**数据结构课程设计《家谱管理系统》项目说明文档**

2253893 苗君文 软件工程

**1. 项目简介**

**1.1. 项目背景**

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，查询家庭成员的功能。

**1.2. 项目要求**

本项目的实质是完成对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。

**1.3. 输入格式**

本项目首先需要用户建立家谱，即输入祖先的姓名，而后可以输入要执行的操作（大小写均可）以完成对家谱管理系统的操作。在不同的操作选项中，用户需要输入如某家庭成员的姓名、家庭成员的个数等。

**1.4. 输出格式**

本项目针对输入有详细的输入错误处理，若用户输入错误，则程序会输出相关提示信息，并要求重新输入。而针对用户选择的操作，可以输出相应的操作结果并输出当前操作的家庭成员的第一代子女。

**2. 设计思路**

**2.1. 数据结构设计**

由于家谱的性质，必然考虑使用树的结构。而又由于家庭成员的子女数量是不确定的，若使用一般的树来存放家庭成员，势必会浪费空间，并且在查找家庭成员时所用时间也会较长。因此，在本项目中采用了子女-兄弟链表表示法来存储家谱信息。

这种存储表示是一种二叉树表示法，每个结点的度d=2，是最节省存储空间的树的存储表示，因为对于不同家庭成员的子树，他们共享一个父亲节点，可以减少重复存储。它的每个结点由3个域组成，分别是data，firstChild和nextSibling。假设某结点A有两个孩子B和C，而A没有兄弟，那么A的firstChild域就要指向它的长子B，而B由于有兄弟C，则它的nextSibling就需要指向它的兄弟C。因此这种存储方法也被称为“左子女，右兄弟”。

这种表示方法非常适合用于频繁寻找子女的应用，其时间复杂度是O(d)，d是树的度；并且添加新子女时，只需要调整nextSibling指针而不需要移动整个字数，可以提高插入操作得到效率。而寻找父节点需要遍历二叉链表，时间复杂度为O(n)，其中n是树中结点个数。

**2.2. 类设计**

TreeNode类和Tree类共同使用，以表示和管理家谱树，为与家庭相关的操作如插入，删除，查询提供了灵活且高效的结构。TreeNode类封装了每个家庭成员的属性，而Tree类则具有各种对家谱树的操作，二者构成了左子女-右兄弟的链表二叉树，满足了项目的功能。

**2.2.1. TreeNode类（树的结点类）**

TreeNode类代表家谱中的一个节点。每个节点对应一个家庭成员，包含有关该成员的信息，以及指向其第一个孩子和下一个兄弟的指针。

成员变量：

* name：代表家庭成员的姓名。
* firstChild：指向当前家庭成员的第一个孩子节点的指针。
* nextSibling：指向当前家庭成员的下一个兄弟节点的指针。

成员函数：

* 构造函数：用给定的姓名、第一个孩子和下一个兄弟初始化TreeNode对象。

**2.2.2. Tree类（左子女-右兄弟二叉树类）**

Tree类代表整个家谱树，管理对树的操作，如添加成员、删除局部家庭、查找成员以及显示信息等。

成员变量：

* root：指向家谱树根节点的指针。
* current：指向树中当前处理节点的指针。

成员函数：

* 构造函数：初始化一个Tree对象，将root和current设置为NULL。
* BuildRoot：建立家谱树的根节点，并将其设置为当前节点。
* Root：将树的根节点设置为当前节点。
* Parent：将当前节点设置为其父节点。
* FirstChild：将当前节点设置为其第一个孩子节点。
* NextSibling：将当前节点设置为其下一个兄弟节点。
* FindParent：在以t为根的树中找到节点p的父节点。
* Find：在整个树中找到姓名为target的节点，并将其设置为当前节点。
* IsEmpty：检查树是否为空。
* Find(TreeNode\* p, string target)：在以p为根的子树中找到姓名为target的节点，并将其设置为当前节点。
* DisplayChildren：显示当前节点的第一代子孙。
* InsertChild：将具有给定姓名的孩子插入到当前节点。
* DeleteSubtree：删除以给定节点为根的子树。
* Menu：显示用户与家谱树交互的菜单，允许执行不同的操作，如建立家庭、添加成员、解散家庭、更改姓名和查询成员等。

**2.3. 主程序设计**

主程序首先需要创建一个Tree对象（myTree）用于表示整个家谱。通过调用其Menu函数，程序通过交互式菜单供用户选择不同的操作。Menu函数中程序首先要求建立一个家谱，用户需要输入祖先姓名，而后程序会创建以祖先为根节点的家谱二叉树。之后，进入主循环，根据用户选择执行相应的操作，直到用户选择退出程序。在循环中调用相应的二叉树操作函数，如插入家庭成员、展示第一代子女、删除家庭成员、删除子树、寻找家庭成员等，然后输出相应的处理结果。

**2.4. 输入错误处理设计**

**2.4.1. 单个参数的输入错误处理**

在输入某家庭成员的子女数时，使用 while 循环不断尝试获取用户输入，直到输入满足要求。使用 cin.fail() 及min、max来判断输入是否出错，检查输入是否在有效范围内。如果输入无效，则输出错误信息，清除输入缓冲区并忽略之后的字符，并重新输出输入的提示信息。通过一个字符来获取输入一个数之后的字符以检查输入字符的个数，若不正确，也输出错误信息，并重新输出输入的提示信息，执行相应清除操作。如果输入有效，则跳出循环。

类似的操作方法可以用在输入操作选项中，区别仅仅在于输入的内容是char型而非int型，用户输入的操作选项是大小写均可的，否则输入内容或输入个数错误都需要重新输入。

**2.4.2. 与家谱操作相关的错误处理**

在许多操作选项中，都需要输入要操作的家庭成员的姓名，如果用户指定的家庭成员通过Find函数未查找到，则需要输出用户所写家庭成员并不存在于家谱中，无法对其进行相应的操作。并且输入姓名时，用户可能会输入空格，那么，为防止第一个空格后的内容会影响后续的输入，因此每一次输入姓名后，都需要清空缓冲区。

在所有操作选项中，都需要展示当前家庭成员的第一代子孙，但也会出现当前家庭成员并没有任何子女的情况，因此通过firstChild指针是否为空来判断，若无子孙，则需要输出相关的提示信息。

**3. 功能实现**

**3.1. Menu函数**

此函数是实现家谱管理系统的核心部分， 函数首先会输出菜单界面并让用户通过输入家谱姓名以建立家谱，从而可以选择想要执行的操作选项。

用户选择完善家谱后，程序要求输入要完善家谱的成员，并找到该成员将此节点设置为当前结点，并让用户输入其儿女个数，再依次输入他们的姓名，通过调用InsertChild函数将这些儿女添加到家谱中。完成后输出当前结点的第一代子女。

用户选择添加家庭成员后，程序要求输入要添加儿子（或女儿）的人的姓名，找到后将其设置为当前结点，并要求输入其子女的姓名，然后通过调用InsertChild函数，在当前节点下添加了一个新的孩子。完成添加后输出当前结点的第一代。

用户选择解散局部家庭后，程序要求输入要解散家庭的人的姓名。通过调用 Find 和 DeleteSubtree函数，找到该人的节点并删除其子树，达到解散局部家庭的效果。完成删除后输出当前结点的第一代子女。

用户选择更改家庭成员姓名后，程序要求输入要更改姓名的人的目前姓名和更改后的姓名。如果姓名与原姓名相同，则提示相关信息；若不同则通过调用 Find 方法找到该人的节点，并将其姓名更新。

用户选择查询家庭成员后，程序要求输入要查询的人的姓名。通过调用Find和DisplayChildren方法，找到该人的节点并显示其第一代子孙。

用户选择退出程序后，程序输出退出消息并结束运行。

**3.2. InsertChild函数**

在多数操作中，都要用到InsertChild函数才能实现，该函数是向当前节点添加一个孩子节点，表示当前家庭成员的新儿女。以下是该函数的实现思路：

首先，创建一个新的TreeNode对象，该对象代表新儿女，其name成员被设置为输入的孩子姓名。然后，检查当前节点是否有孩子（即firstChild是否为空）。如果当前节点没有孩子，将新节点设置为当前节点的第一个孩子（firstChild指针指向新节点）。如果当前节点已经有孩子，需要找到当前节点孩子链表中的最后一个兄弟节点，以便将新节点插入到其后。遍历孩子链表，找到最后一个兄弟节点。将新节点插入到最后一个兄弟节点的后面，使其成为最后一个兄弟节点的下一个兄弟。

这样，通过InsertChild函数，程序可以在当前节点下添加一个新的儿女节点，保持了家谱的正确结构。这个函数的设计考虑到了家庭成员可能有多个儿女的情况，使用了左子女-右兄弟二叉树的特性，使得在家谱中插入新成员变得简单而高效。

**3.3. DeleteSubtree函数**

该函数对于解散局部家庭这一操作非常重要，因为解散该家庭成员意味着该家庭成员的所有孩子都要被删除，也就是删除以给定节点为根的子树。以下是该函数的实现思路：

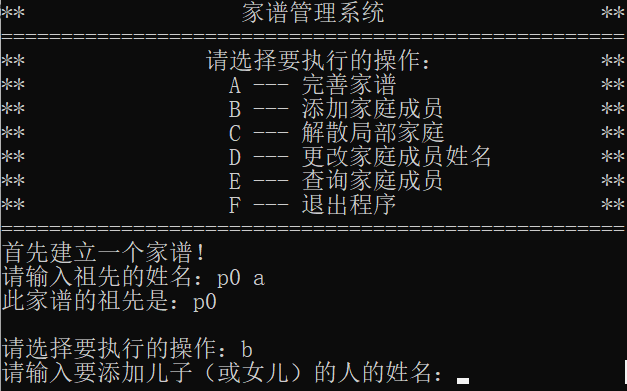
首先，检查输入的节点是否为空。如果为空，直接返回，不进行任何操作。然后，通过一个循环，遍历节点的所有孩子节点。对于每一个孩子节点，递归调用DeleteSubtree函数，以删除其子树。在递归调用之后，删除当前孩子节点，释放其内存。循环继续，处理下一个孩子节点，直到所有孩子节点都被处理完毕。最后，将节点的firstChild指针设置为NULL，表示不再有子节点。

DeleteSubtree函数的递归调用确保了深度优先地删除整个子树。递归调用的目标是子树的每个孩子节点。递归过程中，函数会自动处理孩子节点的孩子节点，以此类推，直到叶子节点。在递归过程中，每个节点都会被删除，并释放相应的内存。

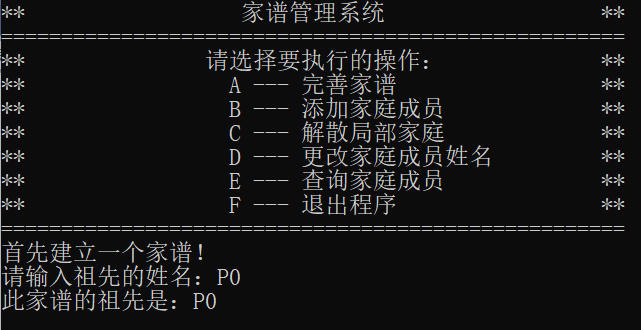
**4. 测试结果（包括边界测试）**

**4.1. 建立一个家谱**

（1）输入的祖先姓名含有空格，则程序会只接受第一个空格前的字符串，而之后输入的内容会在缓冲区被清空，不会影响后续的输入。

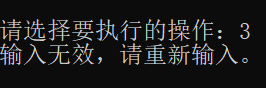


（2）输入正确

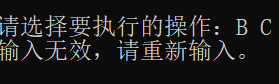


**4.2. 选择操作选项**

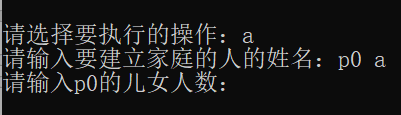
（1）输入内容错误



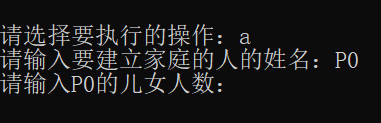
（2）输入个数错误



（3）输入的姓名含有空格，则程序会只接受第一个空格前的字符串，而之后输入的内容会在缓冲区被清空，不会影响后续的输入。

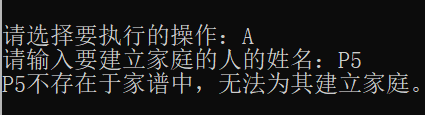


（3）输入正确即可进入相应的操作（大小写均可）



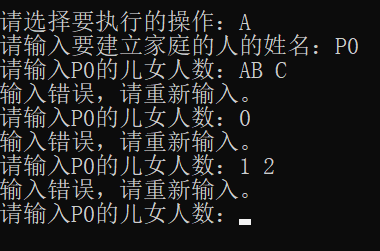
**4.3. 完善家谱（选项A）**

（1）输入的姓名不存在于家谱中

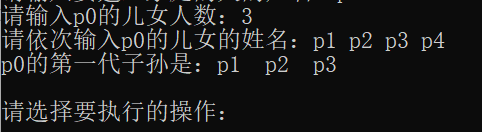


（2）输入的儿女人数错误

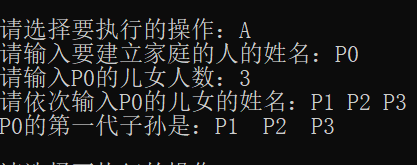
包括输入的数据类型错误，输入的数字个数错误，输入的数字范围错误。



（3）输入儿女姓名却多输入，程序会清除缓冲区，不影响后续输入。

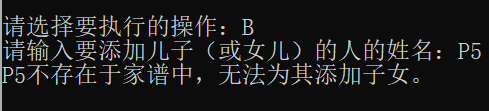


（3）输入正确（即姓名存在于家谱中，输入的儿女人数也正确）



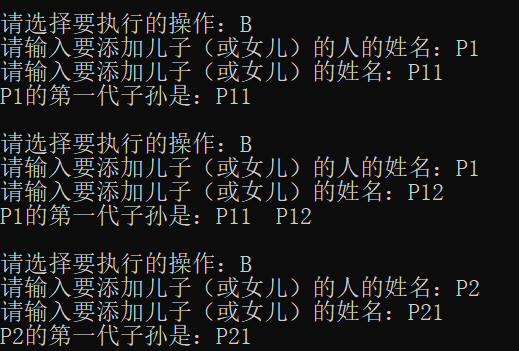
**4.4. 添加家庭成员（选项B）**

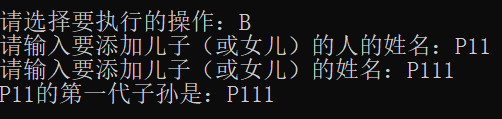
（1）输入的姓名不存在于家谱中



（2）输入正确（即姓名存在于家谱中）

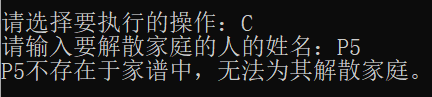
包括为无子女的人添加家庭成员与为有子女的人添加家庭成员。



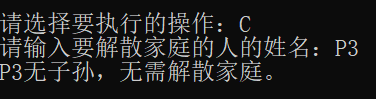


**4.5. 解散局部家庭（选项C）**

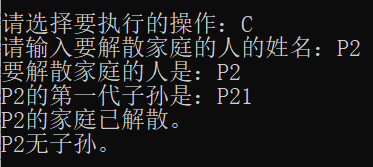
（1）输入的姓名不存在于家谱中



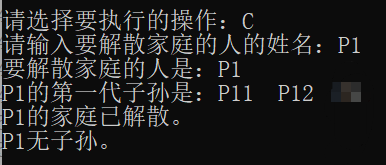
（2）输入的姓名无子女（无法解散家庭）



（3）输入的姓名只有一个子女



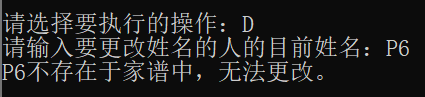
（4）输入的姓名有多代子女



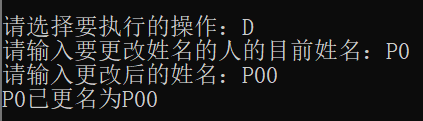


**4.6. 更改姓名（选项D）**

（1）输入的姓名不存在于家谱中

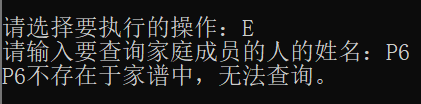


（2）输入正确（即姓名存在于家谱中）

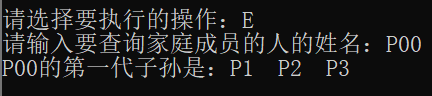


**4.7. 查询家庭成员（选项E）**

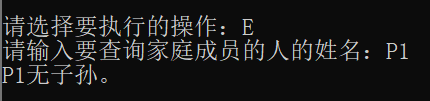
（1）输入的姓名不存在于家谱中



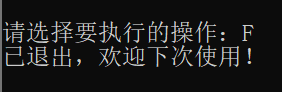
（2）输入的姓名有子孙



（3）输入的姓名无子孙



**4.8. 退出程序（选项F）**



**4.9. 程序的测试结果总览（此处仅展示输入正确有效的情况）**

