项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 杨煜

学 号： 1850217

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc26872470)

[1.1 项目简介 1](#_Toc26872471)

[1.2 项目要求 1](#_Toc26872472)

[2 设计 1](#_Toc26872473)

[2.1 数据结构设计 1](#_Toc26872474)

[2.2 树类结构设计 1](#_Toc26872475)

[2.3 成员与操作设计 1](#_Toc26872476)

[3 实现 2](#_Toc26872477)

[3.1 构造函数的实现 2](#_Toc26872478)

[3.1.1 构造函数 2](#_Toc26872479)

[3.1.2 构造函数核心代码 2](#_Toc26872480)

[3.2 查找节点的实现 3](#_Toc26872481)

[3.2.1 查找函数 3](#_Toc26872482)

[3.2.2 查找函数核心代码 3](#_Toc26872483)

[3.3 添加子节点的实现 4](#_Toc26872484)

[3.3.1 添加子节点 4](#_Toc26872485)

[3.3.2 添加子节点核心代码 4](#_Toc26872486)

[3.4 输出子节点的实现 5](#_Toc26872487)

[3.4.1 输出子节点 5](#_Toc26872488)

[3.4.2 输出子节点核心代码 5](#_Toc26872489)

[3.5 插入子树的实现 5](#_Toc26872490)

[3.5.1 插入子树 5](#_Toc26872491)

[3.5.2 插入子树核心代码 5](#_Toc26872492)

[3.6插入节点的实现 6](#_Toc26872493)

[3.6.1 插入节点 6](#_Toc26872494)

[3.6.2 输出功能核心代码 6](#_Toc26872495)

[3.7 删除功能的实现 7](#_Toc26872496)

[3.7.1 删除功能 7](#_Toc26872497)

[3.7.2 删除功能核心代码 7](#_Toc26872498)

[3.8 修改功能的实现 8](#_Toc26872499)

[3.8.1 修改功能 8](#_Toc26872500)

[3.8.2 修改功能核心代码 8](#_Toc26872501)

[3.9 总体功能的实现 9](#_Toc26872502)

[3.9.1 总体功能 9](#_Toc26872503)

[3.9.2 总体功能核心代码 9](#_Toc26872504)

[3.9.3 总体功能流程图 11](#_Toc26872505)

[4 测试 11](#_Toc26872506)

[4.1 功能测试 11](#_Toc26872507)

[4.1.1 构建家谱 11](#_Toc26872508)

[4.1.2 添加家庭成员 12](#_Toc26872509)

[4.1.3 解散局部家庭 13](#_Toc26872510)

[4.1.4 更改姓名 14](#_Toc26872511)

[4.2 出错测试 15](#_Toc26872512)

[4.2.1 操作码错误 15](#_Toc26872513)

[4.2.2 找不到此人 15](#_Toc26872514)

# 1 分析

## 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。

## 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。

# 2 设计

## 2.1 数据结构设计

如上述系统功能要求所述，本程序使用树的数据结构来实现该系统。该树的存储结构为链表，拥有两个指针节点，左指针指向子节点，右指针指向兄弟节点。同时具有增加删除，修改等功能。

## 2.2 树类结构设计

本例的树结构使用链表实现，经典的链表一般包括两个抽象数据类型（ADT）——树结点类（TreeNode）与树链表类（Tree），而两个类之间的耦合关以采用嵌套、继承等多种关系。为方便处理，本程序采用结构的方式使得链表结点类以及链表类可以互相访问

## 2.3 成员与操作设计

**树结点类（TreeNode）**

struct TreeNode

{

string data;//数据内容

TreeNode\* firstChild, \*nextSibling;//子孙及兄弟

TreeNode() { firstChild = NULL; nextSibling = NULL; }

};

**树类（Tree）**

class Tree

{

private:

TreeNode\* root, \*current;//根节点

bool Find(string s, TreeNode\* p);//搜索

void addChild(string str, TreeNode\* p);//添加子节点

void printChild(TreeNode\* p);//打印孩子节点

public:

Tree();//构造函数

~Tree() {}//析构函数

void Insertsubtree();//添加子树

void Insertnode();//添加子节点

void Remove();//移除

void Change();//修改

}

# 3 实现

## 3.1 构造函数的实现

### 3.1.1 构造函数

构造函数构造了一个新的节点，作为祖先节点，内容存放祖先的姓名

### 3.1.2 构造函数核心代码

Tree::Tree()

{

root = new TreeNode;

cout << "首先建立一个家谱" << endl;

cout << "请输入祖先的姓名：";

cin >> root->data;

cout << "此家谱的祖先是：" << root->data;

}

## 3.2 查找节点的实现

### 3.2.1 查找函数

查找函数接受的参数为名字和开始的节点，然后判断节点数据是否为名字，如果是则直接返回。如果不是则节点移动到子节点递归搜索，如果任然不是则移动到兄弟节点。

### 3.2.2 查找函数核心代码

bool Tree::Find(string s, TreeNode\* p)

{

if (p->data == s)

{

current = p;

return true;

}

else

{

TreeNode\* q = p->firstChild;//移到子节点

while (!(q == NULL))

{

if (Find(s, q))//递归

{

return true;

}

else

{

q = q->nextSibling;//移到兄弟节点

}

}

}

return false;

}

## 3.3 添加子节点的实现

### 3.3.1 添加子节点

函数的参数为名字和所要添加的双亲节点。如果双亲节点没有子节点则添加入，否则，寻找子节点的兄弟节点直到为空，再添加节点。

### 3.3.2 添加子节点核心代码

void Tree::addChild(string str, TreeNode\* p)

{

TreeNode\* child = new TreeNode;

child->data = str;

if (p->firstChild == NULL)//子节点为空

{

p->firstChild = child;

}

else

{

TreeNode\* lastChild = p->firstChild;

while (lastChild->nextSibling!= NULL) //寻找兄弟节点为空的子节点

{

lastChild = lastChild->nextSibling;

}

lastChild->nextSibling = child;

}

}

## 3.4 输出子节点的实现

### 3.4.1 输出子节点

通过递归寻找子节点的兄弟节点，将该节点的子节点内容全部输出。

### 3.4.2 输出子节点核心代码

void Tree::printChild(TreeNode\* p)

{

TreeNode\* child = p->firstChild;

while (child->nextSibling != NULL)

{

cout << child->data << '\t';

child = child->nextSibling;

}

cout << child->data;

}

## 3.5 插入子树的实现

### 3.5.1 插入子树

输入要建立家庭的人命，如果找到则返回正确，如果未找到则，输出错误信息。随后输入子女个数通过调用addChild，依次插入。结束后，调用printChild，依次输出其中的内容。

### 3.5.2 插入子树核心代码

void Tree::Insertsubtree()

{

cout << "输入要建立家庭的人的姓名：";

string name;

cin >> name;

if (!(Find(name,root)))

{

cout << "没有找到这个人" << endl;

}

else

{

int num;

cout << "请输入" << name << "的儿女人数：";

cin >> num;

cout << "请依次输入" << name << "的儿女姓名：";

TreeNode\* node = current;

while (num > 0)

{

string child;

cin >> child;

addChild(child, node);

num--;

}

cout << node->data << "的第一代子孙是：";

printChild(node);

}

}

## 3.6插入节点的实现

### 3.6.1 插入节点

找到要插入节点的双亲节点随后插入该节点。

### 3.6.2 输出功能核心代码

void Tree::Insertnode()

{

cout << "输入要添加儿子（或女儿）的人的姓名：";

string name;

cin >> name;

if (!(Find(name, root)))

{

cout << "没有找到这个人" << endl;

}

else

{

TreeNode\* node = current;

cout << "请输入" << name << "新添加的儿子（或女儿）的姓名：";

string child;

cin >> child;

addChild(child, node);

cout << node->data << "的第一代子孙是：";

printChild(node);

}

}

## 3.7 删除功能的实现

### 3.7.1 删除功能

首先找到要删除的子树的根节点，即祖先节点，随后依次递归删除节点。

### 3.7.2 删除功能核心代码

void Tree::Remove()

{

cout << "输入要解散家庭的人的目前姓名：";

string name;

cin >> name;

if (!(Find(name, root)))

{

cout << "没有找到这个人" << endl;

}

else

{

TreeNode\* node = current;

cout << "要解散家庭的人的姓名是" << name << endl;

if (node->firstChild == NULL)

{

cout << node->data << "已经没有家庭了";

}

else

{

cout << node->data << "的第一代子孙是：";

printChild(node);

node->firstChild = NULL;

}

}

}

## 3.8 修改功能的实现

### 3.8.1 修改功能

首先通过调用find函数找到，所要修改的节点，随后依次修改节点所在内容。

### 3.8.2 修改功能核心代码

void Tree::Change()

{

cout << "输入要更改姓名的人的目前姓名：";

string name;

cin >> name;

if (!(Find(name, root)))

{

cout << "没有找到这个人" << endl;

}

else

{

TreeNode\* node = current;

cout << "请输入更改后的姓名：";

string new\_name;

cin >> new\_name;

node->data = new\_name;

cout << name << "已更名为" << new\_name;

}

}

## 3.9 总体功能的实现

### 3.9.1 总体功能

首先建立输出的表格，随后依据输入内容，使用switch，分别进行相应的函数调用。

### 3.9.2 总体功能核心代码

int main()

{

cout << "\*\*" << setw(14) << ' ' << "家谱管理系统" << setw(16) << ' ' << "\*\*" << endl;

cout << "==============================================" << endl;

cout << "\*\*" << setw(13) << ' ' << "请选择要执行的操作" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "A --- 完善家谱" << setw(15) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "B --- 添加家庭成员" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "C --- 解散局部家庭" << setw(11) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "D --- 更改家庭成员姓名" << setw(7) << ' ' << "\*\*" << endl;

cout << "\*\*" << setw(13) << ' ' << "E --- 退出程序" << setw(15) << ' ' << "\*\*" << endl;

cout << "==============================================" << endl;

Tree A;

int loop = 1;

while (loop) {

cout << endl << endl << "请选择要执行的操作：";

char id;

cin >> id;

switch (id)

{

case 'A':

A.Insertsubtree();

break;

case 'B':

A.Insertnode();

break;

case 'C':

A.Remove();

break;

case 'D':

A.Change();

break;

case 'E':

loop = 0;

break;

default:

cout << "请输入正确的序号" << endl;

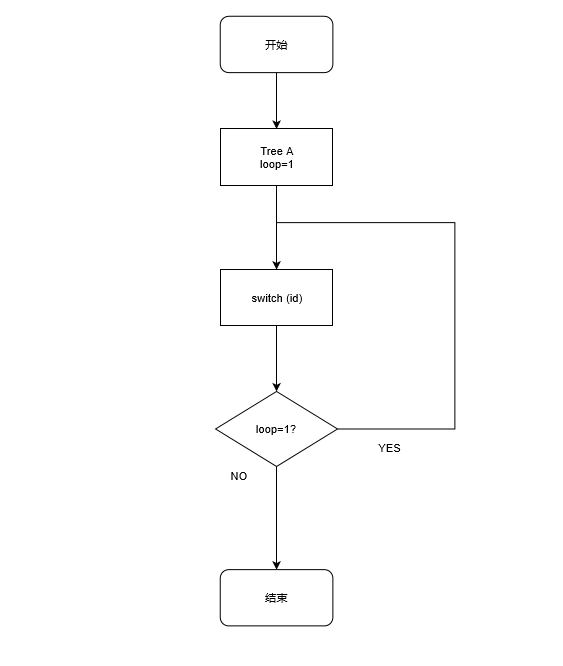
break;

}

}

}

### 3.9.3 总体功能流程图



# 4 测试

## 4.1 功能测试

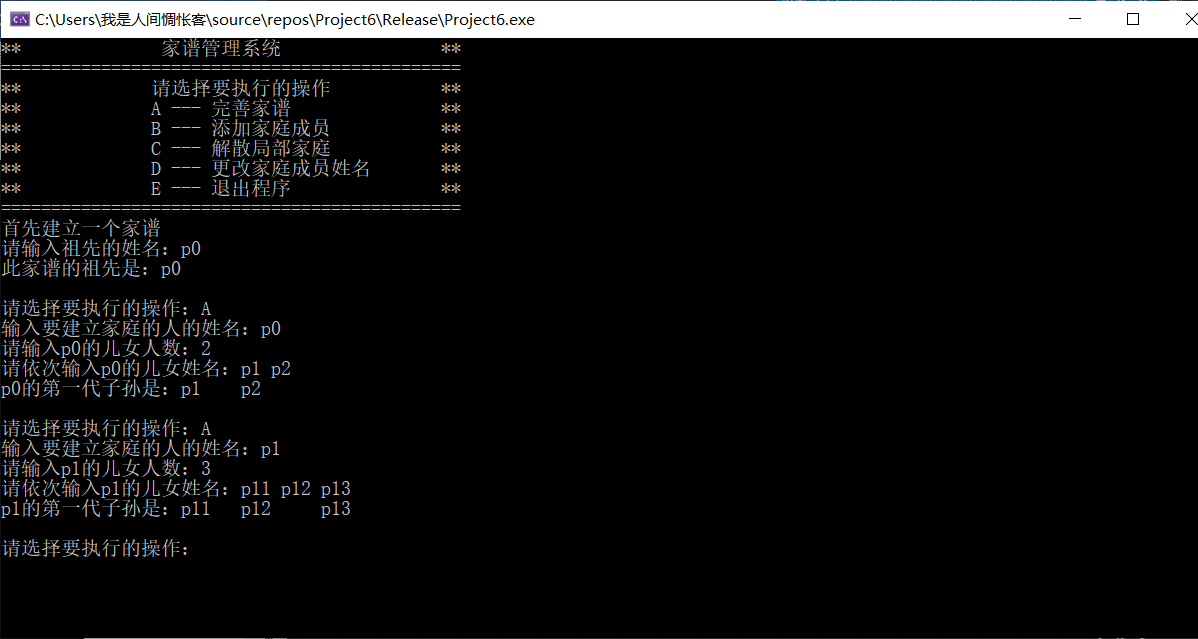
### 4.1.1 构建家谱

**测试用例**：

A p0 2 p1 p2

A p0 3 p11 p12 p13

**实验结果**

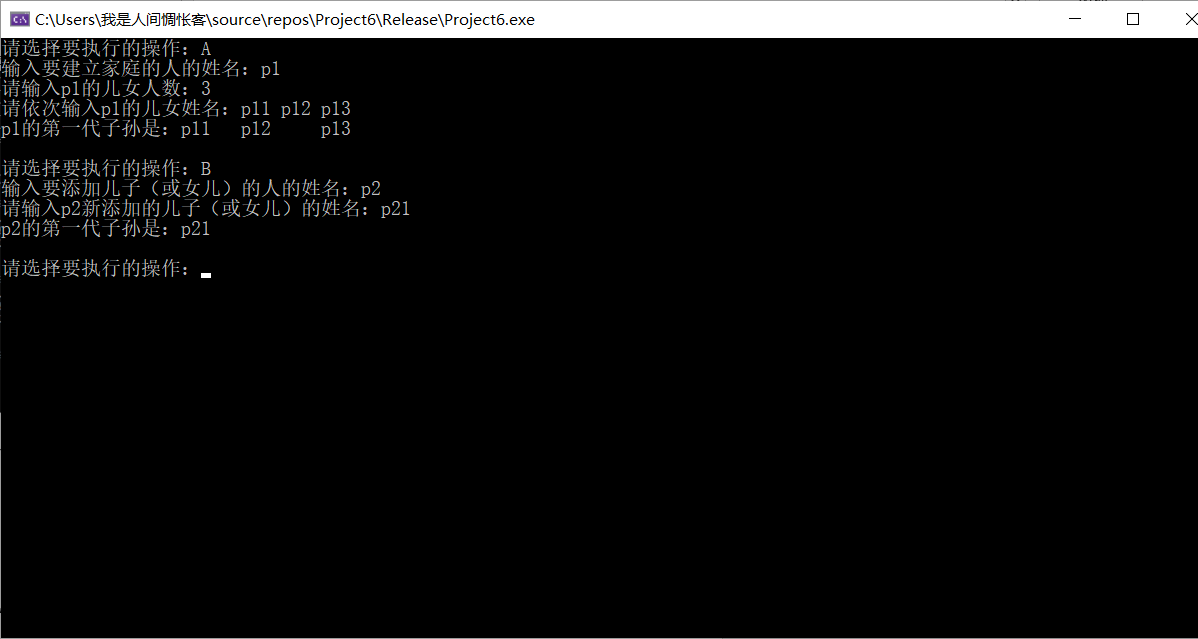


### 4.1.2 添加家庭成员

**测试用例：**

**B p2 p21**

**实验结果：**

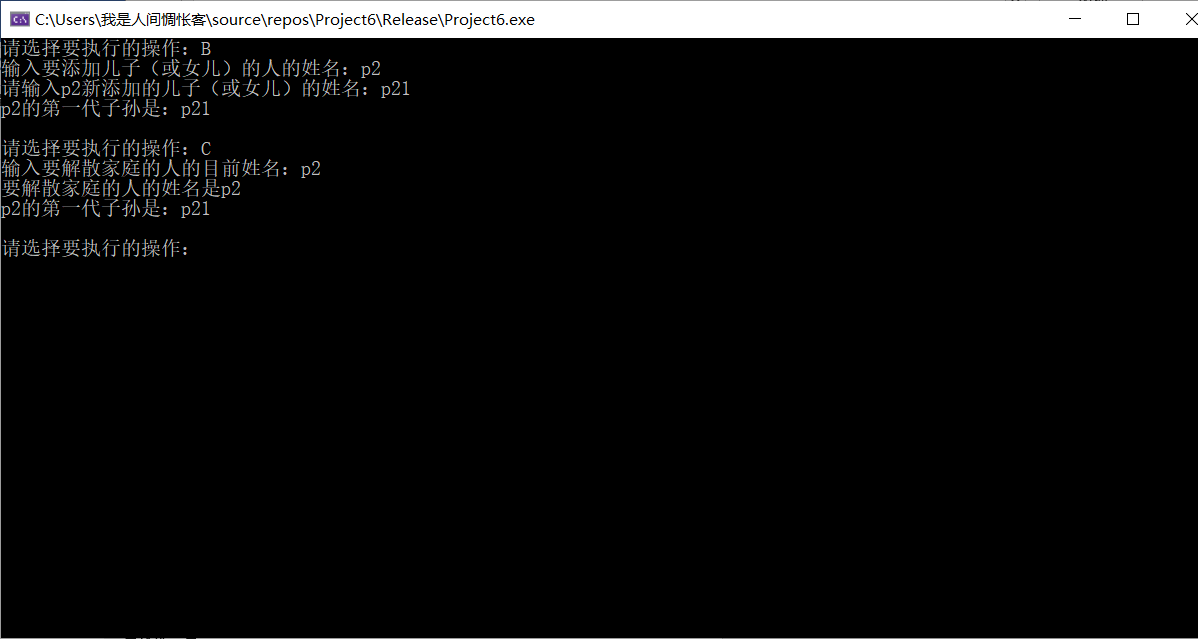


### 4.1.3 解散局部家庭

**测试用例：**

**C p2**

**实验结果：**

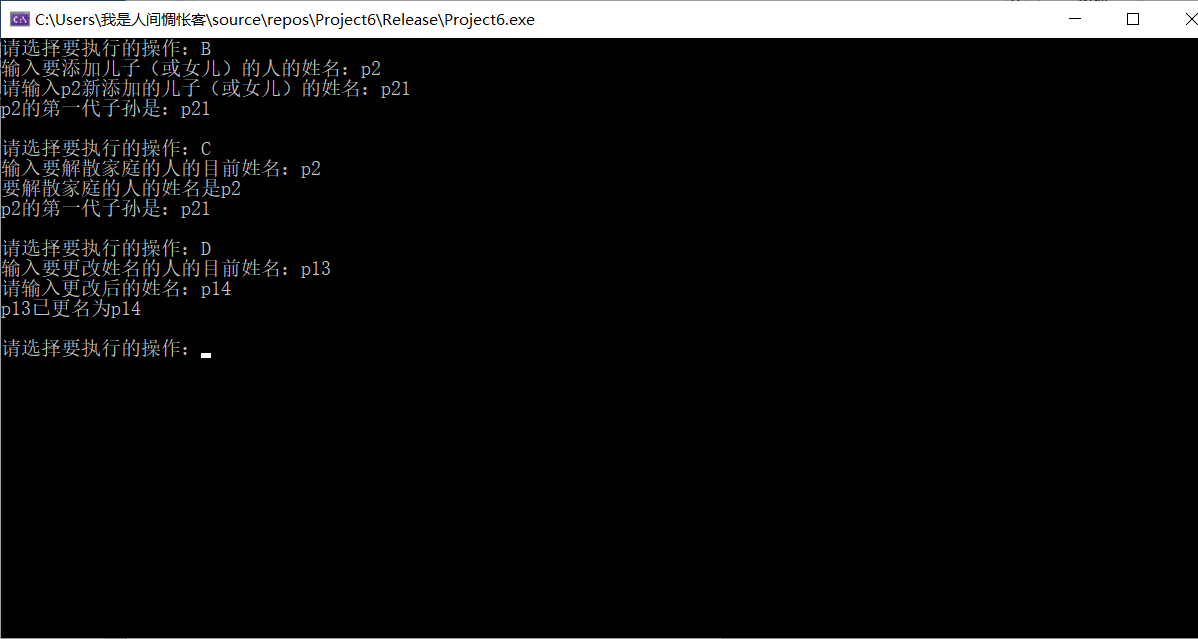


### 4.1.4 更改姓名

**测试用例：**

**D p13 p14**

**实验结果：**



## 4.2 出错测试

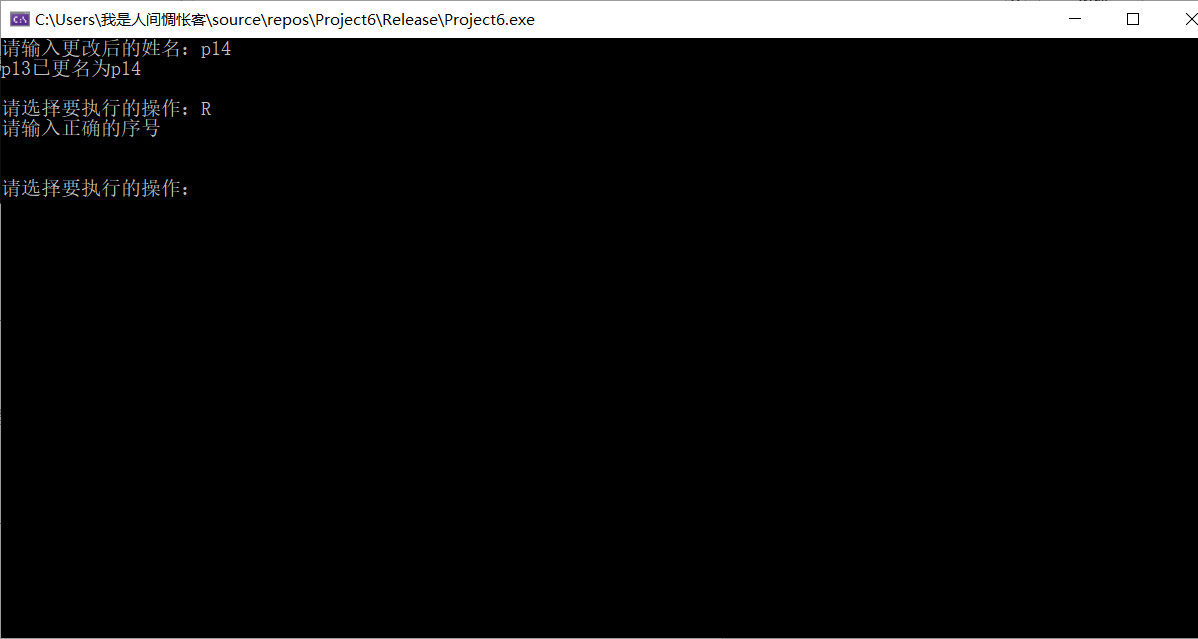
### 4.2.1 操作码错误

**测试用例：**

R

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**



### 4.2.2 找不到此人

**测试用例：**

D p13

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

