数据结构课程设计

1751650 蒋伟博

家谱管理系统

0. 项目简介

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。本项目通过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. 概述

- 项目要求
  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。

- 实现方法
  使用树完成家谱的构建。由于每一代子女数量不一，故采用儿子-兄弟的方法构建树的节点。

2. 类及类成员介绍

- **Node**类

  - 成员变量

成员名称	属性	类型	描述
son	public	Node*	指向长子的指针
brother	public	Node*	指向兄弟的指针
isInFamilyTree	public	bool	描述此节点是否还在家谱中
name	private	string	此节点成员姓名

  说明：设计isInFamilyTree是因为在解散家庭时，对家庭的主人采用了懒惰删除，故用此bool变量描述此节点是否已被删除。

  - 成员函数

函数名称	返回值类型	描述
Node	无	构造函数
getName	string	返回节点的姓名

函数名称	返回值类型	描述
changeName	void	修改节点姓名
addSon	bool	为节点添加子女
addBrother	bool	为节点添加兄弟

- **Tree**类
  - 成员变量

成员名称	属性	类型	描述
ancestor	private	Node*	指向家谱祖先的指针

  - 成员函数

函数名称	返回值类型	描述
Tree	无	构造函数
~Tree	无	析构函数
releaseNode	void	释放某一节点
find	Node*	通过姓名查找节点
addFamily	bool	添加一个家庭（多个成员）
addOneSon	bool	添加一个子女
dismissFamily	bool	解散一个家庭
changeName	bool	修改一个节点的名字
showSons	void	展示节点的所有子女

3. 核心代码解释

- 为当前节点添加儿子

```
bool Node::addSon(Node* newSon)
{
  if (son == NULL)//如果此节点还没有儿子，直接将新儿子置于其下
  {
    son = newSon;
    return true;
  }
  else//若有儿子，将新儿子置为旧儿子兄弟
  {
    if (son->addBrother(newSon))
      return true;
```

```
    else
      return false;//插入失败
  }
}

bool Node::addBrother(Node* brother)
{
  Node* temp = this;
  while (temp->brother != NULL)//找到brother链表的末尾
  {
    temp = temp->brother;
  }
  temp->brother = brother;
  return true;
}
```

对当前节点执行添加儿子操作时分为两类情况考虑：一是此节点没有儿子，这种情况下直接将此节点的儿子指针指向要添加的儿子节点即可；二是此节点已经有儿子（不论此儿子是否仍处于家谱中），此时需要将要插入的儿子节点放到此节点的儿子链表末尾，即执行addBrother操作。

- 按姓名查找家族成员

```
Node* Tree::find(string name, Node* node)
{
  if (node == NULL)//已遍历到树的最深处
    return NULL;
  else
  {
    if (node->isInFamilyTree && node->getName() == name)//找到节点
      return node;
    else//未找到节点
    {
      Node* temp = node->brother;
      temp = find(name, temp);
      if (temp != NULL)//在node的兄弟中找到了节点
        return temp;
      else//node的兄弟中不包含要寻找的节点
        temp = node->son;

      temp = find(name, temp);
      if (temp != NULL)//在node的子孙中找到了节点
        return temp;
      else
        return NULL;
    }
  }
}
```

整个查找过程采用了深度优先遍历，在找到第一个符合要求（仍处在家谱中且名字相符）的节点后返回指向节点的指针。此函数没有特意为重名的情况准备解决方案。

- 解散局部家庭

```
bool Tree::dismissFamily()
{
  string father;
  cout << "请输入要解散家庭的人的姓名：";
  cin >> father;

  Node* temp = find(father, ancestor);
  if (temp == NULL)
  {
    cerr << "不存在此人！" << endl;
    return false;
  }

  cout << "要解散家庭的人是：" << father << endl;
  showSons(temp);
  releaseNode(temp->son);
  temp->isInFamilyTree = false;//进行懒惰删除
  temp->son = NULL;

  /*为了防止删空家族树，规定祖先不可删除*/
  if (ancestor->isInFamilyTree == false)//家族树已空
    ancestor->isInFamilyTree = true;
  return true;
}
```

解散家庭时，该家庭的祖先（根节点）将不会被释放，而是标记之后进行懒惰删除。这是因为任一节点的第一代子女采用单向链表的方式链接，直接将其中一名子女所在的节点进行释放会破坏整条链表，使之断裂。而要解散的家庭的祖先即处于这一种尴尬的境地，他的子孙属于一个整体，可以被直接释放掉而不破坏整个家谱其他部分的完整性。
除此之外，为了避免删空整个家族树使得无法进行操作，程序特别规定家族的原始祖先是不允许被删除的（即使是懒惰删除）。在解散家族祖先的家庭时，祖先会被保留下来。

4. 项目运行效果

- 确定祖先
  在程序运行之初，用户会被要求输入该家谱祖先的名字，该祖先将不被允许删除，在解散家庭时也会被保留。

  ```
  Node* temp = new Node(name);
  ...
  ancestor = temp;
  ```

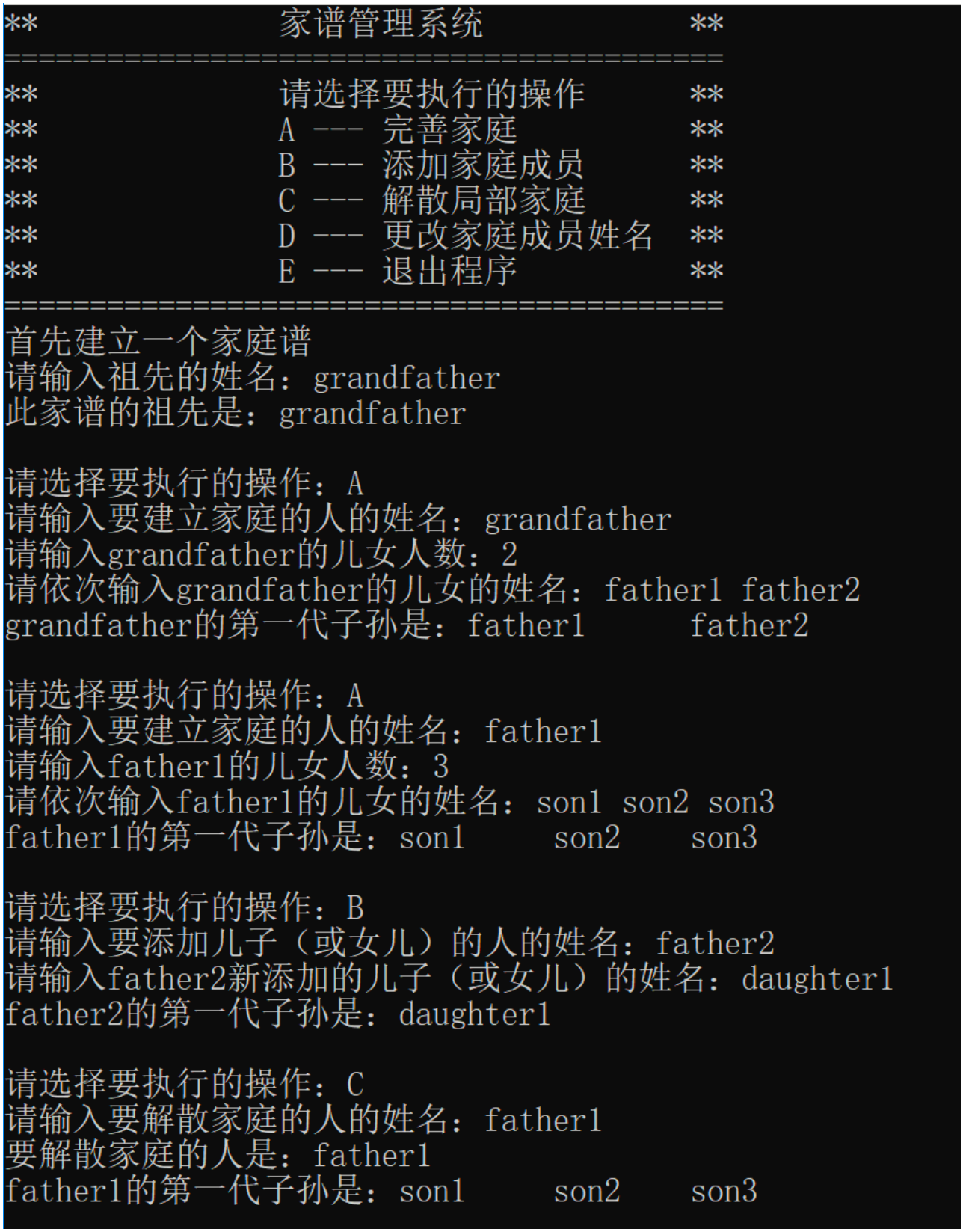
- 完善家庭
  用户选择此项功能后，将被要求输入需要被完善的人的名字，添加儿女的数量和儿女的名字。不允许完善不存在的人。

- 添加家庭成员
  为某人添加一个子女，要求输入此人的名字和儿女的名字。不允许为不存在的人添加子女。

- 解散局部家庭
  效果为删除某人和他的所有子孙后代，祖先不允许被通过这种方式删除。

- 更改家庭成员姓名
  用户被要求输入某人现在的名字和更改后的名字。不允许修改不存在的人的名字。

- 运行截屏



```
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：father2
请输入father2新添加的儿子（或女儿）的姓名：daughter2
father2的第一代子孙是：daughter1        daughter2

请选择要执行的操作：D
请输入要更改姓名的人的目前的姓名：daughter1
请输入更改后的姓名：daughter3
daughter1已更名为daughter3
```