目录

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.需求分析

设计一个家谱管理系统，实现以下功能：

- 建立家谱
- 完善家谱
- 添加家庭成员
- 解散局部家庭

- 显示家庭信息
- 更改家庭成员姓名

3.设计思路

数据结构

根据家谱的特点，最适合家谱的数据结构应该选择树，而同一父亲下可能会有多个子女，因此选用多叉树作为存储家谱的数据结构，家谱的建立、增添、删除均可以通过多叉树的创建、增添节点、删除节点实现

功能实现

功能	设计思路
建立家谱	创建一个多叉树类的对象
完善家谱	节点添加多个子节点
添加家庭成员	节点添加单个子节点
解散局部家庭	删除该节点及所有后代节点
显示家庭信息	遍历子节点
更改家庭成员姓名	搜索节点+信息修改

4.核心代码说明

节点类

```
//Create node class
class member {
public:
    string name;
    //Use vector to save the node's children
    vector<member*> children;
    member(string m_name):name(m_name){}
};
```

一个节点存储一个家庭成员，包含姓名(name)，子女节点指针的vector(children)，带名称的构造函数

多叉树类

```
//Create the muti-way tree class
class familyTree {
private:
    member ancestor;

public:
```

```
    //Return the address of ancestor
    member* getAncestor() { return &ancestor; }
    //Overload constructor
    familyTree(member& m_ancestor):ancestor(m_ancestor){}
    //Search member by name
    member* searchMember(string name,member* node,bool& flag,bool child_model);
    //Add child node
    void addChild(string name, int children_number,vector<string>& children_name);
    //Output node information
    void showNode(string name);
    //Delete subtree
    void deleteMember(string name);
    //Delete node
    void deleteNode(member* node);
    //Change name of node
    void changeName(string pre_name,string new_name);
};
```

包含一个私有成员变量`ancestor`为该多叉树的根节点，该多叉树的成员函数可以实现多叉树的建立、增添节点、查找节点、显示节点信息、删除节点、删除子树、修改节点信息等功能。具体成员函数的功能见代码注释

函数功能说明

搜索节点

该功能作为程序的基础功能，通过输入成员名作为参数，如果在多叉树中找到该成员名的节点，返回节点地址，反之提示“未找到该成员”。搜索功能是添加子节点、删除节点、删除子树、显示节点信息等功能的基础

```
//Search node of parent node by name and return its address
member* familyTree::searchMember(string name, member* node,bool& flag,bool child_model) {
  member* find = NULL;
  if (node) {
    //Search node by name
    if (node->name == name && !child_model) {
      find = node;
      flag = true;
    }
    //Search parent node
    else if (child_model) {
      for (auto i : node->children) {
        if (i->name == name) {
          find = node;
          flag = true;
        }
      }
    }
    if (!flag) {
      for (auto iter = node->children.begin(); iter != node->children.end(); iter++) {
        find = searchMember(name, *iter, flag, child_model);
        if (flag)
          break;
      }
    }
```

```
  }
  return find;
}
```

通过递归方法实现多叉树的搜索，有两种搜索模式：

`child_model`为false：搜索节点名与关键词相同的节点

`child_model`为true：搜索子节点包含关键词的节点

搜索逻辑为：比较当前节点姓名与关键字异同，相同则返回当前节点地址，停止递归调用，不同则通过迭代器将子节点作为参数递归调用`searchMember`函数。（如果`child_model`为true则比较子节点姓名与关键字）

建立家谱

通过重载过的带祖先姓名的构造函数实现，在main函数中先要求用户输入祖先姓名，然后作为参数新建多叉树对象

main部分

```
cout << "首先建立一个家庭！" << endl;
cout << "请输入祖先的姓名:";
cin >> ancestor_name;

cout <<"此家族的祖先是："<<ancestor_name<<endl;

member ancestor(ancestor_name);
familyTree family(ancestor);
```

完善家谱

通过`addChild`函数实现，在main中获取到需要添子女节点的成员名，以及子女节点的姓名（用vector存储），以此作为参数调用

main部分

```
cout << "请输入要建立家庭人的姓名：";
cin >> name_select;
cout<< "请输入" << name_select << "的儿女人数:";
cin >> number;
children_name.clear();
cout << "请依次输入" << name_select << "的儿女的姓名:";
for (int i = 0; i < number; i++) {
  cin >> temp_name;
  children_name.push_back(temp_name);
}
cout << name_select << "的第一代子孙是:";
for (auto i : children_name)
  cout << i << " ";
family.addChild(name_select, number, children_name);
```

addChild函数

```cpp
//Add children to a node
void familyTree::addChild(string name,int children_number,vector<string>& children_name) {
    bool flag = false;
    member* parent = searchMember(name, getAncestor(), flag,false);
    if (parent == NULL) {
        cout << "没有找到该成员!" << endl;
    }
    else {
        for (int i = 0; i < children_number;i++) {
            member* child = new member(children_name[i]);
            parent->children.push_back(child);
        }
    }
}
```

首先通过`searchMember`函数搜索到被添加节点，然后利用传入的子节点姓名的vector，new新的member对象，通过push_back添加进该节点的存储子节点地址的vector中

添加家庭成员

原理与完善家谱的相同，不过vector中只存放一个子节点的名称

main部分

```cpp
cout << "请输入要添加儿子（或女儿）的人的姓名：";
cin >> name_select;
cout << "请输入" << name_select << "新添加的儿子（或女儿）的姓名：";
cin >> temp_name;
children_name.push_back(temp_name);
family.addChild(name_select, 1, children_name);
cout << name_select << "的第一代子孙是：" << temp_name;
```

解散局部家庭

即删除子树，首先调用`searchMember`通过名称找到对应节点的父节点，然后利用迭代器在父节点vector中找到并删除该节点的指针，随后将该节点地址传入`deleteNode`函数中，`deleteNode`函数将其子节点作为参数进行递归调用，自底向上删除节点，最后删除整个子树

```cpp
//Delete a member
void familyTree::deleteMember(string name) {
  bool flag = false;
  member* temp = searchMember(name, getAncestor(), flag, true);
  if (temp == NULL) {
    cout << "家族中没有找到该节点信息！" << endl;
    return;
  }
  for (vector<member*>::iterator ite = temp->children.begin(); ite != temp->children.end();)
{
    if ((*ite)->name == name) {
      showNode(name);
      deleteNode(*ite);
      ite = temp->children.erase(ite);
```

```
      break;
    }
    else
      ite++;
  }
}
```

```
//Delete a node
void familyTree::deleteNode(member* node) {
  if (node->children.size() != 0) {
    for (int i = node->children.size() - 1; i >= 0;i--) {
      deleteNode(node->children[i]);
      node->children.pop_back();
    }
  }
  if(node->children.size() == 0)
    delete node;
}
```

显示家庭信息

首先调用`searchMember`通过名称找到对应节点，然后输出该节点的vector信息（size,子节点名称）

```
//Display the node information
void familyTree::showNode(string name) {
  bool flag = false;
  member* temp = searchMember(name,&ancestor,flag,false);
  if (temp == NULL) {
    cout << "没有找到该节点!" << endl;
    return;
  }
  else {
    cout << name << "节点有" << temp->children.size() << "个子女，分别为：" << endl;
    for (auto i : temp->children) {
      cout << i->name << " ";
    }
  }
}
```

更改家庭成员姓名

首先调用`searchMember`通过名称找到对应节点，然后将其name更改为传入的新值

```cpp
//Change the node info
void familyTree::changeName(string pre_name,string new_name) {
  bool flag = false;
  member* temp = searchMember(pre_name,getAncestor(),flag,false);
  if (temp == NULL) {
    cout << "家族中没有找到该节点信息！" << endl;
    return;
  }
  temp->name = new_name;
  cout << pre_name << "已更名为" << new_name;
}
```

函数接口说明

返回值类型	成员函数名	参数	属性	功能
\	familyTree	(member& m_ancestor)	public	带参数的构造函数
memeber*	getAncestor	()	public	返回祖先的地址
member*	searchMember	(string name,member* node,bool& flag,bool child_model)	public	搜索节点并返回地址
void	addChild	(string name, int children_number,vector& children_name)	public	添加子节点
void	showNode	(string name)	public	输出节点信息
void	deleteNode	(member* node)	public	删除子树
void	deleteMember	(string name)	public	删除子树
void	changeName	(string pre_name,string new_name)	public	修改节点名称

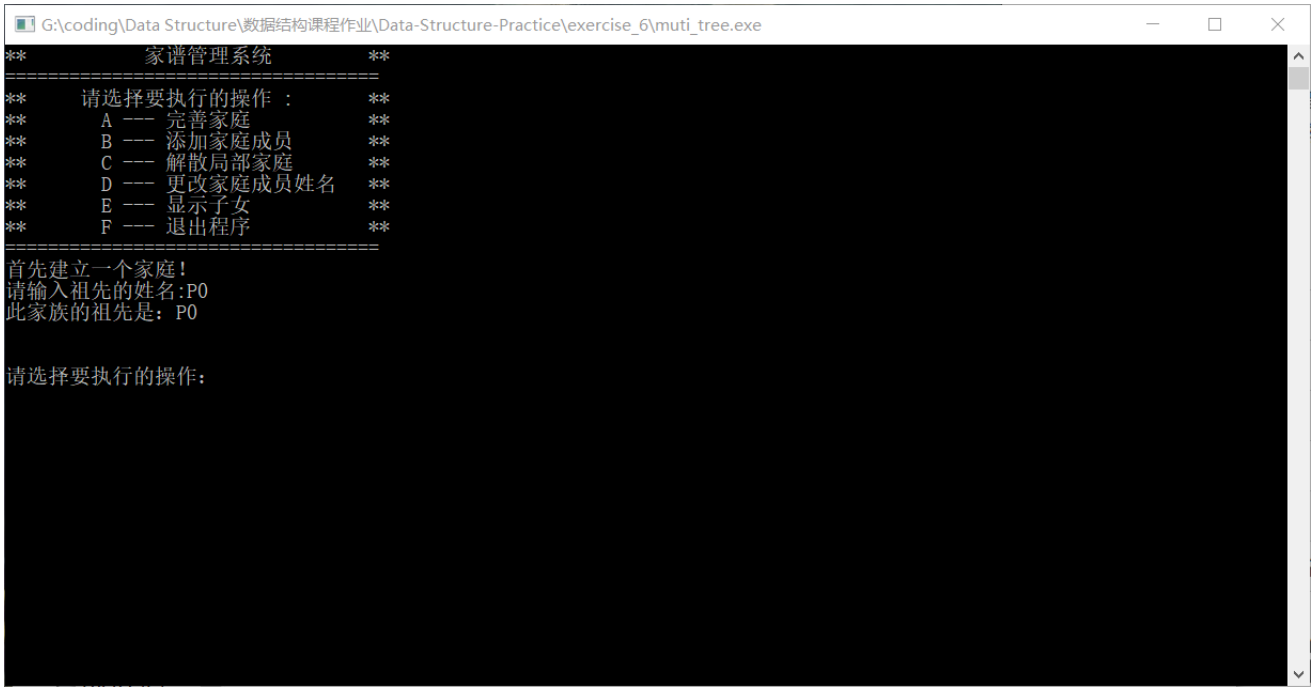
5.使用说明及函数功能演示

运行程序

双击`muti_tree.exe`打开程序，系统会出现功能选择菜单，并提示用户建立家谱

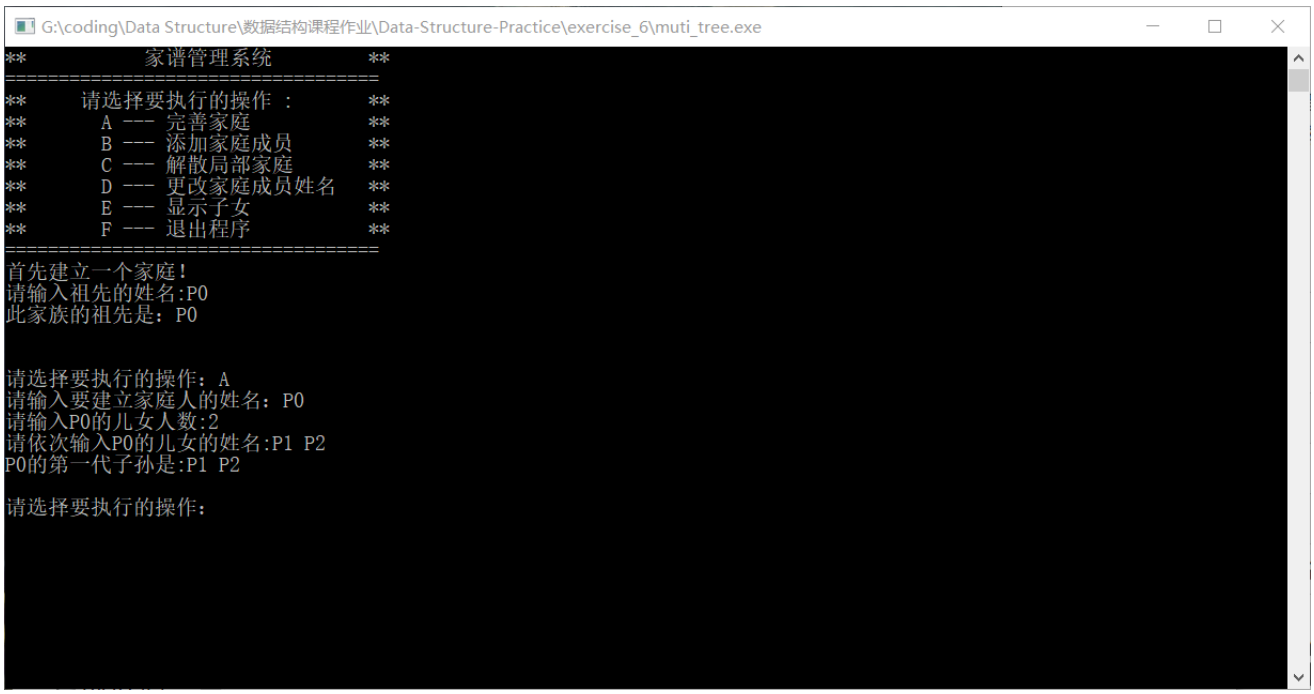
建立家谱

根据系统提示输入家族祖先名称，建立家谱



完善家庭

输入A选择该功能，随后会提示输入要建立家庭的人姓名、子女数、子女名称

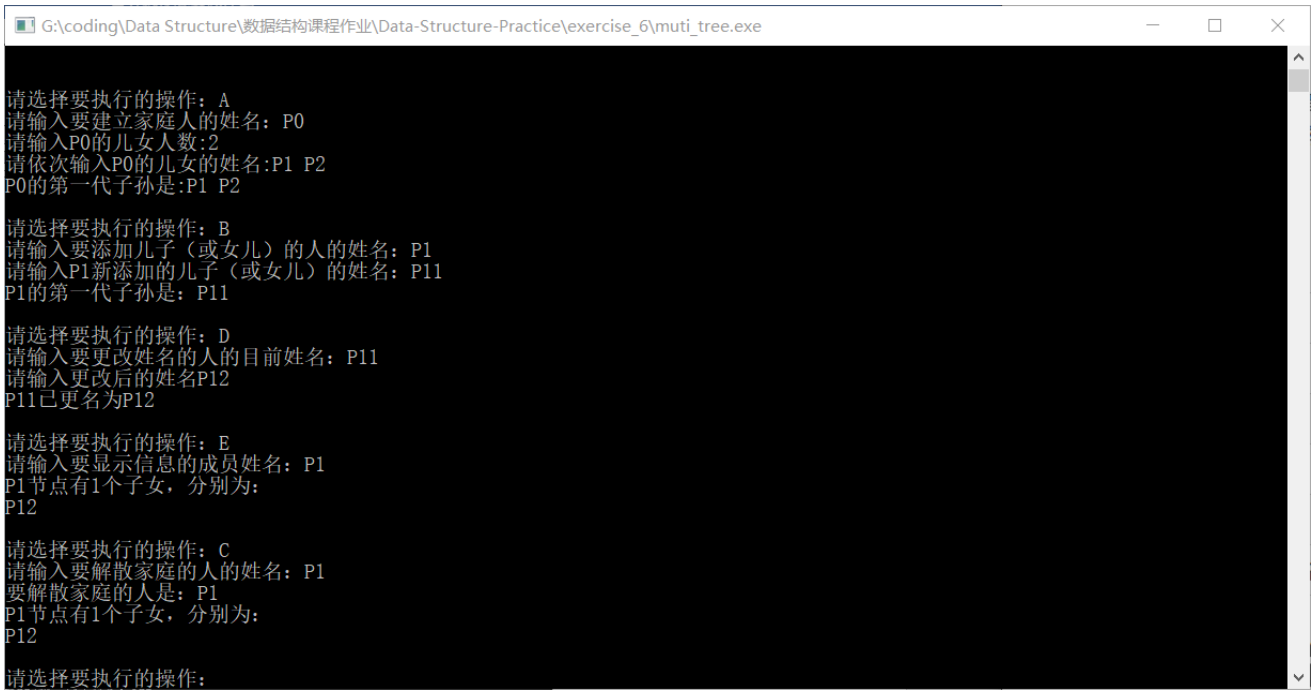


添加家庭成员

输入B选择该功能，随后按提示依次输入要添加家庭成员的人的姓名、子女名称，需注意一次只能添加一位子女

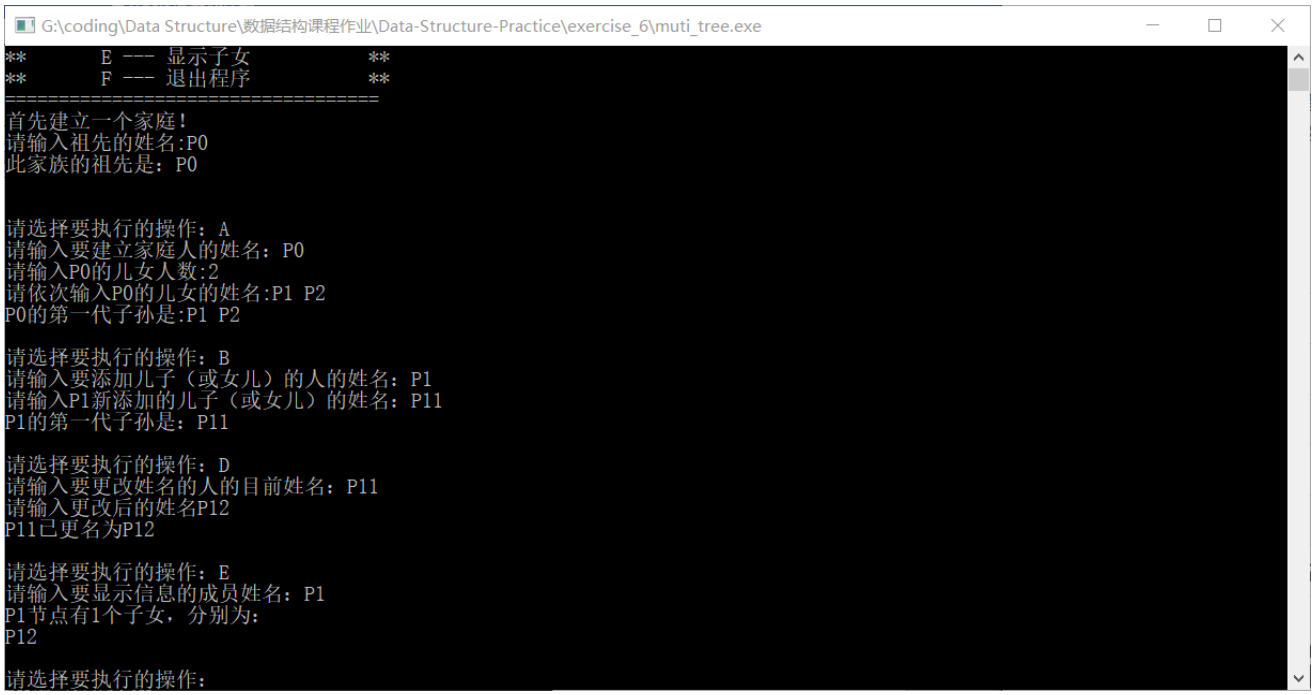
解散局部家庭

输入C选择该功能，随后按提示输入需要解散家庭的人的姓名，系统将会输出该成员的子女人数和子女姓名，随后通过删除该名成员及其所有后代



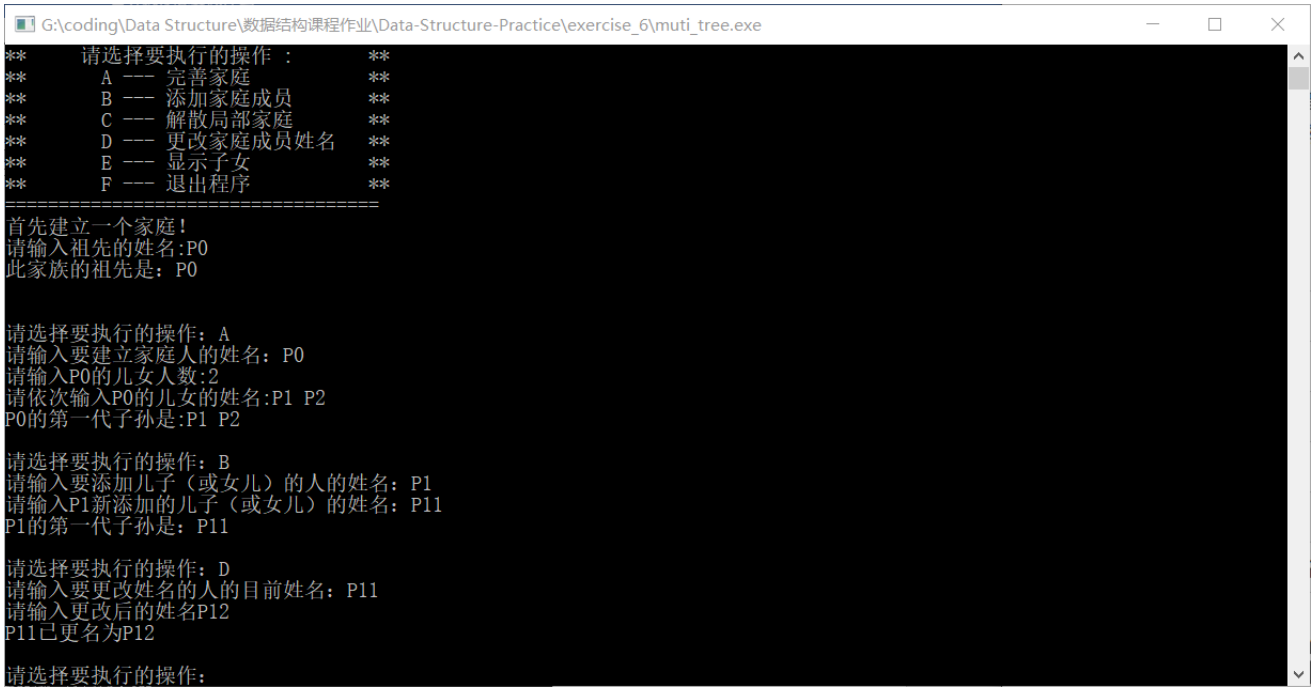
显示家庭信息

输入E选择该功能，按提示输入要显示信息的家庭成员名称，系统将会显示子女数及子女名称



更改家庭成员姓名

输入D选择该功能，按提示输入要修改姓名的人的姓名，再依照提示输入新姓名，即修改完成



6.环境说明

运行环境为windows10 1709 家庭中文版，使用IDE为Visual Studio2017,经测试发现在其他环境下可能出现多叉树存储子女节点地址的vector自动清空现象