项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 祝新元

学 号： 1751629

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 分析 1](#_Toc532243492)

[1.1 背景分析 1](#_Toc532243493)

[1.2 功能分析 1](#_Toc532243494)

[2 设计 2](#_Toc532243495)

[2.1 数据结构设计 2](#_Toc532243496)

[2.2 类结构设计 2](#_Toc532243497)

[2.3 成员与操作设计 2](#_Toc532243498)

[2.4 系统设计 4](#_Toc532243499)

[3 实现 4](#_Toc532243500)

[3.1 建立家庭的实现 4](#_Toc532243501)

[3.1.1 建立家庭核心代码 4](#_Toc532243502)

[3.1.2 建立家庭截屏示例 5](#_Toc532243503)

[3.2 添加家庭成员的实现 5](#_Toc532243504)

[3.2.1 添加家庭成员核心代码 5](#_Toc532243505)

[3.2.2 添加家庭成员截屏示例 7](#_Toc532243506)

[3.3 解散家庭的实现 7](#_Toc532243507)

[3.3.1 核心代码 7](#_Toc532243508)

[3.3.2 解散家庭截图示例 8](#_Toc532243509)

[3.4 修改姓名的实现 8](#_Toc532243510)

[3.4.2 修改姓名核心代码 8](#_Toc532243511)

[3.4.3 修改功能截屏示例 10](#_Toc532243512)

[3.5 总体系统的实现 10](#_Toc532243513)

[3.6.1 总体系统核心代码 10](#_Toc532243514)

[3.6.2 总体系统截屏示例 12](#_Toc532243515)

[4 测试 13](#_Toc532243516)

[4.1 功能测试 13](#_Toc532243517)

[4.1.1 完善家庭测试 13](#_Toc532243518)

[4.1.2 添加家庭成员测试 14](#_Toc532243519)

[4.1.3 解散家庭测试 14](#_Toc532243520)

[4.1.4 修改功能测试 15](#_Toc532243521)

# 1 分析

## 1.1 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，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.

随着计算机科学技术的不断成熟，使用计算机对家谱系统进行管理，具有手工管理所无法比拟的优势。因此，开发一套家谱管理系统具有十分重要的意义。

## 1.2 功能分析

本项目的实质是完成兑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。

综上所述，一个考试报名系统至少应该具有输入、输出、插入、删除、修改、退出的功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的增加、删除、修改操作，并且涉及到继承的关系，每个家庭的子女个数不确定，为了对二叉树进行增加、删除等操作，考虑使用左子女右兄弟的链表树结构。同时，为了实现简易，在第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。

## 2.2 类结构设计

包括两个抽象数据类型（ADT）——结点类（TreeNode）与二叉树类（Tree），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述结点类（TreeNode），这样使得二叉树类（Tree）可以访问链表结点。

同时为体现面向对象编程的思想，在设计时所采用的数据结构均为模板类，template<class T>，之后不在重复。

## 2.3 成员与操作设计

**结点类（**TreeNode**）**

**公有成员：**

T data;//结点数据

TreeNode<T>\* firstChild, \*nextSibling;//子女及兄弟节点

**公有操作：**

TreeNode(T value = 0,TreeNode<T> \*fc = NULL,TreeNode<T> \*ns = NULL)

:data(value),firstChild(fc),nextSibling(ns) {}//构造函数

**二叉树类（**Tree**）**

**私有成员：**

TreeNode<T> \*root, \*current; //指针指向根结点和当前结点

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);//在以t为根节点的树中搜索p的父节点

公有操作：

bool Tree<T>::FirstChild()

//在树中找到当前节点的长子，并使之成为当前节点

bool Tree<T>::NextSibling()

//在树中找到当前节点的下一个兄弟，并使之成为当前节点

Tree<T>::Tree()

//二叉树的构造函数，将根节点和当前结点设为空

bool Tree<T>::Parent()

//在树中找当前节点的父节点，使之成为当前节点

bool Tree<T>::Find(T value)

//在树中搜索含有target的结点，使之成为当前结点

void Tree<T>::AddChild(T parName, T childName)

//添加左子女结点

void Tree<T>::AddSibling(T childName, T Sibling)

//添加右兄弟结点

void Tree <T> ::BuildRoot(T rootVal)

/建立树的根结点,并使之成为树的当前结点

void Tree<T>::showAllChild(T parName)

//展示t结点的所有孩子

bool Tree<T>::RemoveFromPar(T parName)

//删除t结点

T Tree<T>::GetFinalSibling(T parName)

//得到当前结点的最后一个兄弟

void Tree<T>::ChangeName(T nowName, T chName)

//将nowName修改成chName

## 2.4 系统设计

系统首先完成对家谱的创建和输入数据工作，然后根据用户所输入的操作码执行对应的成员函数。定义一个vector来存储要加入家庭的子女名。

# 3 实现

## 3.1 建立家庭的实现

### 3.1.1 建立家庭核心代码

//t是二叉树，childNum是子女数量，parName为建立家庭的人,chName记录了要加入家庭的子女名

//没有发现这个人

if (!t->Find(parName))

{

cout << "查无此人" << endl;

continue;

}

chName.clear();

//输入要加入家庭的子女们

for (int i = 0; i < childNum; i++)

{

string childName;

cin >> childName;

chName.push\_back(childName);

}

//子女们加入家庭，都成为第一个子节点的兄弟

t->AddChild(parName, chName[0]);

for (int i = 0; i < childNum-1; i++)

{

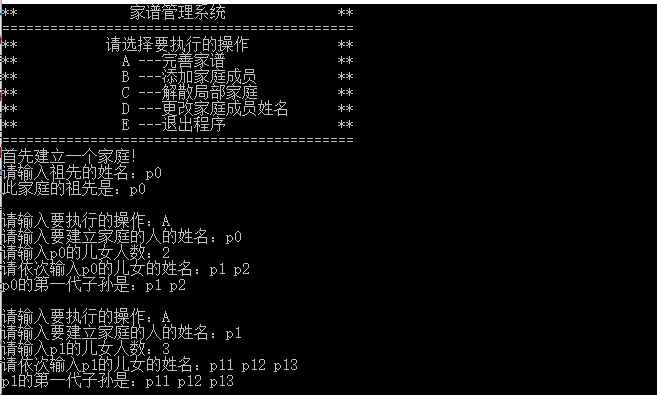
t->AddSibling(chName[i], chName[i + 1]);

}

cout << parName << "的第一代子孙是：";

t->showAllChild(parName);

### 3.1.2 建立家庭截屏示例



## 3.2 添加家庭成员的实现

### 3.2.1 添加家庭成员核心代码

template<class T>

void Tree<T>::AddChild(T parName, T childName)

{

//将当前结点设为parName，将它的子节点设为childName

if (Find(parName))

{

TreeNode<T> \*p = new TreeNode<T>(childName);

current->firstChild = p;

}

}

template<class T>

void Tree<T>::AddSibling(T childName, T Sibling)

{

//将当前结点设为childName，将它的右兄弟设为Sibling

if (Find(childName))

{

TreeNode<T> \*s = new TreeNode<T>(Sibling);

current->nextSibling = s;

}

}

//t是二叉树，parName为建立家庭的人，childName是新添加的子女的姓名

//没有发现这个人

if (!t->Find(parName))

{

cout << "查无此人" << endl;

continue;

}

//如果父节点原来没有子女，直接添加，否则加在最后

if (!t->FirstChild())

{

t->AddChild(parName, childName);

}

else

{

string finalSibling = t->GetFinalSibling(parName);

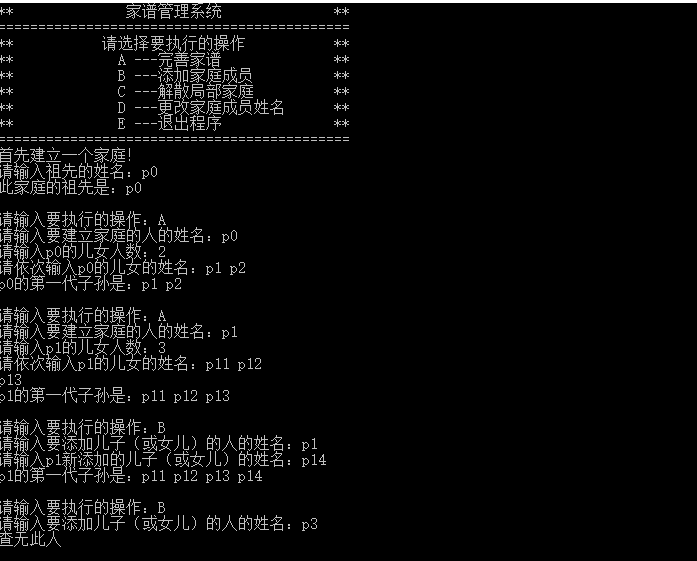
t->AddSibling(finalSibling, childName);

}

cout << parName << "的第一代子孙是：";

t->showAllChild(parName);

### 3.2.2 添加家庭成员截屏示例



## 3.3 解散家庭的实现

### 3.3.1 核心代码

template<class T>

bool Tree<T>::RemoveFromPar(T parName)

{

//找到当前结点并且删除

if (Find(parName))

{

current->firstChild = NULL;

current->nextSibling = NULL;

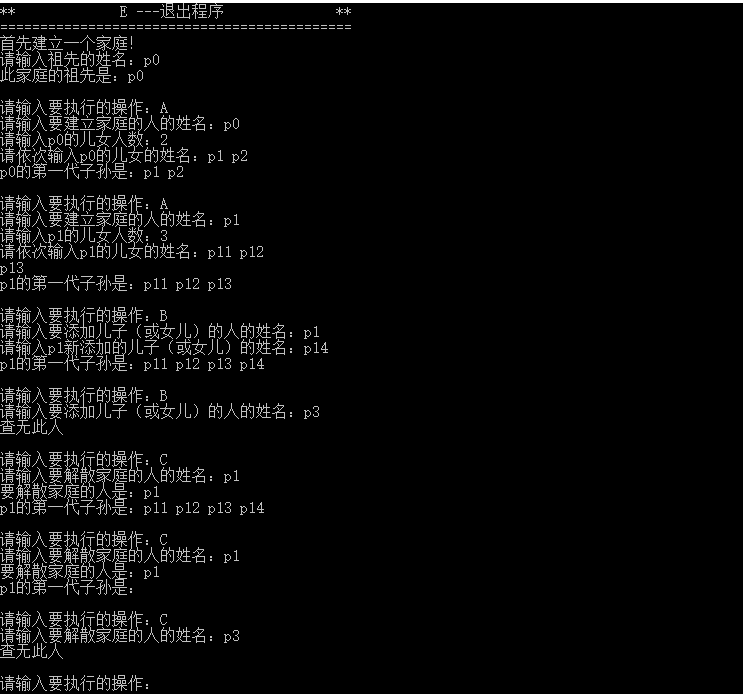
return true;

}

return false;

}

### 3.3.2 解散家庭截图示例



## 3.4 修改姓名的实现

### 3.4.2 修改姓名核心代码

template<class T>

void Tree<T>::ChangeName(T nowName, T chName)

{

//如果找到了nowName就更改其名字

if (Find(nowName))

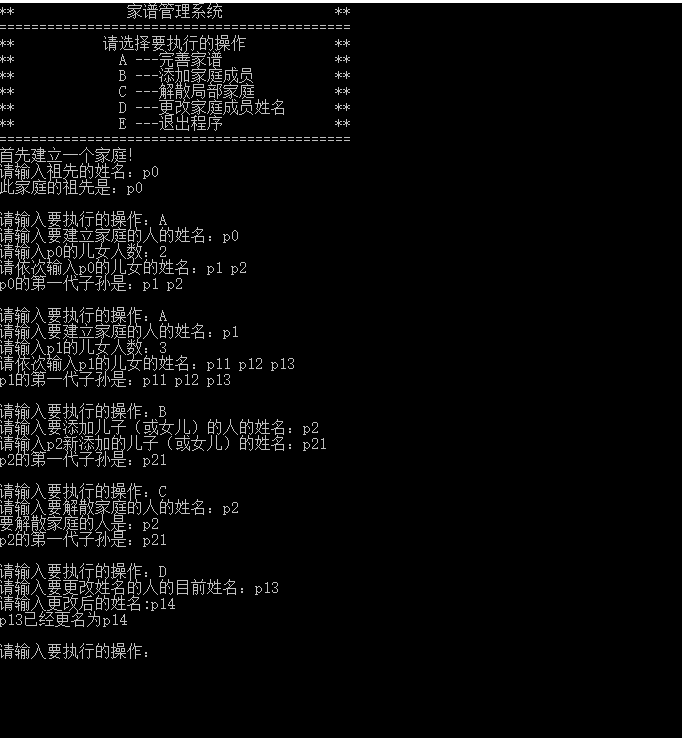
{

current->data = chName;

}

}

### 3.4.3 修改功能截屏示例



## 3.5 总体系统的实现

### 3.6.1 总体系统核心代码

while (1)

{

cout << endl;

cout << "请输入要执行的操作：";

cin >> order;

switch (order)

{

case 'A': （建立家庭） continue;

case 'B': （添加成员） continue;

case 'C': （解散家庭） continue;

case 'D': （更改家庭成员姓名） continue;

case 'E': exit(1);

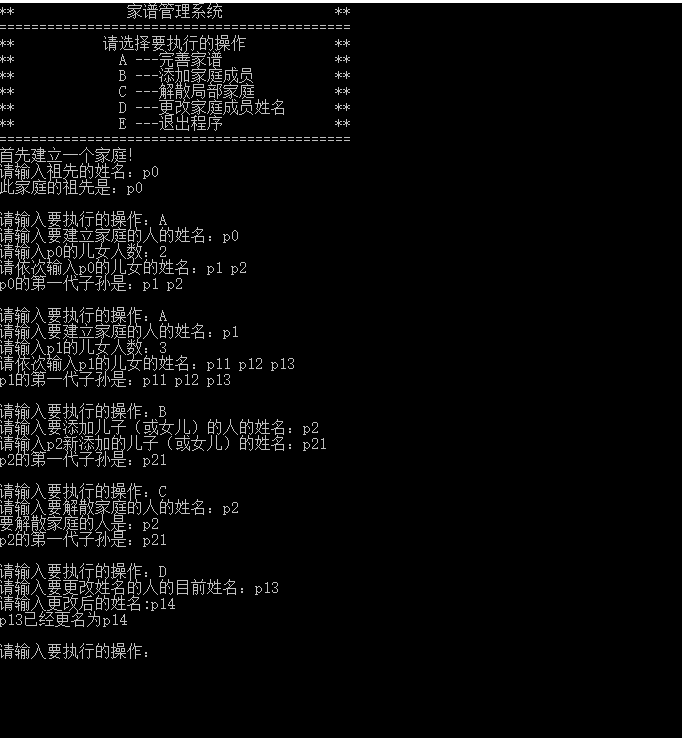
//如果用户输入操作要求错误，提示用户重新输入

default: cout << "抱歉，您想要进行的操作不存在，请重新输入！" << endl; continue;

}

}

### 3.6.2 总体系统截屏示例



# 4 测试

## 4.1 功能测试

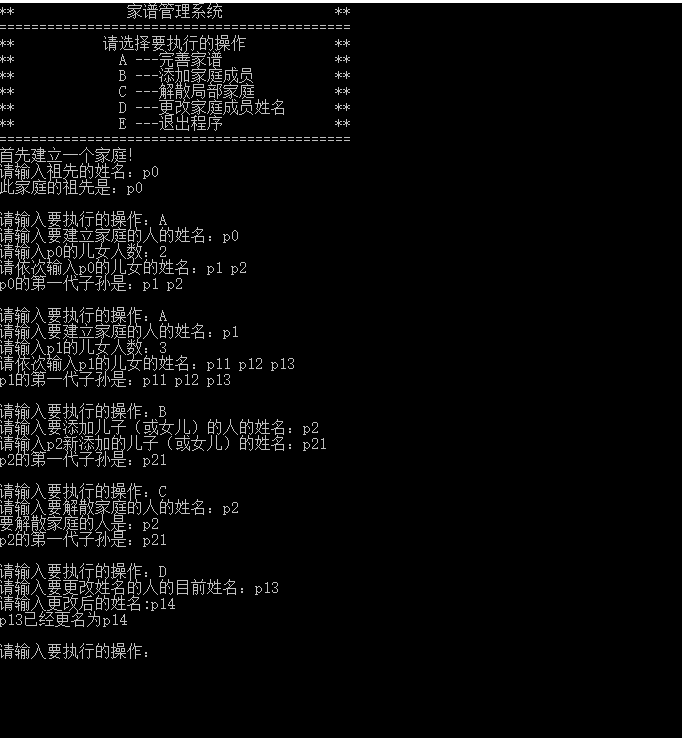
### 4.1.1 完善家庭测试

**测试用例**：2 p1 p2

**预期结果**：

p1 p2

**实验结果**



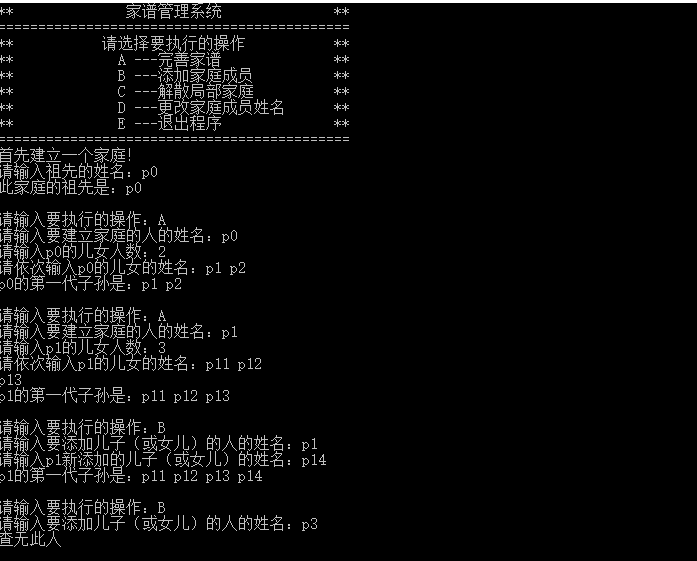
### 4.1.2 添加家庭成员测试

**测试用例：**p3

**预期结果：**

查无此人

**实验结果：**



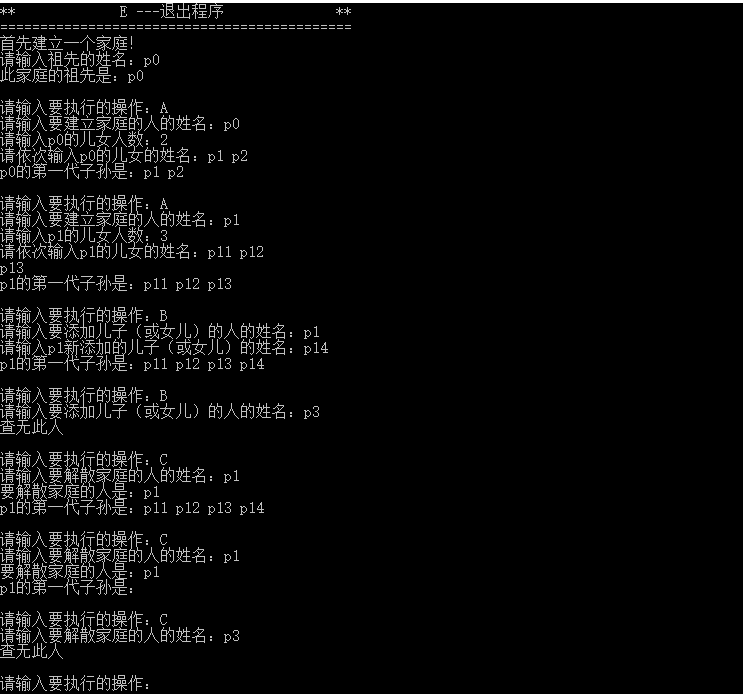
### 4.1.3 解散家庭测试

**测试用例：p3**

**预期结果：**

查无此人

**实验结果：**



### 4.1.4 修改功能测试

**测试用例：**p13 ->p14

**预期结果：**

p14已经更名为p14

**实验结果：**

