数据结构课程设计

项目说明文档

家谱管理系统

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |

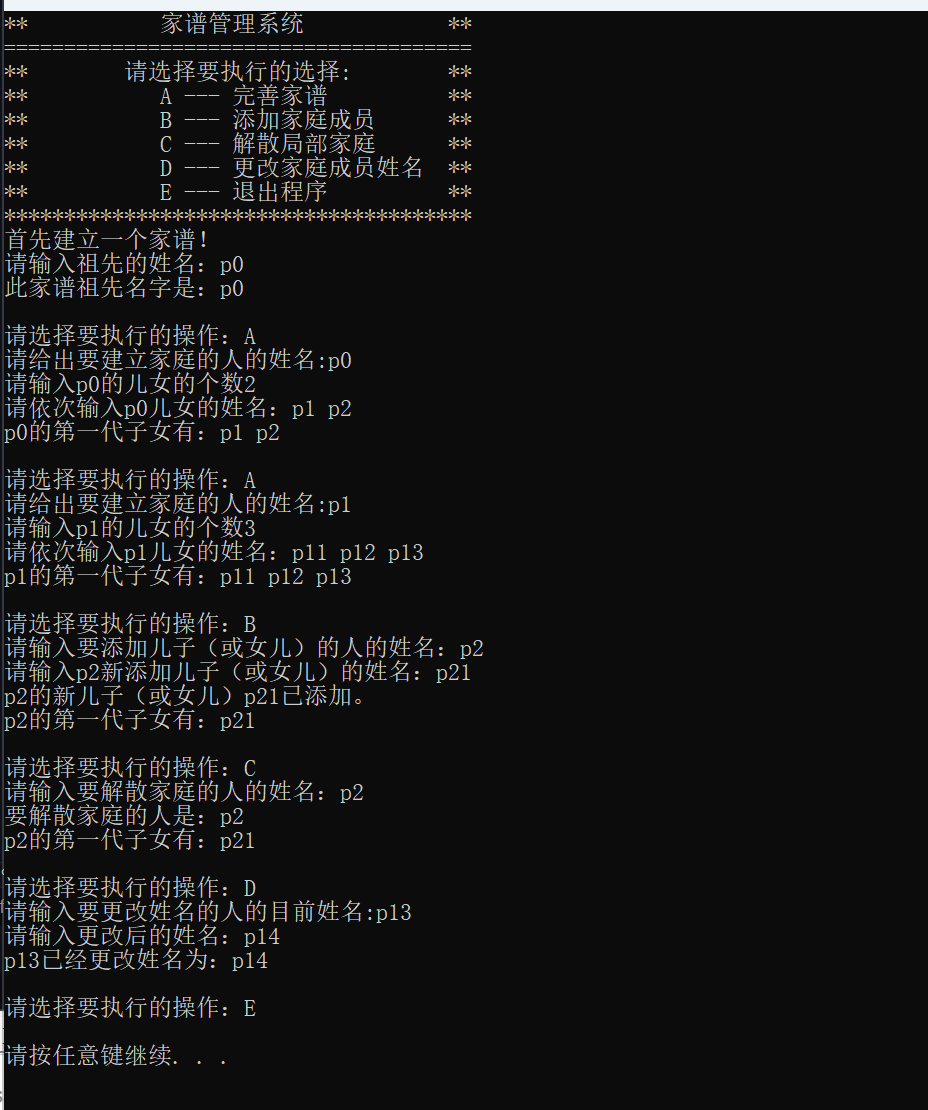


同济大学

Tongji University

1. **题目简介**

**1.实验背景**

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。

**2.实验功能**

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。

1. **项目设计**

**1.数据结构设计**

**struct node**

作用: 表示家谱中的一个成员。

属性:

char name[50]: 存储成员的名字。

node\* firstchild: 指向该成员的第一个孩子节点。

node\* firstbrother: 指向该成员的下一个兄弟节点。

node\* parent: 指向该成员的父节点。

构造函数: 提供了无参和有参构造函数，有参构造函数用于创建具有特定属性值的节点。

**class relation\_tree**

作用: 管理和维护家谱树。

私有成员:

node\* root: 指向家谱树的根节点。

void delete\_tree(node\* current): 递归删除以 current 为根的子树。

公共成员:

构造函数和析构函数。

node\* search(node\* current, const char\* name): 在家谱树中搜索特定名字的成员。

void build\_root(): 建立家谱树的根节点。

void build\_tree(): 为家谱树中的某个成员添加子树（即添加其子女）。

void display\_tree(const char\* name): 显示特定成员的子女。

void add\_family(): 为特定成员添加子女。

void delete\_family(): 解散以特定成员为根的子树。

void modify\_name(): 修改家谱树中特定成员的名字。

**2.类结构设计**

**relation\_tree**

这个类是程序的核心，负责管理整个家谱树。

它使用 node 结构体作为基本的数据单位。

提供了用户界面操作与家谱树数据操作之间的接口。

**3.成员与操作设计**

**构造与析构**: 类构造函数初始化根节点为 nullptr，析构函数负责调用 delete\_tree 清理所有动态分配的内存。

build\_root: 询问用户输入祖先的名字，并创建根节点。

build\_tree: 以某个节点为父节点，添加子节点（子女），包括输入验证和错误处理。

display\_tree: 显示某个节点的直系子孙。

add\_family: 为已有节点添加新的子节点。

delete\_family: 删除某个节点下的所有子节点，包括该节点本身。

modify\_name: 修改家谱中某个成员的名字。

search: 递归搜索特定名字的成员。

delete\_tree: 递归删除以某个节点为根的子树，用于内存清理。

## 4.算法设计

**1. 搜索算法 (search)**

目的: 在家谱树中找到具有特定名字的成员节点。

实现: 使用递归遍历。从当前节点开始，首先检查当前节点的名字是否匹配，然后递归地在其所有子节点中搜索。

时间复杂度: 最坏情况下为 O(N)，其中 N 是家谱树中的节点总数。

**2. 构建家庭树算法 (build\_tree)**

目的: 为家谱中的某个成员添加子女，构建该成员的家庭树。

实现: 首先使用 search 方法找到对应的父节点，然后根据用户输入的子女数和名字创建子节点，并将这些子节点作为父节点的子女链接起来。

特点: 包含错误处理（如父节点不存在、已有子女等情况），并使用循环确保所有子女都被添加。

**3. 删除家庭树算法 (delete\_family)**

目的: 删除以某个成员为根的整个家庭树。

实现: 先找到要删除的节点，然后递归地删除该节点的所有子孙节点。使用 delete\_tree 函数实现递归删除。

特点: 考虑了节点在家谱中的位置，包括根节点和非根节点的情况，确保正确地更新父节点和兄弟节点的链接。

**4. 修改成员姓名算法 (modify\_name)**

目的: 更改家谱中某个成员的名字。

实现: 首先使用 search 方法找到对应的节点，然后根据用户输入更改该节点的名字。

特点: 包括对新名字的有效性检查，确保不与家谱中其他成员的名字重复。

**5. 展示家庭树算法 (display\_tree)**

目的: 显示某个成员的直系子孙。

实现: 查找给定名字的成员节点，然后遍历其所有直接子节点，打印出子节点的名字。

特点: 简单而直接，使用循环遍历子节点。

1. **项目实现**

**3.1. 根节点创立与建立字树**

1. **void** relation\_tree::build\_root()
2. {
3. cout << "首先建立一个家谱！" << endl;
4. cout << "请输入祖先的姓名：";
5. **char** name[50];
6. cin >> name;
7. //为根节点建立空间
8. root = **new** node(name);
9. //错误处理
10. **if**(root==nullptr)
11. {
12. cerr << "内存分配错误" << endl;
13. exit(1);
14. }
15. //给出祖先名字
16. cout << "此家谱祖先名字是：" << root->name << endl;
17. }
19. **void** relation\_tree::build\_tree()
20. {
21. cout << "请给出要建立家庭的人的姓名:";
22. **char** name[50];
23. cin >> name;
24. node\* father = search(root, name);
25. **if**(father==nullptr)
26. {
27. cout << "本家谱内不存在该家庭成员！" << endl;
28. **return**;
29. }
30. **if**(father->firstchild!=nullptr)
31. {
32. cout << "该成员已经建立过家庭！" << endl;
33. **return**;
34. }
35. cout << "请输入" << name << "的儿女的个数:";
36. **int** child\_num;
37. **while**(1)
38. {
39. cin >> child\_num;
40. **if**(child\_num<0||!cin.good())
41. {
42. cin.clear();
43. cin.ignore(1024, '\n');
44. cout << "输入有误，儿女个数必须为正整数" << endl;
45. **continue**;
46. }
47. **else**
48. {
49. **break**;
50. }
51. }
52. cout << "请依次输入" << name << "儿女的姓名：";
53. **char** child\_name[50];
54. **while**(**true**)
55. {
56. cin >> child\_name;
57. **if**(search(root, child\_name)==nullptr)
58. {
59. father->firstchild = **new** node(child\_name);
60. **if**(father->firstchild==nullptr)
61. {
62. cerr << "内存分配错误" << endl;
63. exit(1);
65. }
66. father->firstchild->parent = father;
67. **break**;
69. }
70. **if**(search(root, child\_name)!=nullptr)
71. {
72. cout << "第一位成员已经存在，请重新输入" << endl;
73. cin.clear();
74. cin.ignore(1024, '\n');
75. **continue**;
76. }
77. }
78. **for**(**int** i=1;i<child\_num;i++)
79. {
80. **while**(**true**)
81. {
82. cin >> child\_name;
83. **if**(search(root, child\_name)==nullptr)
84. {
85. node\* current = father->firstchild;
86. **while** (current->firstbrother!=nullptr)
87. {
88. current = current->firstbrother;
89. }
90. current->firstbrother = **new** node(child\_name);
91. **if**(current->firstbrother==nullptr)
92. {
93. cerr << "内存分配错误" << endl;
94. exit(1);
95. }
96. current->firstbrother->parent = father;
97. **break**;
98. }
99. **else**
100. {
101. cout << "姓名为" << child\_name << "成员名字重复，请重新输入！" << endl;
102. cin.clear();
103. cin.ignore(10000, '\n');
104. **continue**;
105. }
107. }
108. }
109. display\_tree(name);
111. }

**创建家谱的根节点（build\_root）**

该功能允许用户初始化家谱，通过创建一个根节点来代表家族的祖先。用户输入祖先的名字后，程序为之创建一个新节点，并设为家谱的起点。如果内存分配成功，家谱的创建即告完成，并显示祖先的名字作为确认。这是整个家谱管理系统的起始步骤，所有后续的家族成员添加和家谱扩展都基于这个根节点。

**为家族成员添加子女（build\_tree）**

这个功能用于在已存在的家谱中添加新的家庭成员。首先，用户指定要为其添加子女的家族成员，然后输入该成员的儿女数量及其姓名。程序首先验证指定成员的存在性和是否已有家庭，然后逐个添加子女节点。这个过程中，程序确保不会添加重名的成员，并在完成添加后显示新更新的家庭成员列表，从而使用户能够逐渐构建出完整的家族树。

**3.2. 添加家庭成员功能**

1. **void** relation\_tree::add\_family()
2. {
3. cout << "请输入要添加儿子（或女儿）的人的姓名：";
4. **char** parentName[50];
5. cin >> parentName;
6. node\* parent = search(root, parentName);
7. **if** (parent == nullptr)
8. {
9. cout << "家谱中不存在该成员信息" << endl;
10. **return**;
11. }
13. **while** (**true**)
14. {
15. cout << "请输入" << parentName << "新添加儿子（或女儿）的姓名：";
16. **char** childName[50];
17. cin >> childName;
19. **if** (search(root, childName) != nullptr)
20. {
21. cout << "该成员已经存在，请重新输入！" << endl;
22. cin.clear(); // 清除缓冲区中的输入错误
23. cin.ignore(1000, '\n'); // 忽略之前的错误输入
24. **continue**; // 继续循环，等待新的输入
25. }
27. // 根据是否已有子女来添加新子女
28. **if** (parent->firstchild == nullptr)
29. {
30. parent->firstchild = **new** node(childName);
31. parent->firstchild->parent = parent;
32. }
33. **else**
34. {
35. node\* current = parent->firstchild;
36. **while** (current->firstbrother != nullptr)
37. {
38. current = current->firstbrother;
39. }
40. current->firstbrother = **new** node(childName);
41. current->firstbrother->parent = parent;
42. }
44. cout << parentName << "的新儿子（或女儿）" << childName << "已添加。" << endl;
45. display\_tree(parentName); // 显示更新后的家庭成员
46. **break**; // 成功添加后退出循环
47. }
48. }

函数 add\_family 在家谱管理系统中实现了为特定成员添加新的子女的功能。用户首先输入要添加子女的家族成员的姓名，然后程序通过 search 函数检查此成员是否存在于家谱中。如果成员存在，用户继续输入子女的姓名。程序确保这个新姓名在家谱中尚未存在，然后将其作为新节点添加到指定成员的子节点列表中。若父母节点已有子节点，新节点则被添加为最后一个子节点的兄弟节点。完成添加后，程序展示父母节点的更新后子女列表，以确认新成员已被正确添加到家谱中。

**3.3. 解散家庭功能实现**

1. **void** relation\_tree::delete\_family()
2. {
3. cout << "请输入要解散家庭的人的姓名：";
4. **char** name[50];
5. cin >> name;
6. node\* father = search(root, name);
7. **if**(father==nullptr)
8. {
9. cout << "本家谱内不存在该家庭成员！" << endl;
10. **return**;
11. }
12. cout << "要解散家庭的人是：" << name << endl;
13. display\_tree(father->name);
14. **if**(father!=root)
15. {
16. **if**(father->parent->firstchild==father)
17. {
18. father->parent->firstchild = father->firstbrother;
19. }
20. **else**
21. {
22. node\* current = father->parent->firstchild;
23. **while**(**true**)
24. {
25. **if**(current->firstbrother==father)
26. {
27. **break**;
28. }
29. **else**
30. {
31. current = current->firstbrother;
32. }
33. }
34. current->firstbrother = father->firstbrother;
35. }
36. delete\_tree(father);
37. }
38. **else**
39. {
40. delete\_tree(father);
41. }
42. }

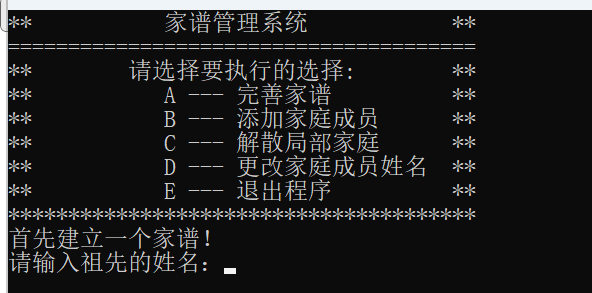
函数 delete\_family 在家谱管理系统中用于解散以指定成员为根的家庭，删除该成员及其所有后代。用户首先输入要解散家庭的成员的姓名。如果该成员在家谱中存在，程序会显示该成员的直系子女，然后从家谱中移除这个成员及其所有后代。这包括从其父母的子节点列表中断开连接，并递归删除该成员及其所有子孙节点。如果指定的成员是家谱的根节点，程序将删除整个家谱。这个功能允许用户在家谱中进行重要的编辑操作，如解散某个分支家庭。

**3.4. 修改家庭成员姓名功能实现**

1. **void** relation\_tree::modify\_name()
2. {
3. cout << "请输入要更改姓名的人的目前姓名:";
4. **char** name[50];
5. cin >> name;
6. node\* current = search(root, name);
7. **if**(current==nullptr)
8. {
9. cout << "本家谱内不存在该家庭成员！" << endl;
10. **return**;
11. }
12. cout << "请输入更改后的姓名：";
13. **while** (**true**)
14. {
15. **char** new\_name[50];
16. cin >> new\_name;
17. **if** (search(root, new\_name) == nullptr)
18. {
19. cout << current->name << "已经更改姓名为：" << new\_name << endl;
20. strcpy(current->name, new\_name);
21. **break**;
22. }
23. **else**
24. {
25. cout << "该姓名已经被使用，请更换新的姓名：";
26. cin.clear();
27. cin.ignore(1024, '\n');
29. }
30. }

函数 modify\_name 用于更改家谱中某个成员的姓名。用户首先输入需要更改姓名的成员的当前姓名。如果这个成员存在于家谱中，程序接着提示用户输入新的姓名。在更改之前，程序会检查新姓名在家谱中是否已存在，以避免重名。如果新姓名不重复，程序将更新该成员的姓名，并显示更改后的姓名以确认更改。如果新姓名已存在，程序将提示用户重新输入另一个姓名。这个功能使得用户能够灵活地更新家谱信息，确保家谱中的成员姓名始终是最新且准确的。

**3.5. 主菜单功能实现**



**健全性测试：**

在这个家谱管理系统中，错误处理主要体现在以下几个方面：

**1. 内存分配错误处理**

在创建新节点（如在 build\_root 或 add\_family 函数中）时，程序会检查 new 操作是否成功。如果内存分配失败（即新创建的节点为 nullptr），程序将输出错误信息并退出。

**2. 成员存在性验证**

在多个函数中，如 build\_tree、delete\_family 和 modify\_name，程序首先会使用 search 函数检查用户指定的家族成员是否存在于家谱中。如果成员不存在，程序会输出相应的错误信息并返回。

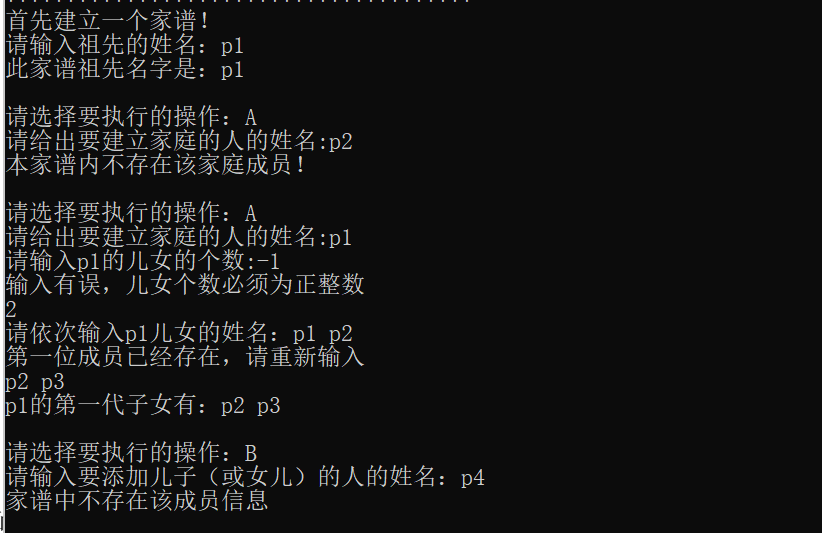
**3. 输入验证**

在输入家族成员的儿女个数时（如在 build\_tree 函数中），程序检查输入是否为正整数。如果输入无效，程序会提示错误并要求用户重新输入。

**4. 防止重名**

在添加新家族成员时（如在 add\_family 和 modify\_name 函数中），程序会检查新输入的姓名是否已在家谱中存在。如果发现重名，程序会要求用户重新输入。

**错误测试：**

****

1. **项目创新点**

本项目的创新之处在于采用了孩子兄弟表示法来构建和管理复杂的家谱树结构，有效地表示了家族成员之间的层级关系。这种数据结构的应用不仅增强了程序处理家族关系的灵活性，也使得添加、删除和修改家族成员变得更加高效。此外，程序通过综合应用递归搜索和动态内存管理技术，实现了对家谱数据的精准操作，允许用户灵活地构建和更新家族信息。错误处理机制的有效实施也确保了程序在不同情境下的稳定运行。

**五、实验小结**

通过这次实验，不仅展示了家谱管理系统的设计和实现，还加深了对于树结构数据管理和递归算法的理解。实验中对错误处理和用户输入验证的关注，提高了程序的健壮性和用户友好性。同时，此项目也揭示了动态内存管理在复杂数据结构中的重要性。虽然程序在功能实现方面表现良好，但对于未来的扩展，如图形界面的加入或更高级的数据检索功能，仍有进一步的开发空间。总体来说，这个项目不仅成功地解决了特定问题，也为更复杂的数据结构应用提供了宝贵的实践经验。