家谱管理系统

1651573 刘客

功能介绍

- 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，删除家族成员的功能。

  具体功能为**插入**，**搜索**，**删除**，**修改**。

编译说明

- 在windows平台下的.exe文件
- 在Linux平台下的out文件

一 . 设计

1. 数据结构设计

   > 由于家谱的结构是树形结构,因此应该设计树来实现该项目。
   > 另外由于该家谱管理系统要求实现插入、删除等操作且可能存在分支较多的情况
   > 故使用**链表**来实现树这一结构。

2. 类的设计

- ## FamilyNode 节点

  - 成员变量

成员名称	属性	类型	描述
name	private	string	当前节点成员的姓名
childrenNum	private	int	当前节点成员的孩子数量
childrenNode	private	vector<FamilyNode*>	存储孩子节点的数组

- 成员函数

函数名称	返回值类型	描述
FamilyNode	无	构造函数
getName	string	获得当前节点成员的姓名
getNum	int	获得当前节点成员的孩子数量
setNum	void	设置当前节点成员的孩子数量
setName	void	设置当前学生的姓名

|getNodeVector|vector<FamilyNode>|获得指向存储孩子节点数组的指针 |getNode|FamilyNode*|获得指向当前节点第i个孩子的指针

- **FamilyTree**
  - 成员变量

成员名称	属性	类型	描述
root	private	FamilyNode*	祖宗节点

  - 成员函数

函数名称	返回值类型	描述
FamilyTree	无	构造函数
createFamily	void	创建一个家庭
addFamilyMember	void	添加家庭成员
dissovePartFamily	void	解散局部家庭
changeNameForOneFameliyMember	void	为某一家庭成员更改名字

|findNode|FamilyNode*|搜索特定节点 |coutFirstChildren|void|输出第一代子孙

二. 类的具体实现

1. FamilyTree

```
FamilyTree::FamilyTree(FamilyNode * node) {
    root = node;
}
```

2. 搜索特定节点函数

> 两个参数分别为root节点和搜寻的成员的名字
> 当找到该成员节点时,将节点返回,若未找到,则返回NULL

```
FamilyNode* FamilyTree::findNode(FamilyNode* node,string name) {
    if (node != NULL) {
        if (node->getName() == name) {
            return node;
        }
        else {
            FamilyNode* tempNode;
            for (auto i = 0; i < node->getNum(); i++) {
                tempNode = findNode(node->getNode(i), name);
                if (tempNode != NULL) {
                    return tempNode;
                }
            }
            return NULL;
        }
    }
    return NULL;
}
```

3. 创建家庭函数

> 首先通过私有查找函数找到节点,若返回值为NULL,则输出错误信息
> 如果该节点已经有了家庭,也输出错误信息。若满足创建家庭条件,则进行处理。

```
void FamilyTree::createFamily(string name) {
    FamilyNode* tempNode = findNode(root, name);
    if (tempNode == NULL) {
        cout << "查无此人,请确认是否输入姓名正确!!!\n";
        return;
    }
    else if (tempNode->getNum() != 0) {
        cout << "此人已拥有家庭,请确认是否输入姓名正确!!!\n";
        return;
    }
    int num;
    cout << "请输入" << name << "的儿女人数: ";
    cin >> num;
    tempNode->setNum(num);
    cout << "请依次输入" << name << "的儿女的姓名: ";
    for (auto i = 0; i < num; i++) {
        string tempName;
        cin >> tempName;
        FamilyNode* node = new FamilyNode(tempName);
        tempNode->setNode(node);
    }
    coutFirstChildren(tempNode, name);
}
```

4. 添加家庭成员

> 仍使用查找函数查找节点,并针对返回值进行处理

```
```

```
void FamilyTree::addFamilyMember(string name) {
    FamilyNode* node = findNode(root,name);
    if (node == NULL) {
        cout << "查无此人,请确认是否输入姓名正确!!!\n";
        return;
    }
    int num = node->getNum();
    num++;
    node->setNum(num);
    cout << "请输入"<<name<<"新添加儿子(或女儿)的姓名: ";
    string tempName;
    cin >> tempName;
    FamilyNode* tempNode = new FamilyNode(tempName);
    node->setNode(tempNode);
    coutFirstChildren(node, name);
}
```

5. 解散局部家庭

> 仍使用查找函数,寻找相应节点,并对返回值进行处理。
> 当解散时,将当前节点所存储子节点指针的数组清空。

```
 void FamilyTree::dissovePartFamily(string name) {
     FamilyNode* node = findNode(root,name);
     if (node == NULL) {
         cout << "查无此人,请确认是否输入姓名正确!!!\n";
         return;
     }
     cout << "要解散家庭的人是:" << node->getName() << endl;
     cout << node->getName() << "的第一代子孙是: ";
     for (auto i = 0; i < node->getNum(); i++) {
         cout << node->getNode(i)->getName() << "\t";
         delete node->getNode(i);
     }
     node->setNum(0);
     //清空存储子节点的vector
     node->getNodeVector()->clear();
     cout << "\n\n";
 }
```

6. 更改家庭成员姓名函数

> 仍使用查找函数,寻找相应节点,并对返回值进行处理。

```
void FamilyTree::changeNameForOneFameliyMember(string name) {
    FamilyNode* node = findNode(root, name);
    if (node == NULL) {
        cout << "查无此人,请确认是否输入姓名正确!!!\n";
        return;
    }
    string newName;
    cout << "请输入更改后的姓名：";
    cin >> newName;
    cout << name << "已更名为";
    node->setName(newName);
    cout << node->getName()<<endl<<endl;
}
```

三. 主程序实现

- 函数

函数名称	返回值类型	描述
welcomeImage	void	绘制操作台函数
processMethod	bool	处理输入指令函数
main	int	主函数

- 绘制操作台函数

```
void welcomeImage() {
    cout << "**\t\t" << "家谱管理系统" << "\t\t\t**\n";
    cout << "==================================================\n";
    cout << "**\t       请选择要执行的操作 :         \t**\n";
    cout << "**\t\tA --- 完善家谱 \t\t\t**\n";
    cout << "**\t\tB --- 添加家庭成员\t\t**\n";
    cout << "**\t\tC --- 解散局部家庭\t\t**\n";
    cout << "**\t\tD --- 更改家庭成员姓名\t\t**\n";
    cout << "**\t\tE --- 退出程序\t\t\t**\n";
    cout << "==================================================\n";
    cout << "首先建立一个家谱!\n";
}
```

- 处理输入指令函数

> 当指令等于E时,返回true终结循环

```
bool processMethod(char method,FamilyTree * tree) {
    string name;
    switch (method)
    {
    case 'A':
        cout << "请输入要建立家庭的人的姓名：";
```

```
            cin >> name;
            tree->createFamily(name);
            return false;
            break;
        case 'B':
            cout << "请输入要添加儿子(或女儿)的人的姓名: ";
            cin >> name;
            tree->addFamilyMember(name);
            return false;
            break;
        case 'C':
            cout << "请输入要解散家庭的人的姓名: ";
            cin >> name;
            tree->dissovePartFamily(name);
            return false;
            break;
        case 'D':
            cout << "请输入要更改姓名的人的目前姓名: ";
            cin >> name;
            tree->changeNameForOneFameliyMember(name);
            return false;
            break;
        case 'E':
            return true;
        default:
            cout << "你输入的命令有误,请确认后再输入!!!!!\n";
            return false;
            break;
        }
    }
```

- 循环实现

```
FamilyTree* tree = new FamilyTree(indi);
while (1) {
    char Method;
    cout << "请选择要执行的操作: ";
    cin >> Method;
    if (processMethod(Method,tree)) {
        break;
    }
}
```

四. 程序运行效果

1. 进入
2. 创建祖先
3. 执行A操作
4. 执行B操作

5. 执行C操作
6. 执行D操作
7. 执行E操作

> 直接退出

五. 边界测试

1. 输入指令错误
2. 搜索节点不存在
3. 孩子个数不为正数