项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 刘畅

学 号： 2054164

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](#_Toc91009571)

[1.1 背景分析 1](#_Toc91009572)

[1.2 功能分析 1](#_Toc91009573)

[2 设计 1](#_Toc91009574)

[2.1 数据结构设计 1](#_Toc91009575)

[2.2 类结构设计 1](#_Toc91009576)

[2.3 成员与操作设计 1](#_Toc91009577)

[2.4 系统设计 2](#_Toc91009578)

[3 实现 2](#_Toc91009579)

[3.1 建立家谱功能的实现 2](#_Toc91009580)

[3.1.1 实现方法 2](#_Toc91009581)

[3.1.2 核心代码 2](#_Toc91009582)

[3.2 组建家庭功能、添加子女功能的实现 3](#_Toc91009583)

[3.2.1 组建家庭功能实现方法 3](#_Toc91009584)

[3.2.2 添加子女功能实现方法 3](#_Toc91009585)

[3.2.3 核心代码 3](#_Toc91009586)

[3.3 解散家庭功能的实现 4](#_Toc91009587)

[3.3.1 实现方法 4](#_Toc91009588)

[3.3.2 解散家庭功能核心代码 5](#_Toc91009589)

[3.4 搜索功能的实现 6](#_Toc91009590)

[3.4.1 实现方法 6](#_Toc91009591)

[3.4.2 搜索功能核心代码 6](#_Toc91009592)

[4 测试 6](#_Toc91009593)

[4.1 功能测试 6](#_Toc91009594)

[4.1.1 家谱初始化 6](#_Toc91009595)

[4.1.2 建立家庭、添加子女 7](#_Toc91009596)

[4.1.3 解散家庭 7](#_Toc91009597)

[4.1.4 更改姓名 8](#_Toc91009598)

[4.2 错误测试 8](#_Toc91009599)

[4.2.1 输入姓名不存在 8](#_Toc91009600)

[4.2.2 输入人数非法 8](#_Toc91009601)

# 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。

## 1.2 功能分析

本项目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。

# 2 设计

## 2.1 数据结构设计

家谱的父子关系和树形数据结构的父子结点关系十分相似，因此可以使用“树”来模拟家谱结构。由于每一个家庭的孩子数目不定，所以是最普遍意义上的树结构。由于本题需要实现数据的插入、删除、查询等功能，因此采用三叉链表和左子右兄弟法存储树的结构，牺牲空间的代价以加快插入、删除、查询的速度。

## 2.2 类结构设计

使用模板类封装三叉链表树的实现，在main.cpp中生成类的对象并进行操作。

## 2.3 成员与操作设计

**树结点类（TreeNode）**

T \_data;

TreeNode<T>\* \_brother;//兄弟结点，构成链表

TreeNode<T>\* \_parent;

TreeNode<T>\* \_firstson;//第一个儿子结点

**树类（Tree）**

bool setRoot(T& data) {

//根结点，brother parent firstson都是NULL，目前只有数据

if (\_root != NULL) {

//根结点已存在，只修改数据

\_root->\_data = data;

return true;

}

else {

//根结点未建立

\_root = new TreeNode<T>(data);

return \_root != NULL ? true : false;

}

}

bool IfEmpty() { return \_root == NULL; }

TreeNode<T>\* getRoot()const { return \_root; }

TreeNode<T>\* getParent(TreeNode<T>\* tar);

TreeNode<T>\* getSons(TreeNode<T>\* tar);//获取firstson，可以根据firstson迭代得到其所有sons

bool Insert(TreeNode<T>\* tar, TreeNode<T>\* firstson);

bool Remove(TreeNode<T>\* tar);//删除tar及其所有子树

TreeNode<T>\* search(TreeNode<T>\* tar, const T& data);

void PreOrder(TreeNode<T>\* cur);

## 2.4 系统设计

系统首先提示输入祖先的名字，因为一个家谱必须有且仅有一个祖先，接着调用PrintInterface()打印提示界面，等待用户输入操作码，通过while(true)的循环让用户不断输入操作码，直到操作码为E的时候退出程序。

# 3 实现

## 3.1 建立家谱功能的实现

### 3.1.1 实现方法

输入祖先名字，调用树的setRoot函数，将其设置为根节点。若原来的根节点不为空，则只修改根节点的数据，若原来为空，则重新new一个根节点设置数据。

### 3.1.2 核心代码

bool setRoot(T& data) {

//根结点，brother parent firstson都是NULL，目前只有数据

if (\_root != NULL) {

//根结点已存在，只修改数据

\_root->\_data = data;

return true;

}

else {

//根结点未建立

\_root = new TreeNode<T>(data);

return \_root != NULL ? true : false;

}

}

## 3.2 组建家庭功能、添加子女功能的实现

### 3.2.1 组建家庭功能实现方法

先输入要建立家庭的人的姓名（相当于添加子树，先输入子树的根节点），在原树中查找该姓名，若不存在，则输出查无此人并跳出“组建家庭功能”；若此人已有子女，则应当使用添加子女功能（而非组建家庭），故也输出提示词并退出。如果合法，则将包含此人信息的树结点给到tar指针，表示插入的目标位置。而后先输入此人（根节点）的子女数目，再依次输入每个子女的姓名，对每个姓名都new一个新的树结点，并使用“右兄弟”指针将他们串接起来，保留第一个子女的指针信息作为insert\_data，而后调用family\_tree. Insert(tar, insert\_data)，将全部子女作为一个链表添加到父节点的firstson指针，并将全部子女地parent指针设置为tar。

### 3.2.2 添加子女功能实现方法

与组建家庭使用的方法如出一辙，唯一的变化就是添加子女允许目标结点已经存在子女。在代码上的区别就是，Insert时不能将insert\_data直接赋值为tar的firstson，这样可能覆盖掉原来的子女，所以应该将insert\_data放置在tar的firstson结点的最右端。

### 3.2.3 核心代码

输入根节点：

string parent;

cin >> parent;

TreeNode<string>\* tar = family\_tree.search(family\_tree.getRoot(), parent);

插入子女：

cout << "请依次输入" << parent << "的" << n << "个儿女姓名：" << endl;

TreeNode<string>\* insert\_data = new TreeNode<string>;

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

TreeNode<string>\* temp = new TreeNode<string>;

temp->\_brother = insert\_data;

insert\_data = temp;

}

//申请了n块内存

for (TreeNode<string>\* iter = insert\_data; iter != NULL; iter = iter->\_brother) {

cin >> iter->\_data;

}

if (family\_tree.Insert(tar, insert\_data)) { cout << "建立成功！" << endl; }

else { cout << "未知原因，建立失败！" << endl; }

Insert()函数实现：

bool Insert(TreeNode<T>\* tar, TreeNode<T>\* firstson) {

if (tar == NULL || firstson == NULL) {

return false;

}

//将所有子女结点的parent指针全部置成父结点

for (TreeNode<T>\* iter = firstson; iter != NULL; iter = iter->\_brother) {

iter->\_parent = tar;

}

//在cur下插入子结点

if (tar->\_firstson == NULL) {

tar->\_firstson = firstson;

}

else {

tar = tar->\_firstson;

while (tar->\_brother != NULL){

tar = tar->\_brother;

}

tar->\_brother = firstson;

}

return true;

}

## 3.3 解散家庭功能的实现

### 3.3.1 实现方法

先输入要解散家庭的人的姓名，然后在树中查找是否存在，不存在则输出提示词，若为根结点，也输出提示词，因为无法删除根结点。而后处理该结点的子女链表关系：即若它是parent的firstson，则将parent的firstson指向该结点的brother；若不为firstson，则可以统一处理，即从parent的firstson开始向右迭代，直到到达被删结点，将被删结点的brother赋值给前一个结点即可。

然后再处理该结点的删除，使用递归删除法，即每个删除过程中先依次递归删除其子结点，最后再删除当下的结点。

### 3.3.2 解散家庭功能核心代码

删除结点前处理“右兄弟”结点关系：

cout << "请输入要解散家庭的人的姓名：";

string parent; cin >> parent;

TreeNode<string>\* tar = family\_tree.search(family\_tree.getRoot(), parent);

if (tar == family\_tree.getRoot()) {

cout << "无法删除祖先！" << endl;

continue;

}

else if (tar == NULL) {

cout << "要删除的结点不存在！" << endl;

continue;

}

else {

TreeNode<string>\* par = tar->\_parent;

if (par->\_firstson == tar) {

//要删除的结点是firstson，需要特殊处理连接

par->\_firstson = tar->\_brother;

}

else {

par = par->\_firstson;

while (par->\_brother != tar) {

par = par->\_brother;

}

par->\_brother = tar->\_brother;

}

}

if (family\_tree.Remove(tar)) { cout << "解散成功！" << endl; }

else { cout << "未知原因，解散失败！" << endl; }

递归删除结点：

bool Remove(TreeNode<T>\* tar) {

//拿到一个结点，直接删它的子女

TreeNode<T>\* son = tar->\_firstson;

while (son != NULL) {

TreeNode<T>\* tmp = son->\_brother;

Remove(son);

son = tmp;

}

delete tar;

return true;

}

## 3.4 搜索功能的实现

### 3.4.1 实现方法

给定一个键值（此题中是家谱成员的姓名），查找是否已经在树中，若存在则返回包含该键值的树结点指针，不然则返回NULL。相当于遍历整个树，采用递归实现前序遍历。

### 3.4.2 搜索功能核心代码

TreeNode<T>\* search(TreeNode<T>\* tar, const T& data) {

if (tar->\_data == data)return tar;

if (tar == NULL)return NULL;

//先寻找其子女

if (tar->\_firstson != NULL) {

TreeNode<T>\* search\_son = search(tar->\_firstson, data);

if (search\_son != NULL) { return search\_son; }

}

//再寻找其兄弟

if (tar->\_brother != NULL) {

TreeNode<T>\* search\_brother = search(tar->\_brother, data);

if (search\_brother != NULL) { return search\_brother; }

}

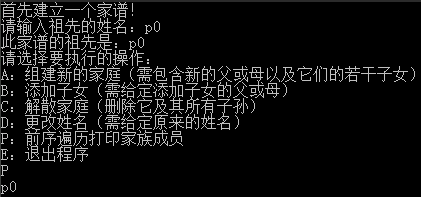
return NULL;//找不到

}

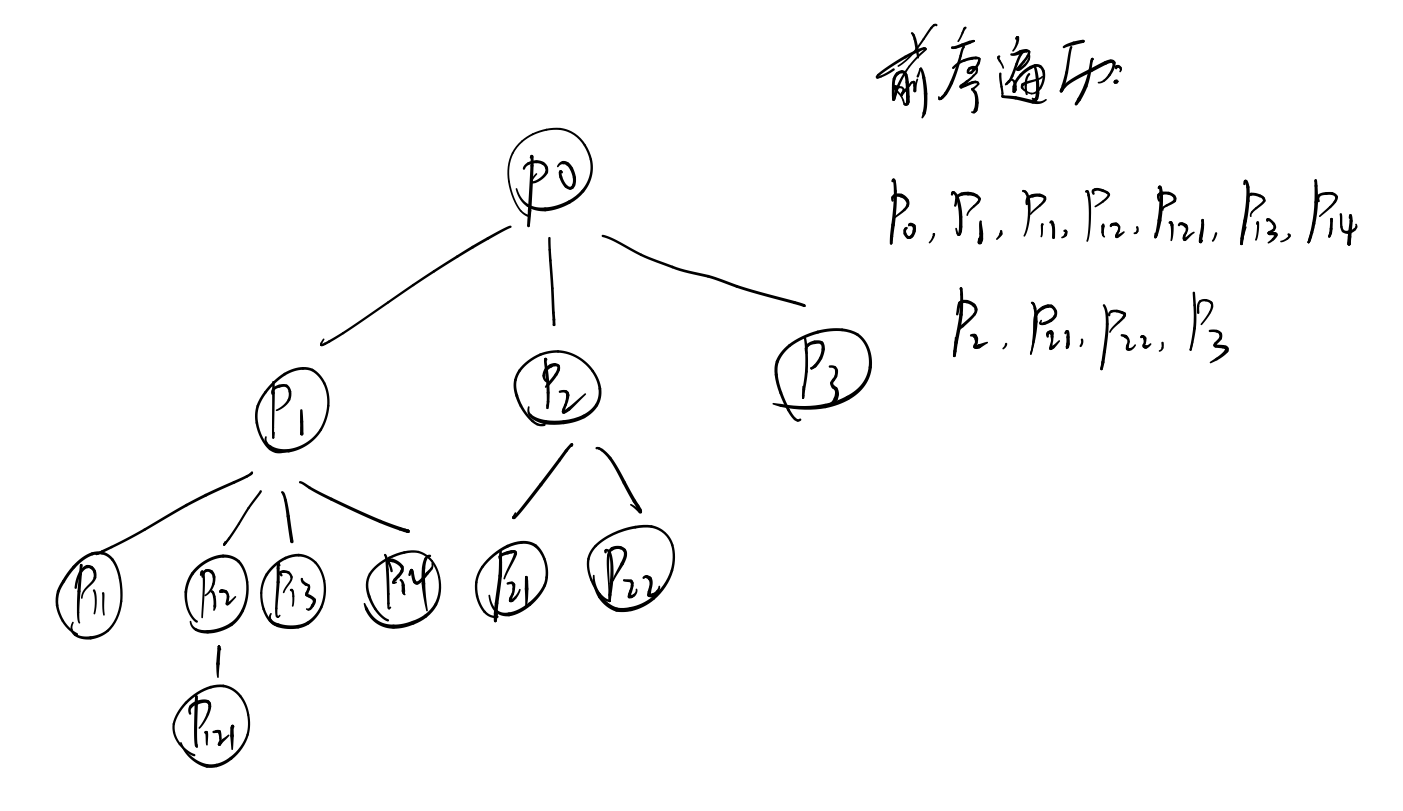
# 4 测试

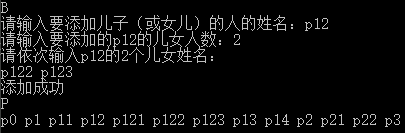
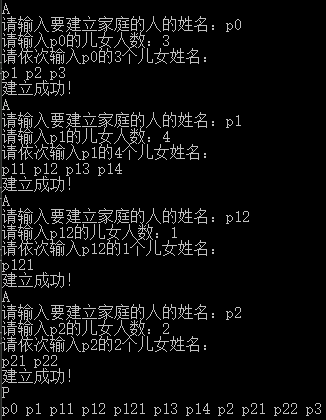
## 4.1 功能测试

### 4.1.1 家谱初始化



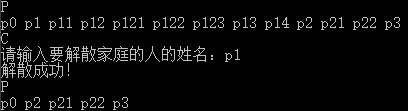
### 4.1.2 建立家庭、添加子女



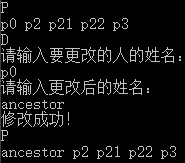


### 4.1.3 解散家庭

解散p1家庭，即删除p1及其所有子孙

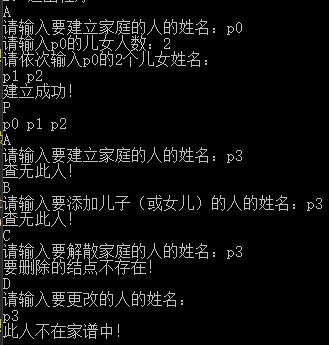


### 4.1.4 更改姓名



## 4.2 错误测试

### 4.2.1 输入姓名不存在



### 4.2.2 输入人数非法

