**项目说明文档**

**数据结构课程设计**

**——家谱管理系统**

作 者 姓 名： 张梓瀚

学 号： 2051943

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 - 3 -](#_Toc121208556)

[1.1项目背景 - 3 -](#_Toc121208557)

[1.2 项目要求 - 3 -](#_Toc121208558)

[1.2.1 功能要求 - 3 -](#_Toc121208559)

[1.2.2 输入格式 - 3 -](#_Toc121208560)

[1.2.3 输出格式 - 3 -](#_Toc121208561)

[1.2.4 项目示例 - 4 -](#_Toc121208562)

[2 项目设计 - 4 -](#_Toc121208563)

[2.1 数据结构设计 - 4 -](#_Toc121208564)

[2.2 类设计 - 4 -](#_Toc121208565)

[2.2.1 树结点类（tree\_node） - 5 -](#_Toc121208566)

[2.2.2 二叉树类（my\_tree） - 5 -](#_Toc121208567)

[2.2.3 家谱树类（geneology） - 6 -](#_Toc121208568)

[2.3 家谱管理系统算法 - 6 -](#_Toc121208569)

[2.3.1 主要思想 - 6 -](#_Toc121208570)

[2.3.2代码 - 7 -](#_Toc121208571)

[3 项目测试 - 10 -](#_Toc121208572)

[3.1 正常测试四种操作 - 10 -](#_Toc121208573)

[3.2 解散局部家庭后再解散同个家庭 - 10 -](#_Toc121208574)

[3.3 解散局部家庭后再查询、修改、删除该家庭成员 - 10 -](#_Toc121208575)

[3.4 更名后查询家庭成员原名 - 11 -](#_Toc121208576)

[3.5 多代单传家庭测试 - 11 -](#_Toc121208577)

[3.6 只有祖先结点家庭测试 - 11 -](#_Toc121208578)

# 1 项目分析

## 1.1项目背景

在日常生活中，我们常常需要管理树形结构的数据，例如家谱管理、组织架构管理等，这类数据往往可以用多叉树的形式描述。然而，对多叉树结构进行增删改查并不方便，因此我们引入左儿子右兄弟的二叉树结构，将多叉树转化为二叉树，极大简化了数据处理的过程，对于日常生产生活有很大意义。

## 1.2 项目要求

### 1.2.1 功能要求

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。

### 1.2.2 输入格式

（见项目示例）

### 1.2.3 输出格式

（见项目示例）

### 1.2.4 项目示例



# 2 项目设计

## 2.1 数据结构设计

如上述项目背景部分所述，为了方便数据的增删改查、简化代码，我在本项目中选用左儿子右兄弟的二叉树结构来存储家谱树。

## 2.2 类设计

本题需要实现的类主要包括树结点和二叉树类，为了提高代码的复用性，我使用模板类的方法编写这两个类的代码；同时为了方便用户操作，编写了一个二叉树类的派生类——家谱树类，在其中加入了输入输出提示、错误处理等部分。

### 2.2.1 树结点类（tree\_node）

树的结点中存储的数据有结点数据data、长子结点指针\*first\_child\_、最近兄弟结点指针\*next\_sibling\_和前驱节点\*prev（此结点的设置是为了方便后续的删除等操作）。

template<class T>

class tree\_node

{

T\* data = nullptr;

tree\_node<T>\* first\_child\_ = nullptr;

tree\_node<T>\* next\_sibling\_ = nullptr;

tree\_node<T>\* prev = nullptr;

public:

//默认构造函数

tree\_node(){};

//赋值构造函数

tree\_node(const T& src);

//获取结点的第一个孩子

tree\_node<T>\* child();

//获取结点的第一个兄弟

tree\_node<T>\* sibling();

//删除结点所有孩子

void delete\_children();

//删除结点的一个兄弟

void delete\_sibling();

//为结点增加孩子

void add\_child(const T& src);

//为结点增加兄弟

void add\_sibling(const T\* str);

//获取结点对应数据

const T get\_data();

//设置结点数据

void set\_data(const T& new\_data);

friend class my\_tree<T>;

};

### 2.2.2 二叉树类（my\_tree）

template<class T>

class my\_tree

{

public:

//默认构造函数

my\_tree();

//赋值构造函数

explicit my\_tree(tree\_node<T>\* src);

//析构函数

~my\_tree();

//获取指定结点的双亲节点

tree\_node<T>\* get\_parent(tree\_node<T>\* c, tree\_node<T>\* current\_node);

//寻找目标结点（使用布尔类型函数指针equal\_sign判断是否是目标结点）

tree\_node<T>\* find\_node(const T dest, tree\_node<T>\* current\_node, bool(\*equal\_sign)(T, T));

//前序遍历

void pre\_traversal(tree\_node<T>\* root);

//前序遍历封装

void pre\_traversal();

tree\_node<T>\* root, \* current;

};

### 2.2.3 家谱树类（geneology）

class Geneology : public my\_tree<char\*>

{

private:

//函数指针，作为查找结点时比较依据，以函数参数形式传入

bool(\*equal\_sign)(char\*, char\*);

public:

Geneology();

//完善家谱

void improve\_family();

//添加家庭成员

void add\_member();

//解散局部家庭

void delete\_family();

//改变家庭成员名称

void change\_member\_name();

};

## 2.3 家谱管理系统算法

### 2.3.1 主要思想

开始时初始化祖先结点，根据操作类型作不同处理：对于完善家庭操作，找到对应的家长结点并为其添加左儿子结点，若有多个子结点则依次添加到第一个儿子的右侧作为兄弟；对于添加家庭成员操作，找到该新结点的双亲结点，再找到双亲结点的最右一个孩子，将待添加结点添加到这个孩子的右侧；对于解散局部家庭操作，先找到该家庭的家长结点，再递归删除所有后代结点；对于改名操作，直接找到待改名结点进行单点修改即可。

### 2.3.2代码

Geneology::Geneology()

{

cout << "首先建立一个家谱！" << endl;

cout << "请输入祖先的姓名：";

char\* name = new char[MAX\_NAME\_LENGTH];

cin >> name;

tree\_node<char\*>\* new\_root = new tree\_node<char\*>(name);

root = new\_root;

equal\_sign = my\_strcmp;

cout << "此家谱的祖先是：" << root->get\_data() << endl << endl;

cout << "\*\*" << setw(14) << ' ' << "家谱管理系统" << setw(16) << ' ' << "\*\*" << endl;

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

cout << "\*\*" << setw(13) << ' ' << "请选择要执行的操作" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "A --- 完善家谱" << setw(15) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "B --- 添加家庭成员" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "C --- 解散局部家庭" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "D --- 更改家庭成员姓名" << setw(7) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "E --- 退出程序" << setw(15) << ' ' << "\*\*" << endl;

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

}

inline void Geneology::improve\_family()

{

cout << "输入要建立家庭的人的姓名：";

char\* name = new char[MAX\_NAME\_LENGTH];

cin >> name;

name[strlen(name)] = '\0';

//查找对应的结点是否在已有的家庭树中

tree\_node<char\*>\* parent = find\_node(name, root, equal\_sign);

if (parent == nullptr)

cout << "查无此人" << endl;

else

{

int num;

cout << "请输入" << name << "的儿女人数：";

cin >> num;

cout << "请依次输入" << name << "的儿女姓名：";

while (num--)

{

char\* child = new char[MAX\_NAME\_LENGTH];

cin >> child;

parent->add\_child(child);

}

cout << parent->get\_data() << "的第一代子孙是：";

tree\_node<char\*>\* child = parent->child();

while (child != nullptr)

{

cout << ' ' << child->get\_data();

child = child->sibling();

}

//清理可能留在缓冲区中的超限输入

cin.clear();

cin.ignore(1024, '\n');

cout << endl;

}

}

inline void Geneology::add\_member()

{

cout << "输入要添加儿子（或女儿）的人的姓名：";

char\* name = new char[MAX\_NAME\_LENGTH];

cin >> name;

//清理可能留在缓冲区中的超限输入

cin.clear();

cin.ignore(1024, '\n');

//查找对应的结点是否在已有的家庭树中

tree\_node<char\*>\* parent = find\_node(name, root, equal\_sign);

if (parent == nullptr)

cout << "查无此人" << endl;

else

{

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

char\* new\_child = new char[MAX\_NAME\_LENGTH];

cin >> new\_child;

//清理可能留在缓冲区中的超限输入

cin.clear();

cin.ignore(1024, '\n');

//添加子结点

parent->add\_child(new\_child);

cout << parent->get\_data() << "的第一代子孙是：";

tree\_node<char\*>\* child = parent->child();

while (child != nullptr)

{

cout << ' ' << child->get\_data();

child = child->sibling();

}

cout << endl;

}

}

inline void Geneology::delete\_family()

{

cout << "输入要解散家庭的人的目前姓名：";

char\* name = new char[MAX\_NAME\_LENGTH];

cin >> name;

tree\_node<char\*>\* current\_node = find\_node(name, root, equal\_sign);

if (current\_node == nullptr)

cout << "查无此人" << endl;

else {

cout << "要解散家庭的人的姓名是" << name << endl;

if (current\_node->child() == nullptr)

cout << current\_node->get\_data() << "暂无家庭" << endl;

else

{

cout << current\_node->get\_data() << "的第一代子孙是：";

tree\_node<char\*>\* child = current\_node->child(), \* tmp = new tree\_node<char\*>;

while (child != nullptr)

{

cout << ' ' << child->get\_data();

child = child->sibling();

}

cout << endl;

//调用删除所有子结点的函数

current\_node->delete\_children();

}

}

}

inline void Geneology::change\_member\_name()

{

cout << "输入要更改姓名的人的目前姓名：";

char\* name = new char[MAX\_NAME\_LENGTH];

cin >> name;

//清理可能留在缓冲区中的超限输入

cin.clear();

cin.ignore(1024, '\n');

tree\_node<char\*>\* current\_node = find\_node(name, root, equal\_sign);

if (current\_node == nullptr)

cout << "查无此人" << endl;

else

{

cout << "请输入更改后的姓名：";

char\* new\_name = new char[MAX\_NAME\_LENGTH];

cin >> new\_name;

//清理可能留在缓冲区中的超限输入

cin.clear();

cin.ignore(1024, '\n');

//更改名字

current\_node->set\_data(new\_name);

cout << name << "已更名为" << new\_name << endl;

}

}

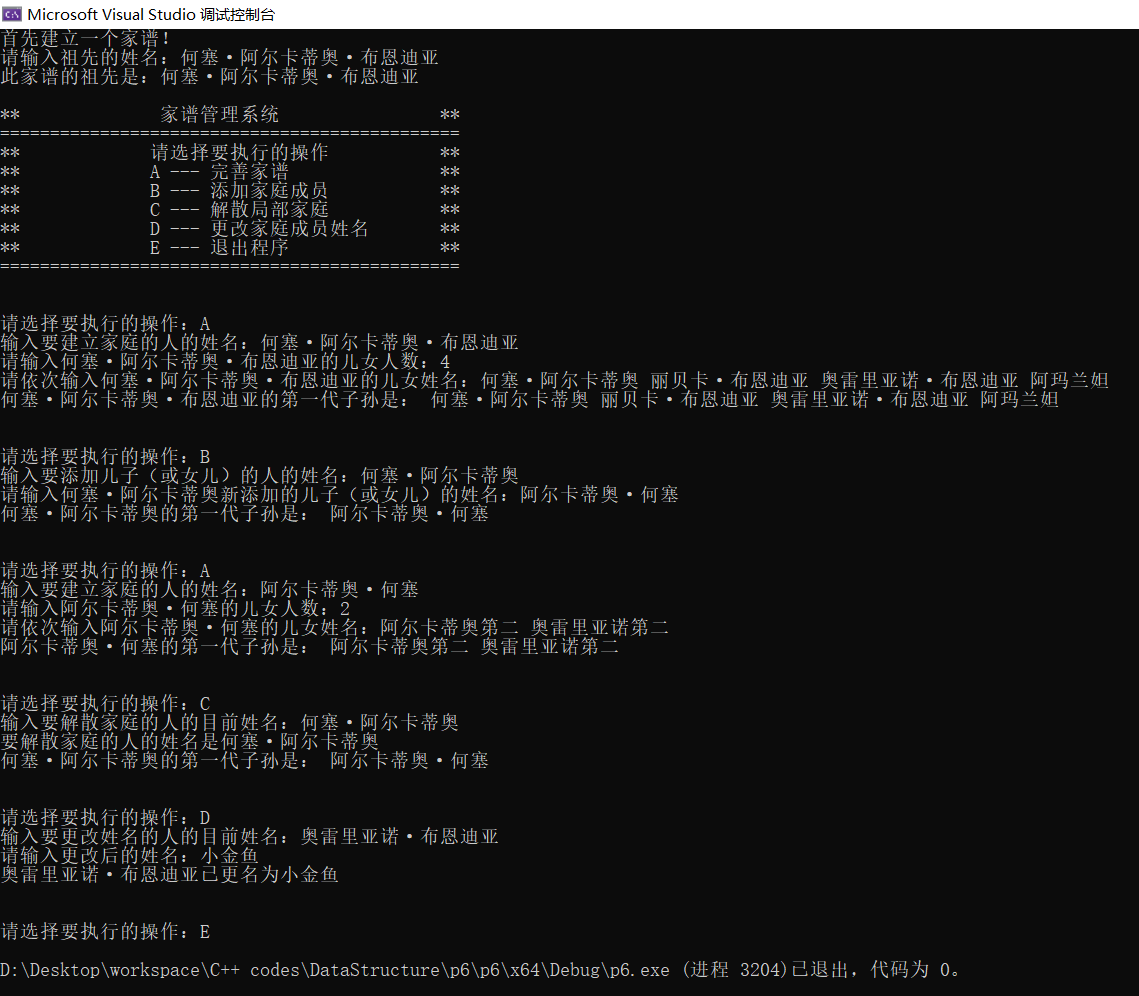
# 3 项目测试

## 3.1 正常测试四种操作

测试用例：

预期结果：

实验结果：

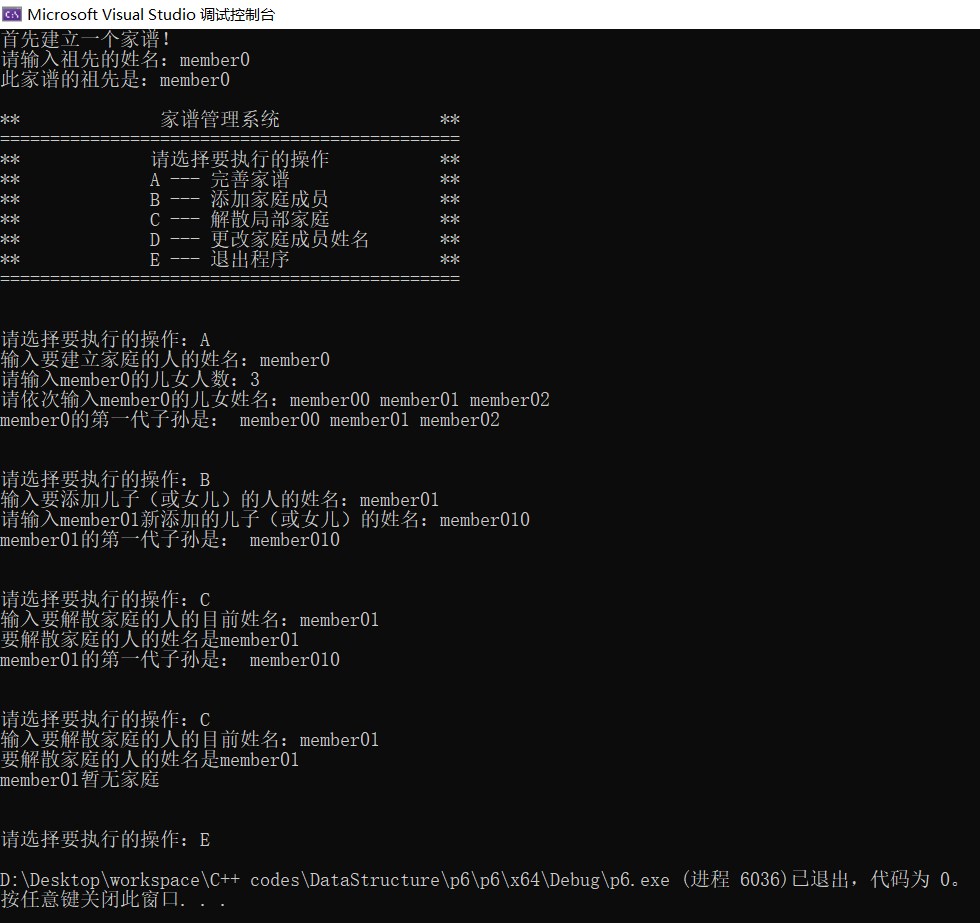


## 3.2 解散局部家庭后再解散同个家庭

测试用例：

预期结果：

实验结果：

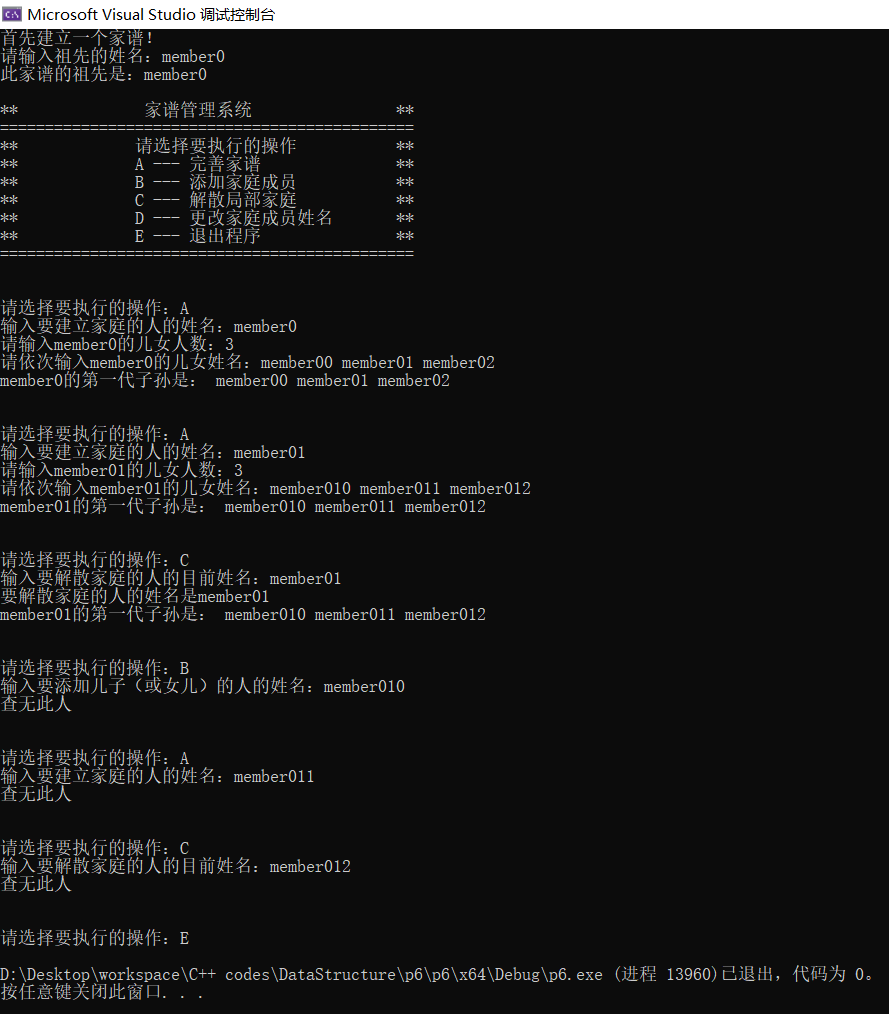


## 3.3 解散局部家庭后再查询、修改、删除该家庭成员

测试用例：

预期结果：

实验结果：

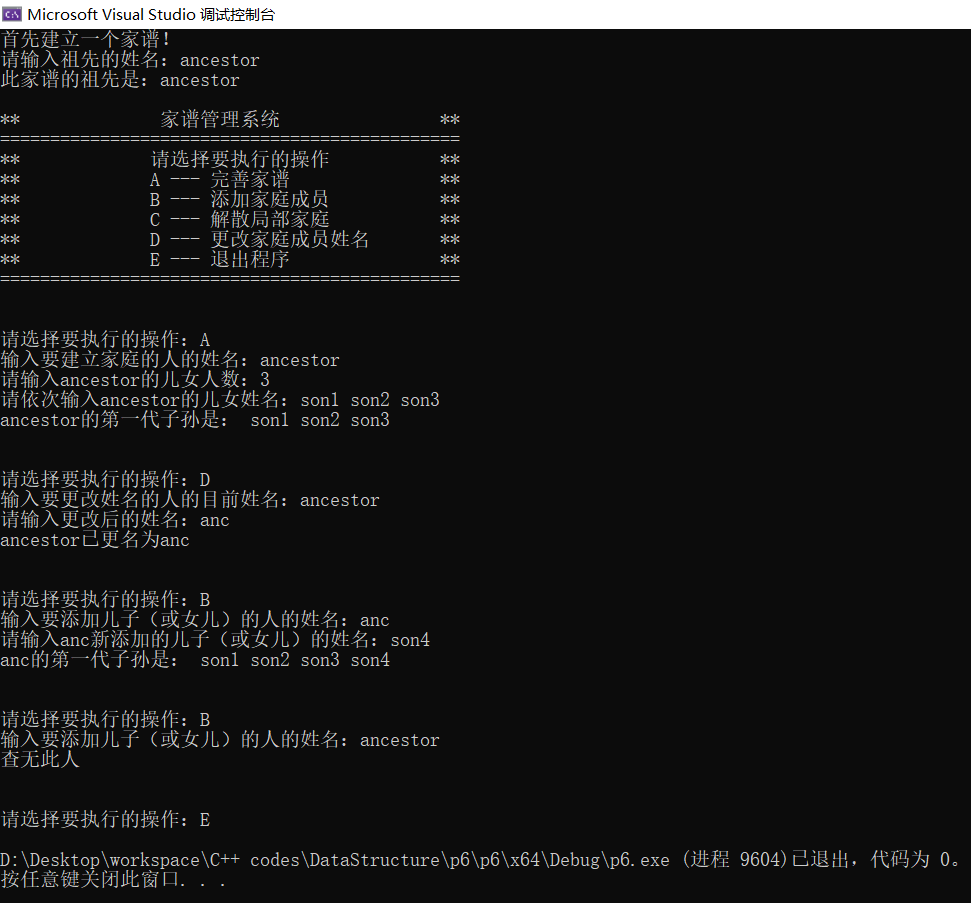


## 3.4 更名后查询家庭成员原名

测试用例：

预期结果：

实验结果：

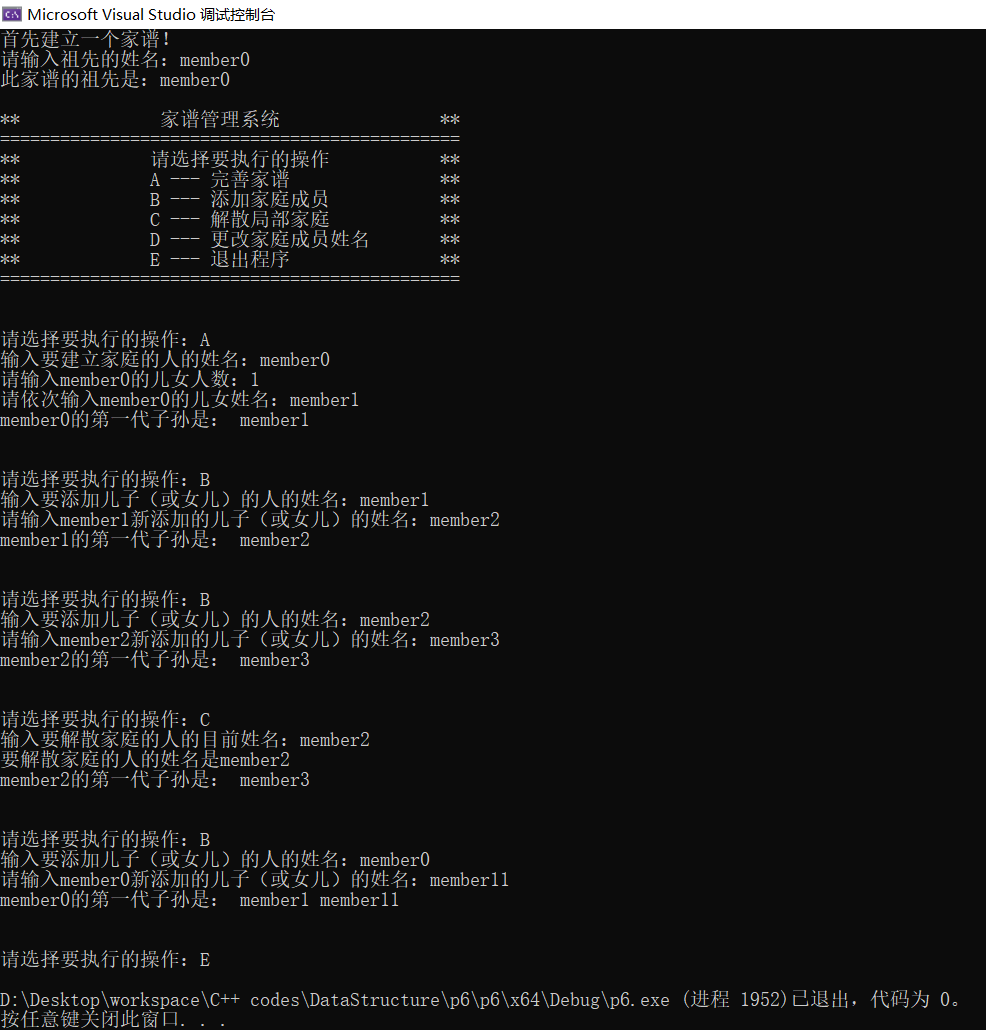


## 3.5 多代单传家庭测试

测试用例：

预期结果：

实验结果：



## 3.6 只有祖先结点家庭测试

测试用例：

预期结果：

实验结果：

