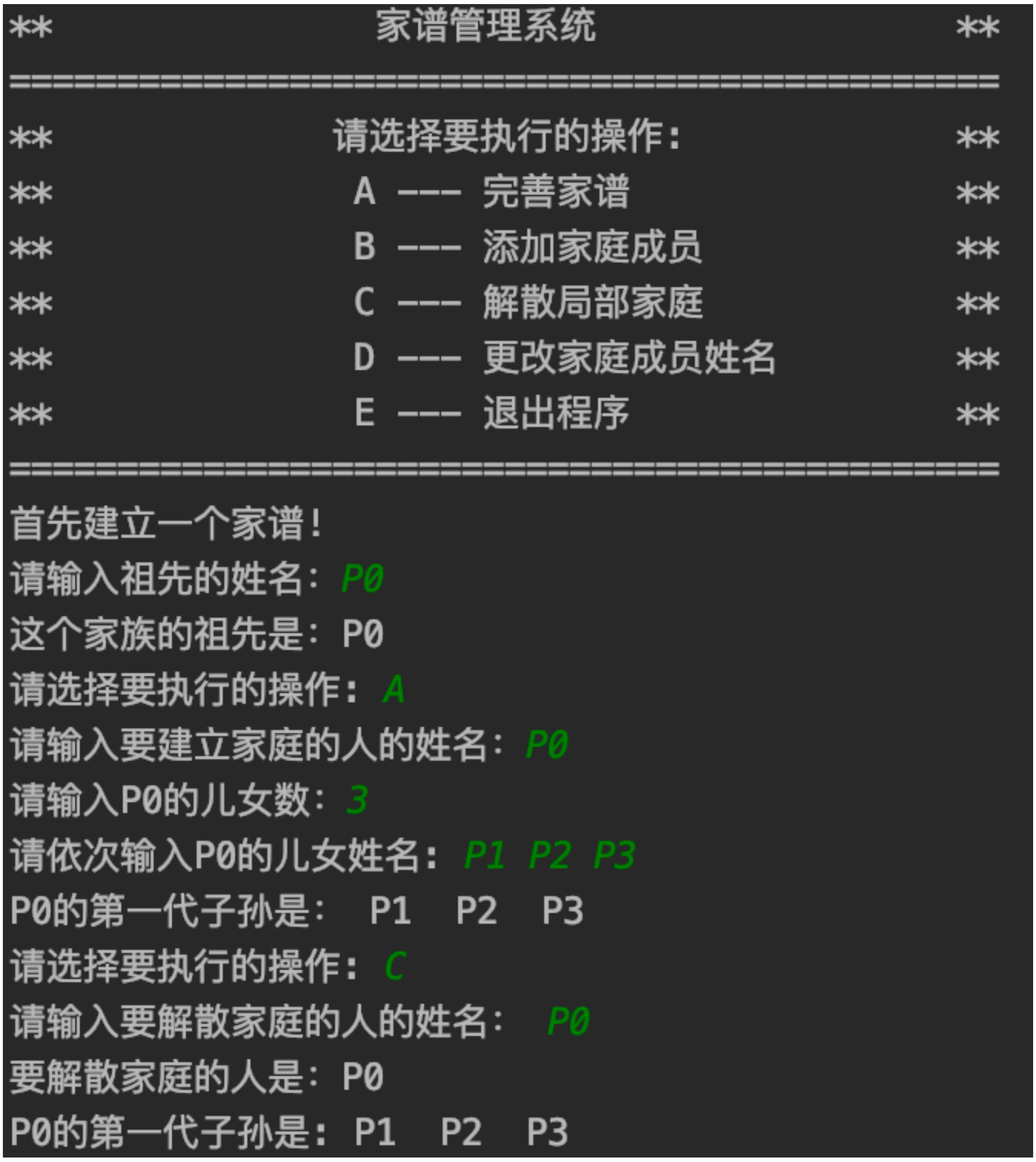
4. Test

4.1 Boundary test

1. Family tree becomes an empty tree after deletion

Experimental result:

All child nodes are successfully deleted.

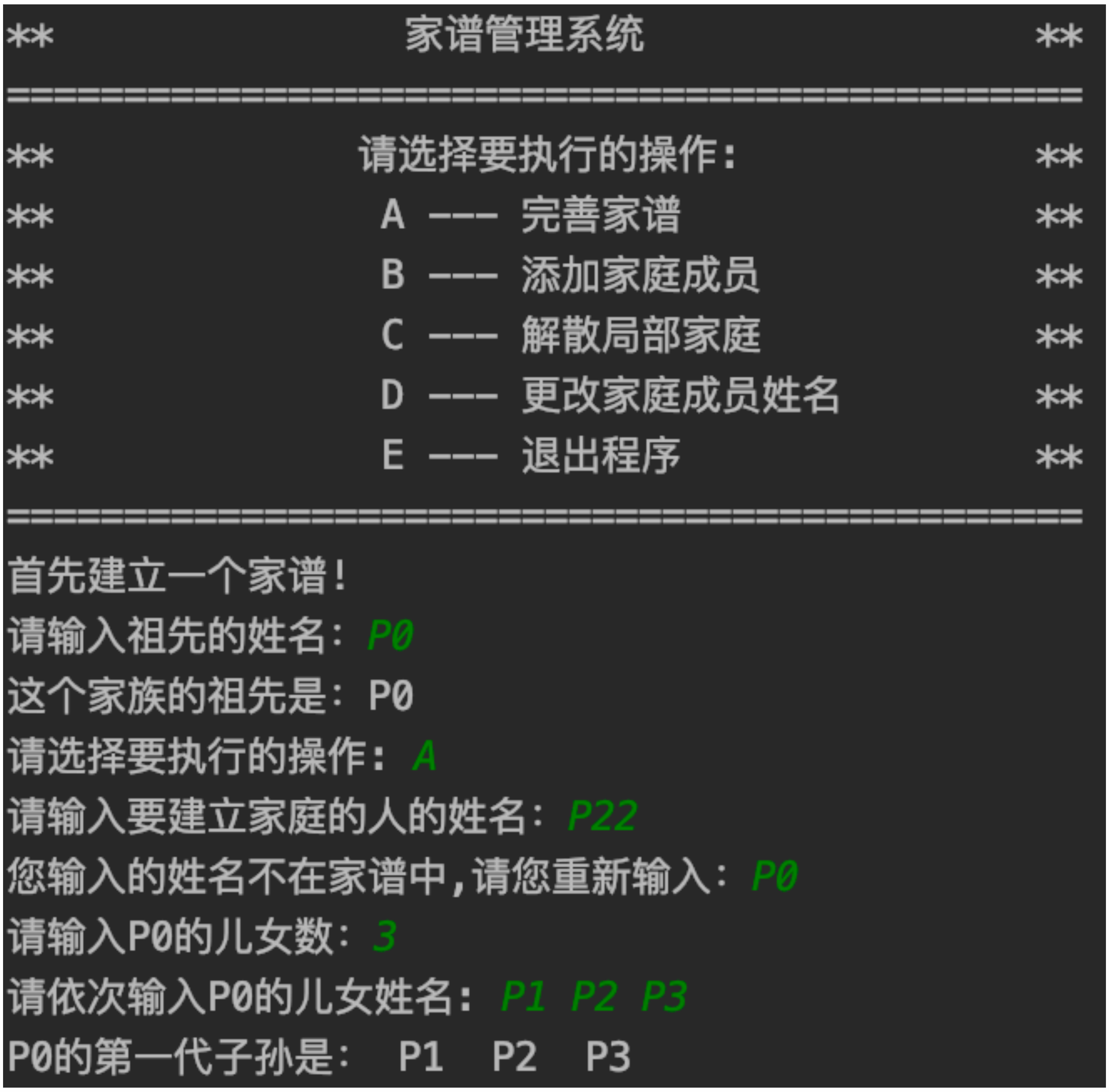


```
**                    家谱管理系统                      **
========================================================
**                  请选择要执行的操作:                 **
**                   A --- 完善家谱                     **
**                   B --- 添加家庭成员                 **
**                   C --- 解散局部家庭                 **
**                   D --- 更改家庭成员姓名             **
**                   E --- 退出程序                     **
========================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1  P2  P3
请选择要执行的操作: C
请输入要解散家庭的人的姓名:  P0
要解散家庭的人是: P0
P0的第一代子孙是: P1  P2  P3
```

4.3 Error test

1. Wrong name entered when creating family tree

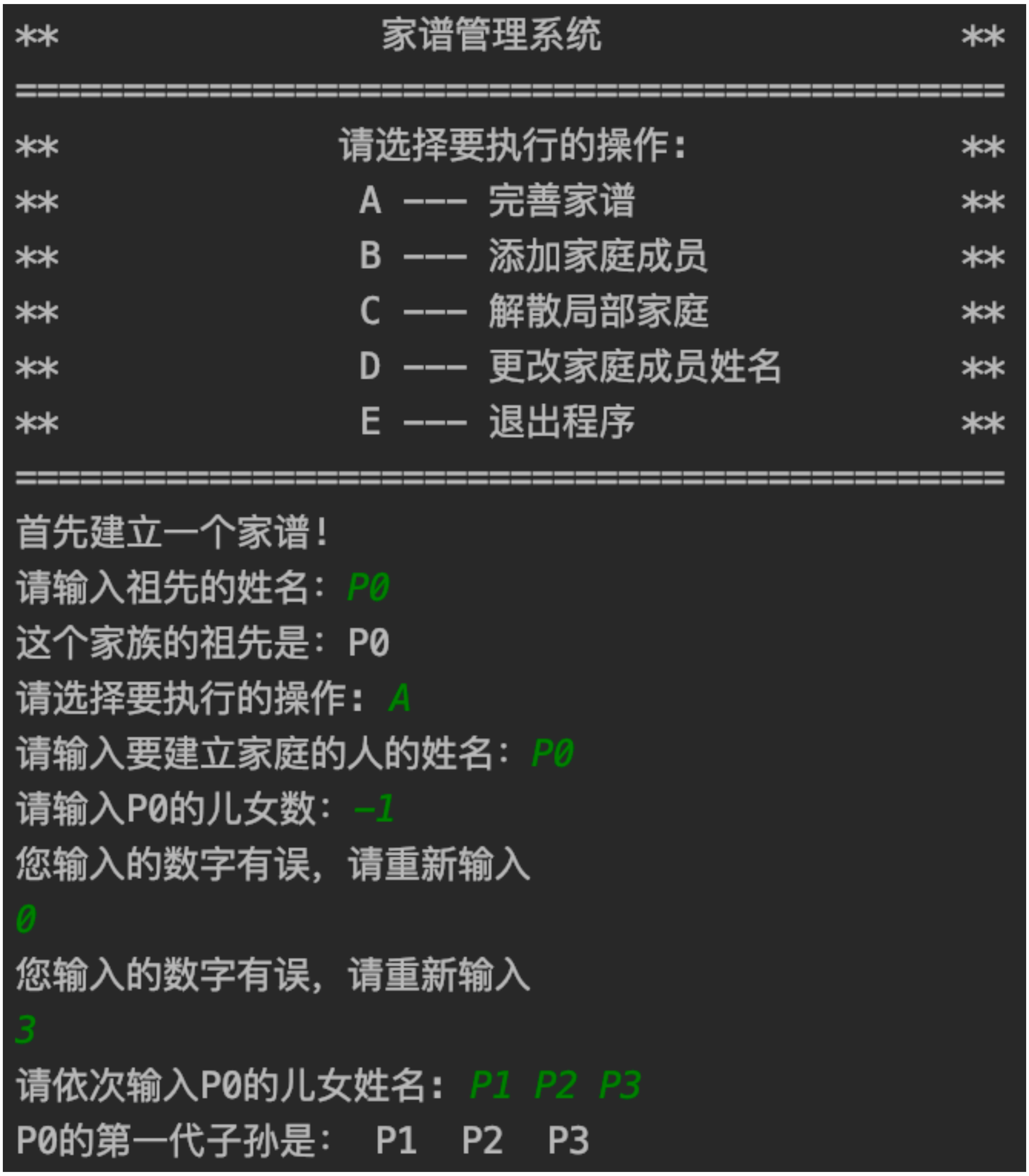
Experimental result:



```
**                    家谱管理系统                    **
======================================================
**                 请选择要执行的操作:                 **
**                  A --- 完善家谱                    **
**                  B --- 添加家庭成员                 **
**                  C --- 解散局部家庭                 **
**                  D --- 更改家庭成员姓名              **
**                  E --- 退出程序                    **
======================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P22
您输入的姓名不在家谱中,请您重新输入: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1  P2  P3
```

2. The number of descendants entered is not positive when creating a family tree

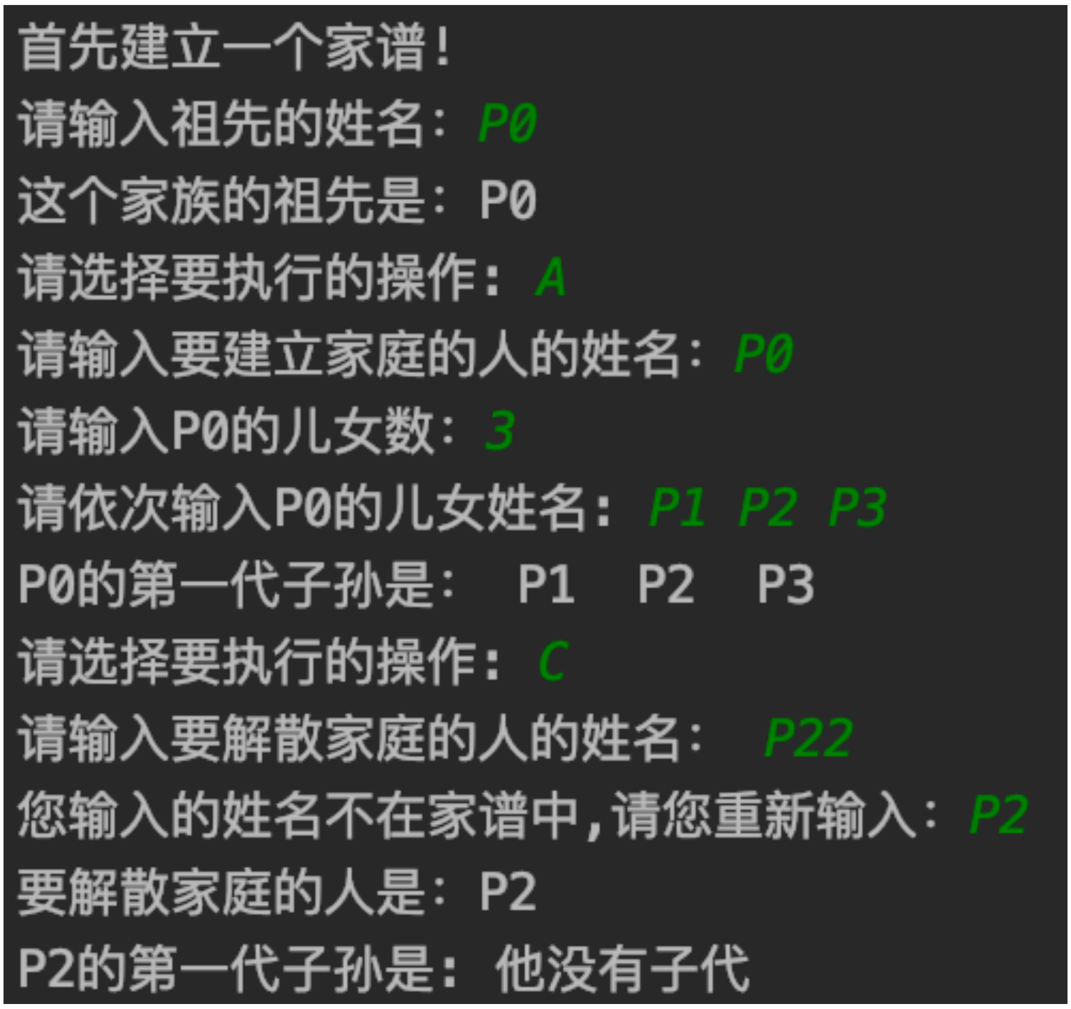
Experimental result:



```
**                      家谱管理系统                      **
==========================================================
**                    请选择要执行的操作:                  **
**                     A --- 完善家谱                     **
**                     B --- 添加家庭成员                  **
**                     C --- 解散局部家庭                  **
**                     D --- 更改家庭成员姓名              **
**                     E --- 退出程序                     **
==========================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: -1
您输入的数字有误, 请重新输入
0
您输入的数字有误, 请重新输入
3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1  P2  P3
```

3. The name entered when the family was dissolved does not exist

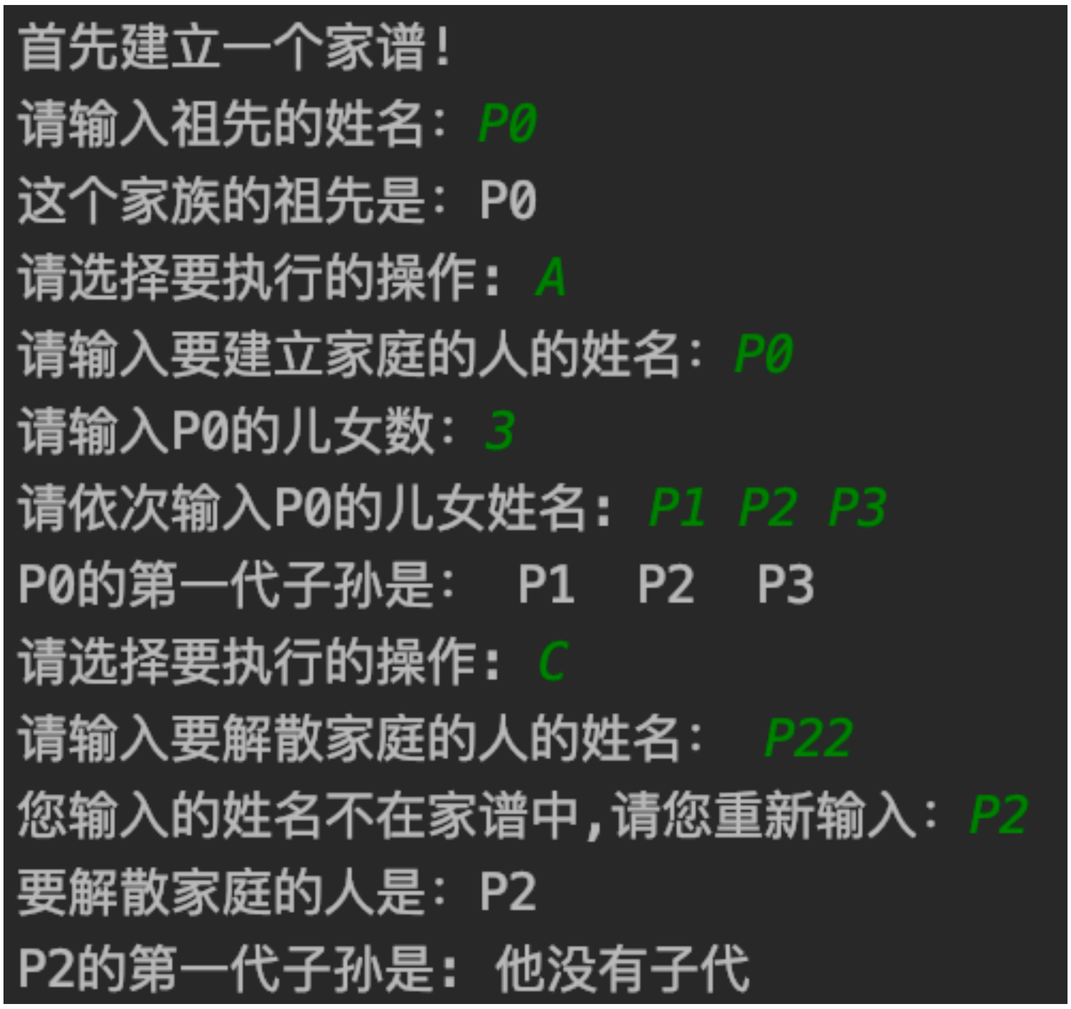
Experimental result:



```
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1   P2   P3
请选择要执行的操作: C
请输入要解散家庭的人的姓名:  P22
您输入的姓名不在家谱中,请您重新输入: P2
要解散家庭的人是: P2
P2的第一代子孙是: 他没有子代
```

4. Dismissed family member has no children

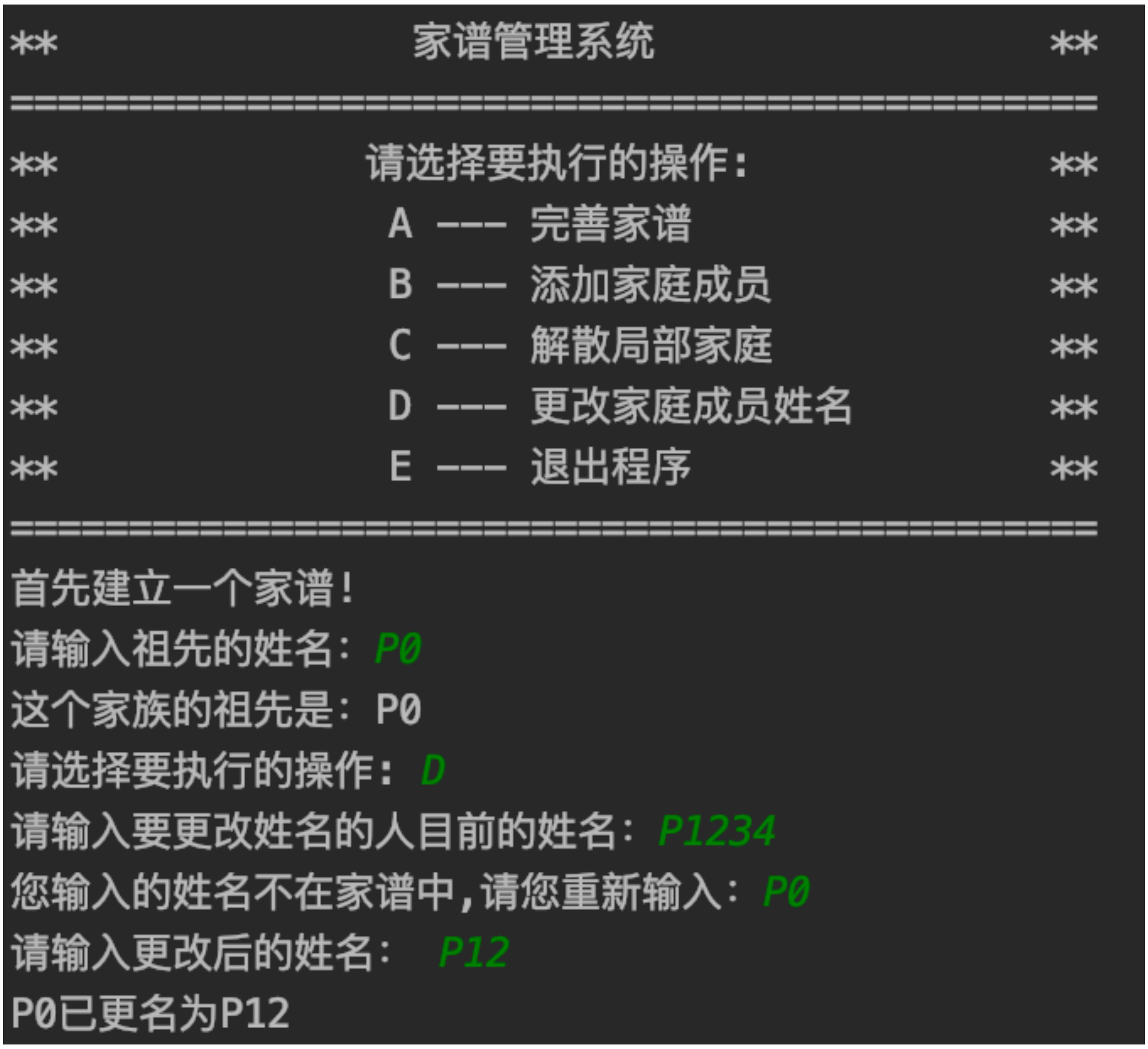
Experimental result:



```
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: A
请输入要建立家庭的人的姓名: P0
请输入P0的儿女数: 3
请依次输入P0的儿女姓名: P1 P2 P3
P0的第一代子孙是:  P1   P2   P3
请选择要执行的操作: C
请输入要解散家庭的人的姓名:  P22
您输入的姓名不在家谱中,请您重新输入: P2
要解散家庭的人是: P2
P2的第一代子孙是: 他没有子代
```

5. The member who changed the name is not in the family tree

Experimental result：



```
**                   家谱管理系统                 **
===================================================
**               请选择要执行的操作:               **
**                A --- 完善家谱                  **
**                B --- 添加家庭成员               **
**                C --- 解散局部家庭               **
**                D --- 更改家庭成员姓名            **
**                E --- 退出程序                  **
===================================================
首先建立一个家谱!
请输入祖先的姓名: P0
这个家族的祖先是: P0
请选择要执行的操作: D
请输入要更改姓名的人目前的姓名: P1234
您输入的姓名不在家谱中,请您重新输入: P0
请输入更改后的姓名:  P12
P0已更名为P12
```