

项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 陆诚彬

学 号： 2254321

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 项目背景 4](#_Toc10478)

[2 项目需求分析 4](#_Toc20926)

[2.1 功能需求 4](#_Toc29572)

[2.2 非功能需求 4](#_Toc9855)

[2.3 项目输入输出需求 4](#_Toc15831)

[2.3.1 输入格式 4](#_Toc15200)

[2.3.2 输出格式 4](#_Toc15043)

[2.3.3 项目示例 5](#_Toc19604)

[3 项目设计 5](#_Toc16230)

[3.1 数据结构设计 5](#_Toc9612)

[3.1.1 treeNode 结构 5](#_Toc20731)

[3.1.2 myVector 类 5](#_Toc10237)

[3.2 类设计 6](#_Toc32309)

[3.2.1 类内主要函数 6](#_Toc29758)

[4 项目实现 7](#_Toc18625)

[4.1 建立家族树实现 7](#_Toc1356)

[4.2建立家族实现 8](#_Toc12654)

[4.3 添加家族成员实现 9](#_Toc29876)

[4.4 删除家族实现 10](#_Toc17969)

[4.5 重命名家族成员实现 11](#_Toc31165)

[4.6 查找家族成员实现 12](#_Toc5664)

[4.7 显示家谱实现 13](#_Toc9120)

[4.8 系统总体功能流程图 14](#_Toc3269)

[5 设计小结 15](#_Toc24781)

[6 软件测试 16](#_Toc17855)

[6.1 输入测试 16](#_Toc24000)

[6.1.1 正常输入 16](#_Toc24738)

[6.1.2 输入超界/非法 16](#_Toc569)

[6.2 输出测试 16](#_Toc15851)

# 1 项目背景

家谱，作为一种独特的文书体裁，是中国深厚文化传统和家族历史的重要载体。它不仅仅是记录家族血脉传承的工具，更是一份承载着丰富历史信息、社会风俗和文化特色的宝贵遗产。家谱的编纂和保存对于历史学、民俗学、人口学、社会学及经济学等领域的研究具有极高的价值，是探索中国传统社会和文化的关键。

随着信息技术的发展和数字化时代的到来，传统的家谱管理方式面临着诸多挑战。纸质家谱容易遭受物理损坏和时间侵蚀，且查找和更新信息效率低下。因此，传统家谱的数字化和信息化管理成为了一个迫切需要解决的问题。

# 2 项目需求分析

## 2.1 功能需求

1. **成员信息建立与管理：**系统应能够有效地存储和管理家族成员的个人信息，包括但不限于姓名、性别、出生日期、逝世日期、家族关系等。
2. **信息查询：**提供高效的查询功能，使用户能够轻松地查找特定家族成员的详细信息。
3. **信息插入与更新：**允许用户插入新的家族成员信息，并对现有信息进行更新和修改。
4. **信息删除：**在必要时，用户应能夠删除某些家族成员的信息。

## 2.2 非功能需求

1. **用户友好的界面：**界面应简洁明了，方便用户操作。
2. **数据安全与隐私保护：**确保家族信息的安全存储，防止数据泄露。
3. **可扩展性：**考虑到家族的不断扩展，系统应设计为可轻松扩展和升级。

## 2.3 项目输入输出需求

### 2.3.1 输入格式

输入成员的姓名或者数量。

### 2.3.2 输出格式

输出相应的提示信息。

### 2.3.3 项目示例



# 3 项目设计

## 3.1 数据结构设计

本家谱管理系统项目的核心在于高效地组织和管理家族成员信息。为此，我们设计了以下关键的数据结构：

### 3.1.1 treeNode 结构

**目的：**用于表示家族树中的每个节点，即每个家庭成员。

**结构：**

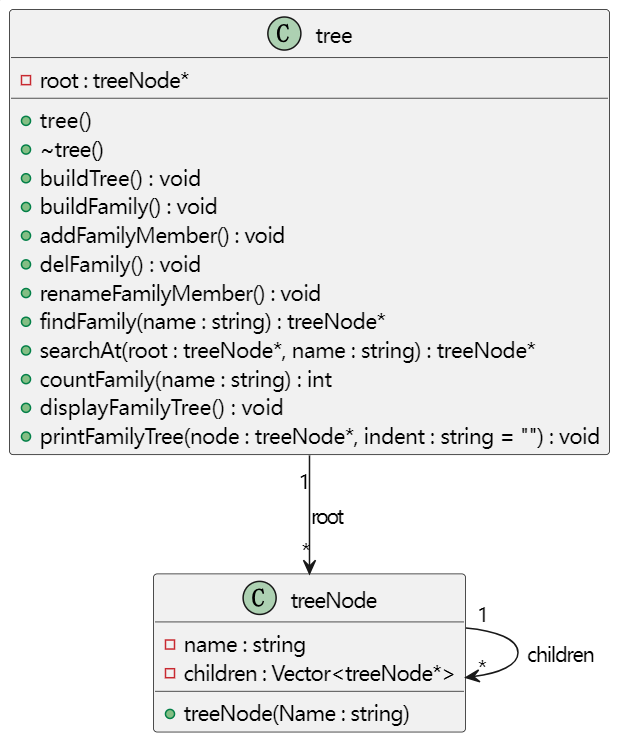
1. name: string 类型，存储家族成员的名字。
2. children: Vector<treeNode\*> 类型，动态数组，存储指向子节点的指针，即该家庭成员的后代。

### 3.1.2 myVector 类

**目的：**一个自己实现的泛型动态数组，用于存储指向 treeNode 的指针。提供动态数组的标准操作，如添加元素、访问元素、清空数组等。

## 3.2 类设计

tree 成为家谱管理系统的核心，承担着所有与家谱数据交互的重要功能，包括家谱的创建、修改、查询和展示。UML图如下：



### 3.2.1 类内主要函数

**公共方法：**

1. **void** buildTree(): 用于建立家族树的根节点。
3. **void** buildFamily(): 用于在特定家庭成员下建立家族。
5. **void** addFamilyMember(): 用于添加新的家族成员。
7. **void** delFamily(): 用于删除特定家族成员及其后代。
9. **void** renameFamilyMember(): 用于重命名家族成员。
11. treeNode\* findFamily(**const** string& name) **const**: 用于查找特定名字的家族成员。
13. treeNode\* searchAt(treeNode\* root, **const** string& name) **const**: 在指定的树节点下递归查找特定名字的家族成员。
15. **int** countFamily(**const** string& name) **const**: 计算具有特定名字的家族成员的数量。
17. **void** displayFamilyTree() **const**: 用于显示整个家谱树的结构。
19. **void** printFamilyTree(treeNode\* node, **const** string& indent = "") **const**: 辅助函数，递归打印家族树的每个成员及其子成员。

# 4 项目实现

## 4.1 建立家族树实现

### 4.1.1 建立家族树功能简介

**功能描述：** buildTree 方法用于初始化家谱树，创建根节点，即祖先的节点。

**实现细节：**

1. 首先检查 root 是否已经存在，若存在，提示家族树已建立。
2. 若 root 不存在，则提示用户输入祖先的名字，并以此创建新的树节点作为根节点。

### 4.1.2 初始化棋盘核心代码

1. **void** tree::buildTree()
2. {
3. **if** (root != nullptr)
4. {
5. cout << "已经建立了家族树" << endl;
6. **return**;
7. }
9. cout << "请输入家族的祖先的名字：";
10. string name;
11. cin >> name;
12. root = **new** treeNode(name);
13. cout << "已经建立了家族树" << endl;
14. }

## 4.2建立家族实现

### 4.2.1 建立家族功能简介

**功能描述：**

buildFamily 方法允许用户在特定成员下添加其后代，从而构建家族。

**实现细节：**

1. 首先确认家族树 (root) 已经存在。
2. 提示用户输入要建立家族的成员名字，并通过 findFamily 方法查找该成员。
3. 若找到该成员，提示输入其子女数量和名字，并将这些子女作为新节点添加到该成员的 children 中。

### 4.2.2 建立家族核心代码

1. **if** (root == nullptr) {
2. cout << "请先建立家族树" << endl;
3. **return**;
4. }
5. cout << "请输入要建立家族的人的名字：";
6. string name;
7. cin >> name;
8. treeNode\* ptr = findFamily(name);
9. **if** (ptr == nullptr) {
10. cout << "没有找到这个人" << endl;
11. **return**;
12. }
13. cout << "请输入" << name << "的儿女个数：";
14. **int** num;
15. cin >> num;
16. cout << "请依次输入" << name << "的儿女的姓名：";
17. **for** (**int** i = 0; i < num; ++i) {
18. cin >> name;
19. **if** (!countFamily(name)) {
20. ptr->children.push\_back(**new** treeNode(name)); // 添加新成员
21. }
22. **else** {
23. cout << "姓名为 " << name << " 的人已经存在" << endl;
24. }
25. }
26. cout << "已经建立了家族" << endl;

## 4.3 添加家族成员实现

### 4.3.1 添加家族成员功能简介

**功能描述：**

addFamilyMember 方法用于在特定家族成员下添加新的成员。

**实现细节：**

1)确认家族树已存在。

2)提示输入要添加子女的家族成员名字，通过 findFamily 查找该成员。

3)若找到，提示输入新成员的名字，并作为新节点添加到 children。

### 4.3.2 添加家族成员核心代码

1. **void** tree::addFamilyMember()
2. {
3. **if** (root == nullptr)
4. {
5. cout << "请先建立家族树" << endl;
6. **return**;
7. }
9. cout << "请输入要添加家族成员的人的名字：";
10. string name;
11. cin >> name;
12. treeNode\* ptr = findFamily(name);
13. **if** (ptr == nullptr)
14. {
15. cout << "没有找到这个人" << endl;
16. **return**;
17. }
19. cout << "请输入" << name << "新添加儿子（或女儿）的姓名：";
20. cin >> name;
21. **if** (!countFamily(name)) {
22. ptr->children.push\_back(**new** treeNode(name)); // 添加新成员
23. }
24. **else** {
25. cout << "姓名为 " << name << " 的人已经存在" << endl;
26. }
27. cout << "已经添加了家族成员" << endl;
28. }

## 4.4 删除家族实现

### 4.4.1 删除家族功能简介

**功能描述：**

delFamily 方法用于删除特定成员及其所有后代。

**实现细节：**

1. 确认家族树已存在。
2. 提示输入要删除的家族成员名字，通过 findFamily 查找该成员。
3. 若找到，清空该成员的 children 向量，从而删除所有后代。

### 4.4.2 删除家族核心代码

1. **void** tree::delFamily()
2. {
3. **if** (root == nullptr)
4. {
5. cout << "请先建立家族树" << endl;
6. **return**;
7. }
9. cout << "请输入要删除家族的人的名字：";
10. string name;
11. cin >> name;
12. treeNode\* ptr = findFamily(name);
13. **if** (ptr == nullptr)
14. {
15. cout << "没有找到这个人" << endl;
16. **return**;
17. }
19. cout << "要解散家族的人是" << name << endl;
20. ptr->children.clear();
21. cout << "已经删除了家族" << endl;
22. }

## 4.5 重命名家族成员实现

### 4.5.1 重命名家族成员功能简介

**功能描述：**

renameFamilyMember 方法用于更改特定家族成员的名字。

**实现细节：**

1. 确认家族树已存在。
2. 提示输入要重命名的成员的当前名字，并通过findFamily查找该成员。
3. 若找到，提示输入新名字，并更新该成员的name属性。

### 4.5.2 重命名家族成员核心代码

1. **void** tree::renameFamilyMember()
2. {
3. **if** (root == nullptr)
4. {
5. cout << "请先建立家族树" << endl;
6. **return**;
7. }
8. cout << "请输入要重命名的人的名字：";
9. string name;
10. cin >> name;
11. treeNode\* ptr = findFamily(name);
12. **if** (ptr == nullptr)
13. {
14. cout << "没有找到这个人" << endl;
15. **return**;
16. }
18. cout << "请输入" << name << "的新名字：";
19. cin >> name;
20. **if** (!countFamily(name)) {
21. cout << "已经将" << ptr->name << "重命名为" << name << endl;
22. ptr->name = name;
23. }
24. **else** {
25. cout << "姓名为 " << name << " 的人已经存在" << endl;
26. }
27. cout << "已经重命名了家族成员" << endl;
28. }

## 4.6 查找家族成员实现

### 4.6.1 查找家族成员功能简介

**功能描述：**

findFamily 方法用于在家族树中查找具有特定名字的成员。

**实现细节：**

从根节点 root 开始，递归调用 searchAt 方法在树中搜索名字。

### 4.6.2 查找家族成员核心代码

1. treeNode\* tree::findFamily(**const** string& name) **const**
2. {
3. **if** (root == nullptr)
4. **return** nullptr;
5. **else**
6. **return** searchAt(root, name);
7. }
9. treeNode\* tree::searchAt(treeNode\* root, **const** string& name) **const**
10. {
11. **if** (root->name == name)
12. **return** root; // 找到了
13. treeNode\* result = nullptr;
14. **for** (auto ptr : root->children)
15. {
16. result = searchAt(ptr, name); // 递归查找
17. **if** (result != nullptr)
18. **break**;
19. }
20. **return** result;
21. }

## 4.7 显示家谱实现

### 4.7.1 显示家谱功能简介

**功能描述：**

displayFamilyTree 方法用于以树状格式显示整个家族的结构。

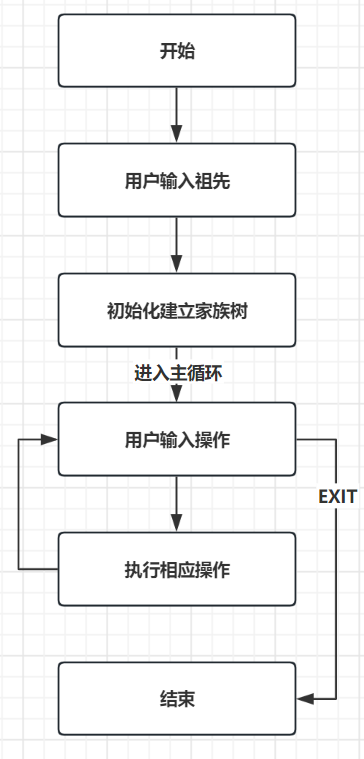
**实现细节：**

1. 首先检查 root 是否为空，若为空，提示家谱树为空。
2. 若不为空，则递归调用 printFamilyTree 方法，从根节点开始打印每个成员及其后代。

### 4.7.2 显示家谱核心代码

1. **void** tree::displayFamilyTree() **const** {
2. **if** (root == nullptr) {
3. cout << "家谱树为空。" << endl;
4. } **else** {
5. cout << "家谱树：" << endl;
6. printFamilyTree(root); //从根开始打印家谱树
7. }
8. }
10. **void** tree::printFamilyTree(treeNode\* node, **const** string& indent) **const** {
11. **if** (node != nullptr) {
12. cout << indent << "- " << node->name << endl; //打印当前节点
13. **for** (auto child : node->children) {
14. printFamilyTree(child, indent + "  "); //递归打印子节点，增加缩进
15. }
16. }
17. }

## 4.8 系统总体功能流程图



# 5 设计小结

本项目旨在开发一个家谱管理系统，实现了创建家族树、管理家族成员、显示家谱等核心功能。系统通过命令行界面与用户交互，允许用户进行家族成员的添加、修改、查询和删除操作。项目的设计注重了用户交互的直观性和数据结构的高效性。

## 5.1 数据结构和类设计

使用 treeNode 结构来代表家谱树中的每个成员。treeNode 包含成员名字和指向其子成员的指针列表。

tree 类是系统的核心，提供了构建家族树、添加成员、删除家族、重命名成员等方法。方法设计允许灵活的家族成员管理，包括添加新成员、扩展现有家族、重命名成员等。

## 5.2 功能与非功能需求

功能涵盖从家族树的创建到家族成员的管理，每个功能都有明确的逻辑流程。提供了用户友好的命令行界面，使得用户可以轻松选择和执行不同的操作。实现了家族树的递归显示，使得家族关系一目了然。

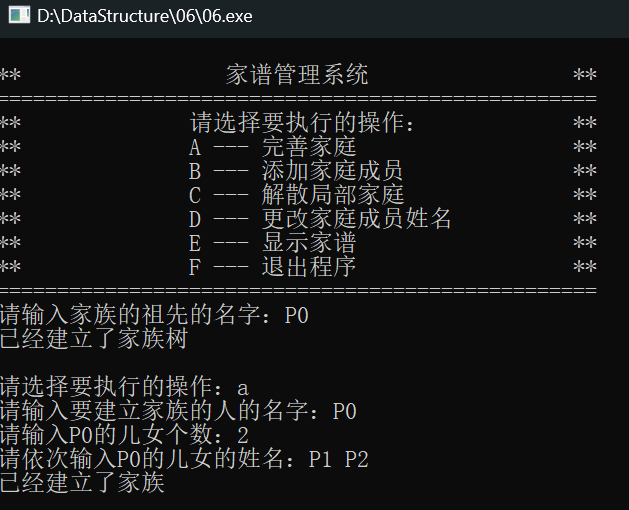
通过命令行界面，用户可以轻松地进行操作选择和数据输入。系统提供清晰的指示和反馈，确保用户在每个步骤都能明确自己的操作。

本家谱管理系统项目成功地将传统的家族记录方式数字化，提供了一个高效、易用的解决方案来管理和展示家族信息。该系统的设计和实现考虑了易用性和灵活性，同时保留了家谱文化的核心价值。未来的改进可能包括图形界面的引入、更复杂的家族关系处理、以及数据持久化等功能。

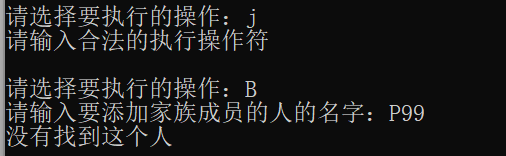
# 6 软件测试

## 6.1 输入测试

### 6.1.1 正常输入

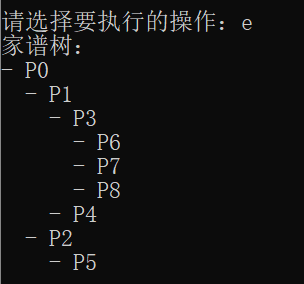


### 6.1.2 输入超界/非法



**结论：**符合输入逻辑判断

## 6.2 输出测试



**结论：**符合输出逻辑，且正确。