项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 崔鑫宇

学 号： 1853444

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

家谱是一种中国传统的表谱形式，记载一个以血缘关系为主体的家族世袭繁衍和重要人物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。但随着代数的增长与家系的增大，如何管理如此庞大的数据显得极为复杂，传统的手工管理工作量大且容易出错。

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。由于计算机快速稳定的特点，相比于以往的人工管理方式有着显而易见的优势。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

作为一个最简易的家谱管理系统，首先应该可以实现查看祖先和子孙的个人信息的功能，这也是最基本的功能。其次还应该可以实现插入家庭成员、删除家庭成员、 以及修改家庭成员身份信息的功能。

综上所述，一个家谱管理系统至少应该至少可以兑现完成对家谱成员信息的建立、查找、插入、修改以及删除等功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，家谱管理系统要求大量的增加、删除、修改、插入操作，且一个节点可能拥有多个子节点，因此考虑使用树数据结构。然后将每个功能作为一个成员函数来完成对数据的操作，完成最后的主函数以验证各个函数功能并得到运行结果。

## 2.2 类结构设计

在建立家谱树的类结构之前中，我先定义了成员函数的结构体（Members），然后才正式进行类的结构设计，Members结构体中包含一个字符串用来存储每个成员的姓名，其中还包含两个返回Members\*的指针。而在类结构中除了题目中要求的那几个成员函数外我又额外添加了查找函数以及添加子节点和兄弟节点的函数

## 2.3 成员与操作设计

**存储成员信息的结构体（Members）**

string name //用来存储成员姓名

Members\* FirstChildNode //子节点

Members\* NextBrotherNode//兄弟节点

**家谱管理系统类（GenealogyManagement）**

成员函数：

void buildGenealogy（Members\*）//建立家谱系统

void addMember（Members\*）//添加成员

void deleteBranch（Members\*）//删除某一分支

void changeName（Members\*）//修改某一成员姓名

Members\* findParentNode（Members\* ，string）//找到目的节点

void addSonNode（Members\* ，string）//添加子节点

void addBrotherNode（Members\* ，string）//添加兄弟节点

## 2.4 系统设计

系统首先调用PrintOption()函数实现对菜单的创建，然后自动完成家谱树的创建，然后根据用户所输入的操作码（option）执行树的对应的成员函数。

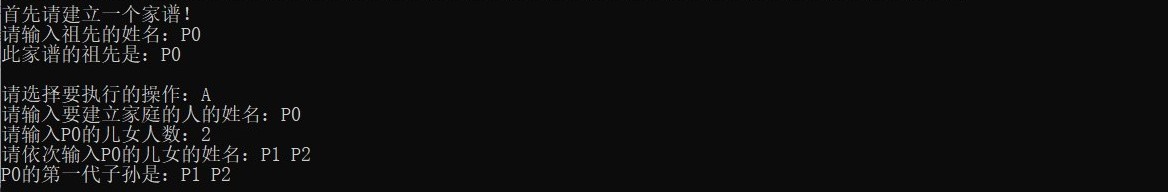
# 3 实现

## 3.1 建立家谱树

### 3.1.1 建立家谱树实现方式

首先请使用者给定祖先节点，即（root）的姓名，然后以创建一个新节点，其成员变量name为root的姓名。

### 3.1.2 建立家谱树截屏示例

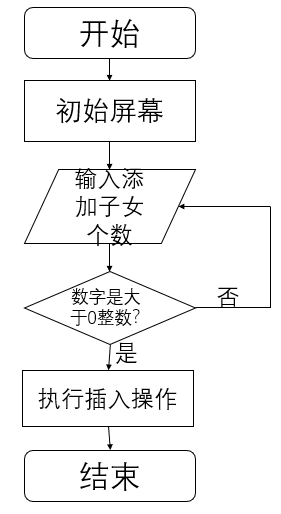


## 

## 

## 3.2 添加成员功能的实现

### 3.2.1 添加成员功能流程图



### 3.2.2 添加成员功能核心代码

void GenealogyManagement::addSonNode(Members\* parent,string addName) {

if (parent->FirstChildNode == NULL) {//若上一辈子节点为空添加一新结点在他的子结点上

Members\* childNode = new Members;

parent->FirstChildNode = childNode;

childNode->name = addName;//否则在他的子节点的兄弟节点上添加一个新结点

}

else

addBrotherNode(parent->FirstChildNode, addName);

}

//成员函数：添加子节点

void GenealogyManagement::addBrotherNode(Members\* brother, string addName) {

if (brother->NextBrotherNode == NULL) {//寻找第一个空的兄弟节点

Members\* BrotherNode = new Members;

brother->NextBrotherNode = BrotherNode;

BrotherNode->name = addName;

}

else

addBrotherNode(brother->NextBrotherNode, addName);

}

//成员函数：添加兄弟节点

void GenealogyManagement::addMember(Members\* root) {

cout << "请输入要添加儿子（或女儿）的人的姓名：";

string ParentName, ChildName;

cin >> ParentName;

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

cin >> ChildName;

parent = new Members;

parent = findParentNode(root, ParentName);

addSonNode(parent, ChildName);

Members\* NewChild = new Members;

cout << ParentName << "第一代子孙是：" << ChildName;

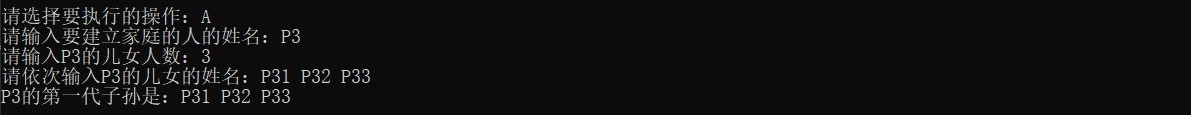
cout << endl;

cout << endl;

}

//成员函数：添加成员

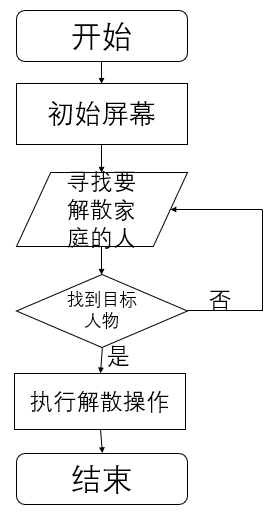
### 3.2.3 添加成员功能截屏示例



## 

## 3.3 解散家庭功能的实现

### 3.3.1 解散家庭功能流程图



### 3.3.2 解散家庭功能核心代码

void GenealogyManagement::deleteBranch(Members\* root) {

cout << "请输入要解散的家庭的人的姓名：";

string ParentName;

cin >> ParentName;

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

parent = new Members;

parent = findParentNode(root, ParentName);//找到要删除的成员节点

parent = NULL;

cout << endl;

cout << endl;

### }

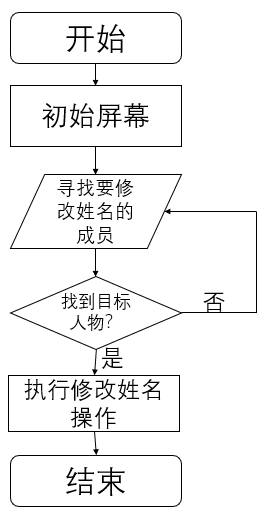
//成员函数：删除某一分支

### 3.3.3 解散家庭功能截图示例



## 3.4 修改成员姓名功能的实现

### 3.4.1 修改成员姓名功能流程图



### 

### 3.4.2 修改成员姓名功能核心代码

void GenealogyManagement::changeName(Members\* root) {

cout << "请输入要更改姓名的人的目前姓名：";

string MemberName;

cin >> MemberName;

parent = new Members;

parent = findParentNode(root, MemberName);//找到要修改姓名的成员节点

cout << "请输入更改后的姓名：";

string MemberName\_New;

cin >> MemberName\_New;

cout << MemberName << "已更名为" << MemberName\_New<<endl;

parent->name = MemberName\_New;

cout << endl;

}

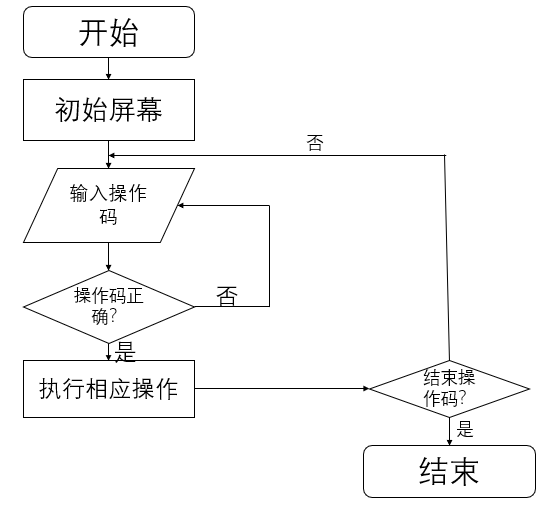
//成员函数：修改某一成员姓名

3.4.3 修改成员姓名功能截屏示例



## 3.5 总体系统的实现

### 3.5.1 总体系统流程图



### 3.5.2 总体系统核心代码

PrintOption();

cout << "首先请建立一个家谱！" << endl;

string name;

cout << "请输入祖先的姓名：";

cin >> name;

cout << "此家谱的祖先是：" << name << endl;

cout << endl;

Members\* root = new Members;

root->name = name;

GenealogyManagement Genealogy;

char option = 'A';

while (option == 'A' || option == 'B' || option == 'C' || option == 'D') {

cout << "请选择要执行的操作：";

cin >> option;

switch (option) {

case 'A':Genealogy.buildGenealogy(root); break;//建立家谱

case 'B':Genealogy.addMember(root); break;//添加成员

case 'C':Genealogy.deleteBranch(root); break;//删除分支

case 'D':Genealogy.changeName(root); break;//修改姓名

default: break;

}

}

delete root;

return 0;

### 3.6.3 总体系统截屏示例

# 

# 4 测试

## 4.1 功能测试

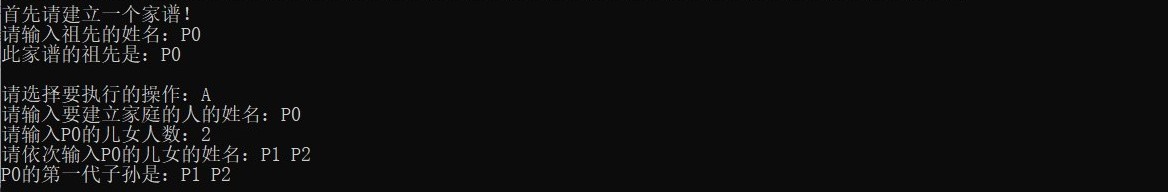
### 4.1.1 建立家谱树功能测试

**测试用例**：P0

**预期结果**：

P0

**实验结果**



### 

### 4.1.2 添加成员功能测试

**测试用例：**为P0添加后代P1、P2，将P1改名为P3后，再为P3添加后代P31、P32、P33

**预期结果：**

P0后代为P1、P2

P1改为P3后P3后代为P31、P32、P33

**实验结果：**



### 

### 

### 

### 4.1.3 添加成员功能测试

**测试用例：**为P2添加后代P21

**预期结果：**

P2后代为P21

**实验结果：**



### 

### 4.1.4 修改功能测试

**测试用例：**如测试4.1.1中第二步

**预期结果：**

如测试4.1.1中第二步

**实验结果：**

如测试4.1.1中第二步