**项目说明文档**

**数据结构课程设计**

**——家谱管理系统**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二三 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.完善家谱功能 1

1.3.2.添加家庭成员功能 1

1.3.3.解散局部家庭功能 1

1.3.4.更改家庭成员姓名功能 1

1.3.5.统计家庭成员功能 2

1.3.6.异常处理功能 2

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[结构体与类设计 2](#_bookmark14)

2.2.1 LinkNode 结构体的设计 2

2.2.2 LinkedQueue 类的设计 3

2.2.3 BinTreeNode 结构体的设计 3

2.2.4 BinaryTree 类的设计 3

2.2.5 Person 结构体的设计 5

* 1. .[项目主体架构设计 5](#_bookmark32)

[3.项目功能实现 6](#_bookmark33)

3.1.项目主体架构的实现 6

3.2.完善家谱功能的实现 6

3.3.添加家庭成员功能的实现 6

3.4.解散局部家庭功能的实现 7

3.5.更改家庭成员姓名功能的实现 7

3.6.统计家庭成员功能的实现 7

4.项目测试 8

4.1.输入家庭成员姓名功能测试 8

4.2.完善家谱功能测试 8

4.3.添加家庭成员功能测试 9

4.4.解散局部家庭功能测试 9

4.5.更改家庭成员姓名功能测试 9

4.6.统计家庭成员功能测试 10

1. **项目分析**
   1. 项目背景分析

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，是珍贵的人文资料。

本项目是一个家谱管理系统，用于创建和管理一个家族的家谱。家谱管理系统用于记录和维护家族的历史、成员以及家族之间的关系。它可以帮助用户理解家族历史，追溯血缘关系，并保存重要的家族信息。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)实现对家族成员信息的完善、添加、解散、更改、统计等功能，确保数据的准确、高效管理；

(2)设计简单直观的控制台界面，使操作便捷、容易上手，适应不同用户的操作习惯；

(3)选择合适的数据结构，以支持对家族成员信息的高效操作，同时考虑信息的关联性和复杂度；

(4)实现异常处理机制，确保系统稳定性和安全性，避免因用户输入错误导致系统崩溃或信息丢失；

* 1. 项目功能分析

本项目旨在通过模拟家谱管理过程，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、统计家庭成员等功能，从而实现对家族成员的高效管理。需要设计合适的数据结构、开发用户友好的控制台界面，并考虑系统的稳定性、安全性以及未来的扩展性。通过该项目的实施，可以提高家谱管理系统的效率和准确性。下面对项目的功能进行详细分析。

* + 1. 完善家谱功能

允许用户为特定成员添加直系后代。这是建立家谱时的关键功能，它允许用户逐步构建完整的家族树。

* + 1. 添加家庭成员功能

用户可以添加新的家庭成员到现有的家谱中。

* + 1. 解散家庭成员功能

提供删除特定家庭成员的后代的选项，这对于维护准确和最新的家谱数据很重要。

* + 1. 更改家庭成员姓名功能

允许修改家庭成员的姓名。这一功能在家庭成员改名或者录入错误时非常有用。

* + 1. 统计家庭成员功能

提供统计和查看特定家庭成员及其后代的功能，便于用户了解家族规模和成员构成。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，家谱管理系统的设计中选择使用二叉树结构作为数据结构而不是一般树结构，主要基于以下几个考虑：

(1)简化结构和操作：在家谱管理中，每个家庭成员通常有多个孩子，但使用二叉树可以将这种复杂性简化。将第一个孩子作为左孩子，其余孩子作为右孩子的兄弟链表可以简化添加、删除等操作。

(2)时间和空间效率：二叉树的遍历、搜索、插入和删除操作相对一般树来说更加简单。二叉树相较于一般树在存储上更加高效。每个节点只需存储两个子节点的链接，减少了存储空间的浪费。

(3)易于扩展和修改：二叉树的结构使得对家谱的扩展和修改变得容易。例如，添加新成员或更改现有成员关系时，只需调整有限的节点链接。对于家族成员的插入和删除操作，二叉树提供了较高的灵活性和效率。

(4)便于家族层级展示：使用二叉树可以方便地展示家族成员的层级关系。通过左孩子和右孩子的关系，可以清晰地表达家族成员之间的直系和旁系血缘关系。家谱的图形化展示在使用二叉树时更加直观和易于理解。

2.2.结构体与类设计

2.2.1.LinkNode结构体的设计

LinkNode 结构体是一个用于构建链表节点的模板结构体。该结构体用于表示链表中的每个节点，其中包括节点存储的数据以及指向下一个节点的指针。本项目希望链表结点类可以直接访问链表结点，所以使用 struct 而不是 class 描述链表结点类。其数据成员和构造函数定义及含义如下：

T data：数据域，存储节点的数据。

LinkNode<T>\* link：指针域，指向下一个节点的指针。

LinkNode(LinkNode\* ptr = NULL)：构造函数，初始化指针域。

LinkNode(const T& item, LinkNode\* ptr = NULL)：构造函数，初始化数据域和指针域。

2.2.2.LinkedQueue 类的设计

该通用模板类LinkedQueue用于存储和管理数据元素，遵循先进先出的原则。该队列是基于链表数据结构实现的，链表节点由MyLinkNode 结构体表示，其中包含数据和指针域。此模板类LinkedQueue允许在队列的前端添加元素，以及从队列的前端移除元素。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

LinkNode<T>\* front：指针指向队列的前端，即队列中的第一个元素

LinkNode<T>\* rear：指针指向队列的末端，即队列中的最后一个元素

LinkedQueue()：默认构造函数，创建一个空的队列

~LinkedQueue()：析构函数，清空队列并释放内存。

bool EnQueue(const T item)：将元素添加到队列的末尾

bool DeQueue(T& item)：移除队列的前端元素并将其值通过引用返回

bool getFront(T& item) const：获取队列前端元素的值

void makeEmpty()：清空队列，释放所有元素占用的内存

bool IsEmpty() const：检查队列是否为空

int getSize() const：获取队列中元素的数量

2.2.3.BinTreeNode 结构体的设计

BinTreeNode 结构体是一个模板结构体，用于构建和表示一个二叉树的节点。二叉树是一种重要的数据结构，广泛应用于各种计算机科学领域。在这个结构体中，每个节点包含一个数据元素和两个指向其子节点的指针：左子节点和右子节点。其数据成员，构造函数定义及含义如下：

T data：数据域，存储节点的数据

BinTreeNode<T>\* leftChild：指针域，指向左子节点的指针

BinTreeNode<T>\* rightChild：指针域，指向右子节点的指针

BinTreeNode() : data(T()),leftChild(NULL), rightChild(NULL) {}：构造函数。

BinTreeNode(T item, BinTreeNode<T>\* l = NULL, BinTreeNode<T>\* r = NULL) : data(item), leftChild(l), rightChild(r) {}：构造函数，初始化数据域和指针域。

2.2.4.BinaryTree 类的设计

BinaryTree 类是一个模板类，用于创建和管理二叉树结构。二叉树是一种基础且重要的数据结构，广泛应用于多种场景，如管理、排序和搜索算法等。本模板类提供了二叉树的基本操作，包括插入、查找、遍历、修改和删除节点等功能。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

BinTreeNode<T>\* root：指向树的根节点的指针

BinTreeNode<T>\* copy(const BinTreeNode<T>\* subTree)：深拷贝一个子树。

void outputNode(BinTreeNode<T>\* subTree)：输出一个节点的数据。

void (BinaryTree::\* visitFunc)(BinTreeNode<T>\*)：函数指针，指向BinaryTree的成员函数，指向的函数接受一个指向BinTreeNode<T>类型的指针作为参数，并且返回值为void。

BinaryTree(): root(NULL) {}：默认构造函数，初始化一个空的二叉树，即根节点设置为NULL。

BinaryTree(T& item)：转换构造函数，创建一个二叉树，并设置根节点的数据为提供的 item，适 用于已知根节点数据的情况，便于直接构建含有根节点的二叉树。

BinaryTree(BinaryTree<T>& other)：复制构造函数，创建一个新的二叉树，其结构和数据是另一个二叉树（other） 的深拷贝。

~BinaryTree()：析构函数，用于在 BinaryTree 类的对象不再需要时，安全地销毁该对 象。它负责释放二叉树中所有节点占用的内存资源，防止内存泄漏。

void destroy(BinTreeNode<T>\*& subTree)：递归地销毁整个子树。

bool isEmpty(void)：检查树是否为空。

int getHeight(BinTreeNode<T>\* subTree)：计算树或子树的高度。

int getSize(BinTreeNode<T>\* subTree)：计算树或子树的高度。

BinTreeNode<T>\* getRoot(void)：获取树的根节点。

BinTreeNode<T>\* getParent(BinTreeNode<T>\* current, BinTreeNode<T>\* subTree)：查找指定节点的父节点。

BinTreeNode<T>\* getLeftChild(BinTreeNode<T>\* current)：查找指定节点的左子节点。

BinTreeNode<T>\* getRightChild(BinTreeNode<T>\* current)：查找指定节点的右子节点。

BinTreeNode<T>\* findNode(const T& item, BinTreeNode<T>\* subTree)：在树中查找包含特定数据的节点。

void inOrder(BinTreeNode<T>\* subTree, void (BinaryTree::\* visitFunc)(BinTreeNode<T>\*))：前序遍历树。

void preOrder(BinTreeNode<T>\* subTree, void (BinaryTree::\* visitFunc)(BinTreeNode<T>\*))：中序遍历树。

void postOrder(BinTreeNode<T>\* subTree, void (BinaryTree::\* visitFunc)(BinTreeNode<T>\*))：后序遍历树。

void levelOrder(BinTreeNode<T>\* subTree, void (BinaryTree::\* visitFunc)(BinTreeNode<T>\*))：层序遍历树。

bool insertFrontLeft(BinTreeNode<T>\*& subTree, const T& item)：在指定节点插入左子节点。

bool insertFrontRight(BinTreeNode<T>\*& subTree, const T& item)：在指定节点插入右子节点。

void inOrderOutput(BinTreeNode<T>\* subTree)：前序遍历树输出。

void preOrderOutput(BinTreeNode<T>\* subTree)：中序遍历树输出。

void postOrderOutput(BinTreeNode<T>\* subTree)：后序遍历树输出。

void levelOrederOutput(BinTreeNode<T>\* subTree)：层序遍历树输出。

bool modifyNode(BinTreeNode<T>\*& subTree, T& oldItem, T& newItem)：修改指定节点的数据。

BinaryTree<T>& operator=(const BinaryTree<T>& other)：重载赋值运算符，用于实现树的深拷贝。

2.2.5.Person 结构体的设计

Person 结构体用于表示一个人的基本信息.它主要包含人的姓名，并提供了重载的赋值运算符和比较运算符，以及重载输入输出运算符的友元声明，从而实现简单且直观的数据操作。在此基础上可以继续添加性别、年龄、籍贯等信息，可扩展性强。其数据成员、构造函数、析构函数、公有成员函数，运算符重载定义及含义如下：

char name[MAX\_NAME\_LENGTH + 1]：姓名

Person()：构造函数，初始化name为空字符串

Person(const char\* prompt)：构造函数，初始化name为prompt的内容

Person operator =(const Person& person)：重载赋值运算符，用于将一个Person对象的姓名复制到另一个Person对象。

bool operator ==(Person& person)：重载比较运算符，用于比较两个Person对象的姓名是否相同。

friend std::ostream& operator <<(std::ostream& out, Person& person)：重载输出运算符，用于输出Person对象的姓名。

friend std::istream& operator >>(std::istream& in, Person& person)：重载输入运算符，用于输入Person对象的姓名。

2.3.项目主体架构设计

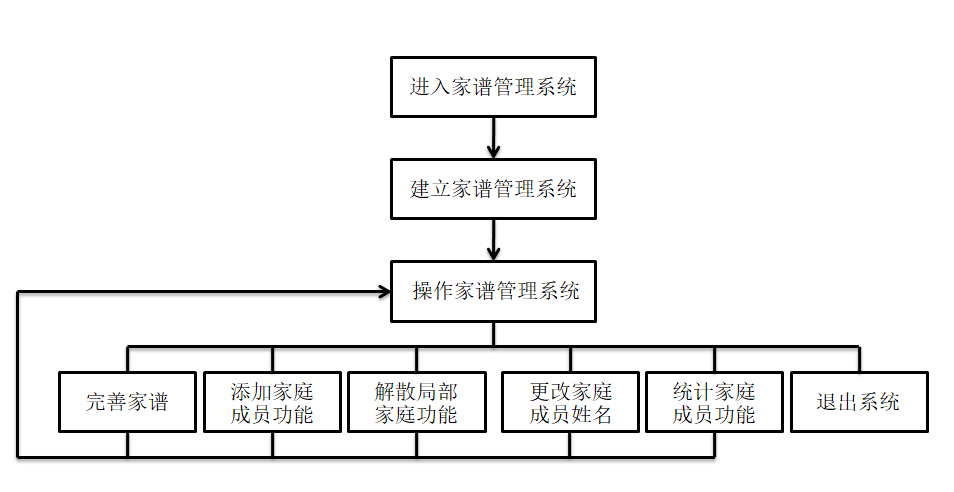


图 2.3 项目主体架构设计流程图

项目主体架构设计为：

(1)进入家谱管理系统；

(2)建立家谱管理系统；

(3)通过循环结构操作家谱管理系统，调用包括完善家谱、添加家庭成员、解散家庭成员、更改家庭成员姓名、统计家庭成员等函数，这些函数实现了对家族成员信息的各种管理和操作；

(4)由用户选择退出家谱管理系统。

**3.项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)进入家谱管理系统：系统启动时，首先进入主函数main。这是整个家谱管理系统的入口点；

(2)建立家谱管理系统：在main函数中，首先进行家谱管理系统的初始化。这包括创建家谱树的实例BinaryTree和录入祖先信息。

(3)通过循环结构操作家谱管理系统：系统进入一个循环，允许用户反复执行不同的操作。这是通过while循环结构和switch语句实现的，循环中首先调用makeChoice函数，这个函数通过用户输入来决定执行哪种操作，根据用户的选择，switch语句调用相应的函数，如：

①buildFamliy：完善家谱功能；

②addFamilyMenber：添加家庭成员功能；

③destoryLocalFamily：解散家庭成员功能；

④modifyFamilyMember：更改家庭成员姓名功能；

⑤statisticFamilyMembers：统计家庭成员功能；

这些函数利用BinaryTree类的各种方法来执行具体的家谱树操作；

(4)由用户选择退出家谱管理系统：在循环的每个周期结束时，用户可以选择退出系统。当用户选择退出，main函数执行结束，家谱管理系统关闭。

3.2.完善家谱功能的实现

完善家谱功能的函数名为buildFamliy，完善家谱功能实现的思路为：

(1)输入家庭成员姓名：函数首先提示用户输入要建立家庭的人的姓名。

(2)查找家庭成员：使用genealogy.findNode在家谱树中查找输入的成员。如果成员不存在，则输出相应的信息并返回；

(3)检查家庭状态：检查找到的家庭成员是否已经有家庭（检查左子节点是否存在）。如果已有家庭，则输出相应信息并返回；

(4)输入子女数量：如果家庭成员没有建立家庭，那么提示用户输入该成员的子女数量。

(5)录入子女信息：根据输入的子女数量，循环提示用户输入每个子女的姓名。对于每个输入的子女姓名，检查其是否已在家谱树中存在。如果存在，则提示错误信息，并要求重新输入。若子女姓名在家谱中不存在，则将其作为新节点添加到家谱树中。这是通过genealogy. insertFrontLeft和genealogy.insertFrontRight实现的；

(6)输出家庭成员信息：完成子女信息的录入后，输出该家庭成员的所有直 系子女的姓名，以确认家谱更新情况。

3.3.添加家庭成员功能的实现

添加家庭成员功能的函数名为addFamilyMenber，添加家庭成员功能实现的思路为：

(1)输入要添加子女的家庭成员姓名：用户首先输入想要为其添加子女的家庭成员的姓名；

(2)在家谱中查找该成员：使用genealogy.findNode在家谱树中查找 输入的成员姓名。如果该成员不存在于家谱中，则输出提示信息并返回；

(3)输入新子女的姓名：提示用户输入新添加的子女的姓名。检查输入的子女姓名是否已经存在于家谱中。如果已存在，则提示错误信息并要求重新输入。

(4)在家谱中添加新子女：再次查找刚输入的家庭成员的节点。如果该成员还没有子女，则将新子女添加为该成员的左子节点。如果该成员已有子女，则沿右子节点链向下查找，直到找到一个没有右子节点的子女节点，并将新子女添加为该节点的右子节点；

(5)输出更新后的家庭成员信息：输出添加新子女后，该家庭成员的所有子女的姓名，以确认家谱更新情况。

3.4.解散局部家庭功能的实现

解散家庭成员功能的函数名为destoryLocalFamily，解散家庭成员功能实现的思路为：

(1)输入要解散家庭的成员姓名：用户首先输入要解散其家庭的家庭成员的 姓名；

(2)在家谱中查找该成员：使用genealogy.findNode在家谱树中查找 输入的成员姓名。如果该成员不存在于家谱中，则输出提示信息并返回；

(3)检查家庭成员状态：检查找到的家庭成员是否有家庭。如果该成员没有家庭成员，则输出提示信息并返回；

(4)输出并解散家庭成员：输出该家庭成员的所有直系家庭成员。调用genealogy.destroy方法删除该成员的及其所有子节点，从而解散该家庭成员的家庭。输出被解散的家庭成员信息。

3.5.更改家庭成员姓名功能的实现

更改家庭成员姓名功能的函数名为modifyFamilyMember，更改家庭成员姓名功能实现的思路为：

(1)输入要更改姓名的家庭成员的当前姓名：首先输入需要更改姓名的家庭成员的当前姓名；

(2)在家谱中查找该成员：使用genealogy.findNode在家谱树中查找输入的成员姓名。如果该成员不存在于家谱中，则输出提示信息；

(3)输入新姓名：用户输入该家庭成员的新姓名。检查新姓名是否已存在于家谱中。如果新姓名已存在，则提示错误信息，并要求用户重新输入，直到输入一个在家谱中不存在的新姓名；

(4)更改家庭成员的姓名：调用genealogy.modifyNode，将家庭成员的姓名更改为新姓名；

(5)输出确认信息：输出一条确认信息，表明家庭成员的姓名已从原姓名更改为新姓名。

3.6.统计家庭成员功能的实现

统计家庭成员功能的函数名为statisticFamilyMembers，统计家庭成员功能实现的思路为：

(1)输入要统计的家庭成员姓名：用户首先输入想要统计其家庭成员的家庭成员的姓名；

(2)在家谱中查找该成员：使用genealogy.findNode在家谱树中查找输入的成员姓名。如果该成员不存在于家谱中，则输出提示信息并返回；

(3)统计家庭成员数量：使用genealogy.getSize计算该家庭成员的后代数量；

(4)输出家庭成员信息：输出该家庭成员的后代总数。如果有后代，使用outputGeneration输出所有后代的姓名。

**4.项目测试**

4.1.输入家庭成员姓名功能测试

家庭成员姓名输入要求为不超过 32 个英文字符或 16 个汉字字符组成的字符串，超出部分将被截断。

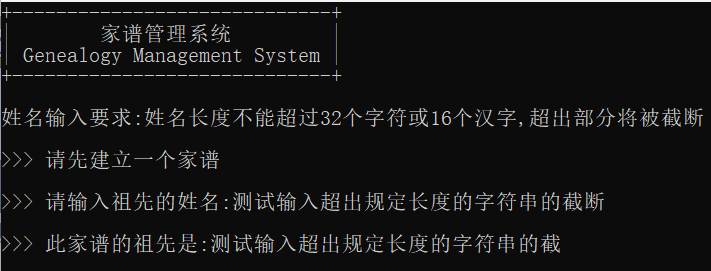


图 4.1 输入家庭成员姓名功能测试

4.2.完善家谱功能测试

输入在家谱中不存在的家庭成员，程序结束操作。

输入某家庭成员的儿女人数时，分别输入超过上下限的整数、浮点数、字符、字符串，程序要求重新输入。

输入在家谱中已存在家庭成员的家庭成员，程序结束操作，可以验证程序对输入家谱中已存在家庭成员的家庭成员的情况进行了处理。

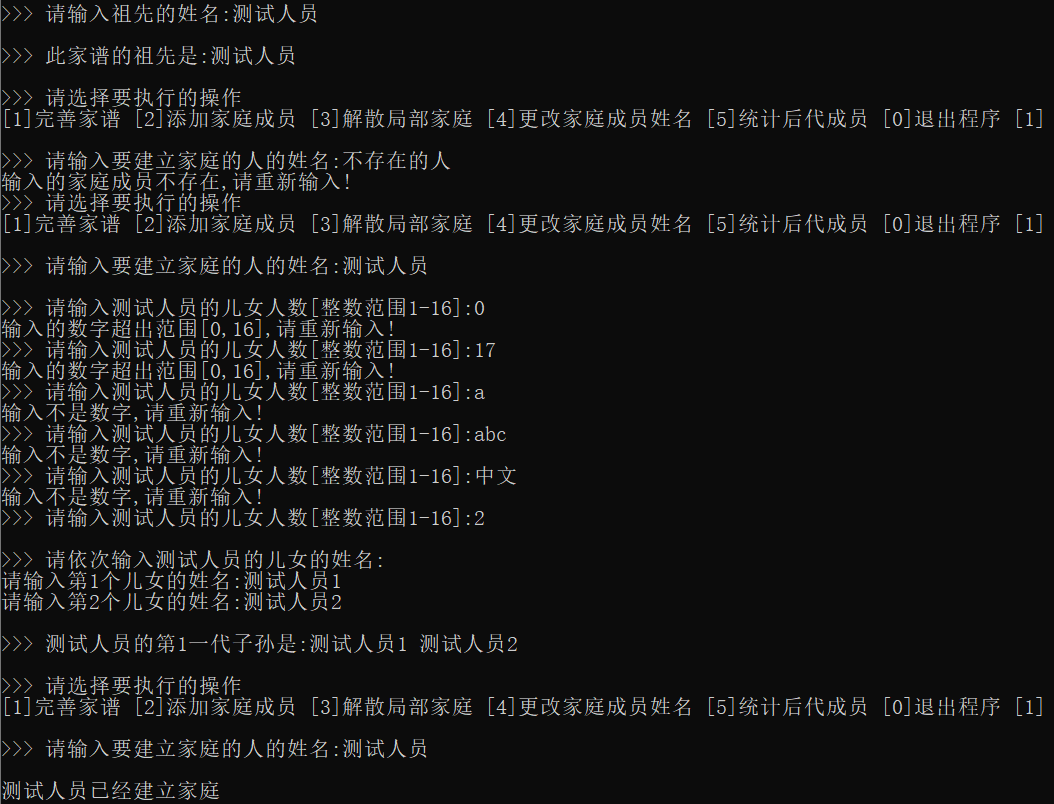


图 4.2 完善家谱功能测试

4.3.添加家庭成员功能测试

输入在家谱中不存在的家庭成员，程序结束操作。

输入某家庭成员的儿女姓名时，输入在家谱中已存在的家庭成员，程序要求重新输入。

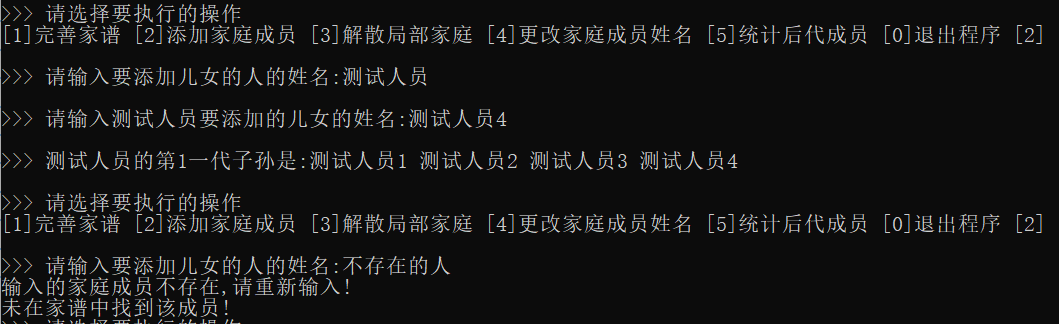


图 4.3 添加家庭成员功能测试

4.4.解散局部家庭功能测试

输入在家谱中不存在的家庭成员，程序结束操作。

输入在家谱中不存在家庭成员的家庭成员，程序结束操作。

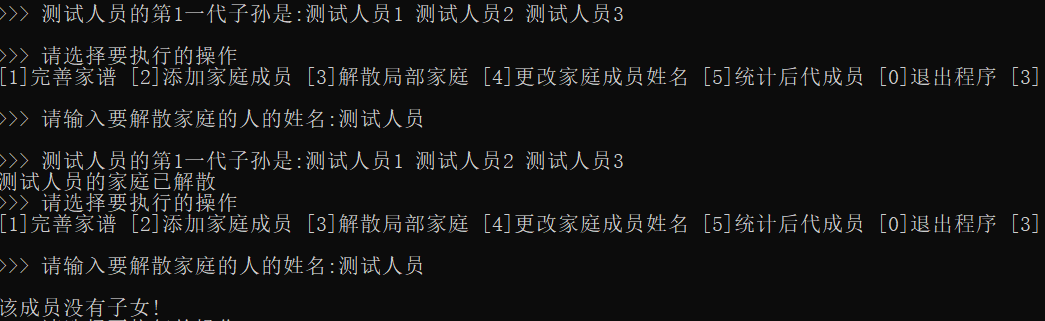


图 4.4 解散局部家庭功能测试

4.5.更改家庭成员姓名功能测试

输入在家谱中不存在的家庭成员，程序结束操作。

输入某家庭成员的更改后姓名时，输入在家谱中已存在的家庭成员，程序要求重新输入。

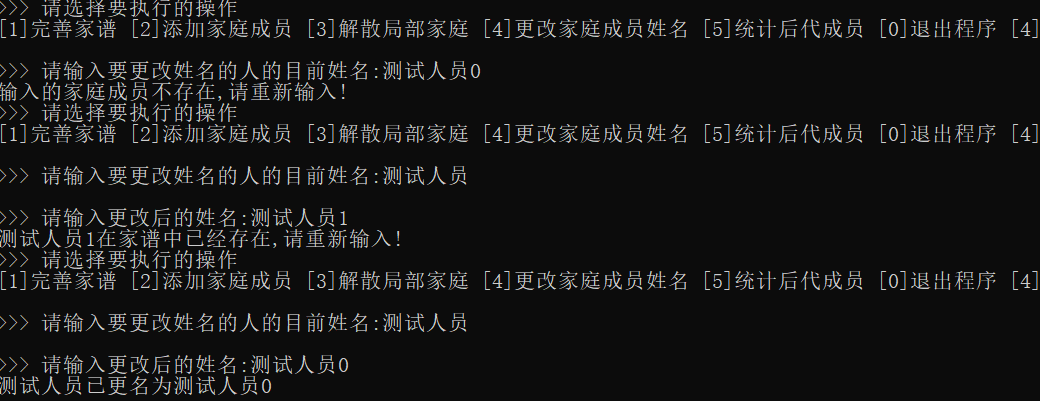


图 4.5 更改家庭成员姓名功能测试

4.6.统计家庭成员功能测试

输入在家谱中不存在的家庭成员，程序结束操作。

输入在家谱中已存在家庭成员的家庭成员，程序输出该家庭成员的儿女数量和儿女姓名。

输入在家谱中不存在家庭成员的家庭成员，程序输出该家庭成员的没有后代。

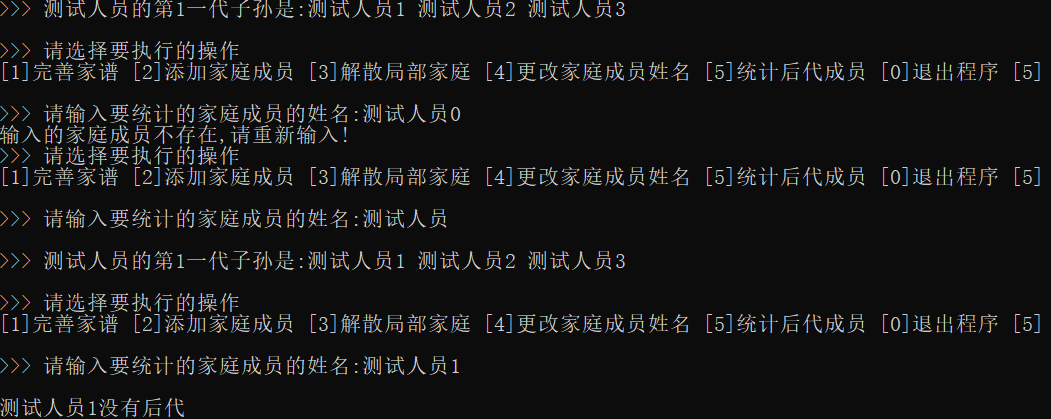


图 4.6 统计家庭成员功能测试