项目说明文档
——家谱管理系统

姓名：吴桐欣
学号：1652677
同济大学 软件学院 软件工程专业

目录

1.项目概述

1.1 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。

1.1.1 功能分析

本项目要求实现功能

（1） 搜索成员；
（2） 添加成员；
（3） 建立家谱
（4） 删除成员；
（5） 更改信息；
（6） 输出家谱。

1.2 文件目录

（1）P06_1652677_吴桐欣_说明文档.docx（本文档）
（2）P06_1652677_吴桐欣.exe（可执行文件）
（3）P06_1652677_吴桐欣.cpp（源文件）
（4）P06_1652677_吴桐欣.h（头文件）

1.3 操作指南

（1）运行程序后，将获得程序提示“首先建立一个家谱 —— 请输入祖先姓名：”
用户输入祖先姓名。
（2）程序提示“请选择要执行的操作：
[S]——查看家谱
[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”
用户根据自身需要输入操作符。

1.4 注意事项

（1）用户需按要求输入信息，不能输入除提供的操作符以外的符号
（2）输入人数信息时，不能输入除数字以外的字符

2.思路与设计

2.1 基本思路

利用子女链表和父指针进行信息存储，并用一个类及其成员函数来实现各项功能。搜索家庭成员用深度优先搜索，递归查找。

2.2 设计

2.2.1 数据结构

建立一个家谱树，根节点为祖先。树上的每一个节点表示一个家庭成员，节点是一个称作 member 的结构体，包含成员姓名，指向父代的指针和存放指向子代的指针的数组。可通过子代寻找父代，也可以通过父代寻找子代。

2.2.2 成员与成员函数

member 内数据包括姓名，指向父代的指针和存放指向子代的指针的数组。search 函数用于递归搜索家庭成员。

```
class member{
    friend class tree;
private:
    string _name; //名字
    member* _parent; //父代
    vector<member*> _child; //子代

    member(string name, member* parent=NULL):_name(name),_parent(parent){};
    member* search(string name);
```

类 tree 是整个家谱的核心，含多个成员函数，实现各项程序主要功能。

```
class tree {
private:
    member* _ancestor;//祖先，家谱树的根节点
public:
    tree(string name) {
        _ancestor = new member(name);
        _ancestor->_name=name;
        cout<<"此家谱的祖先是："<<_ancestor->_name<<endl;
    };
    bool changeName(string oldName, string newName);//更改姓名
    void dissolve(string parentName);//解散家庭
    void addChild(string parentName, string childName);//搜索父代名字，添加后代
    bool addChild(member* parent, string childName);//利用父代指针，添加后代
    void build(string name, int nChild);//建立家庭，即添加多个后代
    member* search(string name){return _ancestor->search(name);}//搜索函数
    void showChildList(member* parent);//输出子代
    void display();//展示家谱
};
```

3.具体实现

3.1 搜索成员

核心代码

```
member* member:: search(string name){
    //是否就是此节点
    if (this->_name == name) {
        return this;
    } else {
        //是否有子节点
        if (this->_child.empty()) {
        } else {
            for (auto& p:_child) {
                member* cur=p->search(name);
                if (cur) {return cur;}
            }
        }
        return NULL;
    }
}
```

search(string name)函数代码

```
member* tree:: search(string name){return _ancestor->search(name);}//
搜索函数
```

说明

递归调用 member 结构体的 search 函数，对每个节点进行判断，并继续对相应节点的子节点进行判断，直到找到要搜索的节点或者遍历家谱为止。
tree 中的 search 函数是为了提供外部接口。

3.2 添加成员

核心代码

```
bool tree:: addChild(member* parent, string childName){
    member* child = new member(childName);
    child->_parent = parent;
    parent->_child.push_back(child);
    //检查是否添加成功
    if (parent->_child.back()->_name == childName) {
        return true;
    } else {
        return false;
    }
}
```

```
void tree:: addChild(string parentName, string childName){
    member* parent=_ancestor->search(parentName);
    if(addChild(parent, childName)){
        showChildList(parent);
    } else {
        cerr<<"操作失败！"<<endl;
    }
}
```

说明

先用 search 函数根据父代名字搜索成员，找到父代后，获得父代指针，添加子代指针到父代的子代指针数组，同时更新子代的父代指针。使用两个函数，是为了在需要添加多个子女的时候，只需要搜索一次父代，不会多次搜索。

3.3 建立家谱

核心代码

```
void tree:: build(string parentName, int nChild){
    member* parent=_ancestor->search(parentName);
    string childName;
    for (int i=0; i<nChild; i++) {
        cin>>childName;
        addChild(parent, childName);
    }
    showChildList(parent);
}
```

说明

根据用户输入的子代个数，循环多次调用 addChild 函数。此函数用于一次添加多个后代，建立家庭。

3.4 删除成员

核心代码

```
void tree:: dissolve(string parentName){
    member* parent=_ancestor->search(parentName);
    cout<<"[操作前]"<<endl;
    showChildList(parent);
    for (auto p:parent->_child){
        delete p;
        parent->_child.pop_back();
    }
    cout<<"[操作后]"<<endl;
    showChildList(parent);
}
```

说明

用 search 函数搜索到要解散家庭的父代，释放所有子代指针指向的空间，清空父代用于存放子代指针的数组。

3.5 更改信息

核心代码

```
bool tree:: changeName(string oldName, string newName){
    member* person = _ancestor->search(oldName);
    person->_name = newName;
    if (person->_name==newName) {
        return true;
    } else {
        return false;
    }
}
```

说明

用 search 函数根据旧姓名搜索到家庭成员，并修改该家庭成员的姓名。

3.6 输出家谱

核心代码

```
void tree:: showChildList(member* parent){
    if (parent->_child.empty()) {
        cout<<parent->_name<<"没有后代";
    } else {
        cout<<parent->_name<<"的第一代后代是:";
        for (auto& p:parent->_child) {
            cout<<p->_name<<" ";
        }
    }
    cout<<endl;
}
```

```
void tree:: display(){
    vector<member*> haveChild;
    if (!_ancestor->_child.empty()) {
        haveChild.push_back(_ancestor);
    } else {
        cout<<"祖先"<<_ancestor->_name<<"没有后代"<<endl;    }
    member* cur;
    while (!haveChild.empty()) {
        cur=haveChild.back();
        showChildList(cur);
        haveChild.pop_back();
        for (auto& p:cur->_child) {
            if (!p->_child.empty()) {
                haveChild.push_back(p);
            }
        }
    }
}
```

说明

根据该家庭成员的存放子代指针的数组，依次输出该家庭成员的所有子代姓名，若无子代则输出“没有后代”。display()函数中建立一个数组 haveChild，存储有子代的成员。从根节点祖先开始自上而下搜索，对每个节点，先输出他的子代，检查子代是否有子代，若有则将该子代成员放入 haveChild 数组。输出子代结束后，检测 haveChild 数组是否为空，若不为空，则继续输出某成员的子代并检查是否存在子代的子代；若为空，则退出循环。

4.测试

4.1 功能测试

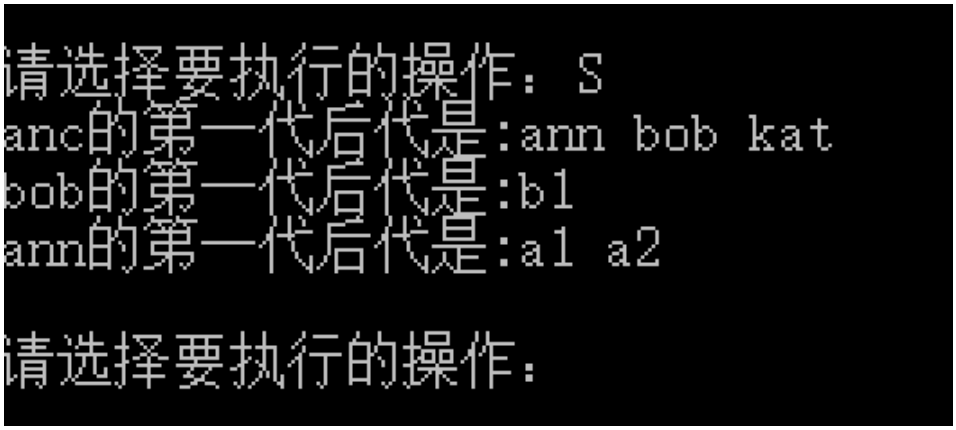
4.1.1 建立家谱

```
[家谱管理系统]
请选择要执行的操作:
[S]——查看家谱
[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 —— 请输入祖先姓名：anc
此家谱的祖先是：anc

请选择要执行的操作：A
请输入要建立家庭的人的姓名：anc
请输入anc的儿女人数：3
请依次输入anc的儿女的姓名：ann bob kat
anc的第一代后代是:ann bob kat

请选择要执行的操作：
```

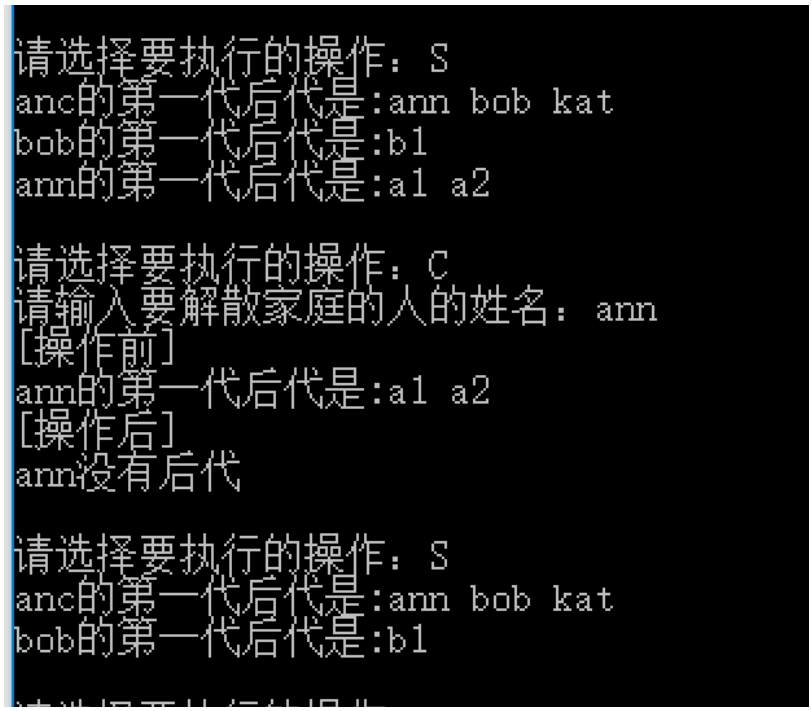
4.1.2 添加家庭成员

```
请选择要执行的操作：B
请输入要添加孩子的人的姓名：bob
请输入添加的孩子的姓名：b1
bob的第一代后代是:b1

请选择要执行的操作：
```

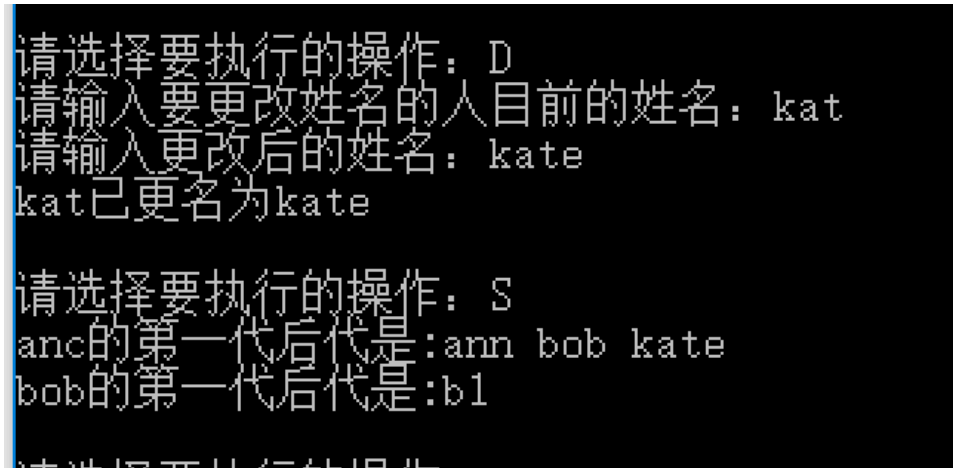
4.1.3 查看家谱



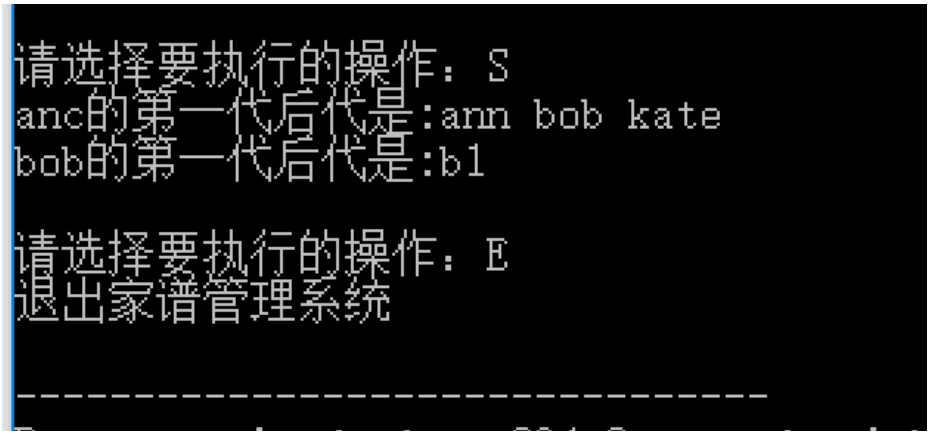
4.1.4 解散局部家庭



4.1.5 更改成员姓名

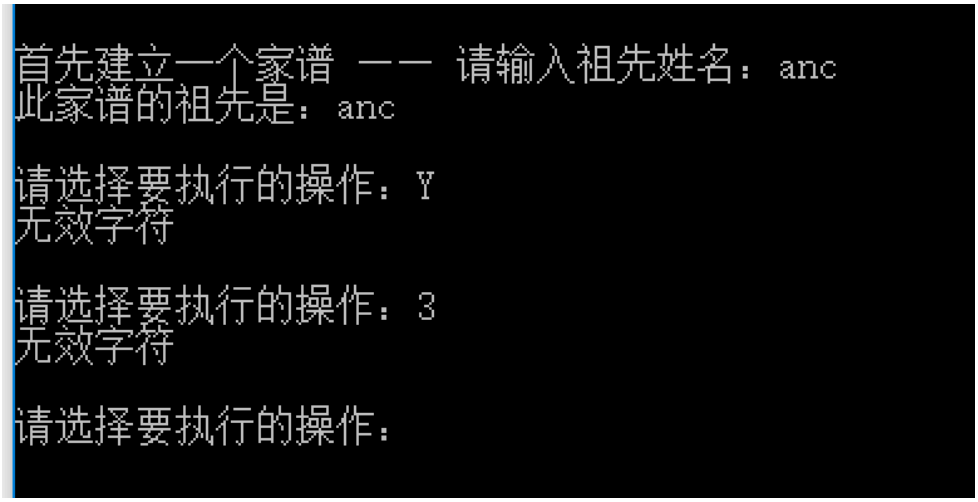


4.1.6 退出程序



4.2 出错测试

4.2.1 输入非法字符



4.2.2 找不到家庭成员