家谱管理系统

1.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.项目功能要求

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。

3.核心代码及其功能

- **声明的数据类**
    - person：用于作为单个节点

```
class person{                                   //用于保存一个人的信息
public:
 string name;                                  //该人姓名
 vector<person*>children;                      //保存指向该人的孩子的指针
};
```

    - multiTree:模拟多叉树结构

```
class multiTree{
public:
 multiTree();
 ~multiTree(){}
 void Create();                                 //完善家谱
```

```
 void Add();                            //向家谱中添加孩子
 void Delete();                         //在家谱中解散成员
 void Modify();                         //修改家谱信息
 void deleteChild(person* start);       //在家谱中找到孩子并删除
 person* findName(person* ptr,string searchName);       //寻找姓名为
searchName的孩子
 void Display(person* ptr);             //展示家谱
private:
 person* head;
};
```

- System:抽象整个系统

```
class multiTree{
public:
 multiTree();
 ~multiTree(){}
 void Create();                         //完善家谱
 void Add();                            //向家谱中添加孩子
 void Delete();                         //在家谱中解散成员
 void Modify();                         //修改家谱信息
 void deleteChild(person* start);       //在家谱中找到孩子并删除
 person* findName(person* ptr,string searchName);       //寻找姓名为
searchName的孩子
 void Display(person* ptr);             //展示家谱
private:
 person* head;
};
```

- **创建家谱**
  这里将创建家谱的过程放在了multiTree的构造函数中，首先接受一个头节点作为整个多叉树的祖先。

```
multiTree::multiTree() {                        //构造家谱（即构造多叉树）
    cout<<"**                  家庭管理系统                        **"<<endl;
    cout<<"================================================="<<endl;
    cout<<"**                请选择要执行的操作                  **"<<endl;
    cout<<"**                A.---完善家谱                      **"<<endl;
    cout<<"**                B.---添加家庭成员                   **"<<endl;
    cout<<"**                C.---解散局部家庭                   **"<<endl;
    cout<<"**                D.---更改家庭成员姓名               **"<<endl;
    cout<<"**                E.---退出程序                      **"<<endl;
    cout<<"================================================="<<endl;
    cout<<"首先建立一个家谱"<<endl;
    cout<<"请输入祖先的姓名:";                  //建立家谱祖先
    auto ptr=new person;
    cin>>ptr->name;
    cout<<"该家谱的祖先是:"<<ptr->name<<endl;
    head=ptr;
}
```

- **寻找姓名为searchName的家庭成员**
  因为在整个系统中平频繁的使用到了寻找某位家庭成员是否存在的功能，所以这里将这一过程抽象为findName函数，其中ptr代表当前的根节点，searchName代表要寻找的家庭成员的姓名，这里

采用了递归搜索的方式，也就是如果当前节点的子节点中没有该姓名的成员孩子那么就进入该成员的子节点中进行查找，直到找到或者查找完所有可查找节点为止。返回值为指向名字为searchName节点的指针，如果没有找到返回NULL。

```
person* multiTree::findName(person* ptr,string searchName){
    if(ptr==NULL||ptr->name==searchName){    //如果找到或者找完就直接返回当前指针
        return ptr;
    }
    else{                                    //如果没有找到或者还可以继续向下查找
        for(int i=0;i<ptr->children.size();i++){
            auto tempPtr=findName(ptr->children[i],searchName);      //递归寻找
            if(tempPtr) {
                if (tempPtr->name == searchName) {
                    return tempPtr;
                }
            }
        }
    }
    return NULL;
}
```

- **完善家谱功能**
  首先调用searchName函数寻找名字为想要寻找的家庭成员，如果没有找到就输出没有找到信息，返回。如果找到就询问其子女人数（检测是否大于0）。这里由于输入的姓名都是按照字符串形式进行读入，所以采用了getline函数遇到‘ ’停止读，但这里需要将最后一个姓名独立出来，因为它的后一个字符为’\0’而不是’ ‘。

```
void multiTree::Create() {
    cout<<"请输入要建立家庭的人的姓名;";
    string searchName;
    cin>>searchName;
    auto ptr=findName(head,searchName);      //寻找名字为searchName的家庭成员
    if(ptr){                                 //如果找到该位家庭成员
        int childNum;
        bool check=1;
        while(check){
            cout<<"请输入"<<searchName<<"的儿女人数:";
            cin>>childNum;
            if(childNum<0){                  //检查输入是否合法
                cout<<"the number of children can not less than 0!"<<endl;
            }
            else{
                check=0;
            }
        }
        cout<<"请依次输入"<<searchName<<"的子女姓名:";
        getchar();                           //去除上次输入的'\n'
        //最后一次读入数据需要独立出来因为最后一个字符串末尾是'\n'而不是' '
        for(int i=0;i<childNum-1;i++){
            auto childPtr=new person;
            getline(cin,childPtr->name,' ');
            (ptr->children).push_back(childPtr);
        }
        auto childPtr=new person;
        getline(cin,childPtr->name);
        (ptr->children).push_back(childPtr);
```

```
        Display(ptr);
    }
    else{                                       //如果没有找到该成员
        cout<<"该家庭无此人!"<<endl;
        return;
    }
}
```

- **删除节点**
  这里首先抽象出一个删除节点的函数，这里采用了递归删除的方式，不断递归寻找其子节点并delete。start指针指向当前遍历到的节点。

```
void multiTree::deleteChild(person *start) {     //递归删除家庭成员从而删除其所有孩子节点
    if(start){
        return;
    }
    else{
        for(int i=0;i<start->children.size();i++){
            deleteChild(start->children[i]);
        }
        delete(start);
    }
}
```

- **解散家庭功能**
  先调用finName函数找到名字为searchName的家庭成员，然后调用deleteChild删除该成员的所有子节点。

```
void multiTree::Delete() {
    cout<<"请输入要解散家庭的人的姓名:";
    string searchName;
    cin>>searchName;
    auto ptr=findName(head,searchName);          //寻找名字为searchName的家庭成员
    if(ptr){                                     //如果找到该位家庭成员
        cout<<"要解散家庭的人是:"<<searchName<<endl;
        Display(ptr);
        deleteChild(ptr);                        //调用函数删除该家庭成员及其子节点
    }
    else{                                        //如果没有找到该位家庭成员
        cout<<"该家庭无此人"<<endl;
    }
}
```

- **修改成员信息功能**
  先调用findName函数找到名字为searchName的家庭成员，然后修改信息

```
void multiTree::Modify(){
    cout<<"请输入要更改姓名的人目前姓名是:";
    string searchName;
    cin>>searchName;
    auto ptr=findName(head,searchName);          //寻找名字为searchName的家庭成员
    if(ptr){                                     //如果找到该位家庭成员
        cout<<"请输入修改后的姓名:";
        string tempName=ptr->name;
```

```
        cin>>ptr->name;
        cout<<tempName<<"已改名为"<<ptr->name<<endl;
    }
    else{                                           //如果没有找到该位家庭成员
        cout<<"该家庭无此人"<<endl;
        return;
    }
}
```

- **添加家庭成员功能**
  先调用findName函数找到名字为searchName的家庭成员，然后将其孩子作为节点push到保存该节点子节点的vector中

```
void multiTree::Add() {
    cout<<"请输入要添加儿子或女儿的姓名:";
    string searchName;
    cin>>searchName;
    auto ptr=findName(head,searchName);          //寻找名字为searchName的家庭成员
    if(ptr){                                     //如果找到该位家庭成员
        cout<<"请输入"<<searchName<<"要添加的儿子（或女儿）的姓名:";
        auto newChild=new person;
        cin>>(newChild->name);
        ptr->children.push_back(newChild);
        Display(ptr);
    }
    else{                                        //如果没有找到该位家庭成员
        cout<<"该家庭无此人"<<endl;
        return;
    }
}
```

4.项目实例

> 注：以下代码均为顺序执行

- **进入界面**

```
/Users/kirito/CLionProjects/untitled/cmake-build-debug/untitled
**                  家庭管理系统                    **
=====================================================
**                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
该家谱的祖先是:p0
```

- **完善家谱功能**

```
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的子女姓名: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的子女姓名:p11 p12 p13
p1的第一代子孙是:p11 p12 p13
-----------
```

- **添加家庭成员功能**

```
请选择要执行的操作:B
请输入要添加儿子或女儿的姓名:p2
请输入p2要添加的儿子（或女儿）的姓名:p21
p2的第一代子孙是:p21
-----------
```

- **解散家庭成员功能**

```
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
-----------
```

- **修改家庭成员信息功能**

```
请选择要执行的操作:D
请输入要更改姓名的人目前姓名是:p13
请输入修改后的姓名:p14
p13已改名为p14
-----------
```

- **非法输入处理和未找到成员处理**

```
请选择要执行的操作:F
it seems that you didn't input a correct operand!
-----------
请选择要执行的操作:B
请输入要添加儿子或女儿的姓名:p4
该家庭无此人
```

- **退出程序**

请选择要执行的操作:E

```
Process finished with exit code 0
```