项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 张诚睿

学 号： 2150998

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](#_Toc122644691)

[1.1 项目背景分析 1](#_Toc122644692)

[1.2 功能分析 1](#_Toc122644693)

[2 设计 1](#_Toc122644694)

[2.1 数据结构设计 1](#_Toc122644695)

[2.2 类结构设计 2](#_Toc122644696)

[2.3 成员与操作设计 2](#_Toc122644697)

[2.4 系统设计 4](#_Toc122644698)

[3 实现 5](#_Toc122644699)

[3.1 完善家谱功能的实现 5](#_Toc122644700)

[3.1.1 功能流程图 5](#_Toc122644701)

[3.1.2 核心代码 5](#_Toc122644702)

[3.1.3 截屏示例 6](#_Toc122644703)

[3.2 添加家庭成员功能的实现 6](#_Toc122644704)

[3.2.1 功能流程图 6](#_Toc122644705)

[3.2.2 核心代码 7](#_Toc122644706)

[3.2.3 截屏示例 7](#_Toc122644707)

[3.3 解散家庭功能的实现 8](#_Toc122644708)

[3.3.1 功能流程图 8](#_Toc122644709)

[3.3.2 核心代码 8](#_Toc122644710)

[3.3.3 截图示例 9](#_Toc122644711)

[3.4 更改姓名功能的实现 10](#_Toc122644712)

[3.4.1 功能流程图 10](#_Toc122644713)

[3.4.2 核心代码 10](#_Toc122644714)

[3.4.3 修改功能截屏示例 11](#_Toc122644715)

[3.5 关键子功能的实现 11](#_Toc122644716)

[3.5.1 搜索姓名的实现 11](#_Toc122644717)

[3.5.2 删除树的实现 11](#_Toc122644718)

[4 测试 11](#_Toc122644719)

[4.1 功能测试 11](#_Toc122644720)

[4.1.1 完善家谱功能测试 11](#_Toc122644721)

[4.1.2 添加家庭成员功能测试 12](#_Toc122644722)

[4.1.3 解散局部家庭功能测试 12](#_Toc122644723)

[4.1.4 更改姓名功能测试 13](#_Toc122644724)

[4.2 边界测试 14](#_Toc122644725)

[4.2.1 初始化根节点姓名特殊 14](#_Toc122644726)

[4.2.2 删除根节点的家庭 14](#_Toc122644727)

[4.3 出错测试 15](#_Toc122644728)

[4.3.1 添加家庭成员数目为非正或字符 15](#_Toc122644729)

[4.3.2 输入错误的父母名字 15](#_Toc122644730)

[4.3.1 添加的人姓名与已有人重名 16](#_Toc122644731)

[4.3.2 操作码错误 16](#_Toc122644732)

# 1 分析

## 1.1 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，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。计算机技术广泛应用的今天，在电脑上实现家谱管理可以方便家谱的记录与使用，本项目的目标即在此。

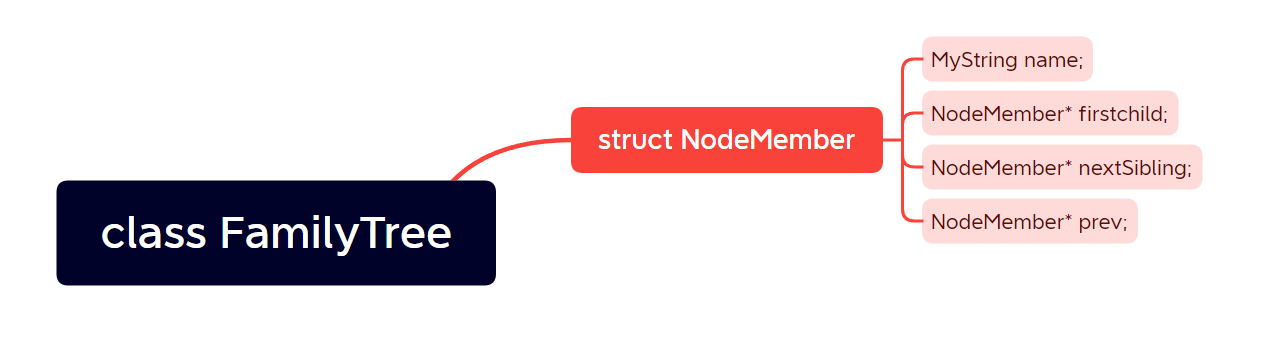
## 1.2 功能分析

本项的目的在于简单实现家谱信息的管理功能，因此必须包含初始化建立祖先节点、为指定叶节点添加子女、以指定节点为父节点添加孩子、解散家庭、更改姓名、自动显示子女的功能。

综上所述，项目可以实现对家谱人员的新建家庭、添加子女、删除家庭等核心操作，相比于纸面记录极大地提高了效率，既方便增删修改，也方便查看指定人的子女情况。

# 2 设计

## 2.1 数据结构设计



每个家谱人员信息所包含的内容仅有姓名一项，每个人的子女数量并不确定。因此，要实现对所有人员的家族逻辑关系的记录需要用到子女-兄弟法，在结构体NodeMember中添加指向第一个子女的指针firstchild、指向后续兄弟的指针nextSibling、和指向前置节点指针prev。再设计一个类FamilyTree，利用上述结点树结构，将头结点记录在私有成员中，就可实现对整个家谱树的操作。

需要注意的是，姓名是字符串，且长度不定，在特殊情况下，可能出现重名，为了区分重名人员，我们通常的做法是添加“（大）”、“（小）”等括号说明年龄差别；还有可能出现与外国人结婚后子女含英文姓名，因此姓名不仅需要同步支持中英文，还需要支持符号。在此设计一个MyString类，用动态申请字符数组的方式实现变长度操作字符串，并重载部分运算符来简化使用操作。

## 2.2 类结构设计

如上所述，本项目采用struct描述结点，这样使得管理家谱树的class可以访问结点信息。

MyString利用动态申请数组方式管理字符串，用char\*类型的私有成员访问数组。

## 2.3 成员与操作设计

**结点类**

struct NodeMember {

MyString name;

NodeMember\* firstchild;

NodeMember\* nextSibling;

NodeMember\* prev;

NodeMember(MyString \_name) :name(\_name), firstchild(NULL), nextSibling(NULL), prev(NULL) {};

};

**家谱树类**

class FamilyTree {

private:

NodeMember\* root; // 根节点

public:

FamilyTree() :root(NULL) {};

~FamilyTree() { MakeEmpty(root); };

void MakeEmpty(NodeMember\* start); // 清空以start为根的整棵树

bool IsEmpty() { return !bool(root); }; // 判断是否为空树

void NewFamilyMember(NodeMember\* Parent, NodeMember\* child); // 添加新成员操作

void InitTree(); // 初始化树，获取root结点信息

void ManageTree(); // 管理家谱

void GetChoice(char& chocie);

void Perfect(); // 完善家谱

void New(); // 添加成员指令

void Break(); // 解散家庭

void ChangeName(); // 更改姓名

void FindName(MyString& target, NodeMember\* now, NodeMember\*& Addr); // 找到指定姓名

void ShowChild(NodeMember\* parent); // 打印某一成员的第一代子孙

};

用root指针记录根结点，从而对整个树进行操作

**MyString字符串管理类**

class MyString {

private:

MyString& operator+=(char ch); // 末尾添加一个字符

int size; // 字符串长度

char\* content; // 内容指针

public:

MyString(const char\* str);

MyString() :size(1) {

content = new char;

if (content == NULL) {

cout << "No Memory" << endl;

exit(EXIT\_FAILURE);

}

\*content = '\0';

};// 无参

~MyString();

MyString(MyString& target); // 深拷贝

friend ostream& operator<<(ostream& os, MyString& target); // 输出内容

friend istream& operator>>(istream& is, MyString& target); // 读取内容，遇空格、回车停止

bool operator==(const MyString& matched); // 判断两字符串是否一致

MyString& operator=(const MyString target);

};

MyString实现对字符串的管理，重载了流运算符以方便输入输出。读入内容时，每次size++，重新申请一块新空间并添加一个字符进content数组，这样操作虽然效率降低，但相比于每次多开辟固定长度的方式节省了空间，达到100%空间利用率，同时考虑到一般情况下考号、姓名等信息不会过长，对效率的影响几乎可以不计，所以采用了最大化利用空间的设计方式。由于涉及动态申请，在拷贝构造函数和重载等号的操作中均使用了深拷贝的方式，避免内存错误。

## 2.4 系统设计

首先建立FamilyTree对象，调用其InitTree()函数获取初始信息，再调用其ManageTree()函数进行管理。根据对应输入的操作码执行完善家谱、添加成员、解散家庭、更改姓名等一系列操作，如指令为E，函数返回，主函数也返回0退出程序。

# 3 实现

## 3.1 完善家谱功能的实现

### 3.1.1 功能流程图

### 3.1.2 核心代码

cout << "请依次输入" << ParentName << "的儿女的姓名: ";

int Count = 0;

while (Count < Num)

{

MyString ChildName;

cin >> ChildName;

NodeMember\* check = NULL; // 检查重名，不允许重名

FindName(ChildName, root, check);

if (check != NULL) // 说明有重名

{

cout << "第" << Count + 1 << "个名字与家谱已有人重名，请从此处重新输入至结尾" << endl;

}

else // 无重名，可添加

{

Count++;

NodeMember\* child = new NodeMember(ChildName);

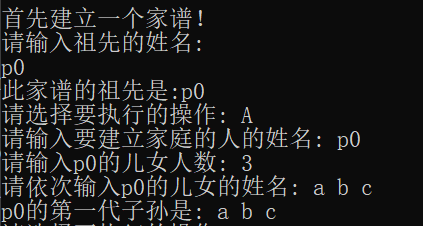
NewFamilyMember(Parent, child);

}

}

ShowChild(Parent);

### 3.1.3 截屏示例



## 3.2 添加家庭成员功能的实现

### 3.2.1 功能流程图

### 3.2.2 核心代码

while (true)

{

cin >> childName;

child = NULL;

FindName(childName, root, child);

if (child == NULL) // 无重名

{

child = new NodeMember(childName);

NewFamilyMember(Parent, child);

break;

}

else

{

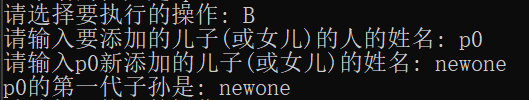
cout << "新名字与家谱已有人重名，请重新输入" << endl;

}

}

ShowChild(Parent);

### 3.2.3 截屏示例



## 3.3 解散家庭功能的实现

### 3.3.1 功能流程图

### 3.3.2 核心代码

if (Parent->prev != NULL)

{

if (Parent->prev->nextSibling == Parent)

{

Parent->prev->nextSibling = Parent->nextSibling;

}

else

{

Parent->prev->firstchild = Parent->nextSibling;

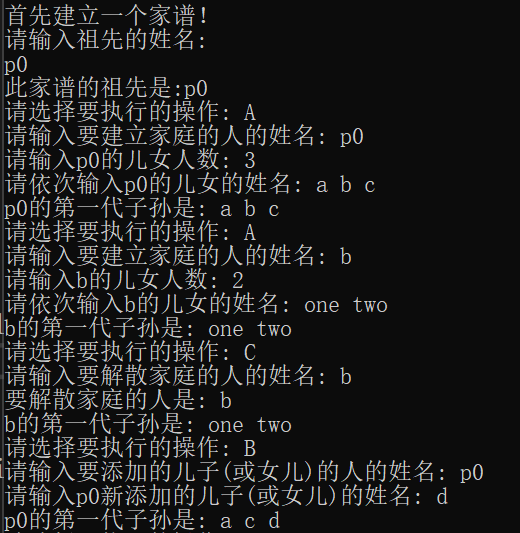
}

}

MakeEmpty(Parent->firstchild);

delete Parent;

### 3.3.3 截图示例



示例中p0为根节点，子女三个a、b、c，b有两个子女one、two；解散b的家庭后，向p0添加新子女d，可见p0的子女仅剩a、c、d。b及其子树已全部被删除

## 3.4 更改姓名功能的实现

### 3.4.1 功能流程图

### 3.4.2 核心代码

while (true)

{

cin >> newName;

check = NULL;

FindName(newName, root, check);

if (check == NULL) // 无重名

{

Parent->name = newName;

cout << ParentName << "已更名为" << newName << endl;

break;

}

else

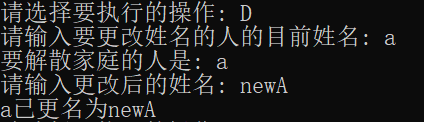
{

cout << "新名字与家谱已有人重名，请重新输入" << endl;

}

}

### 3.4.3 修改功能截屏示例



## 3.5 关键子功能的实现

### 3.5.1 搜索姓名的实现

void FamilyTree::FindName(MyString& target, NodeMember\* now, NodeMember\*& Addr);

递归寻找指定姓名target，找到后修改Addr为当前结点now。在调用时只需将目标性命给到target，起始搜索位置（一般为root）给now，Addr在调用前设为NULL，即可根据函数完成后Addr的值判断是否寻找到目标姓名。例如：

FindName(target,root,Addr);

### 3.5.2 删除树的实现

void FamilyTree::MakeEmpty(NodeMember\* start);

递归寻找叶子结点，将叶子结点删除后回溯，直至全部删除。调用时start为被删除树的根节点，该函数可同时应用于析构函数中，只需将root作为参数即可。例如:

MakeEmpty(root);

# 4 测试

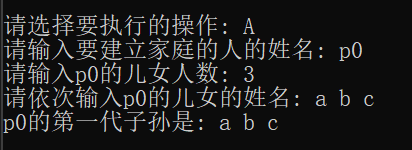
## 4.1 功能测试

### 4.1.1 完善家谱功能测试

**测试用例**：p0为祖先，为其添加三个子女a、b、c

**预期结果**：p0的第一代子孙是: a b c

**实验结果**



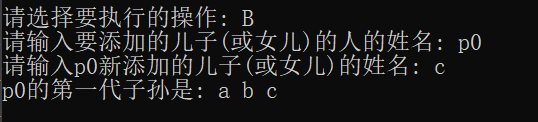
### 4.1.2 添加家庭成员功能测试

**测试用例：**p0为根节点，原有两个子女a、b，添加第三个子女c

**预期结果：**

p0的第一代子孙是: a b c

**实验结果：**



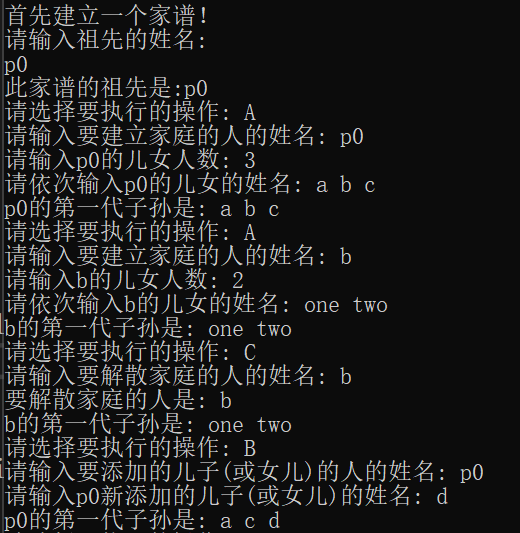
### 4.1.3 解散局部家庭功能测试

**测试用例：**p0为根节点，子女三个a、b、c，b有两个子女one、two；解散b的家庭后，向p0添加新子女d，可见p0的子女仅剩a、c、d。b及其子树已全部被删除

**预期结果：**

p0的第一代子孙是: a c d

**实验结果：**



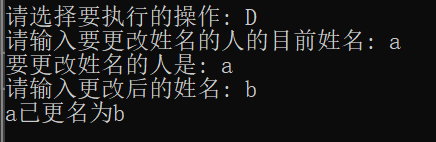
### 4.1.4 更改姓名功能测试

**测试用例：**将p0的子女a姓名修改为b

**预期结果：**

a已更名为b

**实验结果：**



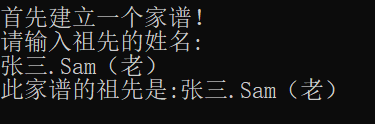
## 4.2 边界测试

### 4.2.1 初始化根节点姓名特殊

**测试用例：**根节点姓名含中英文、标点符号：张三.Sam（老）

**预期结果：**成功输入并建立树

**实验结果：**

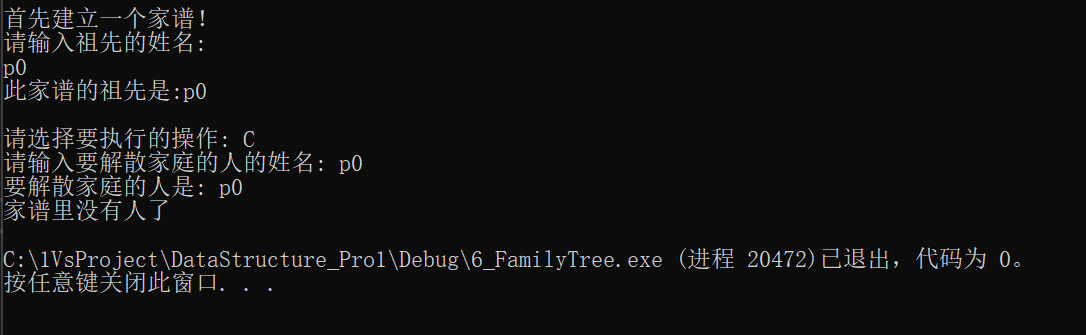


### 4.2.2 删除根节点的家庭

**测试用例：**删除根节点p0

**预期结果：**程序正常运行并在删除后正常退出，不崩溃。

**实验结果：**



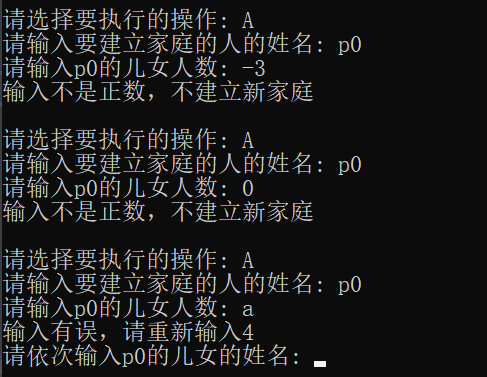
## 4.3 出错测试

### 4.3.1 添加家庭成员数目为非正或字符

**测试用例：**数目输入分别为-3 0 a

**预期结果：**程序正常运行，输入字符时提示错误信息并重新输入直至正确，输入错误数字时退出该指令，不崩溃

**实验结果：**

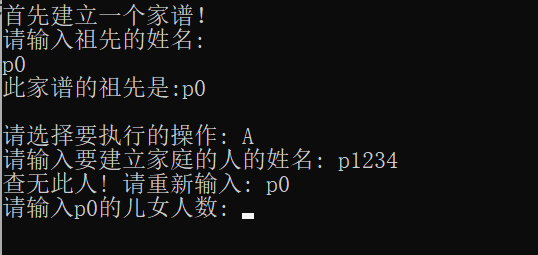


### 4.3.2 输入错误的父母名字

**测试用例**：树中只有p0姓名的人，指令内输入姓名时输入p1234

**预期结果**：提示错误信息，要求重新输入直至正确

**实验结果**：

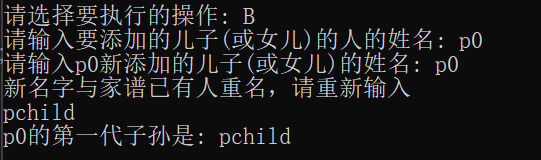


### 4.3.1 添加的人姓名与已有人重名

**测试用例：**根节点为p0，添加的子女姓名仍为p0

**预期结果：**程序给出重名错误提示，重新输入直至输入不重名

**实验结果：**



### 4.3.2 操作码错误

**测试用例：**输入操作码不为A-E的数字或是字符

**预期结果：**程序给出提示信息，重新输入直至输入正确。

**实验结果：**

