

# 数据结构项目文档

---

题目：家谱管理系统

指导教师：张颖

姓名：王亮

学号：1653340

---

## 数据结构项目文档

### 一、题目分析

1. 项目简介
2. 功能需求
3. 设计思路

### 二、设计

1. 数据结构设计
2. 类的设计
  - 2.1 TreeNode
  - 2.2 Tree
  - 2.3 SeqQueue
3. 函数设计
  - 3.1 建立家谱
  - 3.2 菜单功能循环

### 三、实现

1. 创建家族系统
  - 1.1 相关代码
  - 1.2 运行截图
2. 菜单功能循环
  - 2.1 流程图
  - 2.2 相关代码
3. 完善家谱
  - 3.1 流程图
  - 3.2 相关代码
  - 3.3 运行截图
4. 添加家族成员
  - 4.1 流程图
  - 4.2 相关代码
  - 4.3 运行截图
5. 解散局部家庭
  - 5.1 流程图
  - 5.2 相关代码
  - 5.3 运行截图
6. 更改家庭成员姓名
  - 6.1 流程图
  - 6.2 相关代码

- 6.3 运行截图
- 7. 显示完整家谱（创新功能）
  - 7.1 相关代码
  - 7.2 运行截图
- 四、测试
  - 1. 功能测试
  - 2. 出错测试
    - 3.1 建立家庭的人不存在
    - 3.2 建立家庭的人已经拥有家庭
    - 3.3 建立家庭时儿女人数范围有误
    - 3.4 添加儿女、解散家庭、更改姓名的人不存在

## 一、题目分析

---

### 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。

### 2. 功能需求

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。

### 3. 设计思路

首先确定项目采用树作为数据结构，定义类的成员变量和成员函数；然后实现家谱管理系统需要的函数；最后完成主函数以验证程序的功能并得到运行结果。

## 二、设计

---

### 1. 数据结构设计

家谱中存在家庭成员之间各种个样的关系，比如：兄弟、父母、子女、祖先等等。树结构可以较好地实现这样的家族成员关系，表示一个家族从祖先到后辈之间的继承关系，因此选择树作为实现本系统的基本数据结构。

为了更加完整地实现一个树的模板类，我还实现了树的广度优先遍历函数，这个函数主要依赖队列（具体算法不展开叙述），因此直接使用了前面题目中实现的顺序队列。

因此，本项目中共引入了树和队列两个数据结构。

## 2. 类的设计

多叉树有多种存储表示形式，例如：广义表表示法、双亲数组表示法（父指针表示法）、左子女右兄弟表示法（子女-兄弟链表表示法）等。

在这个题目中，我选择了左子女-右兄弟表示法实现了链表。这种表示形式的链表需要两个类：TreeNode类和Tree类，协同表示树。

### 2.1 TreeNode

包含一个数据域，存储家庭成员姓名（string）。

指针域包括两个指针，分别指向第一个子女结点，和下一个兄弟结点。

```
template <typename T>struct TreeNode{
    T data;
    TreeNode<T> *firstChild, *nextSibling;
    TreeNode(T value = 0, TreeNode<T> *fc = NULL, TreeNode<T> *ns = NULL)
        :firstChild(fc), nextSibling(ns){
        data = value;
    }
};
```

### 2.2 Tree

实现了一个功能比较完整的Tree模板类。包括对子女和兄弟的访问、三种顺序的遍历方式、各种对结点的操作、查找、删除等功能。

包括三个成员数据：根结点指针、当前结点指针、数据结束的标志数据。

```
template <typename T>class Tree{
public:
    Tree(){
        root = current = NULL;
    }
    Tree(T value){
        root = current = NULL;
        RefValue = value;
    }
    bool Root(); //置根结点为当前结点
    TreeNode<T> *getRoot() const{
        return root;
    }
    void setRoot(TreeNode<T> *rt){
        root = rt;
    }
    TreeNode<T> *getCurrent() const{
        return current;
    }
    bool IsEmpty(){
```

```

        return root == NULL;
    }
    bool FirstChild();           //将当前结点的第一个子女置为当前结点
    bool NextSibling();         //将当前结点的下一个兄弟置为当前结点
    bool Parent();              //将当前结点的双亲置为当前结点
    bool Find(T value);         //搜索含value的结点，使之成为当前结点

    void PreOrder(void (*visit)(TreeNode<T> *t)){//前序
        TreeNode<T> *p = root;
        while (p){
            PreOrder(p, visit);
            p = p->nextSibling;
        }
    }
    void PostOrder(void (*visit)(TreeNode<T> *t)){//后序
        TreeNode<T> *p = root;
        while (p){
            PostOrder(p, visit);
            p = p->nextSibling;
        }
    }
    void LevelOrder(void (*visit)(TreeNode<T> *t)){//层次
        LevelOrder(root, visit);
    }

    //以先根次序输入树;以RefValue表示某节点没有子女或不再有兄弟
    void CreateTree(istream &in = cin){
        CreateTree(in, root);
    }

    //给current指向的结点添加有n个孩子结点的子树
    bool CreateSubTree(int n);

    //依次显示某个节点的第一代儿子节点
    void showChildren(TreeNode<T>* p);

    //给p指向的结点的儿子节点后面新增一个值为data的儿子节点
    void addChild(TreeNode<T>* p, T data);

    //通过递归删除结点p的所有子孙结点
    void removeAllDescendants(TreeNode<T>* p);

    void changeData(TreeNode<T>* p, T newData)
    {
        p->data = newData;
    }

    void Output(ostream &out = cout){           //树状格式输出
        Output(root, string(), out);
    }

```

```

    }
    void IntendedText(ostream &out = cout){    //将树用缩格文本形式输出
        IntendedText(root, string(), out); //string()是构造函数
    }
    void ShowTree(ostream &out = cout){
        ShowTree(root, out);
    }
    void reSetCurrent(){    //将current重新指向根节点
        current = root;
    }

    friend istream& operator >> (istream &in, Tree<T> &tree); //tree不可误作
Tree
    friend ostream& operator << (ostream &out, Tree<T> &tree); //因为类模板的
友元函数是函数模板，必须在此声明，在类外定义
private:
    TreeNode<T> *root, *current;
    T RefValue;    //数据输入停止标志

    bool Find(TreeNode<T> *p, T value);    //在以p为根的树中搜索value
    void RemovesubTree(TreeNode<T> *p);    //删除以p为根的子树
    bool FindParent(TreeNode<T> *t, TreeNode<T> *p);
    void PreOrder(TreeNode<T> *subTree, void (*visit)(TreeNode<T> *t));
    void PostOrder(TreeNode<T> *subTree, void (*visit)(TreeNode<T> *t));
    void LevelOrder(TreeNode<T> *subTree, void (*visit)(TreeNode<T> *t));
    void Output(TreeNode<T> *subTree, string str, ostream &out);
    void IntendedText(TreeNode<T> *subTree, string str, ostream &out);
    void ShowTree(TreeNode<T> *t, ostream &out);
    void CreateTree(istream& in, TreeNode<T> *& subTree); //以先根次序输入树;以
RefValue表示某节点没有子女或不再有兄弟
};

```

## 2.3 SeqQueue

为了实现一个完整的树的模板类，我还实现了树的广度优先遍历函数（本题目没有用到）。实现树的广度优先遍历需要一个队列，这里直接调用了前面题目实现的顺序队列

## 3. 函数设计

### 3.1 建立家谱

建立一个家谱，需要用户输入祖先的姓名，将其设置为一个空家族树的根结点。

### 3.2 菜单功能循环

显示一个功能菜单，通过循环实现不断接受用户指令，并执行相应的操作。

系统提供的功能菜单：

```

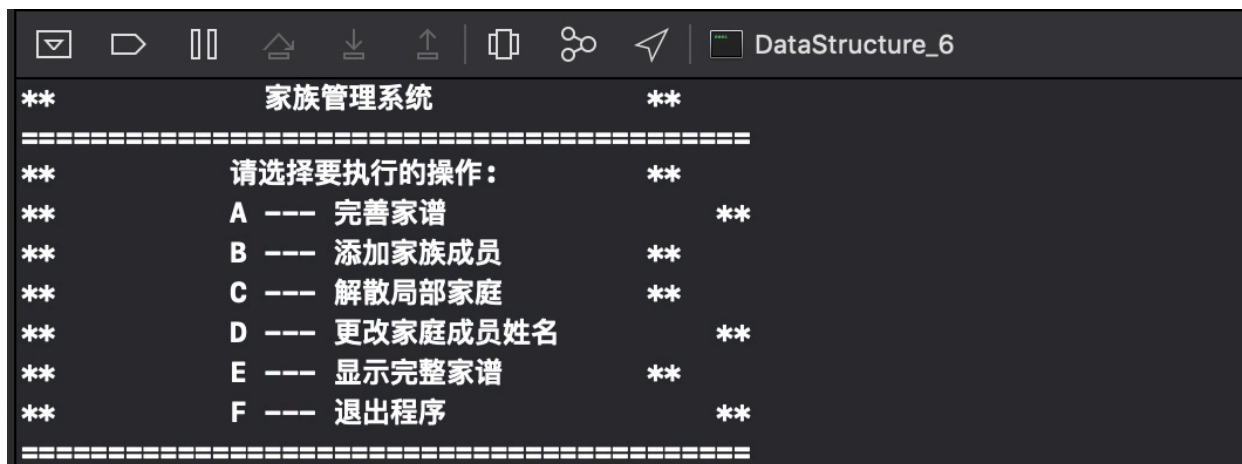
//显示功能菜单
void showMenu()

```

```

{
    cout << "**\t\t\t\t\t 家族管理系统\t\t\t\t\t**\n";
    cout << "=====\n";
    cout << "**\t\t\t\t\t请选择要执行的操作:\t\t\t\t**\n";
    cout << "**\t\t\t\t\tA --- 完善家谱\t\t\t\t\t**\n";
    cout << "**\t\t\t\t\tB --- 添加家族成员\t\t\t\t**\n";
    cout << "**\t\t\t\t\tC --- 解散局部家庭\t\t\t\t**\n";
    cout << "**\t\t\t\t\tD --- 更改家庭成员姓名\t\t\t\t**\n";
    cout << "**\t\t\t\t\tE --- 显示完整家谱\t\t\t\t**\n";
    cout << "**\t\t\t\t\tF --- 退出程序\t\t\t\t\t**\n";
    cout << "=====\n";
}

```



```

**          家族管理系统          **
=====
**          请选择要执行的操作:          **
**          A --- 完善家谱                **
**          B --- 添加家族成员            **
**          C --- 解散局部家庭            **
**          D --- 更改家庭成员姓名        **
**          E --- 显示完整家谱            **
**          F --- 退出程序                **
=====

```

## 三、实现

### 1. 创建家族系统

输入祖先姓名，将其设置为家族树的根结点。

#### 1.1 相关代码

```

//初始化家族树：添加祖先结点
void init(Tree<string>& f, TreeNode<string>& a)
{
    cout << "请先建立一个家谱!" << endl;
    cout << "请输入祖先的姓名:";

    cin>>a.data;
    f.setRoot(&a);

    cout << "此家谱的祖先是" << a.data << endl << endl;
}

```

#### 1.2 运行截图

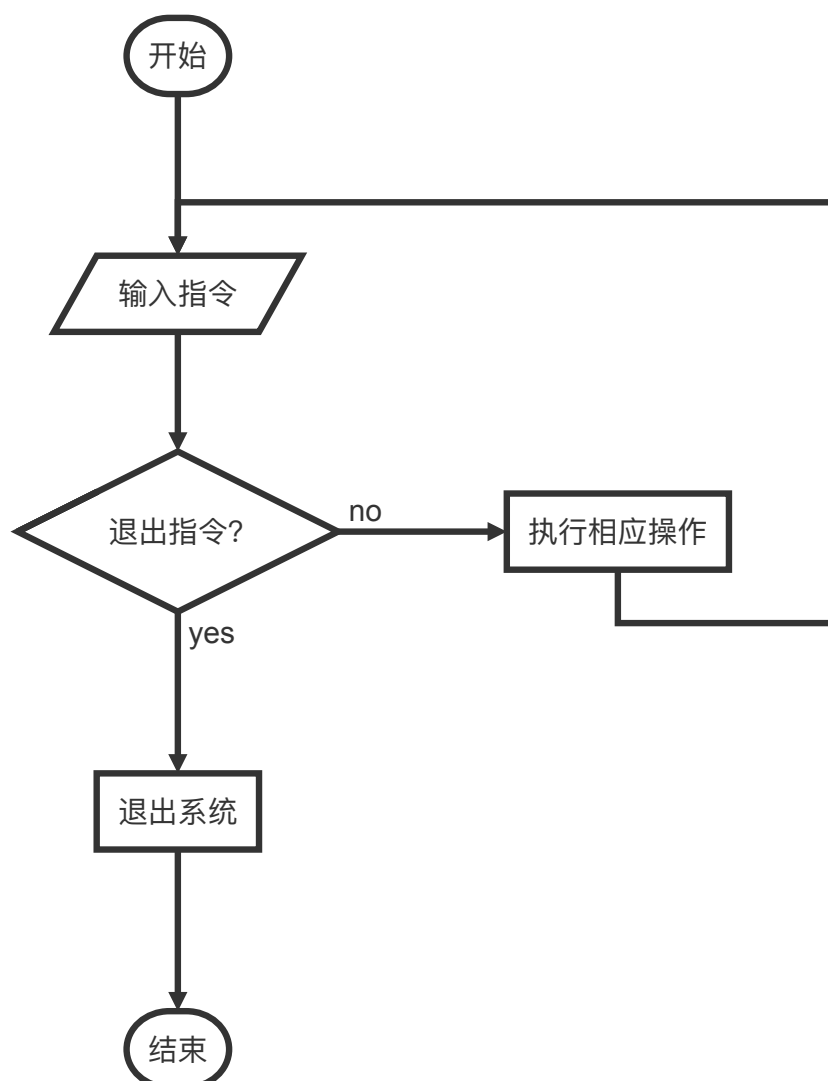
```

**          家族管理系统          **
=====
**          请选择要执行的操作：          **
**          A --- 完善家谱          **
**          B --- 添加家族成员          **
**          C --- 解散局部家庭          **
**          D --- 更改家庭成员姓名          **
**          E --- 显示完整家谱          **
**          F --- 退出程序          **
=====
请先建立一个家谱！
请输入祖先的姓名：老王
此家谱的祖先是老王

```

## 2. 菜单功能循环

### 2.1 流程图



## 2.2 相关代码

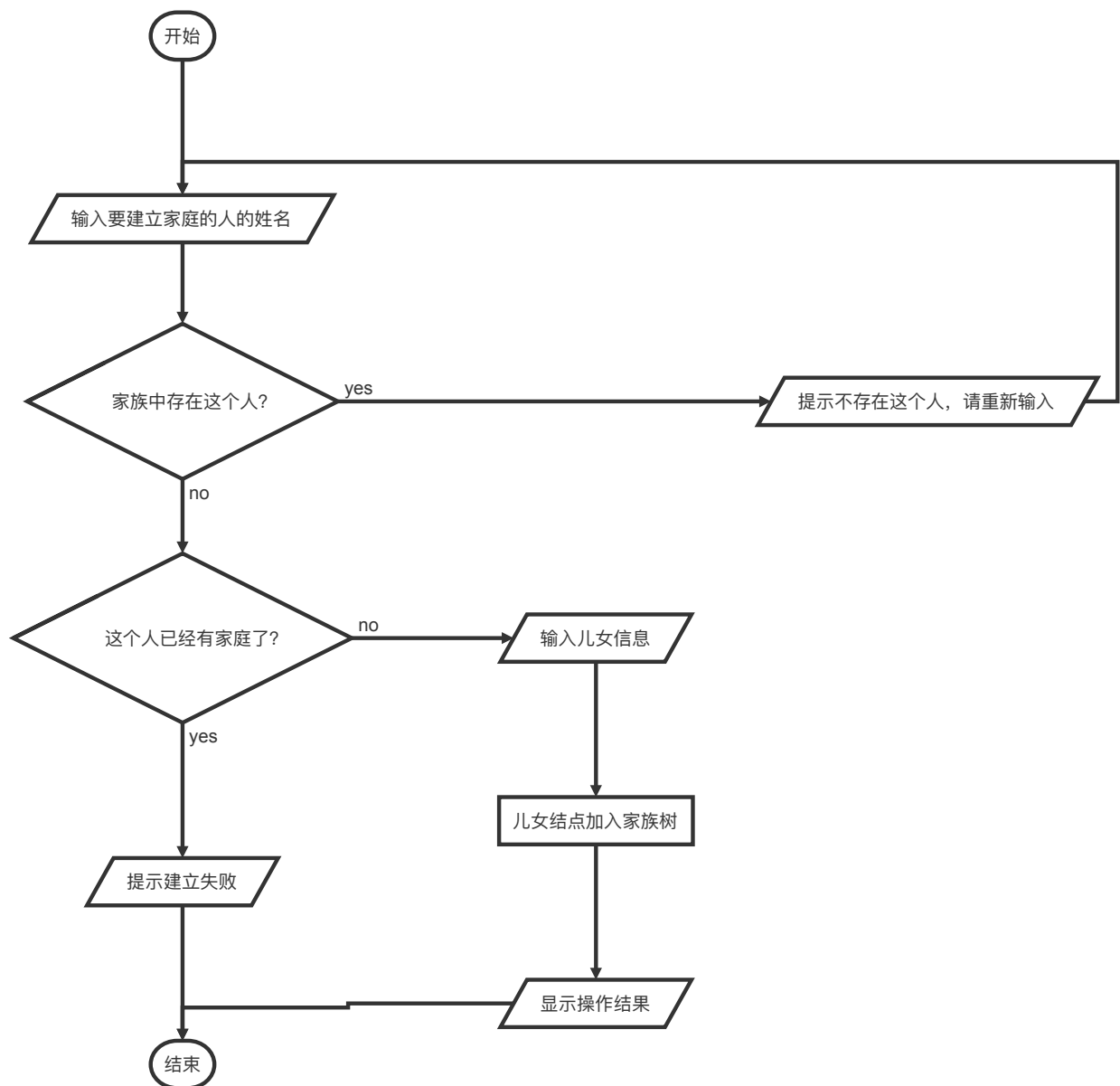
```
//执行菜单功能
void execute(Tree<string>& f)
{

    char command = 'a';
    while(command != 'F')
    {
        cout << "请选择要执行的操作：" ;
        cin >>command;
        switch(command)
        {
            case 'A':
            {
                //建立家庭代码...
            }
            break;
            case 'B':
            {
                //添加子女代码...
            }
            break;
            case 'C':
            {
                //解散家庭代码...
            }
            break;
            case 'D':
            {
                //更改家庭成员代码...
            }
            break;
            case 'E':
                f.Output();
                cout<<endl;
                break;
            case 'F':
                cout <<"\n\n程序已经成功退出！ ";
                break;
            default :
                cout << "输入错误, 请重新输入！ " << endl;
                break;
        }
    }
}
```

## 3. 完善家谱



### 3.1 流程图

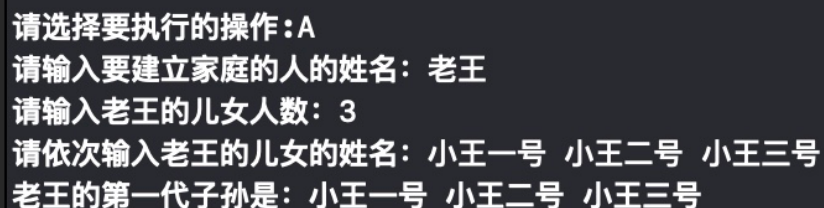


## 3.2 相关代码

```
cout << "请输入要建立家庭的人的姓名: ";
string name;
cin >> name;
while(!f.Find(name))
{
    cout << "找不到这个人, 请重新输入: ";
    cin>>name;
}

if(f.getCurrent()->firstChild != nullptr)
{
    cout << "他已经有家庭了, 请不要破坏! " << endl;
}
else
{
    cout<<"请输入"<<name<<"的儿女人数: ";
    int num;
    cin>>num;
    while(num<=0)
    {
        cout<<"儿女人数的范围有误, 请重新输入: ";
        cin>>num;
    }
    cout<<"请依次输入"<<name<<"的儿女的姓名: ";
    f.CreateSubTree(num);
    cout<<name<<"的第一代子孙是: ";
    f.showChildren(f.getCurrent());
    cout<<endl<<endl;
}
```

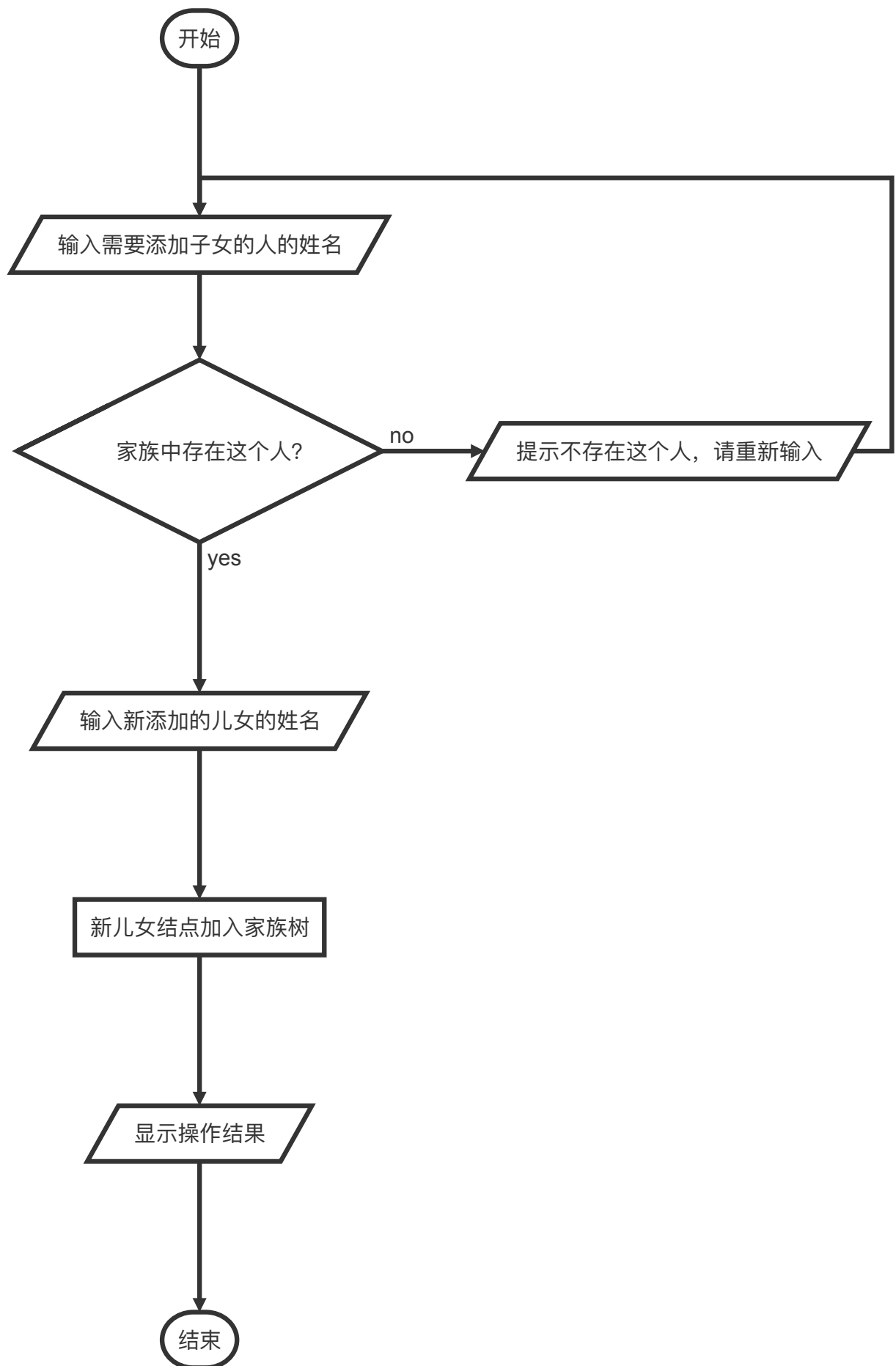
## 3.3 运行截图



请选择要执行的操作:A  
请输入要建立家庭的人的姓名: 老王  
请输入老王的儿女人数: 3  
请依次输入老王的儿女的姓名: 小王一号 小王二号 小王三号  
老王的第一代子孙是: 小王一号 小王二号 小王三号

## 4. 添加家族成员

### 4.1 流程图



## 4.2 相关代码

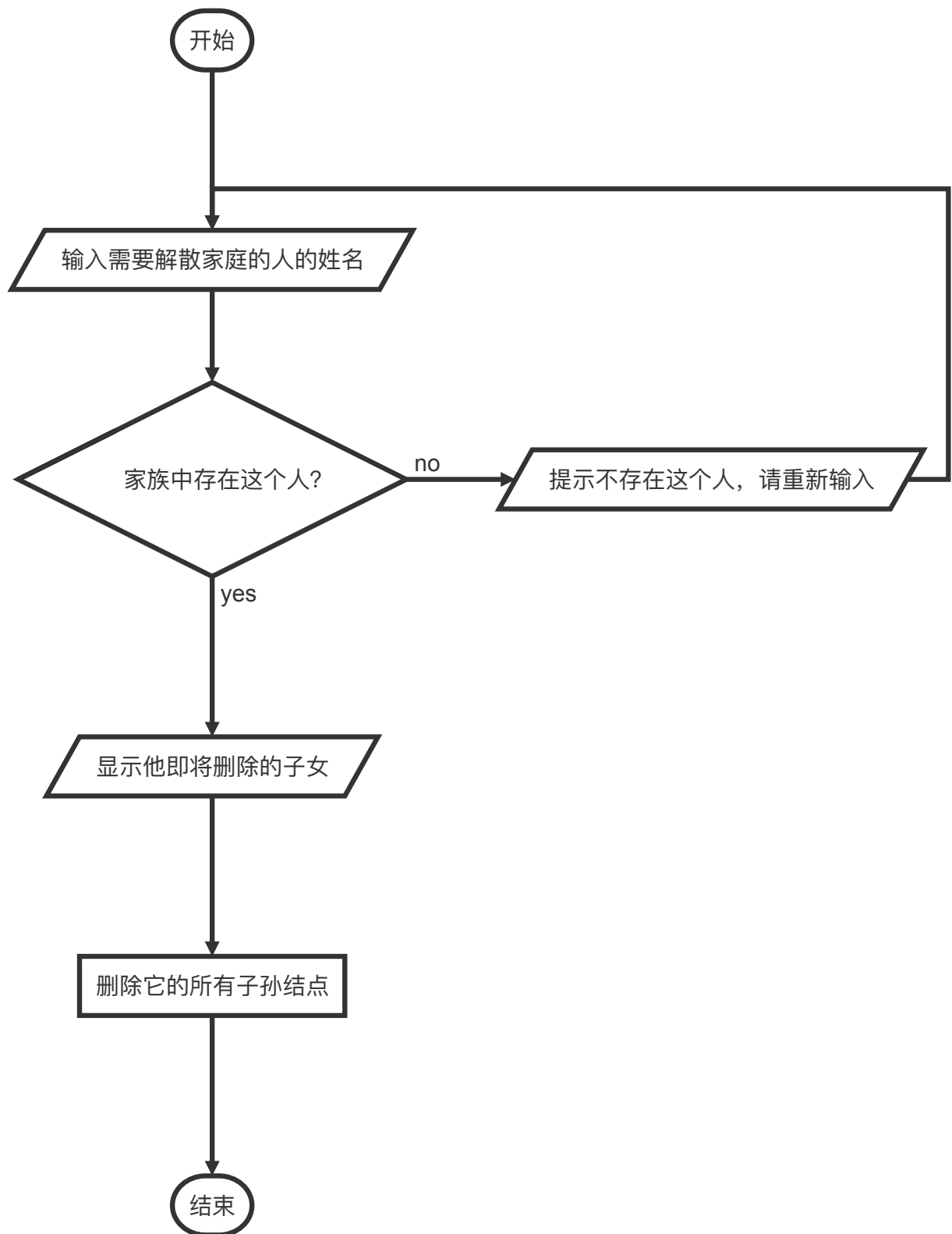
```
cout << "请输入要添加儿子（或女儿）的人的姓名:";
string name;
cin >> name;
while(!f.Find(name))
{
    cout << "找不到这个人，请重新输入: ";
    cin>>name;
}

cout << "请输入"<< name <<"新的添加儿子（或女儿）的人的姓名:";
cin >> name;
f.addChild(f.getCurrent(), name);
cout<<name<<"的第一代子孙是: ";
f.showChildren(f.getCurrent());
cout<<endl<<endl;
```

## 4.3 运行截图

## 5. 解散局部家庭

### 5.1 流程图



## 5.2 相关代码

```
cout << "请输入要解散家庭的人的姓名:";  
string name;  
cin >> name;  
while(!f.Find(name))
```

```

{
    cout << "找不到这个人，请重新输入：";
    cin>>name;
}
cout << "要解散家庭的是" << name << endl;

cout<<name<<"的第一代子孙是：";
f.showChildren(f.getCurrent());
cout<<endl<<endl;

f.removeAllDescendants(f.getCurrent());

```

### 5.3 运行截图

```

请选择要执行的操作：E
老王
├─小王一号
│   ├──小小王一号
│   └──小小王二号
└─小王二号
    └─小小王三号
├─小王三号
└─小王四号

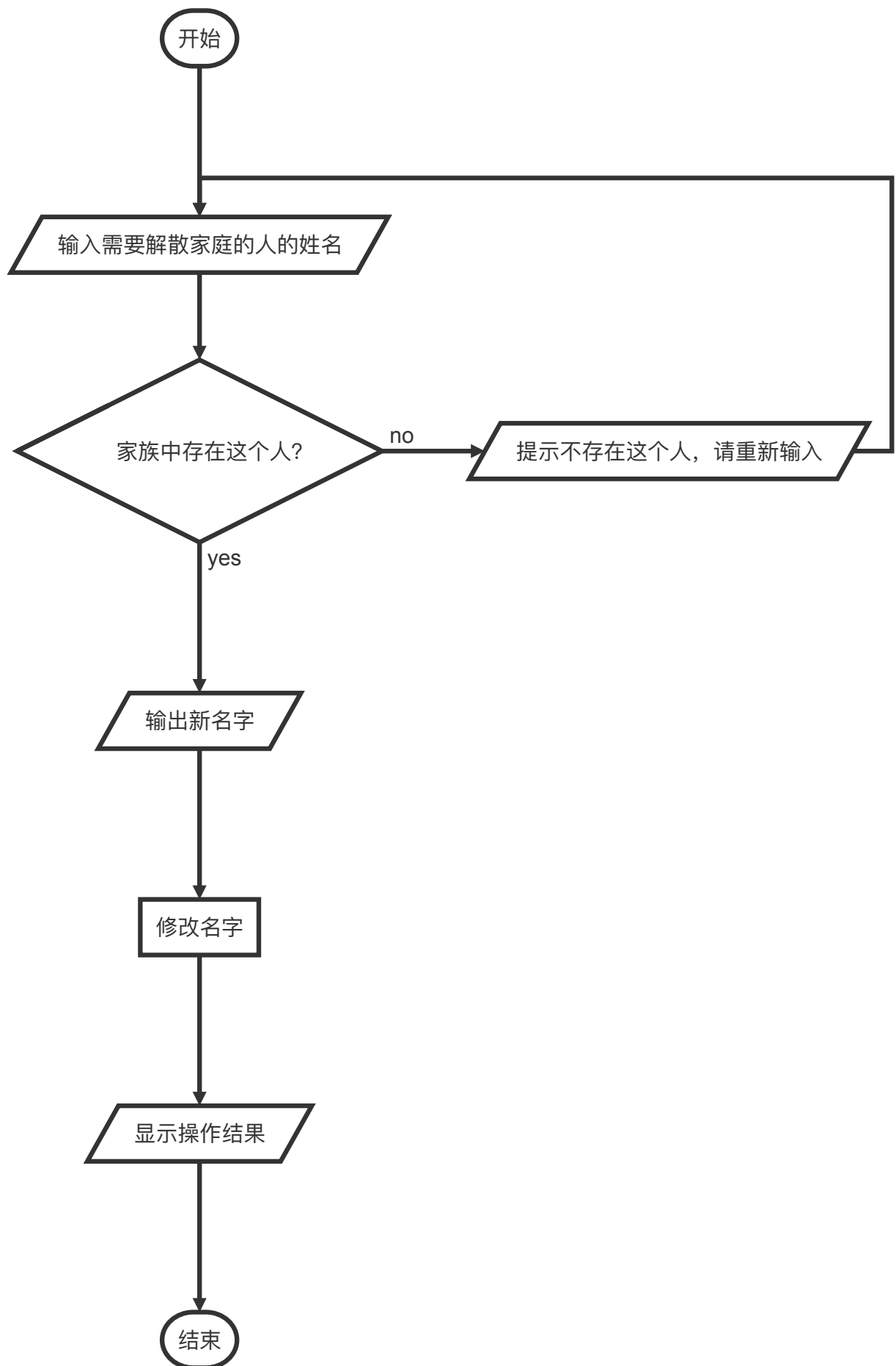
请选择要执行的操作：C
请输入要解散家庭的人的姓名：小王一号
要解散家庭的是小王一号
小王一号的第一代子孙是：小小王一号 小小王二号

请选择要执行的操作：E
老王
├─小王一号
└─小王二号
    └─小小王三号
├─小王三号
└─小王四号

```

## 6. 更改家庭成员姓名

## 6.1 流程图

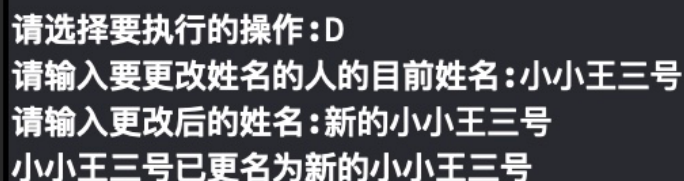




## 6.2 相关代码

```
cout << "请输入要更改姓名的人的目前姓名:";
string oldName, newName;
cin >> oldName;
while(!f.Find(oldName))
{
    cout << "找不到这个人, 请重新输入: ";
    cin>>oldName;
}
cout << "请输入更改后的姓名:";
cin >> newName;
f.changeData(f.getCurrent(), newName);
cout << oldName << "已更名为" << newName <<endl << endl;
```

## 6.3 运行截图



```
请选择要执行的操作:D
请输入要更改姓名的人的目前姓名:小小王三号
请输入更改后的姓名:新的小小王三号
小小王三号已更名为新的小小王三号
```

# 7. 显示完整家谱（创新功能）

使用深度优先遍历的思想依次访问全部家谱成员，按照树形图形结构显示完整家谱。

## 7.1 相关代码

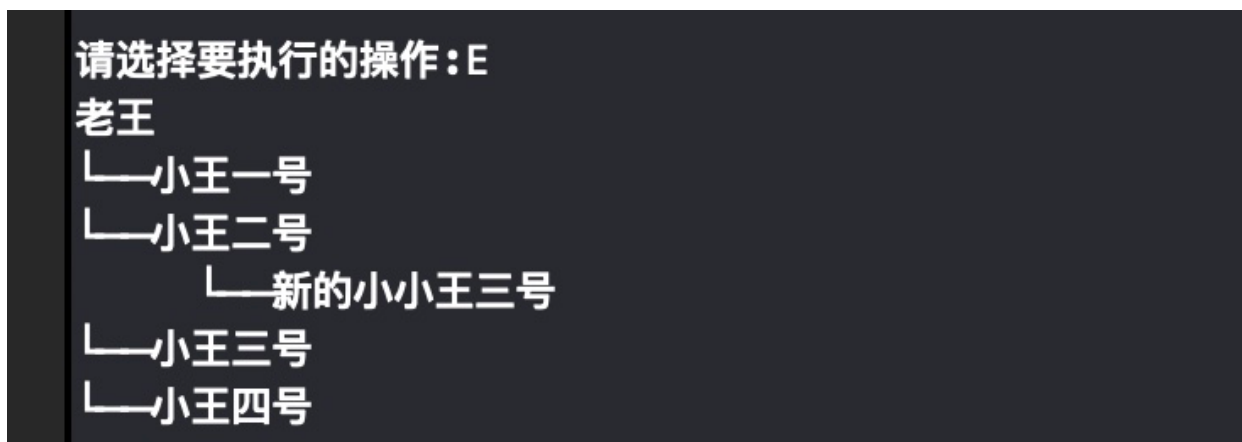
```
template <typename T>void Tree<T>::Output(TreeNode<T> *subTree, string str,
ostream &out){
    if (subTree == root){
        current = root;
    }
    string temp;
    TreeNode<T> *p = subTree;
    while (p){//深度优先
        out << str << p->data << endl;//str开始是空串
        if (p == current && p->firstChild){
            //          str = str + "└─";
            //          temp = str;
            temp= str + "└─";
        }
        else if (p == current && p->nextSibling){
            current = current->nextSibling;
        }
    }
}
```

```

    }
    else if (p->firstChild && p->firstChild->nextSibling){
        temp = string("|    ") + str;
    }
    else{
        temp = string("    ") + str;
    }
    Output(p->firstChild, temp, out); //深度递归
    p = p->nextSibling; //广度循环
}
}

```

## 7.2 运行截图



## 四、测试

### 1. 功能测试

一般情况下的功能测试已经在“三、实现”部分中以运行截图的形式给出，全部测试正常，故不在这里重复叙述。

### 2. 出错测试

#### 3.1 建立家庭的人不存在

```
请选择要执行的操作:E
老王
├─小王一号
├─小王二号
│   └─新的小小王三号
├─小王三号
└─小王四号

请选择要执行的操作:A
请输入要建立家庭的人的姓名: 小小王一号
找不到这个人, 请重新输入: |

All Output
```

提示这个人不存在, 要求重新输入。

测试结果正确。

### 3.2 建立家庭的人已经拥有家庭

```
请选择要执行的操作:E
老王
├─小王一号
│   └─小小王一号
│       └─小小王二号
├─小王二号
│   └─新的小小王三号
├─小王三号
└─小王四号

请选择要执行的操作:A
请输入要建立家庭的人的姓名: 小王一号
他已经有家庭了, 请不要破坏!
请选择要执行的操作:

All Output
```

提示这个人已经拥有家庭, 要求重新输入。

测试结果正确。

### 3.3 建立家庭时儿女人数范围有误

```
请选择要执行的操作:E
老王
├─小王一号
│   ├──小小王一号
│   ├──小小王二号
├─小王二号
│   └─新的小小王三号
├─小王三号
└─小王四号

请选择要执行的操作:A
请输入要建立家庭的人的姓名: 小小王一号
请输入小小王一号的儿女人数: -1
儿女人数的范围有误, 请重新输入: |

All Output
```

提示人数输入有误, 要求重新输入人数。

测试结果正确。

### 3.4 添加儿女、解散家庭、更改姓名的人不存在

实现方法与“建立家庭的人不存在”相同, 经过测试均运行正常, 不再重复展示。