项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 毛凌骏

学 号： 2053058

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 需求 2](#_Toc6974)

[1.1 项目背景 2](#_Toc23740)

[1.2 功能分析 3](#_Toc17077)

[2 设计 3](#_Toc26270)

[2.1 数据结构设计 3](#_Toc14803)

[2.1.1 逻辑结构设计 3](#_Toc534)

[2.1.2 存储结构设计 3](#_Toc10138)

[2.2 类设计 4](#_Toc12369)

[2.2.1 树节点的模板类 4](#_Toc3543)

[2.2.2 树的模板类 5](#_Toc9208)

[2.2.3 家谱管理系统类 5](#_Toc21610)

[3 实现 6](#_Toc11712)

[3.1 建立家谱功能的实现 6](#_Toc22573)

[3.1.1 建立家谱功能实现思路 6](#_Toc4680)

[3.1.2 建立家谱功能实现流程图 6](#_Toc25027)

[3.1.3 建立家谱功能实现代码 6](#_Toc15393)

[3.1.4 建立家谱功能实现样例 7](#_Toc19465)

[3.2 完善家谱功能的实现 7](#_Toc4860)

[3.2.1 完善家谱功能实现思路 7](#_Toc7949)

[3.2.2 完善家谱功能实现流程图 8](#_Toc18763)

[3.2.3 完善家谱功能实现代码 8](#_Toc19376)

[3.2.4 完善家谱功能实现样例 9](#_Toc15316)

[3.3 添加家庭成员功能的实现 9](#_Toc21853)

[3.3.1 添加家庭成员功能实现思路 9](#_Toc6928)

[3.3.2 添加家庭成员功能实现流程图 9](#_Toc25215)

[3.3.3 添加家庭成员功能实现代码 10](#_Toc5193)

[3.3.3 添加家庭成员功能实现样例 10](#_Toc12697)

[3.4 解散局部家庭功能的实现 10](#_Toc8398)

[3.4.1 解散局部家庭功能实现思路 10](#_Toc8239)

[3.4.2 解散局部家庭功能实现流程图 11](#_Toc5057)

[3.4.3 解散局部家庭功能实现代码 11](#_Toc6622)

[3.4.4 解散局部家庭功能实现样例 11](#_Toc833)

[3.5 更改家庭成员姓名功能的实现 12](#_Toc14679)

[3.5.1 更改家庭成员姓名功能实现思路 12](#_Toc20484)

[3.5.2 更改家庭成员姓名功能实现流程图 12](#_Toc29214)

[3.5.3 更改家庭成员姓名功能实现代码 12](#_Toc26267)

[3.5.4 更改家庭成员姓名功能实现样例 13](#_Toc4402)

[3.6 树相关操作的实现 13](#_Toc28659)

[4 测试 15](#_Toc5134)

[4.1 常规测试 15](#_Toc4823)

[4.1.1 建立家谱功能测试 15](#_Toc4259)

[4.1.2 完善家谱功能测试 15](#_Toc13944)

[4.1.3 添加家庭成员功能测试 16](#_Toc32319)

[4.1.4 解散局部家庭功能测试 17](#_Toc30282)

[4.1.5 更改家庭成员姓名功能测试 18](#_Toc28278)

[4.2 错误测试 18](#_Toc14777)

[4.2.1 操作目标不存在 18](#_Toc32394)

[4.2.2 建立家庭目标已有家庭 18](#_Toc13947)

[4.2.3 孩子数量非法 19](#_Toc19689)

# 1 需求

## 1.1 项目背景

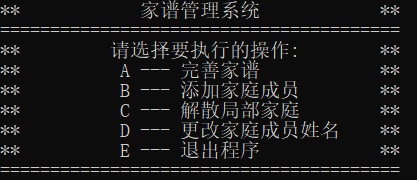
家谱是一种以表谱形式，记载一个以血缘关系为主体的家族世袭繁衍和重要任务事迹的特殊图书体裁。家谱是中国特有的文化遗产，是中华民族的三大文献（国史，地志，族谱）之一，属于珍贵的人文资料，对于历史学，民俗学，人口学，社会学和经济学的深入研究，均有其不可替代的独特功能。传统的家谱通常采用纸质的形式记录，既不易查找翻阅，也不易留存。而本项目通过计算机存储家谱信息，使得用户可随时随地登录系统查找修改家谱，并通过合理设计数据结构解决了传统家谱中数据量过大难以查找的缺点。家谱管理系统使整个家谱管理的操作流程更加高效智能，且不易出错，体现了信息时代家谱管理模式的优越性。

本项目对家谱管理进行简单的模拟，用户通过控制台进行相应的操作，包括查看祖先和子孙个人信息，插入家族成员，删除家族成员等功能，从而完整模拟家谱的建立、查找、修改过程，体现信息化管理的思想。

## 1.2 功能分析

作为一个家谱管理系统，最基本的功能就是建立家谱。除此之外，家谱管理系统还应该遵循最基本的“增、删、改、查”功能，对存储的家谱情况进行增加、删除、修改和查找，并且应当保证系统可以正常退出。同时，对于用户执行的操作应当进行错误处理，例如当用户给一个不存在的对象建立家庭时，程序应给出提示并让用户再次输入，以避免因用户操作不当而导致程序运行错误的现象发生。

因此，本系统的功能应包含：建立家谱功能、创建家庭功能、添加家庭成员功能、解散家庭功能、更改姓名功能、查找成员功能、显示子女功能和退出系统功能、错误处理功能。



# 2 设计

## 2.1 数据结构设计

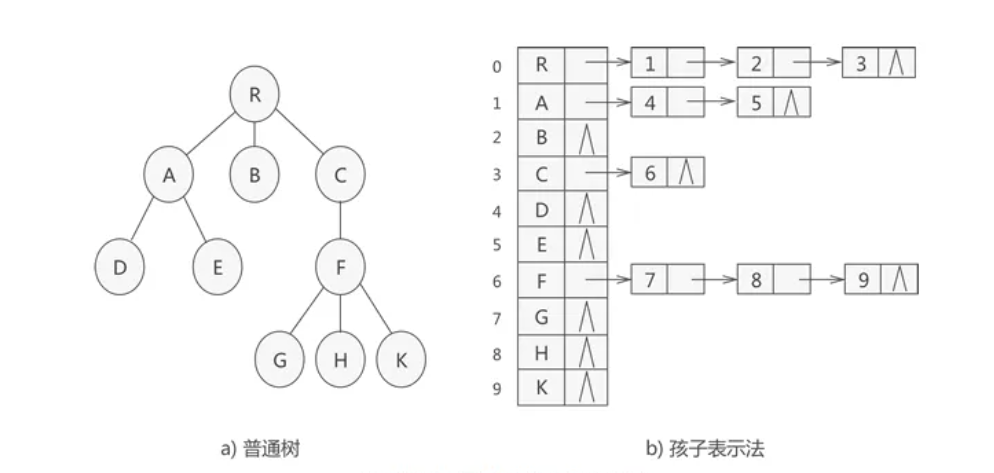
### 2.1.1 逻辑结构设计

本项目是对家谱管理系统进行模拟，其中各个元素之间存在明显的偏序关系或层次关系，比如兄弟姐妹在家谱中应该处于同一层，而父母长辈处于更高的层次，孩子晚辈处于更低的层次。不同层次间的元素构成了“一对多”与“多对一”的映射，因此应当采用非线性逻辑结构存储。而显然家谱满足以下特点：各个元素间有序、整个家谱关系图中不存在环与回边、双亲节点可对应多个孩子节点、孩子节点至多对应一个双亲节点，因此本项目可选用树作为存储逻辑结构。

### 2.1.2 存储结构设计

充分利用顺序存储和链式存储结构的特点，可实现对树存储结构的表示。常用的三种表示法为: 双亲表示法、孩子表示法、孩子兄弟表示法。

双亲表示法定义结构数组存放树的结点，每个结点含数据域和双亲域，其中数据域用于存放结点本身信息，双亲域用于指示本结点的双亲结点在数组中的位置。此存储方式可高效查找孩子的双亲，但查找双亲的孩子则需花费较长的时间。 孩子表示法用二叉链表作树的[存储结构](https://so.csdn.net/so/search?q=%E5%AD%98%E5%82%A8%E7%BB%93%E6%9E%84&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/daocaoren_/article/details/_blank)，链表中每个结点的两个指针域分别指向其第一个孩子结点和下一个兄弟结点，可以快速查找孩子节点但难以找到双亲。孩子兄弟表示法同样使用二叉链表[存储](https://so.csdn.net/so/search?q=%E5%AD%98%E5%82%A8%E7%BB%93%E6%9E%84&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/daocaoren_/article/details/_blank)，链表中每个结点的两个指针域分别指向其第一个孩子结点和下一个兄弟结点。和前两种表示方法相比，孩子兄弟表示法用一棵二叉树替代一棵复杂的树，简化了数据结构，因此本项目采用第三种方法存储家谱信息。



## 2.2 类设计

本项目共设计了三个类，分别是树节点的模板类、树的模板类、家谱管理系统类。其中树节点类与树类主要用于实现树的数据结构，家谱管理系统类则用于存储各项操作，包括完善家谱、添加家庭成员、解散局部家庭、更改家庭成员姓名、打印子女等功能。

### 2.2.1 树节点的模板类

树节点的模板类主要用于存放树中元素的信息，其protected属性包括数据域与指针域，public属性则包含各项操作。树节点的数据域存放的是该节点的值，指针域存放的是指向该元素首孩子的指针和指向该元素下一个兄弟的指针，通过这两个指针将一颗复杂的树结构简化为了一颗二叉树，提高了代码的简洁性，提高了树的搜索效率。

//树节点的模板类

**template** <**class** Type>

**class** treeNode {

**protected**:

    Type value;    //数据域

    treeNode<Type>\* firstChild, \* nextSibling;    //指针域

**public**:

    treeNode(Type v = 0, treeNode<Type>\* f = NULL, treeNode<Type>\* n = NULL) :

        value(v),firstChild(f),nextSibling(n) {}    //构造函数

**friend** **class** Tree<Type>;

**friend** **class** Family;

};

### 2.2.2 树的模板类

树的模板类在树节点的基础上构成了一颗完整的树，其protected属性有两项，分别是指向根节点的指针root与指向当前节点的指针current。树模板类的public属性则用于存放树存储结构可以实现的各项操作，包括查找节点、插入节点、递归删除树、删除子树、寻当前长子、寻当前兄弟、修改节点等。每次操作通过current指针来对当前节点操作，操作结束后更新当前current指针指向的值。

//树的模板类

**template** <**class** Type>

**class** Tree {

**protected**:

    treeNode<Type>\* root, \*current;                 //指针域

**public**:

    Tree();                                         //默认构造函数

    Tree(**const** Type& value);                        //带一个参数的构造函数

**bool** find(treeNode<Type>\* root, Type key);        //查找节点

**void** insertChild(**const** Type& value);            //插入节点

**void** deleteChild(treeNode<Type>\* p);            //递归删除树

**void** deleteChild();                             //删除子树

    treeNode<Type>\* findFirstChild();               //寻当前长子

    treeNode<Type>\* findNextSibling();              //寻当前兄弟

**void** changeNode(**const** Type& value);             //修改节点

**friend** **class** treeNode<Type>;

**friend** **class** Family;

};

### 2.2.3 家谱管理系统类

家谱管理系统类Family是本项目最为直接调用的类，其protected属性存储了一颗家族树，public属性则存储各项家谱管理操作，包括完善家谱、添加家庭成员、解散局部家庭、更改家庭成员姓名、打印子女等。家谱管理系统类为本项目的核心，程序初始先将Family实例化，后续所有操作均通过调用家谱管理系统类的成员函数来实现，最后完成主函数以验证各个函数功能并得到运行结果。由于家谱管理系统类、树节点类、树类高度相关，因此在类设计时将它们都声明为彼此的友元，以便于互相调用。

//家谱管理系统类

**class** Family {

**protected**:

    Tree<string>\* root;

**public**:

    Family();                         //构造函数

**void** build();                     //完善家谱

**void** add();                       //添加家庭成员

**void** cancel();                    //解散局部家庭

**void** change();                    //更改家庭成员姓名

**void** printChild(string name);     //打印子女

**friend** **class** treeNode<string>;

**friend** **class** Tree<string>;

};

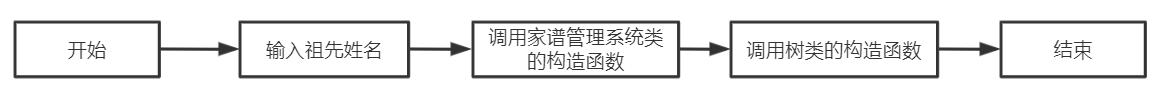
# 3 实现

## 3.1 建立家谱功能的实现

### 3.1.1 建立家谱功能实现思路

建立家谱的主要操作为输入整个家族的祖先信息，在程序中体现为给家族树创建一个根节点，由于家谱管理系统类的主要数据成员为家族树，因此建立家谱可调用家谱管理系统类的构造函数来实现。同时考虑到交互界面部分，在构造家谱管理系统类的时候可以将系统提示一并输出，以提高本系统的用户使用体验。

### 3.1.2 建立家谱功能实现流程图



### 3.1.3 建立家谱功能实现代码

//构造函数

Family::Family() {

    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 --- 退出程序            \*\*" << endl;

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

    cout << "首先建立一个家谱！" << endl;

    cout << "请输入祖先的姓名：";

    string name;

    cin >> name;

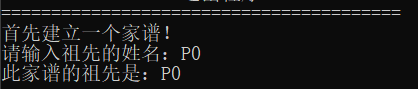
**const** string p = name;

    root = **new** Tree<string>(name);

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

}

### 3.1.4 建立家谱功能实现样例

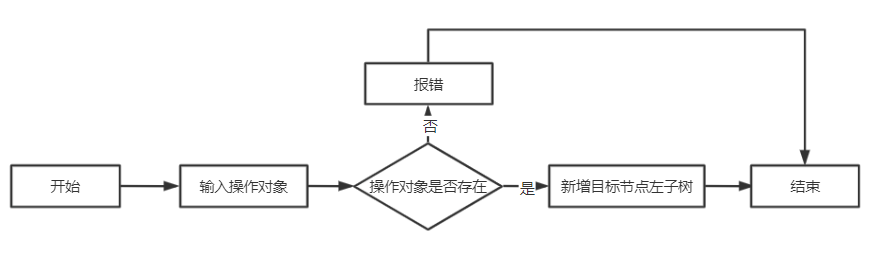


## 3.2 完善家谱功能的实现

### 3.2.1 完善家谱功能实现思路

完善家谱功能在程序中体现为给目标节点插入左子树。程序从家谱树的根节点进行搜索，若目标节点不存在或目标节点已有左子树程序报错并给出提示，否则进行完善家谱操作。由于本项目采用孩子兄弟表示法表示树，因此完善家谱应分为两种情况。当该节点孩子数量为1时只需将新增节点插入current指针指向节点的首孩子firstChild上即可，若该节点孩子数量大于1，则还需将剩余节点插入current的firstChild指针指向节点的右子树上，代表为目标节点孩子的兄弟，依次向右插入即可将所有新增节点插入树中。

### 3.2.2 完善家谱功能实现流程图



### 3.2.3 完善家谱功能实现代码

//完善家谱

**void** Family::build() {

    cout << "请输入要建立家庭的人的姓名：";

    string name;

    cin >> name;

**char** t;

**while** ((t = cin.get()) != '\n');

**if** (root->find(root->root,name) == NULL) {

        cout << "未找到此人" << endl;

**return**;

    }

**else** **if**(root->findFirstChild()!=NULL) {

        cout << "此人已有家庭，可输入B添加子女" << endl;

**return**;

    }

**else** {

        root->find(root->root, name);

**int** number;

        cout << "请输入" << name << "的儿女个数：";

**while** (1) {

            cin >> number;

**if** (cin.fail()) {

                cout << "输入错误，请重新输入：";

                cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

            }

**else** **if** (number < 0) {

                cout << "儿女个数应大于0，请重新输入：";

                cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

            }

**else**

**break**;

        }

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

**for** (**int** i = 0; i < number; i++) {

            string childName;

            cin >> childName;

            root->insertChild(childName);

        }

        cout << name << "的第一代子孙是：";

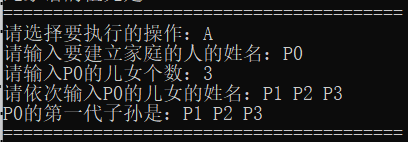
        printChild(name);

        cout << endl;

    }

}

### 3.2.4 完善家谱功能实现样例

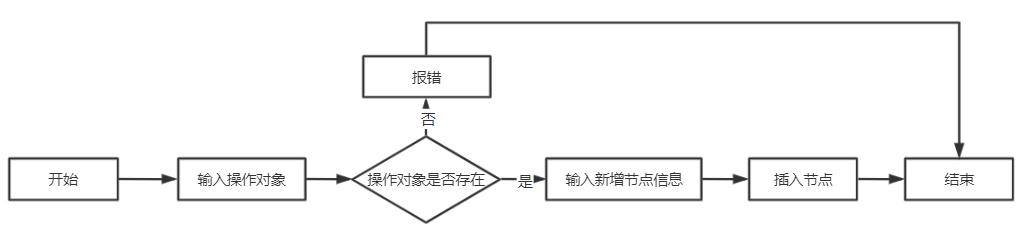


## 3.3 添加家庭成员功能的实现

### 3.3.1 添加家庭成员功能实现思路

添加家庭成员在程序中体现为给特定的树节点增添子女节点，由于本项目采用孩子兄弟表示法，因此在添加成员时应考虑两种情况。一是当前节点无首孩子，此时新节点应赋值给current节点的firstChild指针；二是当前节点已有首孩子，此时新节点应遍历访问current首孩子的nextSibling指针指向的元素，在末尾插入新增节点。同时用户输入待添加子女的节点信息时应当对其做错误处理，若用户输入的目标不在家谱中，程序应报错并给出提示。

### 3.3.2 添加家庭成员功能实现流程图



### 3.3.3 添加家庭成员功能实现代码

//添加家庭成员

**void** Family::add() {

    string name;

    cout << "请输入要添加儿子（或女儿）的人的姓名：";

    cin >> name;

**char** t;

**while** ((t = cin.get()) != '\n');

**if** (root->find(root->root, name) == NULL) {

        cout << "未找到此人" << endl;

**return**;

    }

**else** {

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

        string childName;

        cin >> childName;

        root->find(root->root, name);

        root->insertChild(childName);

        cout << name << "的第一代子孙是：";

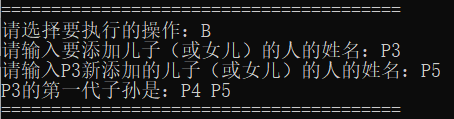
        printChild(name);

        cout << endl;

    }

}

### 3.3.3 添加家庭成员功能实现样例

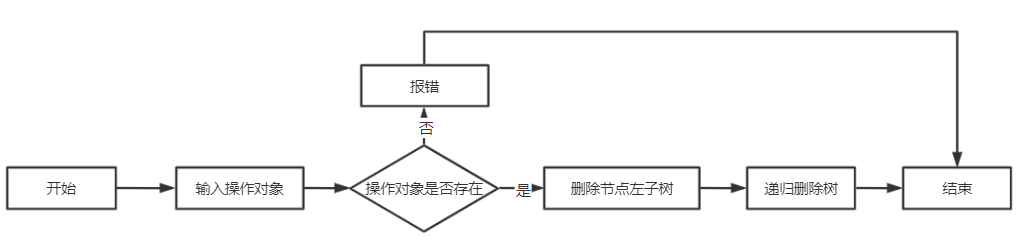


## 3.4 解散局部家庭功能的实现

### 3.4.1 解散局部家庭功能实现思路

解散局部家庭在程序中体现为删除子树信息，在孩子兄弟表示法中体现为删除左子树。由于树的结构是递归的，因此删除子树的操作也可以递归实现。删除子树的操作可整体分为两部分，一是选择current节点的左子树并删除，二是进入current节点的左子树后将当前节点的左右子树均递归删除。同时用户输入待删除子女的节点信息时应当对其做错误处理，若用户输入的目标不在家谱中，程序应报错并给出提示。

### 3.4.2 解散局部家庭功能实现流程图



### 3.4.3 解散局部家庭功能实现代码

//解散局部家庭

**void** Family::cancel() {

    cout << "请输入要解散家庭的人的姓名：";

    string name;

    cin >> name;

**char** t;

**while** ((t = cin.get()) != '\n');

**if** (root->find(root->root, name) == NULL) {

        cout << "未找到此人" << endl;

**return**;

    }

**else** {

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

        cout << name << "的第一代子孙是：";

        printChild(name);

        root->find(root->root, name);

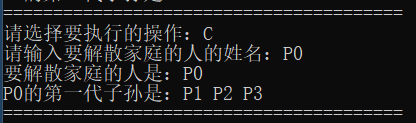
        root->deleteChild();

        cout << endl;

    }

}

### 3.4.4 解散局部家庭功能实现样例

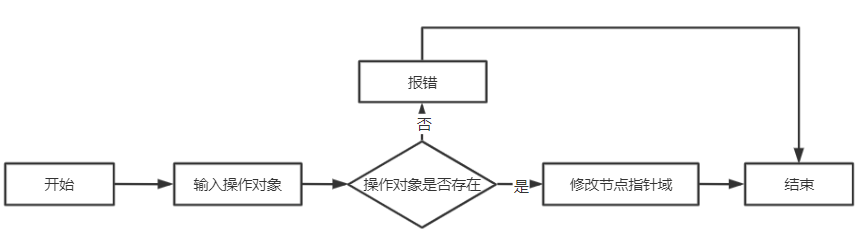


## 3.5 更改家庭成员姓名功能的实现

### 3.5.1 更改家庭成员姓名功能实现思路

更改家庭成员姓名在程序中体现为更改节点数据域，其实现较为简单，首先程序从根节点开始查找待搜索节点，若该节点不存在，则程序报错并给出提示。若搜索到目标节点，则将current置于该节点，调用树的changeNode函数修改该节点的值。

### 3.5.2 更改家庭成员姓名功能实现流程图



### 3.5.3 更改家庭成员姓名功能实现代码

//更改家庭成员姓名

**void** Family::change() {

    cout << "请输入要更改姓名的人的目前姓名：";

    string name;

    cin >> name;

**char** t;

**while** ((t = cin.get()) != '\n');

**if** (root->find(root->root, name) == NULL) {

        cout << "未找到此人" << endl;

**return**;

    }

**else** {

        cout << "请输入更改后的姓名：";

        string nameChange;

        cin >> nameChange;

**char** t;

**while** ((t = cin.get()) != '\n');

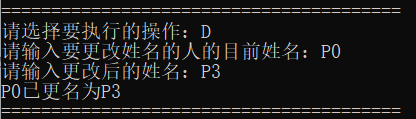
        root->changeNode(nameChange);

        cout << name << "已更名为" << nameChange << endl;

    }

}

### 3.5.4 更改家庭成员姓名功能实现样例



## 3.6 树相关操作的实现

/\* 树类函数的体外实现 \*/

//默认构造函数

**template** <**class** Type>

Tree<Type>::Tree() {

    root = current = NULL;

}

//带一个参数的构造函数

**template** <**class** Type>

Tree<Type>::Tree(**const** Type& value) {

    root = **new** treeNode<Type>(value);

    current = root;

}

//查找节点

**template** <**class** Type>

**bool** Tree<Type>::find(treeNode<Type>\* node, Type key) {

**if** (node == NULL)

**return** **false**;

**if** (node->value == key) {

        current = node;

**return** **true**;

    }

**else** {

**if** (find(node->nextSibling, key) == **true**)

**return** **true**;

**if** (node->firstChild != NULL && find(node->nextSibling, key) == **false**)

**return** find(node->firstChild, key);

    }

**return** **false**;

}

//插入节点

**template** <**class** Type>

**void** Tree<Type>::insertChild(**const** Type& value) {

**if** (current->firstChild == NULL) {

        treeNode<Type>\* p = **new** treeNode<Type>(value);

        current->firstChild = p;

    }

**else** {

        treeNode<Type>\* p = current->firstChild;

**while** (p->nextSibling != NULL)

            p = p->nextSibling;

        treeNode<Type>\* q = **new** treeNode<Type>(value);

        assert(q != NULL);

        p->nextSibling = q;

    }

}

//修改节点

**template** <**class** Type>

**void** Tree<Type>::changeNode(**const** Type& value) {

    current->value = value;

}

//寻当前长子

**template** <**class** Type>

treeNode<Type>\* Tree<Type>::findFirstChild() {

    treeNode<Type>\* temp = current->firstChild;

    current = temp;

**return** temp;

}

//寻当前兄弟

**template** <**class** Type>

treeNode<Type>\* Tree<Type>::findNextSibling() {

    treeNode<Type>\* temp = current->nextSibling;

    current = temp;

**return** temp;

}

//递归删除树

**template** <**class** Type>

**void** Tree<Type>::deleteChild(treeNode<Type>\* p) {

**if** (p != NULL) {

        deleteChild(p->firstChild);

        deleteChild(p->nextSibling);

**delete** p;

    }

}

//删除子树

**template** <**class** Type>

**void** Tree<Type>::deleteChild() {

**if** (current != NULL) {

        deleteChild(current->firstChild);

        current->firstChild = NULL;

    }

}

# 4 测试

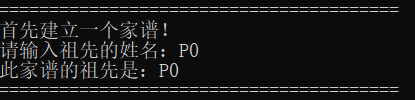
## 4.1 常规测试

### 4.1.1 建立家谱功能测试

预期效果：

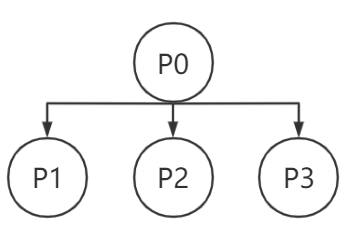


实际效果：

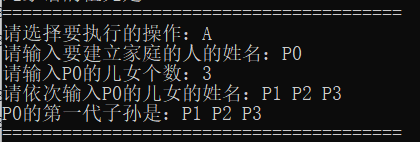


### 4.1.2 完善家谱功能测试

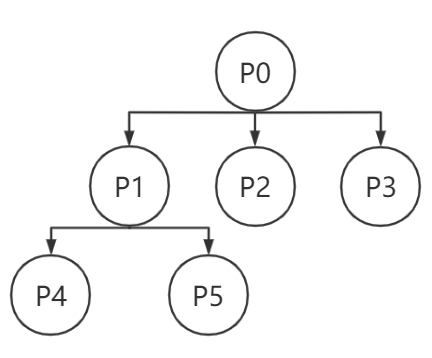
预期效果：



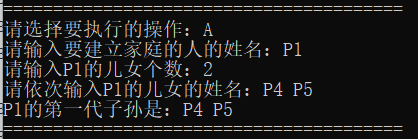
实际效果：



预期效果：

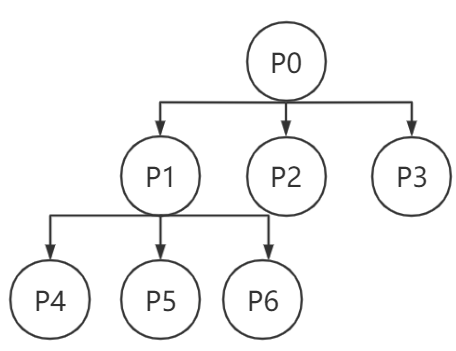


实际效果：

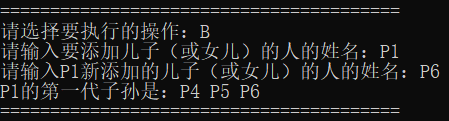


### 4.1.3 添加家庭成员功能测试

预期效果：

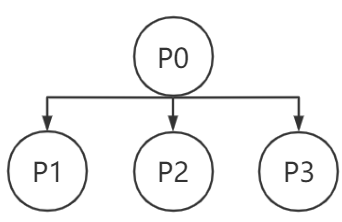


实际效果：

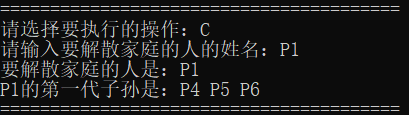


### 4.1.4 解散局部家庭功能测试

预期效果：

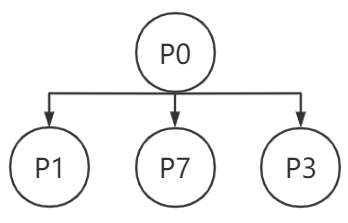


实际效果：

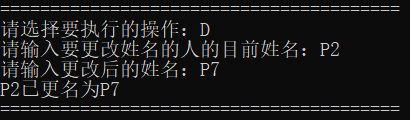


### 4.1.5 更改家庭成员姓名功能测试

预期效果：

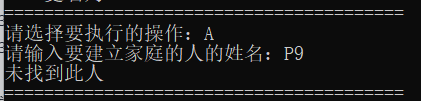


实际效果：

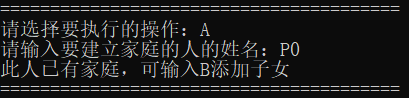


## 4.2 错误测试

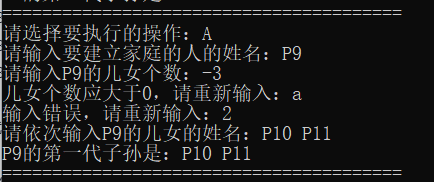
### 4.2.1 操作目标不存在



### 4.2.2 建立家庭目标已有家庭



### 4.2.3 孩子数量非法



4.2.4解散无子女家庭

