同济大学

TONGJI UNIVERSITY

数据结构课程设计

项目名称	家谱管理系统
学　　院	计算机科学与技术学院
专　　业	软件工程
学生姓名	杨瑞晨
学　　号	2351050
指导教师	张颖
日　　期	2024 年 12 月 6 日

目 录

1 项目分析

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。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。

1.2 项目功能分析

1.2.1 功能要求

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。

1.2.2 输入要求

每次操作都输入相对应的操作数字，然后根据提示输入相应的数据，如插入家族成员时，输入家族成员的姓名，性别，年龄等信息。

1.2.3 输出要求

每次操作都会输出相应的操作结果，如插入家族成员时，输出插入成功或失败的信息，查找家族成员时，输出家族成员的信息等。本项目还可输出一个完整的家谱。

1.2.4 项目实例


**　　家谱管理系统　　**
**　　请选择要执行的操作 :　　**
**　　A --- 完善家谱　　**
**　　B --- 添加家庭成员　　**
**　　C --- 解散局部家庭　　**
**　　D --- 更改家庭成员姓名　　**
**　　E --- 退出程序　　**
首先建立一个家谱!
请输入祖先的姓名：P0
此家谱的祖先是：P0
请选择要执行的操作：A
请输入要建立家庭的人的姓名：P0
请输入P0的儿女人数：2
请依次输入P0的儿女的姓名：P1 P2
P0的第一代子孙是：P1　P2
请选择要执行的操作：A
请输入要建立家庭的人的姓名：P1
请输入P1的儿女人数：3
请依次输入P1的儿女的姓名：P11 P12 P13
P1的第一代子孙是：P11　P12　P13
请选择要执行的操作：B
请输入要添加儿子（或女儿）的人的姓名：P2
请输入P2新添加的儿子（或女儿）的姓名：P21
P2的第一代子孙是：P21
请选择要执行的操作：C
请输入要解散家庭的人的姓名：P2
要解散家庭的人是：P2
P2的第一代子孙是：P21
请选择要执行的操作：D
请输入要更改姓名的人的目前姓名：P13
请输入更改后的姓名：P14
P13已更名为P14
请选择要执行的操作：E
Press any key to continue

2 项目设计

2.1 数据结构设计

这个家谱管理系统的数据结构设计反映了典型的树结构，用于表示家族成员之间的层级关系。树结构是一种非线性数据结构，由节点和边组成，每个节点可以有零个或多个子节点，而每个子节点可以有零个或多个子节点，以此类推。树结构的一个重要特点是每个节点只有一个父节点，这种关系可以用指针来表示。c 该项目数据结构设计的特点和优势包括：

（1）层级关系的直观表示：使用树结构可以很自然地表示家族成员之间的父子和兄弟关系。

（2）灵活的成员操作：可以轻松添加或删除成员，同时更新树结构。

（3）递归搜索：递归方法能够有效地在树结构中搜索和操作数据。

2.2 类设计

2.2.1 节点表示（FamilyMember 类）

每个 FamilyMember 对象表示家谱中的一个成员，相当于树的一个节点。成员变量 nextSibling 和 firstChild 分别用于指向同一代中的下一个兄弟节点和该成员的第一个子节点，体现了树结构中的水平和垂直关系。name 存储每个成员的名字信息。

```
class FamilyMember
{
public:
    // 名字
    char name[STRING_SIZE];
    // 指向下一个兄弟
    FamilyMember *nextSibling = nullptr;
    // 指向第一个孩子
    FamilyMember *firstChild = nullptr;
    // 默认构造函数
    FamilyMember() {}
    FamilyMember(const char *name)
    {
        strncpy(this->name, name, STRING_SIZE - 1);
        this->name[STRING_SIZE - 1] = '\0';
    }
    FamilyMember(FamilyMember *sibling, FamilyMember *child, const char *name) :
    ↪ nextSibling(sibling), firstChild(child)
    {
        strncpy(this->name, name, STRING_SIZE - 1);
        this->name[STRING_SIZE - 1] = '\0';
    }
    ~FamilyMember() {}
};
```

2.2.2 树结构表示（FamilyTree 类）

rootMember 表示家族树的根节点，即祖先节点，从这里开始可以访问树中的所有成员。FamilyTree 类提供了多个方法来操作树结构，包括初始化树、添加节点、删除子树、重命名成员等。方法 findMemberAddress 和 findPreviousMemberAddress 使用递归方式遍历家谱树，用于查找特定成员的地址，这反映了深度优先搜索算法的应用。

```
1
2  class FamilyTree
3  {
4      friend class FamilyMember;
5
6  private:
7      // 指向树的根节点的指针
8      FamilyMember *rootMember;
9
10 public:
11     // 默认构造函数
12     FamilyTree() : rootMember(nullptr) {}
13     ~FamilyTree()
14     {
15         deleteSubTree(rootMember);
16         rootMember = nullptr;
17     }
18     // 初始化家谱，获取祖先姓名
19     void initializeTree();
20     // 建立家庭
21     void buildFamily();
22     // 查找成员
23     void findMember();
24     // 根据名字递归地找到某个家庭成员并返回其地址
25     FamilyMember *findMemberAddress(FamilyMember *currentMember, const char *name);
26     // 根据名字递归地找到某个家庭成员的前一个成员并返回其地址
27     FamilyMember *findPreviousMemberAddress(FamilyMember *currentMember, const char *name);
28     // 为某个成员增加孩子
29     void addChildToMember();
30     // 解散局部家庭
31     void dissolveSubTree();
32     // 递归地删除某个家庭成员及其子孙
33     void deleteSubTree(FamilyMember *memberRoot);
34     // 为某个家庭成员改名
35     void renameMember();
36     // 打印家谱
37     void printFamilyMember(FamilyMember *member, int indentLevel = 0);
38     void printFamilyTree();
39 };
```

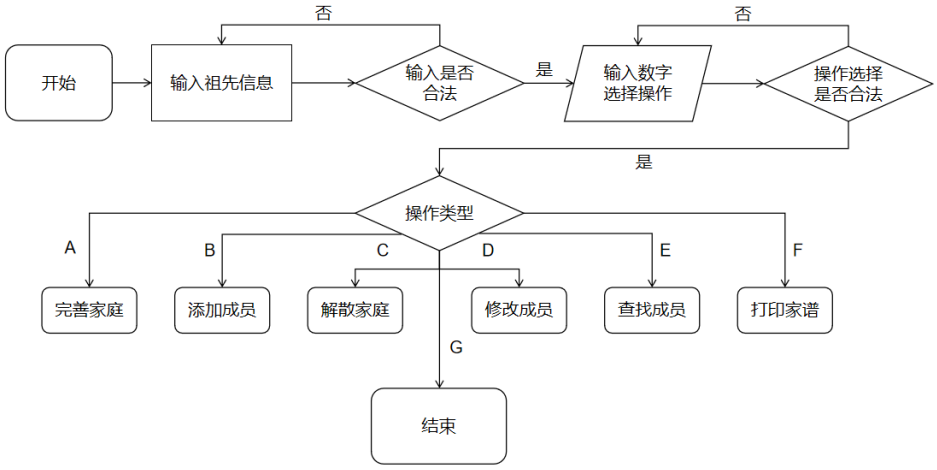
2.2.3 管理器类（FamilyTreeManager 类）

封装了 FamilyTree 对象，提供了用户界面和用户交互的逻辑。printMenu 方法显示操作菜单，run 方法根据用户输入执行对应的操作。

```
1
2   class FamilyTreeManager
3   {
4   public:
5       FamilyTree familyTree;
6       void printMenu();
7       void run();
8   };
9
10  void FamilyTreeManager::printMenu()
11  {
12      cout << "**              家谱管理系统                  **" << endl;
13      cout << "=============================================" << endl;
14      cout << "**            请选择要执行的操作              **" << endl;
15      cout << "**              A --- 完善家谱                **" << endl;
16      cout << "**              B --- 添加家庭成员            **" << endl;
17      cout << "**              C --- 解散局部家庭            **" << endl;
18      cout << "**              D --- 更改家庭成员姓名        **" << endl;
19      cout << "**              E --- 查找家庭成员            **" << endl;
20      cout << "**              F --- 打印所有家庭成员        **" << endl;
21      cout << "**              G --- 退出程序                **" << endl;
22      cout << "=============================================" << endl;
23  }
24
25  void FamilyTreeManager::run()
26  {
27      familyTree.initializeTree();
28
29      char choice;
30      while (true)
31      {
32          cout << " 请选择要执行的操作: ";
33          cin >> choice;
34          cin.clear();
35
36          switch (choice)
37          {
38          case 'A':
39              familyTree.buildFamily();
40              break;
41          case 'B':
42              familyTree.addChildToMember();
43              break;
44          case 'C':
45              familyTree.dissolveSubTree();
46              break;
47          case 'D':
48              familyTree.renameMember();
49              break;
50          case 'E':
51              familyTree.findMember();
52              break;
53          case 'F':
54              familyTree.printFamilyTree();
55              break;
56          case 'G':
57              cout << " 程序结束! " << endl;
58              return;
59          default:
```

```
60              cout << " 输入错误！ " << endl;
61              break;
62          }
63          cout << endl;
64      }
65  }
```

2.2.4 系统流程设计



3 项目实现

3.1 基本功能实现

3.1.1 查找成员地址（findMemberAddress）

递归地查找家谱树中具有指定名字的成员，并返回其地址。

实现步骤：（1） 从当前节点开始，检查名字是否与目标名字匹配。（2） 如果当前节点匹配，则返回当前节点的地址。（3） 如果当前节点不匹配，则递归检查其第一个孩子。（4） 如果孩子链表中没有找到匹配项，递归检查下一个兄弟节点。（5） 如果整个树中都没有找到匹配项，则返回 nullptr。

代码实现

```
FamilyMember *FamilyTree::findMemberAddress(FamilyMember *currentMember, const char *name)
{
    if (!currentMember)
    {
        return nullptr;
    }
    if (strcmp(currentMember->name, name) == 0)
    {
        return currentMember;
    }

    FamilyMember *childAddress = findMemberAddress(currentMember->firstChild, name);
    if (childAddress)
    {
        return childAddress;
    }

    FamilyMember *siblingAddress = findMemberAddress(currentMember->nextSibling, name);
    if (siblingAddress)
    {
        return siblingAddress;
    }

    return nullptr;
}
```

3.1.2 查找前一个成员地址（findPreviousMemberAddress）

递归地查找家谱树中具有指定名字的成员的前一个成员，并返回其地址。

实现步骤：（1） 从当前节点开始，检查其第一个孩子是否与目标名字匹配。（2） 如果匹配，则返回当前节点的地址。（3） 如果当前节点的兄弟与目标名字匹配，则返回当前节点的父节点地址。（4） 递归检查孩子链表。（5） 如果孩子链表中没有找到匹配项，递归检查兄弟链表。（6） 如果整个树中都没有找到匹配项，则返回 nullptr。

代码实现

```cpp
FamilyMember *FamilyTree::findPreviousMemberAddress(FamilyMember *currentMember, const char *name)
{
    if (!currentMember)
    {
        return nullptr;
    }

    // 检查当前节点的第一个孩子
    if (currentMember->firstChild && strcmp(currentMember->firstChild->name, name) == 0)
    {
        return currentMember;
    }

    // 检查当前节点的兄弟节点
    if (currentMember->nextSibling && strcmp(currentMember->nextSibling->name, name) == 0)
    {
        return currentMember;
    }

    // 递归检查孩子
    FamilyMember *childResult = findPreviousMemberAddress(currentMember->firstChild, name);

    if (childResult)
    {
        return childResult;
    }

    // 递归检查兄弟节点
    return findPreviousMemberAddress(currentMember->nextSibling, name);
}
```

3.2 初始化家谱（initializeTree）

初始化家谱树，创建根节点，即祖先节点。

3.2.1 实现步骤

（1）提示用户输入祖先的姓名。（2）创建一个 FamilyMember 对象作为根节点。（3）将根节点的名称设置为用户输入的祖先姓名。

3.2.2 代码实现

```cpp
void FamilyTree::initializeTree()
{
    char ancestorName[STRING_SIZE];
    cout << " 首先建立一个家谱！ " << endl;
    cout << " 请输入祖先的姓名： ";
    cin >> ancestorName;
    rootMember = new FamilyMember(ancestorName);
    cout << " 家谱创建成功！ " << endl;
```

```
    cout << " 此家谱的祖先是: " << rootMember->name << endl;
}
```

3.3 建立家庭（buildFamily）

为指定的家庭成员添加孩子。

3.3.1 实现步骤

（1）提示用户输入要添加孩子的家庭成员姓名。(2) 查找该家庭成员的地址。(3) 提示用户输入子女数目，并依次输入每个孩子的姓名。(4) 为每个孩子创建一个 FamilyMember 对象，并链接到父成员的子女链表中。

3.3.2 代码实现

```
void FamilyTree::buildFamily()
{
    char memberName[STRING_SIZE];
    int num_children;

    cout << " 请输入要建立的家庭成员的姓名: ";
    cin >> memberName;

    FamilyMember *currentMember = findMemberAddress(rootMember, memberName);

    if (!currentMember)
    {
        cout << " 未找到该成员! " << endl;
        return;
    }
    if (currentMember->firstChild)
    {
        cout << " 该成员已有子女! " << endl;
        return;
    }
    cout << " 请输入 " << currentMember->name << " 的子女数目: ";
    cin >> num_children;

    cout << " 请依次输入 " << memberName << " 的子女姓名: " << endl;
    cin >> memberName;
    // 创建第一个孩子
    FamilyMember *newChild = new FamilyMember(memberName);
    currentMember->firstChild = newChild;
    FamilyMember *lastChild = newChild;
    // 创建其他孩子
    for (int i = 1; i < num_children; i++)
    {
        cin >> memberName;

        FamilyMember *nextChild = new FamilyMember(memberName);
        lastChild->nextSibling = nextChild;
```

```
        lastChild = nextChild;
    }
    cout << " 家庭成员添加成功！ " << endl;
    cout << currentMember->name << " 的第一代子孙是: ";
    while (newChild)
    {
        cout << newChild->name << " ";
        newChild = newChild->nextSibling;
    }
    cout << endl;
}
```

3.4 完善家庭（addChildToMember）

为指定的家庭成员添加一个新的孩子。

3.4.1 实现步骤

（1）提示用户输入要添加孩子的家庭成员姓名。(2）查找该家庭成员的地址。(3）如果该成员没有孩子，创建一个新的孩子并将其设置为第一个孩子。(4） 如果该成员已有孩子，找到最后一个孩子，并在其后添加新孩子。

3.4.2 代码实现

```
void FamilyTree::addChildToMember()
{
    char memberName[STRING_SIZE];
    int num_children;

    cout << " 请输入要添加子女的家庭成员的姓名: ";
    cin >> memberName;

    FamilyMember *currentMember = findMemberAddress(rootMember, memberName);

    if (!currentMember)
    {
        cout << " 未找到该成员！ " << endl;
        return;
    }

    if (!currentMember->firstChild)
    {
        // 该成员没有子女
        cout << " 请输入 " << currentMember->name << " 新添加的子女姓名: ";
        cin >> memberName;

        FamilyMember *currentChild = new FamilyMember(memberName);
        currentMember->firstChild = currentChild;
        cout << currentMember->name << " 的第一代子孙是: " << currentChild->name << endl;
        return;
    }
```

```
    FamilyMember *currentChild = currentMember->firstChild;
    while (currentChild->nextSibling)
    {
        currentChild = currentChild->nextSibling;
    }
    cout << " 请输入 " << currentMember->name << " 新添加的子女姓名: ";
    cin >> memberName;

    currentChild->nextSibling = new FamilyMember(memberName);
    cout << currentMember->name << " 的第一代子孙是: ";
    while (currentChild)
    {
        cout << currentChild->name << " ";
        currentChild = currentChild->nextSibling;
    }
    cout << endl;
}
```

3.5 解散局部家庭（dissolveSubTree）

删除指定家庭成员及其所有子孙。

3.5.1 实现步骤

（1）提示用户输入要解散家庭的家庭成员姓名。（2）使用 findPreviousMemberAddress 找到该成员的前一个成员地址。（3） 根据前一个成员地址，删除指定成员及其子孙。

3.5.2 代码实现

```
void FamilyTree::dissolveSubTree()
{
    char memberName[STRING_SIZE];
    cout << " 请输入要解散家庭的家庭成员的姓名: ";
    cin >> memberName;

    // 检查根节点是否就是要删除的节点
    if (strcmp(rootMember->name, memberName) == 0)
    {
        cout << " 正在删除家庭 " << rootMember->name << " 及其所有子孙..." << endl;
        deleteSubTree(rootMember);
        rootMember = nullptr; // 清空根节点
        return;
    }

    // 找到待删除成员的前一个节点
    FamilyMember *previousMember = findPreviousMemberAddress(rootMember, memberName);
    if (!previousMember)
    {
        cout << " 未找到该成员！ " << endl;
        return;
    }
```

```
23
24      FamilyMember *targetSubTree = nullptr;
25
26      // 判断待删除节点是前一个节点的第一个孩子还是兄弟
27      if (previousMember->firstChild && strcmp(previousMember->firstChild->name, memberName) ==
        ↪  0)
28      {
29          targetSubTree = previousMember->firstChild;
30          previousMember->firstChild = targetSubTree->nextSibling; // 更新指针
31      }
32      else if (previousMember->nextSibling && strcmp(previousMember->nextSibling->name,
        ↪  memberName) == 0)
33      {
34          targetSubTree = previousMember->nextSibling;
35          previousMember->nextSibling = targetSubTree->nextSibling; // 更新指针
36      }
37      else
38      {
39          cout << " 未找到该成员！ " << endl;
40          return;
41      }
42
43      // 显示要删除的子孙信息
44      FamilyMember *currentChild = targetSubTree->firstChild;
45      if (!currentChild)
46      {
47          cout << memberName << " 没有子孙！ " << endl;
48      }
49      else
50      {
51          cout << memberName << " 的第一代子孙是： " << endl;
52          while (currentChild)
53          {
54              cout << currentChild->name << " ";
55              currentChild = currentChild->nextSibling;
56          }
57          cout << endl;
58      }
59
60      // 删除子树
61      cout << " 正在删除家庭 " << memberName << " 及其所有子孙..." << endl;
62      deleteSubTree(targetSubTree);
63  }
64
65  void FamilyTree::deleteSubTree(FamilyMember *memberRoot)
66  {
67      if (!memberRoot)
68      {
69          return;
70      }
71
72      // 删除所有子女节点
73      FamilyMember *child = memberRoot->firstChild;
74      while (child)
75      {
76          FamilyMember *nextChild = child->nextSibling;
77          deleteSubTree(child);
78          child = nextChild;
79      }
```

```
80
81      // 删除当前节点
82      delete memberRoot;
83  }
```

3.6 重命名成员（renameMember）

修改指定家庭成员的姓名。

3.6.1 实现步骤

（1） 提示用户输入要修改姓名的家庭成员当前的姓名。（2） 使用 findMemberAddress 函数递归查找成员。（3） 如果找到成员，提示用户输入新姓名，并更新成员的姓名。

3.6.2 代码实现

```
1   void FamilyTree::renameMember()
2   {
3       char memberName[STRING_SIZE];
4       char newName[STRING_SIZE];
5       cout << " 请输入要修改姓名的家庭成员目前的姓名: ";
6       cin >> memberName;
7
8       FamilyMember *currentMember = findMemberAddress(rootMember, memberName);
9       if (!currentMember)
10      {
11          cout << " 未找到该成员! " << endl;
12          return;
13      }
14
15      cout << " 请输入 " << memberName << " 的新姓名: ";
16      cin >> newName;
17
18      strncpy(currentMember->name, newName, STRING_SIZE - 1);
19      currentMember->name[STRING_SIZE - 1] = '\0';
20      cout << memberName << " 已更名为: " << newName << endl;
21  }
```

3.7 查找成员（findMember）

根据姓名查找家谱中的成员，并显示其信息。

3.7.1 实现步骤

（1） 提示用户输入要查找的家庭成员的姓名。（2） 使用 findMemberAddress 找到该成员的地址。（3） 如果找到成员，显示其姓名和子女信息。

3.7.2 代码实现

```
void FamilyTree::findMember()
{
    char memberName[STRING_SIZE];
    cout << " 请输入要查找的家庭成员的姓名: ";
    cin >> memberName;

    FamilyMember *memberAddress = findMemberAddress(rootMember, memberName);

    if (!memberAddress)
    {
        cout << " 未找到该成员! " << endl;
        return;
    }

    cout << " 成员 " << memberName << " 的信息如下: " << endl;
    FamilyMember *currentChild = memberAddress->firstChild;
    if (!currentChild)
    {
        cout << memberName << " 没有子孙! " << endl;
        return;
    }
    cout << memberName << " 的第一代子孙是: " << endl;
    while (currentChild)
    {
        cout << currentChild->name << " ";
        currentChild = currentChild->nextSibling;
    }
    cout << endl;
}
```

3.8 打印家谱（printFamilyTree）

递归打印家谱树的所有成员。

3.8.1 实现步骤

调用 printFamilyMember 函数，从根节点开始递归打印所有成员。

3.8.2 代码实现

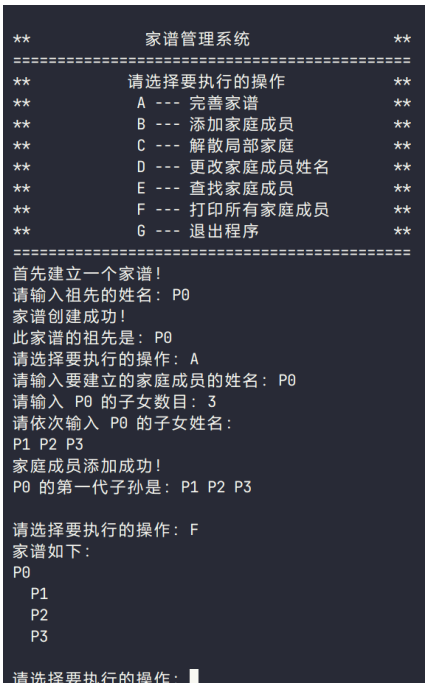
```
void FamilyTree::printFamilyMember(FamilyMember *member, int indentLevel)
{
    if (!member)
    {
        return;
    }
    for (int i = 0; i < indentLevel; i++)
    {
        cout << "  ";
```

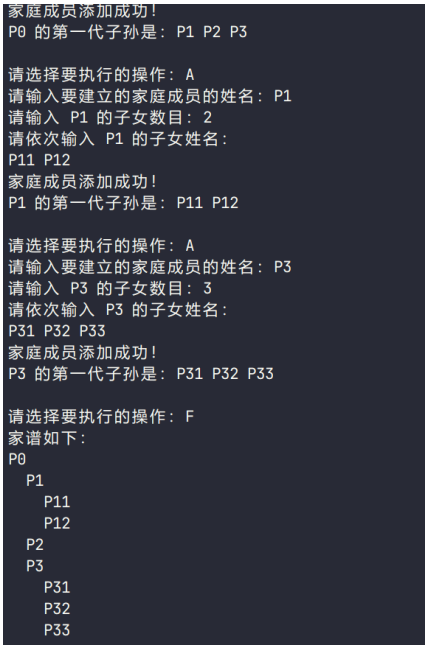
```
    }
    cout << member->name << endl;
    // 递归打印孩子
    FamilyMember *child = member->firstChild;
    while (child)
    {
        printFamilyMember(child, indentLevel + 1);
        child = child->nextSibling;
    }
}

void FamilyTree::printFamilyTree()
{
    cout << " 家谱如下: " << endl;
    printFamilyMember(rootMember);
}
```

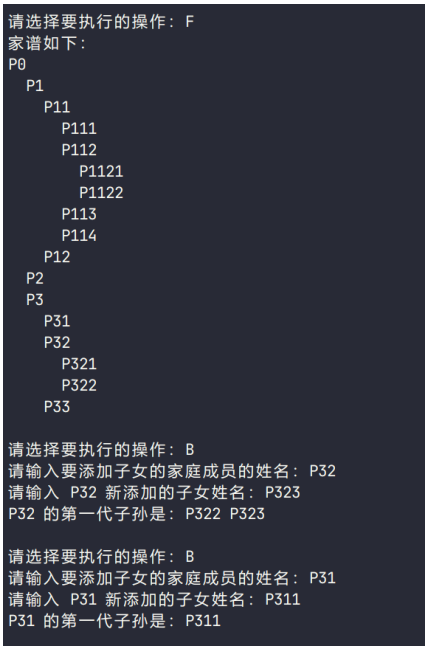
4 项目测试

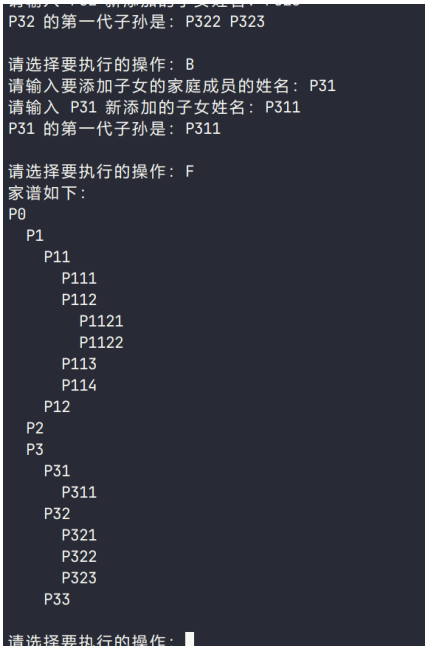
4.1 完善家谱



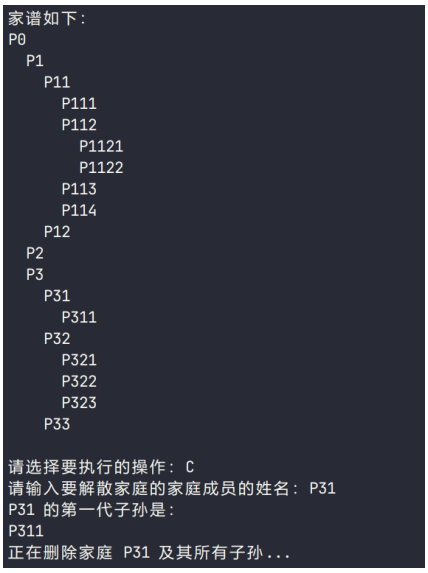


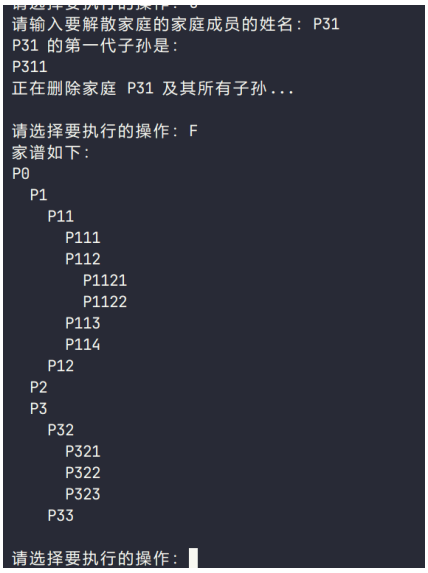
4.2 添加家庭成员



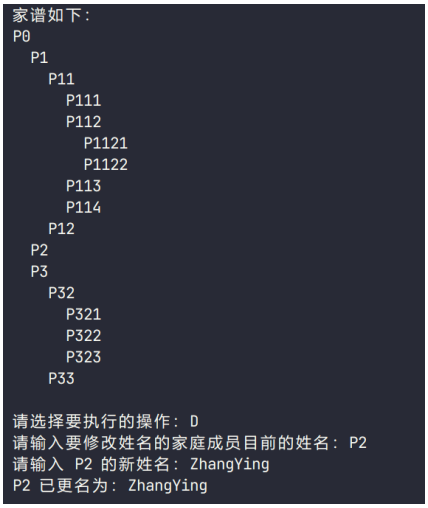


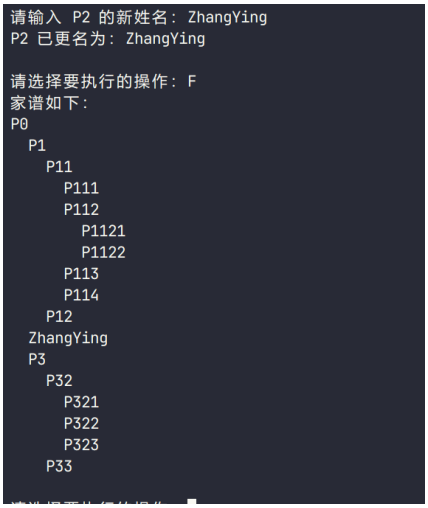
4.3 解散家庭





4.4 修改成员姓名





4.5 查询家庭成员

```
家谱如下：
P0
  P1
    P11
      P111
      P112
        P1121
        P1122
      P113
      P114
    P12
  ZhangYing
  P3
    P32
      P321
      P322
      P323
    P33

请选择要执行的操作： E
请输入要查找的家庭成员的姓名： P11
成员 P11 的信息如下：
P11 的第一代子孙是：
P111 P112 P113 P114

请选择要执行的操作： E
请输入要查找的家庭成员的姓名： P12
成员 P12 的信息如下：
P12 没有子孙！
```

4.6 打印家谱

```
请选择要执行的操作： F
家谱如下：
P0
  P1
    P11
      P111
      P112
        P1121
        P1122
      P113
      P114
    P12
  ZhangYing
  P3
    P32
      P321
      P322
      P323
    P33
```

4.7 健壮性测试

程序在输入错误时能够给出相应的提示，如下图所示。

```
请选择要执行的操作： 3
输入错误！

请选择要执行的操作： A
请输入要建立的家庭成员的姓名： P4
未找到该成员！

请选择要执行的操作： q
输入错误！

请选择要执行的操作： B
请输入要添加子女的家庭成员的姓名： P0
请输入 P0 新添加的子女姓名： sth
P0 的第一代子孙是： P3 sth

请选择要执行的操作： C
请输入要解散家庭的家庭成员的姓名： P22
未找到该成员！

请选择要执行的操作： D
请输入要修改姓名的家庭成员目前的姓名： P8
未找到该成员！
```

5 项目心得与体会

通过参与家谱管理系统的开发，我获得了宝贵的实践经验和深刻的认识，这些不仅增强了我的专业知识，也提升了我的实践技能和解决问题的能力。在设计 FamilyMember 和 FamilyTree 类的过程中，我深刻体会到了数据结构在解决实际问题中的重要性。通过使用链表来表示家庭成员之间的关系，我学会了如何将抽象的家族关系转化为具体的数据结构。这种结构不仅易于实现，而且方便管理和扩展。通过实现 FamilyMember 和 FamilyTree 类，我加深了对面向对象编程（OOP）的理解。我学会了如何封装数据和操作，如何使用继承和多态，以及如何设计易于维护和扩展的类。

总之，这个项目不仅让我掌握了数据结构和算法的具体应用，还提升了我的编程能力、问题解决能力和系统设计能力。这些经验和技能将为我未来的学习和工作奠定坚实的基础。