项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 王星洲

学 号： 1652977

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](#_Toc534122766)

[1.1 背景分析 1](#_Toc534122767)

[1.2 功能分析 1](#_Toc534122768)

[2 设计 1](#_Toc534122769)

[2.1 数据结构设计 1](#_Toc534122770)

[2.2 类结构设计 1](#_Toc534122771)

[2.3 成员与操作设计 2](#_Toc534122772)

[2.4 系统设计 2](#_Toc534122773)

[3 实现 3](#_Toc534122774)

[3.1 创建树 3](#_Toc534122775)

[3.1.1 创建树核心代码 3](#_Toc534122776)

[3.1.2 创建说明 3](#_Toc534122777)

[3.1.3 截屏示例 3](#_Toc534122778)

[3.2 搜索功能的实现 4](#_Toc534122779)

[3.2.1 搜索功能流程图 4](#_Toc534122780)

[3.2.2 搜索功能核心代码 5](#_Toc534122781)

[3.2.3 搜索功能截屏示例 6](#_Toc534122782)

[3.3 建立家庭功能的实现 7](#_Toc534122783)

[3.3.1 建立家庭功能流程图 7](#_Toc534122784)

[3.3.2 建立家庭功能核心代码 8](#_Toc534122785)

[3.3.3 建立家庭功能截屏示例 9](#_Toc534122786)

[3.4 添加子女功能的实现 10](#_Toc534122787)

[3.4.1 添加子女功能流程图 10](#_Toc534122788)

[3.4.2 添加子女功能核心代码 11](#_Toc534122789)

[3.4.3 添加子女功能截图示例 11](#_Toc534122790)

[3.5 解散家庭功能的实现 12](#_Toc534122791)

[3.5.1 解散家庭功能流程图 12](#_Toc534122792)

[3.5.2 解散家庭功能核心代码 13](#_Toc534122793)

[3.5.3 解散家庭功能截屏示例 14](#_Toc534122794)

[3.6 更改姓名功能的实现 15](#_Toc534122795)

[3.6.1 更改姓名功能流程图 15](#_Toc534122796)

[3.6.2 更改姓名功能核心代码 16](#_Toc534122797)

[3.6.3 更改姓名功能截屏示例 17](#_Toc534122798)

[3.7 遍历功能的实现 18](#_Toc534122799)

[3.7.1 遍历功能流程图 18](#_Toc534122800)

[3.7.2 遍历功能核心代码 19](#_Toc534122801)

[3.7.3 遍历功能截屏示例 20](#_Toc534122802)

[3.7.4 遍历功能注意事项 20](#_Toc534122803)

[3.8 总体系统实现 21](#_Toc534122804)

[3.8.1 总体系统流程图 21](#_Toc534122805)

[3.8.2 总体系统核心代码 22](#_Toc534122806)

[4 测试 23](#_Toc534122807)

[4.1 功能测试 23](#_Toc534122808)

[4.1.1 创建家谱功能测试 23](#_Toc534122809)

[4.1.2 建立家庭功能测试 24](#_Toc534122810)

[4.1.3 添加成员功能测试 25](#_Toc534122811)

[4.1.4 解散家庭功能测试 26](#_Toc534122812)

[4.1.5 更改姓名功能测试 27](#_Toc534122813)

[4.1.6 前序遍历功能测试 28](#_Toc534122814)

[4.2 边界测试 29](#_Toc534122815)

[4.2.1 建立家庭时子女数为0 29](#_Toc534122816)

[4.2.2 输入指令无视大小写 30](#_Toc534122817)

[4.3 出错测试 31](#_Toc534122818)

[4.3.1 搜索结点时输入不存在的结点 31](#_Toc534122819)

[4.3.2 操作码错误 32](#_Toc534122820)

[5 亮点 32](#_Toc534122821)

[5.1 人机交互 32](#_Toc534122822)

[5.2 模块化 32](#_Toc534122823)

[5.3 前序遍历 32](#_Toc534122824)

# 1 分析

## 1.1 背景分析

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。

## 1.2 功能分析

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

如上述分析描述，本题主要考查对于树的运用，所以我们需要两个类——树结点以及树类，而本题并不是二叉树而是多叉树，所以子结点的个数无法保证，因此可以使用vector来存储所有子结点，同时，所有的姓名可以使用string来存储，而程序需要用到搜索函数，在树中搜索结点可以使用类似遍历的方式，而便利方式可以使用递归调用，也可以使用循环控制等，这里为了节省时间成本，我考虑使用非递归的方式，这样就需要一个栈stack来存储经过的结点，总体来说，函数的实现需要上述数据结构。

## 2.2 类结构设计

树包括结点TreeNode和树本身Tree两个类。两个类之间的联系方式有很多，这里我选择使用友元的方式，使Tree可以调用TreeNode中的姓名，输出等成员和功能。

## 2.3 成员与操作设计

//树的结点类

class TreeNode

{

public:

//成员名字

string m\_name;

//子结点数组

vector<TreeNode \*> m\_children;

//构造函数

TreeNode() { m\_name = "";}

//添加子女

void AddChild(string name);

//输出子女

void printChild();

//删除子女

void deleteChild();

//更改姓名

void updateName(string name);

};

//树类

class Tree

{

//友元树结点

friend TreeNode;

public:

//根结点

TreeNode \*m\_root;

//构造函数

Tree() { m\_root = new TreeNode; m\_root->m\_name = "";}

Tree(string a) { m\_root = new TreeNode; m\_root->m\_name = a; }

//返回根结点

string getRoot() { return m\_root -> m\_name; }

//搜索姓名为name的结点

TreeNode\* search(string name);

//输出全部成员

void print();

};

## 2.4 系统设计

系统首先调用接收用户的输入，创建一个初始家谱（树），再通过循环和选择语句接收用户的指令，直到退出系统。

# 3 实现

## 3.1 创建树

### 3.1.1 创建树核心代码

Tree& creatFamily()

{

string name;

cout << "首先建立一个家谱" << endl;

cout << "请输入祖先的姓名:";

cin >> name;

Tree Family(name);

cout << "此家谱的祖先是:";

cout << Family.getRoot();

cout << endl;

return Family;

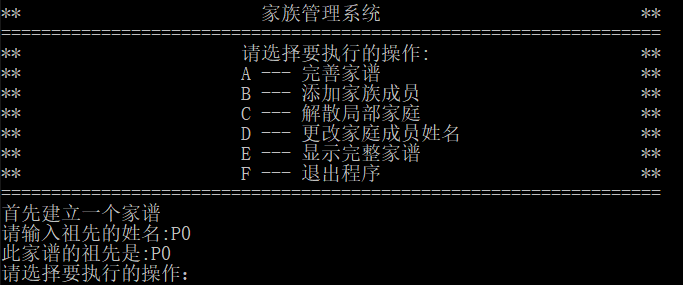
}

Tree(string a) { m\_root = new TreeNode; m\_root->m\_name = a; }

### 3.1.2 创建说明

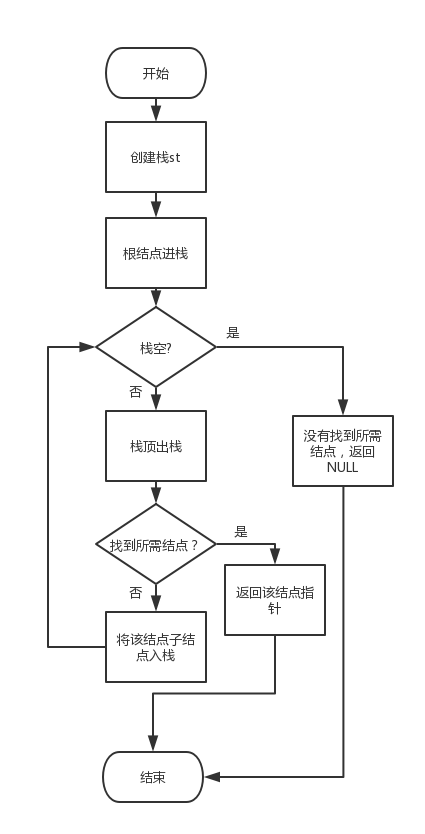
创建树的过程我按照题目要求只创建一个单独的祖先结点，即树的根结点。在获得用户输入的名字后将其根结点的m\_name更改为相应的string，再提供用户反馈即完成。

### 3.1.3 截屏示例



## 3.2 搜索功能的实现

### 3.2.1 搜索功能流程图



### 3.2.2 搜索功能核心代码

//深度搜索姓名为name的结点

TreeNode\* Tree::search(string name)

{

//栈用来存储经过的结点

stack<TreeNode \*> st;

TreeNode \* p;

//将根结点入栈

st.push(m\_root);

//当栈不空

while (!st.empty())

{

//栈顶出栈

p = st.top();

st.pop();

//找到结点

if (p->m\_name == name)

{

return p;

}

//没有找到对应结点

else

{

//将p的子结点入栈

for (int i = 0; i < p->m\_children.size(); i++)

{

if (p->m\_children[i] != NULL)

{

st.push(p->m\_children[i]);

}

}

}

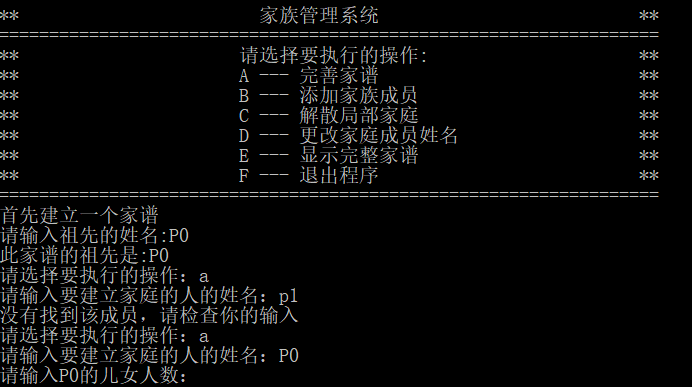
}

//没有找到

return NULL;

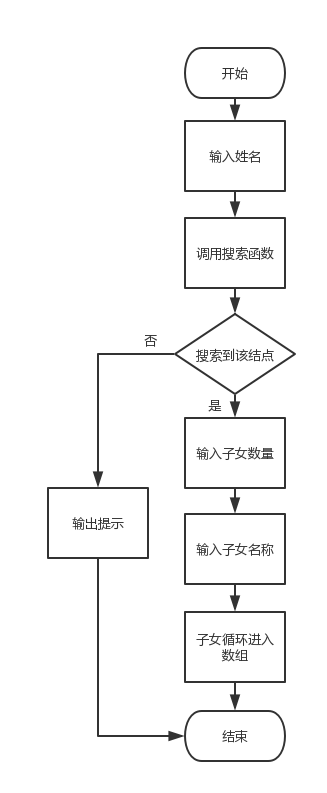
}

### 3.2.3 搜索功能截屏示例



## 3.3 建立家庭功能的实现

### 3.3.1 建立家庭功能流程图



### 3.3.2 建立家庭功能核心代码

//响应a指令建立家庭

void addFamily(Tree& Family)

{

string name, child\_name;

int number = 0;

cout << "请输入要建立家庭的人的姓名：";

cin >> name;

TreeNode \*member = Family.search(name);

if (member == NULL)

{

cout << "没有找到该成员，请检查你的输入" << endl;

return;

}

else

{

cout << "请输入" << name << "的儿女人数：";

while (1)

{

if (!(cin >> number))

{

cout << "输入非法，请重新输入" << endl;

cin.clear();

while (cin.get()!='\n')

{

continue;

}

}

else if (number < 0)

{

cout << "请输入大于等于0的数" << endl;

cin.clear();

while (cin.get() != '\n')

{

continue;

}

}

else

{

break;

}

}

if (number == 0)

{

cout << "该成员没有子女" << endl;

return;

}

else

{

cout << "请依次输入" << name << "的儿女的姓名：";

for (int i = 0; i < number; i++)

{

cin >> child\_name;

//添加子结点

member->AddChild(child\_name);

}

cout << name << "的第一代子孙是：";

member->printChild();

}

}

}

//添加子女

void TreeNode::AddChild(string name)

{

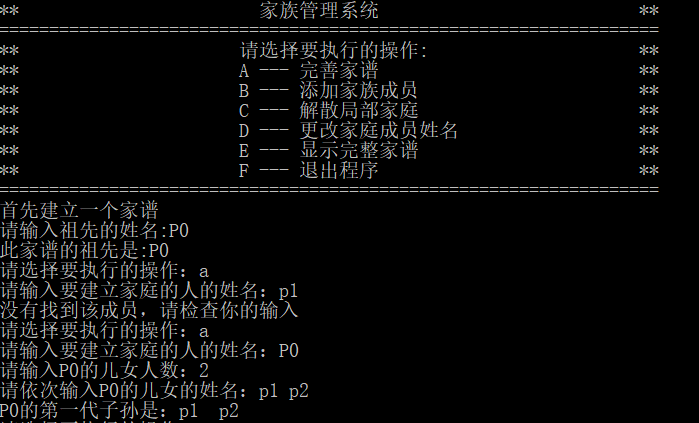
TreeNode \*child = new TreeNode;

child->m\_name = name;

m\_children.push\_back(child);

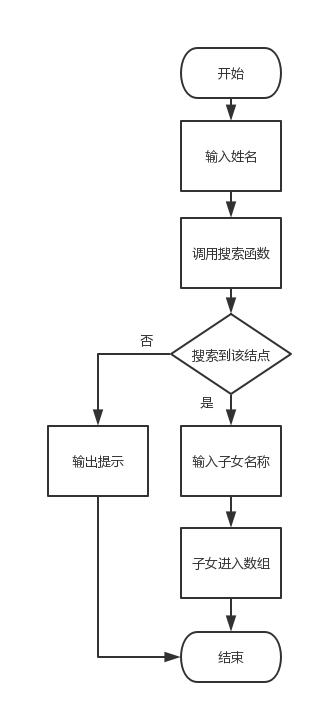
}

### 3.3.3 建立家庭功能截屏示例



## 3.4 添加子女功能的实现

### 3.4.1 添加子女功能流程图



### 3.4.2 添加子女功能核心代码

//响应b指令添加子女

void addChild(Tree& Family)

{

string name;

cout << "请输入要添加儿子(或女儿)的人的姓名：";

cin >> name;

TreeNode \*member = Family.search(name);

if (member == NULL)

{

cout << "没有找到该成员，请检查你的输入" << endl;

return;

}

else

{

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

string child\_name;

cin >> child\_name;

member->AddChild(child\_name);

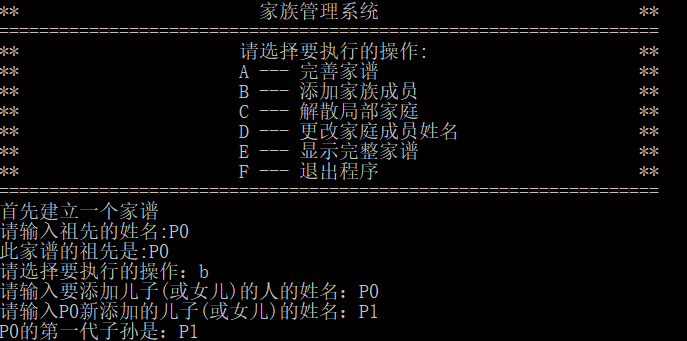
cout << name << "的第一代子孙是：";

member->printChild();

}

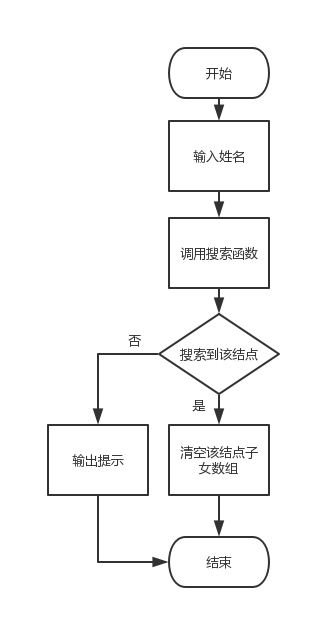
}

### 3.4.3 添加子女功能截图示例



## 3.5 解散家庭功能的实现

### 3.5.1 解散家庭功能流程图



### 3.5.2 解散家庭功能核心代码

//响应c指令解散家庭

void deleteFamily(Tree& Family)

{

string name;

cout << "请输入要解散家庭的人的姓名：";

cin >> name;

TreeNode \*member = Family.search(name);

if (member == NULL)

{

cout << "没有找到该成员，请检查你的输入" << endl;

return;

}

else

{

cout << "要解散家庭的人是：" << name << endl;

cout << name << "的第一代子孙是：";

member->printChild();

member->deleteChild();

cout << "解散成功！他的孩子都删除啦。" << endl;

}

}

//删除所有子女

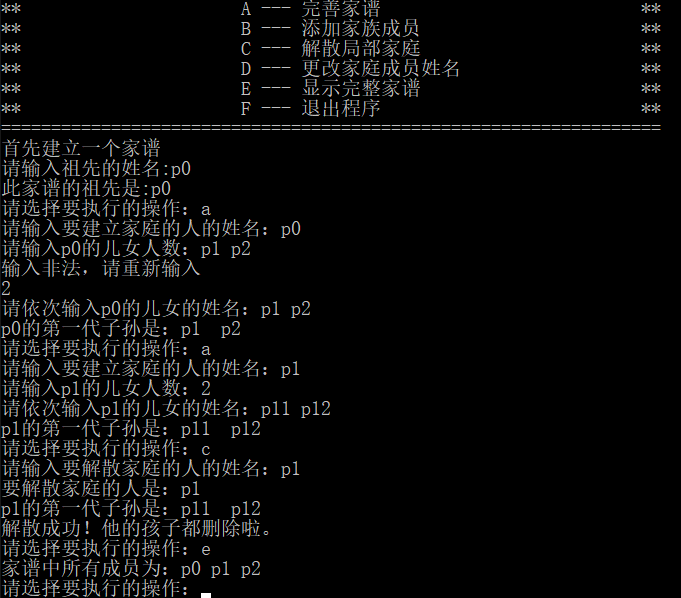
void TreeNode::deleteChild()

{

m\_children.clear();

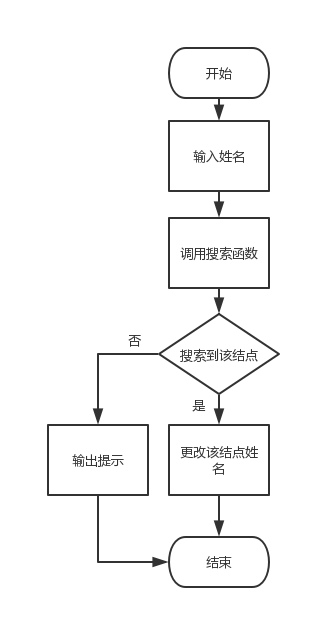
}

### 3.5.3 解散家庭功能截屏示例



## 3.6 更改姓名功能的实现

### 3.6.1 更改姓名功能流程图



### 3.6.2 更改姓名功能核心代码

//响应d指令更改成员姓名

void updateName(Tree& Family)

{

string name;

cout << "请输入要更改姓名的人目前姓名：";

cin >> name;

TreeNode \*member = Family.search(name);

if (member == NULL)

{

cout << "没有找到该成员，请检查你的输入" << endl;

return;

}

else

{

cout << "请输入更改后的姓名：";

string new\_name;

cin >> new\_name;

member->updateName(new\_name);

cout << name << "已更名为" << new\_name << endl;

}

}

//更改姓名

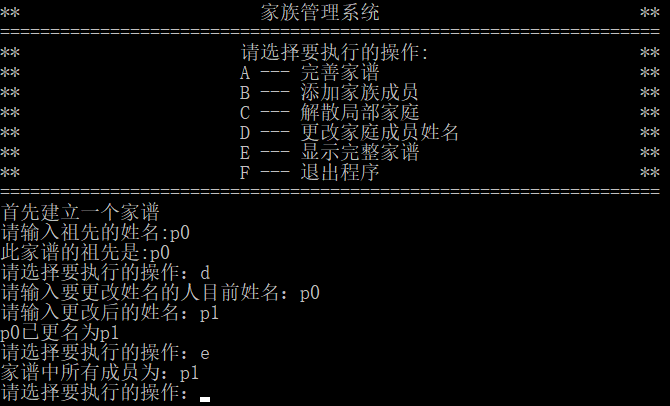
void TreeNode::updateName(string name)

{

m\_name = name;

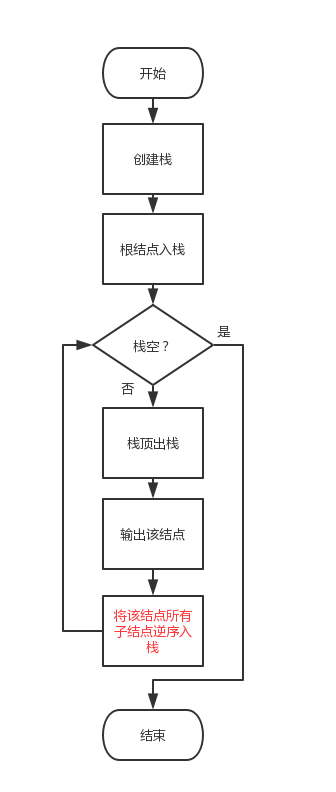
}

### 3.6.3 更改姓名功能截屏示例



## 3.7 遍历功能的实现

### 3.7.1 遍历功能流程图



### 3.7.2 遍历功能核心代码

//响应e指令输出所有成员

void printTree(Tree& Family)

{

cout << "家谱中所有成员为：";

Family.print();

}

//前序输出所有成员

void Tree::print()

{

stack<TreeNode \*> st;

TreeNode \* p;

st.push(m\_root);

while (!st.empty())

{

p = st.top();

st.pop();

cout << p->m\_name << " ";

for (int i = p->m\_children.size() - 1; i >= 0; i--)

{

if (p->m\_children[i] != NULL)

{

st.push(p->m\_children[i]);

}

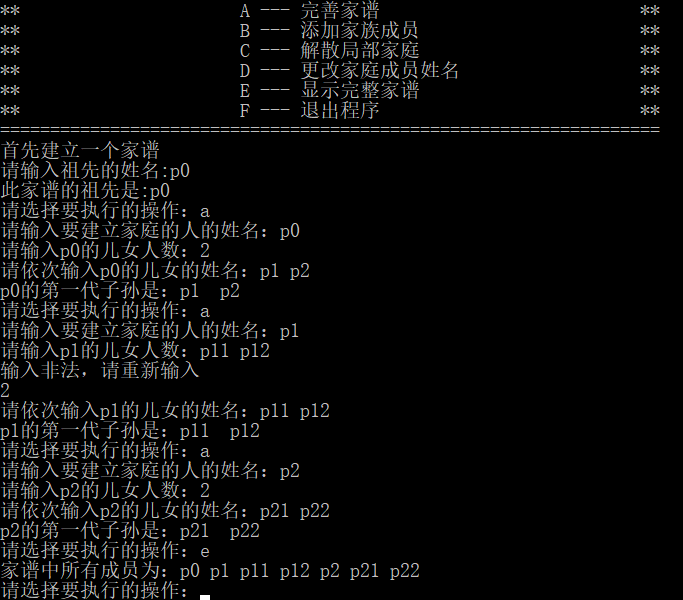
}

}

cout << endl;

}

### 3.7.3 遍历功能截屏示例

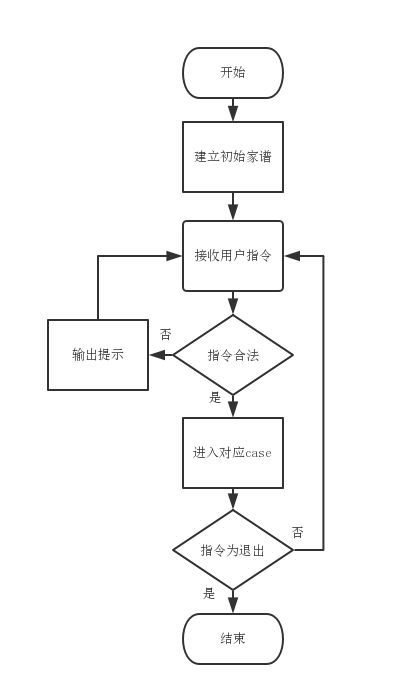


### 3.7.4 遍历功能注意事项

与之前的搜索算法类似，但需要注意的是，由于我打算前序输出，所以对元素入栈的顺序有要求，在将子结点入栈时需要逆序入栈，这样由于栈后进先出的性质就实现了前序输出遍历。

## 3.8 总体系统实现

### 3.8.1 总体系统流程图



### 3.8.2 总体系统核心代码

int main()

{

char order;

showMenu();

Tree Family = creatFamily();

while (1)

{

cout << "请选择要执行的操作：";

cin >> order;

switch (order)

{

//建立家庭

case 'A':

case 'a':

addFamily(Family);

break;

//添加子女

case 'B':

case 'b':

addChild(Family);

break;

//解散家庭

case 'C':

case 'c':

deleteFamily(Family);

break;

//更改成员名字

case 'D':

case 'd':

updateName(Family);

break;

//输出所有成员

case 'E':

case 'e':

printTree(Family);

break;

//退出程序

case 'F':

case 'f':

cout << "退出系统成功" << endl;

system("pause");

return 0;

//指令有错误

default:

cout << "你输入的命令有误，请重新输入" << endl;

break;

}

}

system("pause");

return 0;

}

# 4 测试

## 4.1 功能测试

### 4.1.1 创建家谱功能测试

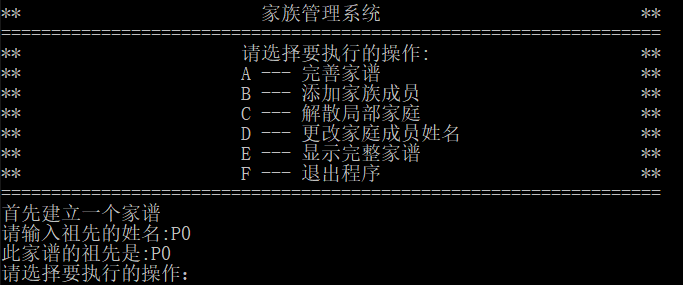
**测试用例**：

祖先为P0

**预期结果**：

创建成功

**实验结果**



### 4.1.2 建立家庭功能测试

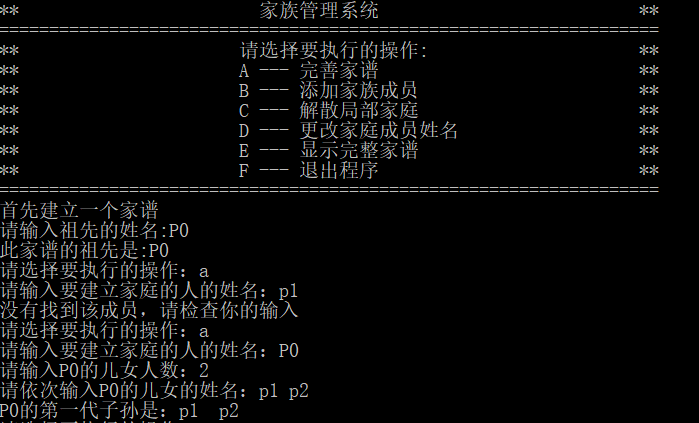
**测试用例：**

P0建立家庭

**预期结果：**

P0家庭建立成功

**实验结果：**



### 4.1.3 添加成员功能测试

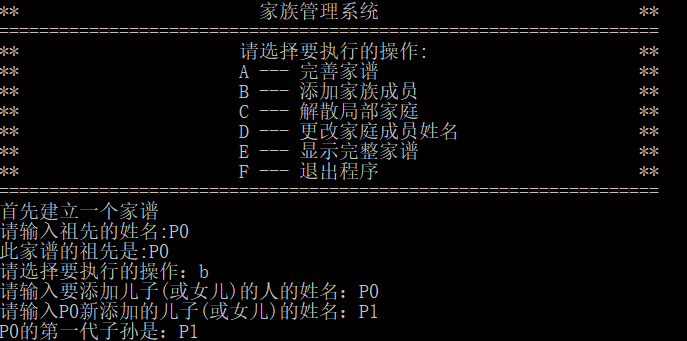
**测试用例：**

P0添加子女P1

**预期结果：**

添加成功

**实验结果：**



### 4.1.4 解散家庭功能测试

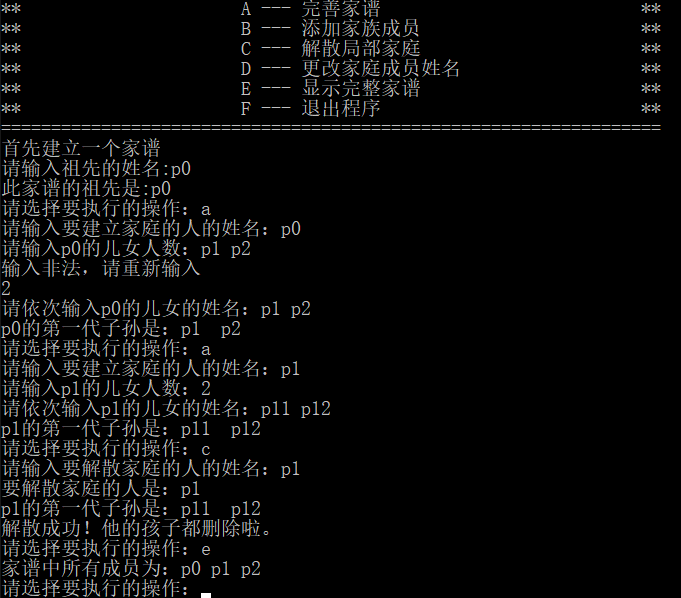
**测试用例：**

P0子女P1 P2，P1子女P11，P12，解散P1家庭

**预期结果：**

仅剩P0，P1，P2

**实验结果：**



### 4.1.5 更改姓名功能测试

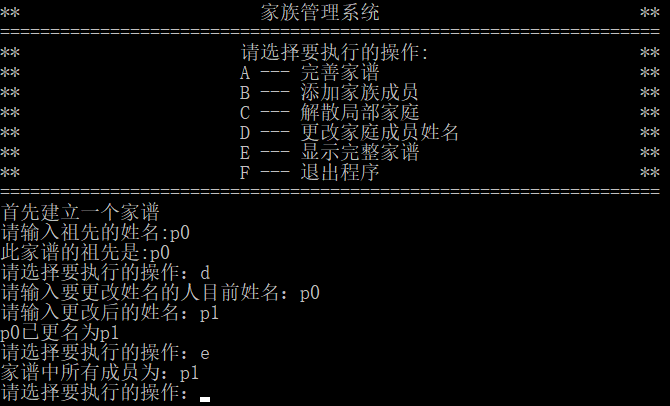
**测试用例：**

P0更名为P1

**预期结果：**

更名成功

**实验结果：**



### 4.1.6 前序遍历功能测试

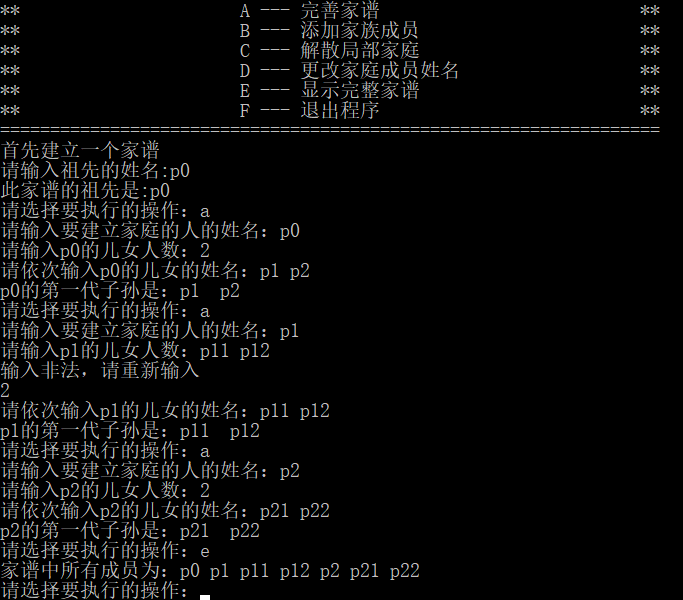
**测试用例：**

P0子女P1 P2，P1子女P11 P12，P2子女P21 P22.

**预期结果：**

P0 P1 P11 P12 P2 P21 P22

**实验结果：**



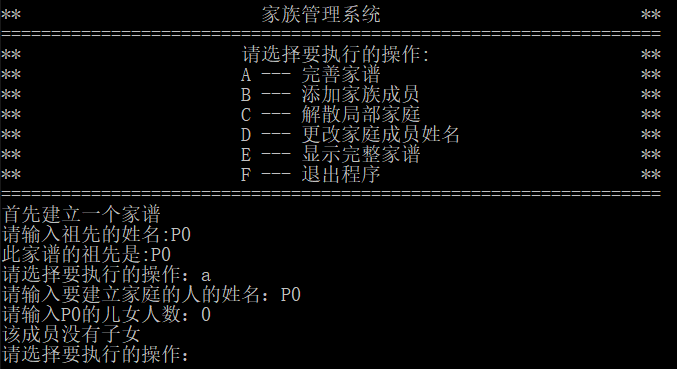
## 4.2 边界测试

### 4.2.1 建立家庭时子女数为0

**测试用例：**建立家庭时子女数为0

**预期结果：**输出提示，程序正确运行

**实验结果：**

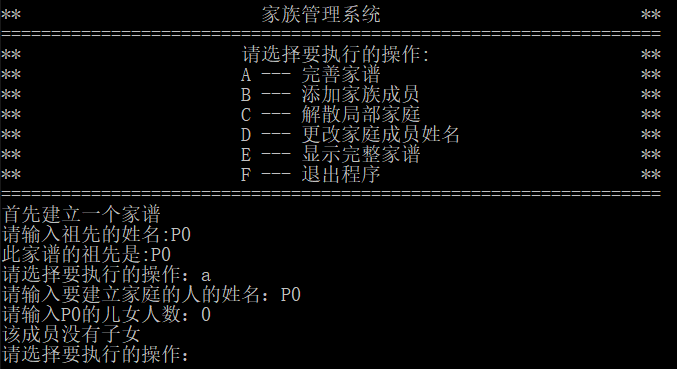


### 4.2.2 输入指令无视大小写

**测试用例：**输入小写指令

**预期结果：**程序正常运行，不崩溃。

**实验结果：**



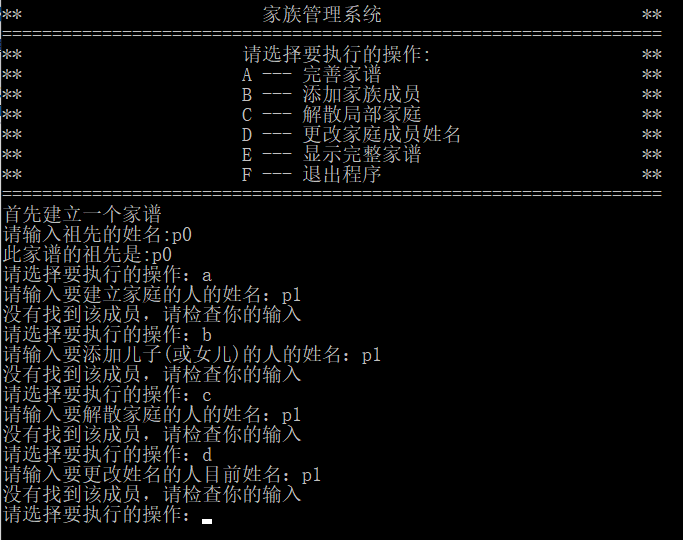
## 4.3 出错测试

### 4.3.1 搜索结点时输入不存在的结点

**测试用例：**输入不存在的结点

**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：**

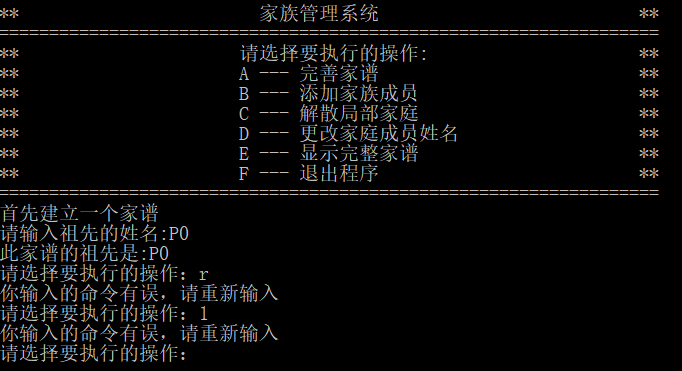


### 4.3.2 操作码错误

**测试用例：**输入操作码错误

**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：**



# 5 亮点

## 5.1 人机交互

本题中我添加了大量人机代码满足人机交互需求，防止意外退出或者崩溃。详情见4.3

## 5.2 模块化

本题中，所有的算法都有对应函数实现，主函数逻辑清晰，结构严谨，便于维护，详情见2.4

## 5.3 前序遍历

本题中搜索和遍历都采用了类似前序遍历的方式，使用中我采用了一个栈，避免了使用递归的方式，其中前序遍历时，对于数组的控制也有特殊说明，详情见3.2，3.7.