项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 曹峰源

学 号： 1951328

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 4](#_Toc59473123)

[1.1 背景分析 4](#_Toc59473124)

[1.2 功能分析 4](#_Toc59473125)

[2. 设计 4](#_Toc59473126)

[2.1 数据结构设计 4](#_Toc59473127)

[2.2 类结构设计 4](#_Toc59473128)

[2.3 类成员函数 4](#_Toc59473129)

[2.4 系统设计 6](#_Toc59473130)

[3. 功能实现 6](#_Toc59473131)

[3.1 创建家谱、确定家谱祖先的功能实现 6](#_Toc59473132)

[3.1.1 流程图 6](#_Toc59473133)

[3.1.2 代码实现 7](#_Toc59473134)

[3.1.3 结果显示 7](#_Toc59473135)

[3.2 遍历家谱的功能实现 7](#_Toc59473136)

[3.2.1 流程图 7](#_Toc59473137)

[3.2.2 代码实现 8](#_Toc59473138)

[3.3 家谱中成员创建家庭的功能实现 9](#_Toc59473139)

[3.3.1 流程图 9](#_Toc59473140)

[3.3.2 代码实现 9](#_Toc59473141)

[3.3.3 结果显示 11](#_Toc59473142)

[3.4 家谱中添加家庭成员的功能实现 11](#_Toc59473143)

[3.4.1 流程图 11](#_Toc59473144)

[3.4.2 代码实现 11](#_Toc59473145)

[3.4.3 结果显示 12](#_Toc59473146)

[3.5 家谱中解散局部家庭的功能实现 13](#_Toc59473147)

[3.5.1 流程图 13](#_Toc59473148)

[3.5.2 代码实现 13](#_Toc59473149)

[3.5.3 结果显示 14](#_Toc59473150)

[3.6 更改家庭成员姓名的功能 15](#_Toc59473151)

[3.6.1 流程图 15](#_Toc59473152)

[3.6.2 代码实现 15](#_Toc59473153)

[3.6.3 结果显示 16](#_Toc59473154)

[3.7 退出程序功能 16](#_Toc59473155)

[3.8 主函数功能 16](#_Toc59473156)

[3.8.1 流程图 16](#_Toc59473157)

[3.8.2 代码实现 17](#_Toc59473158)

[4. 测试 18](#_Toc59473159)

[4.1 边界情况测试 18](#_Toc59473160)

[4.1.1 没有家庭成员时增添成员 18](#_Toc59473161)

[4.1.2 没有家庭成员时解散家庭 18](#_Toc59473162)

[4.2 错误测试 19](#_Toc59473163)

[4.2.1 成员不存在 19](#_Toc59473164)

[4.2.2 操作码错误 19](#_Toc59473165)

[4.2.3 解散家庭的功能测试 19](#_Toc59473166)

[5. 优化方向 20](#_Toc59473167)

1. 分析
   1. 背景分析

对于一个家庭来说，家庭成员的数据记录对于一个家庭来说是十分重要的，家谱的重要性也就体现出来，这是独属于中国人的一种文化传承。在过去，家谱都是纸来记录的，现今，计算机高速发展的时代，我们可以利用计算机来帮助家谱的管理。并且，这种记录方式更加高效并且稳定。

* 1. 功能分析

对于数字化的家谱而言，首先要能够建立家谱的起点，作为家谱的祖先。其次，要能够对家谱进行遍历来使实现家谱家庭的建立、家庭成员的增添、家庭成员的删除等操作。最后程序还应该能够手动退出。

1. 设计
   1. 数据结构设计

家谱的设计与实现用树来实现非常方便，对于树这样的结构，如果使用数组无法得知所需空间的大小，在后续增减人员中非常不方便，所以采用链表的方式，因此也设计了结点类来存放每一个家庭成员的数据。

* 1. 类结构设计

为满足需求，成员类定义了node类，链表树定义了familytree类来实现。此外，因为树的遍历中采取的是前序非递归算法，所以定义了栈类stack来帮助实现。

* 1. 类成员函数

结点类node

1. **class** node
2. {
3. **friend** **class** familytree;
4. **friend** **class** stack;
5. **private**:
6. string name;
7. node\* next = NULL;//next指针用于确定栈中的顺序
8. node\* firstson=NULL;//保存该结点的长子
9. node\* nextbrother=NULL;//保存该结点的下一个兄弟
10. }；

树类familytree

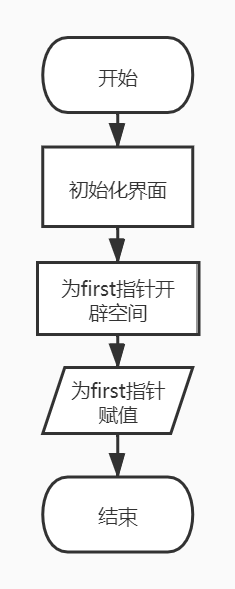
1. **class** familytree
2. {
3. **private**:
4. node\* first=NULL;//根结点
5. node\* current = NULL;
6. stack s;//定义栈s
7. **void** destory(node\* p)//用于将p结点及以下的空间都释放掉
8. {
9. **if** (p != NULL)
10. {
11. **if**(p->firstson!=NULL)destory(p->firstson);
12. **if**(p->nextbrother!=NULL)destory(p->nextbrother);
13. **delete** p;
14. }
15. }
16. **public**:
17. familytree() //初始化界面
18. {
19. cout << "\*\*           家谱管理系统           \*\*" << endl;
20. cout << "======================================" << endl;
21. cout << "\*\*         请选择要执行的操作:        " << endl;
22. cout << "\*\*          A---完善家谱            \*\*" << endl;
23. cout << "\*\*          B---添加家庭成员        \*\*" << endl;
24. cout << "\*\*          C---解散局部家庭        \*\*" << endl;
25. cout << "\*\*          D---更改家庭成员姓名    \*\*" << endl;
26. cout << "\*\*          E---退出程序            \*\*" << endl;
27. cout << "首先建立一个家谱！"<<endl;
28. cout << "请输入祖先的姓名：";
29. string name;
30. cin >> name;
31. create(name);
32. cout << "此家谱的祖先是：";
33. putname(first);
34. cout << endl;
35. cout << endl;
36. }
37. ~familytree()
38. {
39. destory(first);
40. }
41. **bool** traversal(string name);//用于树的遍历
42. **void** create(string name);//用于树的创建
43. **void** putname(node\* p);//用于树成员姓名的输出
44. **void** addfamily();//用于树成员增添家庭
45. **void** addpeople();//用于树中成员增添子女
46. **void** delfamily();//用于树中成员解散家庭
47. **void** changename();//用于树中成员更改姓名
48. }；

栈类stack

1. **class** stack//定义栈类用于树进行遍历时使用
2. {
3. **private**:
4. node\* top;
5. **public**:
6. stack() { top = NULL; }
7. ~stack() { top = NULL; }
8. **bool** isempty();
9. **void** push(node\* p);
10. **void** empty();
11. node\* pop();
12. }；
    1. 系统设计

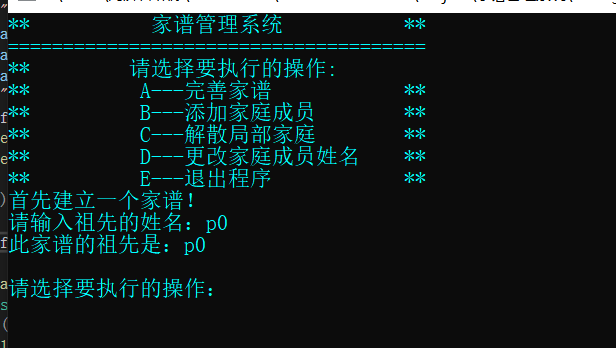
在主函数的设计中，首先定义一个树类，在树中初始化操作台界面，接着由用户输入操作码，根据对应的操作码执行对应的操作。

1. 功能实现
   1. 创建家谱、确定家谱祖先的功能实现
      1. 流程图



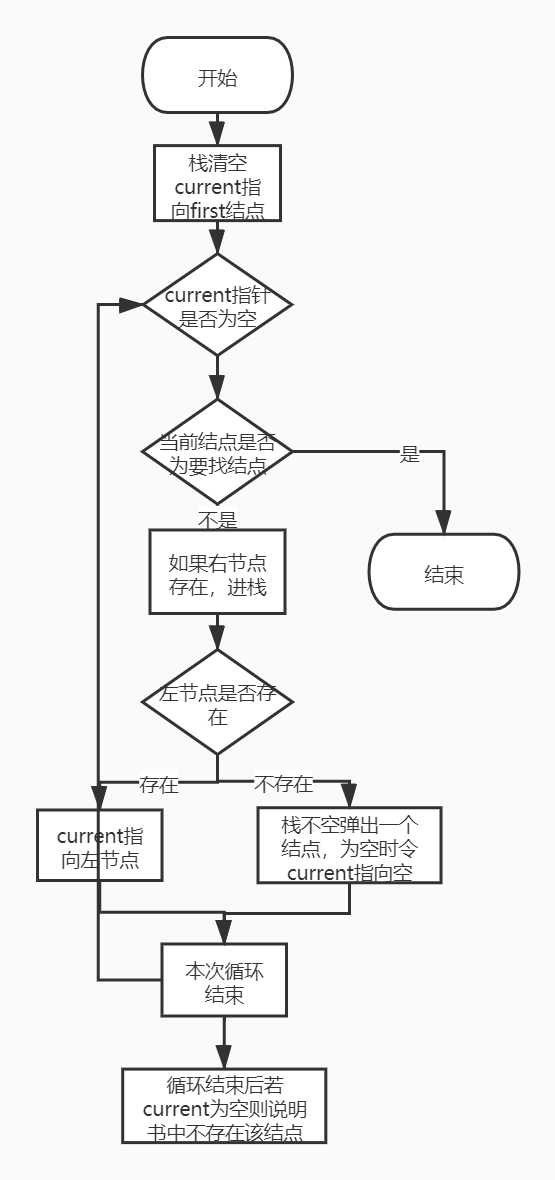
* + 1. 代码实现

1. **void** familytree::create(string name)//树初始化
2. {
3. first = **new** node;
4. first->name = name;
5. current = first;
6. }
   * 1. 结果显示



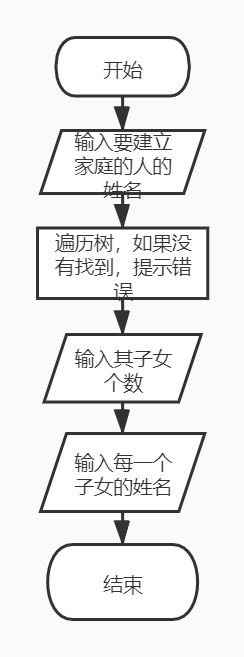
* 1. 遍历家谱的功能实现
     1. 流程图

这段代码使用的是非递归的前序遍历来实现树结构的遍历



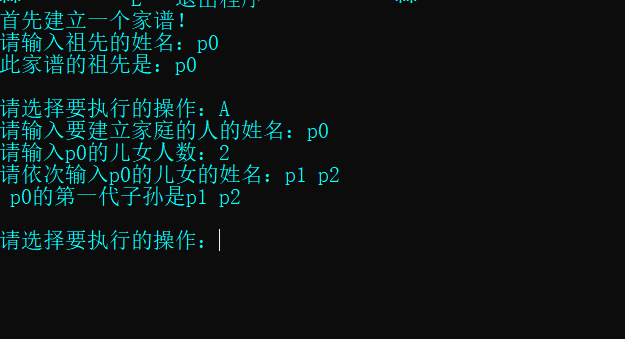
* + 1. 代码实现

1. **bool** familytree::traversal(string name)//遍历函数，如果没有该元素就返回false
2. {
3. s.empty();
4. current=first;
5. **while** (current!=NULL)
6. {
7. **if** (current->name == name) **return** **true**;
8. **if** (current->nextbrother != NULL)
9. {
10. s.push(current->nextbrother);
11. }
12. **if** (current->firstson != NULL) { current = current->firstson; }
13. **else**
14. {
15. **if** (s.isempty()!= 0)
16. {
17. current=s.pop();
18. }
19. **else** current = NULL;
20. }
21. }
22. **if** (current == NULL)
23. {
24. cout << "未能找到该成员" << endl<<endl;
25. **return** 0;
26. }
27. }
    1. 家谱中成员创建家庭的功能实现
       1. 流程图

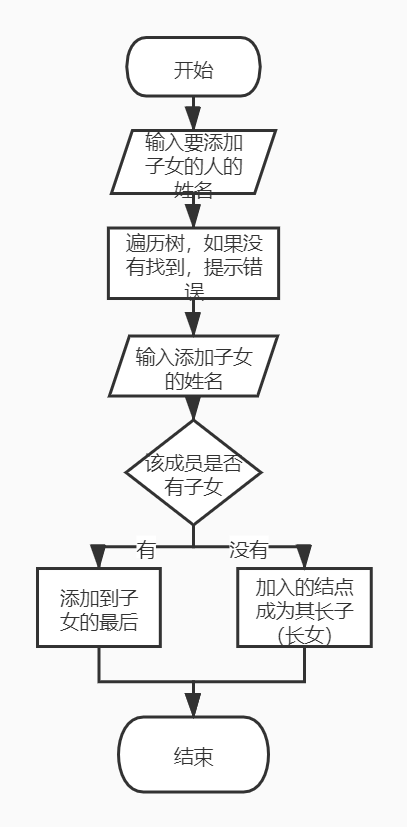


* + 1. 代码实现

1. **void** familytree::addfamily()//建立家庭操作
2. {
3. cout << "请输入要建立家庭的人的姓名：";
4. string name;
5. cin >> name;
6. **if** (traversal(name) == 0)  **return**;//如果没有找到，退出函数
7. cout << "请输入"<<name<<"的儿女人数：";
8. **int** num;
9. cin >> num;
10. **if** (num == 0)
11. {
12. cout << "该成员无子女"<<endl;
13. cout << endl;
14. **return**;
15. }
16. cout << "请依次输入"<<name<<"的儿女的姓名：";
17. string sonname;
18. cin >> sonname;
19. node\* p = **new** node;
20. p->name = sonname;
21. current->firstson = p;
22. **for** (**int** i=1;i<num;i++)
23. {
24. cout << " ";
25. string nextname;
26. cin >> nextname;
27. node\* q=**new** node;
28. q->name = nextname;
29. p->nextbrother = q;
30. p = p->nextbrother;
31. }
32. putname(current);
33. p = current->firstson;
34. cout << "的第一代子孙是";
35. **while** (p!=NULL)
36. {
37. putname(p);
38. cout << " ";
39. p = p->nextbrother;
40. }
41. cout << endl;
42. cout << endl;
43. }
    * 1. 结果显示

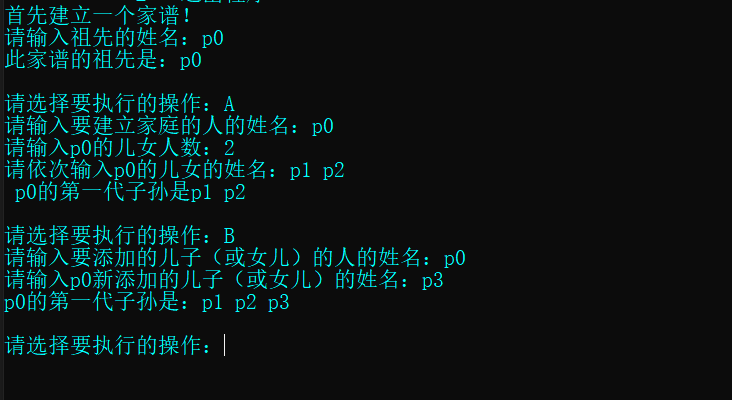


* 1. 家谱中添加家庭成员的功能实现
     1. 流程图

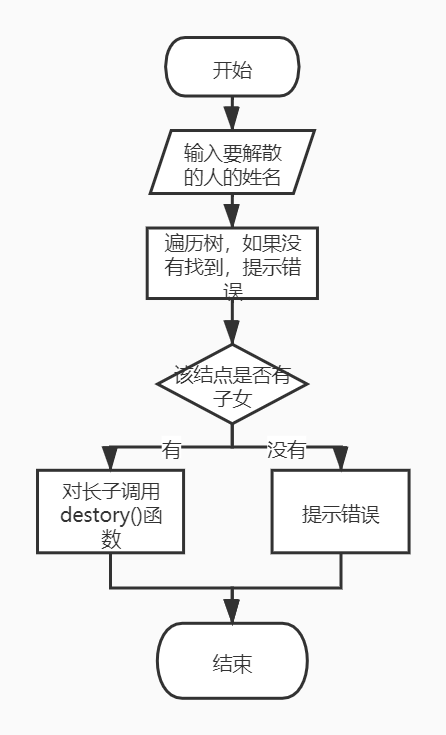


* + 1. 代码实现

1. **void** familytree::addpeople()//向家庭中添加成员的操作
2. {
3. cout << "请输入要添加的儿子（或女儿）的人的姓名：";
4. string name;
5. cin >> name;
6. **if** (traversal(name) == 0) **return**;
7. cout << "请输入"<<name<<"新添加的儿子（或女儿）的姓名：";
8. string sonname;
9. cin >> sonname;
10. node\* p;
11. p = current->firstson;
12. node\* q = **new** node;
13. q->name = sonname;
14. **if**(p==NULL)
15. {
16. current->firstson = q;
17. }
18. **else**
19. {
20. **while** (p->nextbrother!=NULL)
21. {
22. p = p->nextbrother;
23. }
24. p->nextbrother = q;
25. }
26. cout <<name<< "的第一代子孙是：";
27. p = current->firstson;
28. **while** (p!=NULL)
29. {
30. cout << p->name << " ";
31. p = p->nextbrother;
32. }
33. cout << endl;
34. cout << endl;
35. }
    * 1. 结果显示

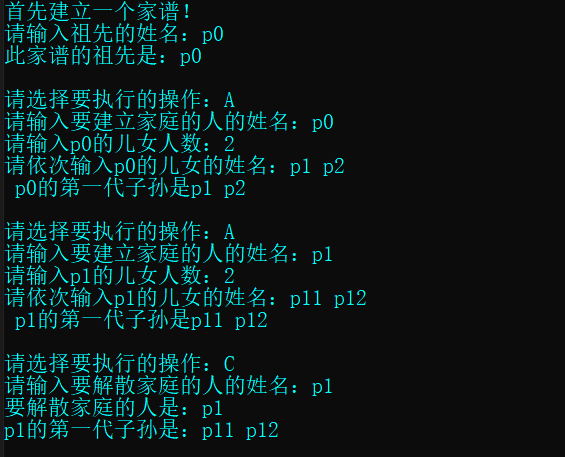


* 1. 家谱中解散局部家庭的功能实现
     1. 流程图

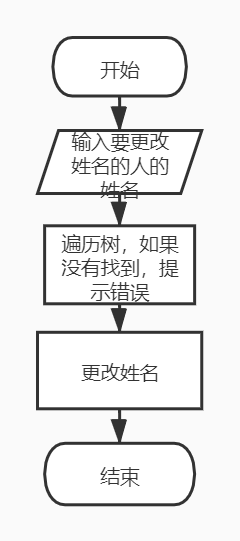


* + 1. 代码实现

1. **void** familytree::delfamily()//解散部分家庭的操作
2. {
3. cout << "请输入要解散家庭的人的姓名：";
4. string name;
5. cin >> name;
6. cout << "要解散家庭的人是："<<name<<endl;
7. **if**(traversal(name)==0) **return**;
8. node\* p;
9. p = current->firstson;
10. **if** (p == NULL)
11. {
12. cout << "该成员没有子孙"<<endl;
13. **return**;
14. }
15. cout << name << "的第一代子孙是：";
16. **while** (p!=NULL)
17. {
18. cout << p->name<<" ";
19. p = p->nextbrother;
20. }
21. p = current->firstson;
22. destory(p);//调用destory()函数
23. current->firstson = NULL;
24. cout << endl;
25. cout << endl;
26. }
    * 1. 结果显示

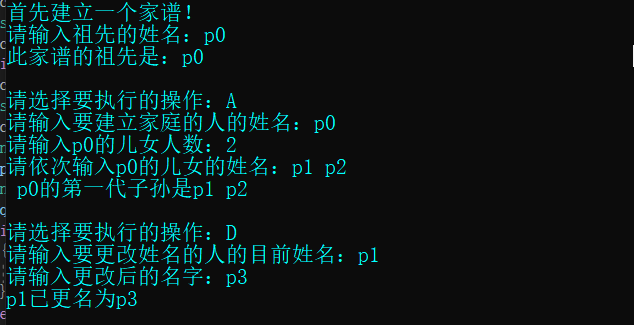


* 1. 更改家庭成员姓名的功能
     1. 流程图



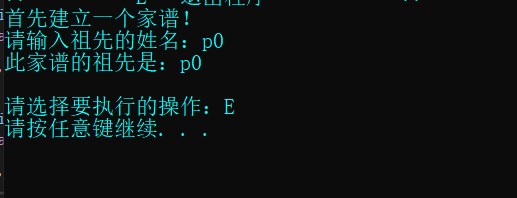
* + 1. 代码实现

1. **void** familytree::changename()//更改家庭成员姓名
2. {
3. cout << "请输入要更改姓名的人的目前姓名：";
4. string name;
5. cin >> name;
6. **if**(traversal(name)==0) **return**;
7. cout << "请输入更改后的名字：";
8. string change;
9. cin >> change;
10. cout << name << "已更名为" << change;
11. current->name = change;
12. cout << endl<<endl;
13. }
    * 1. 结果显示

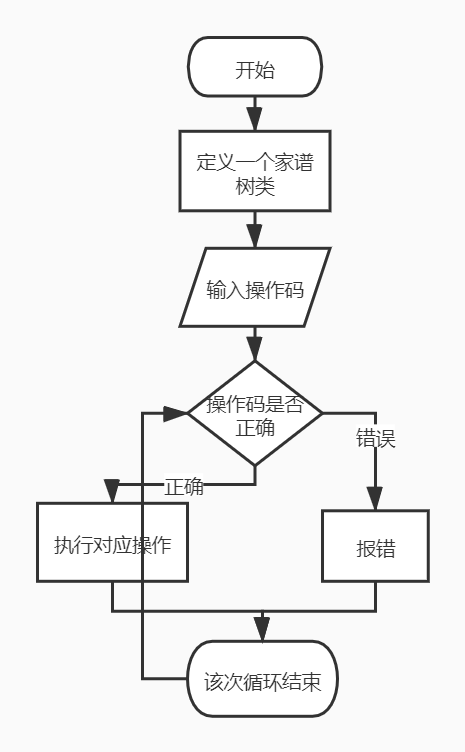


* 1. 退出程序功能

结果显示：



* 1. 主函数功能
     1. 流程图

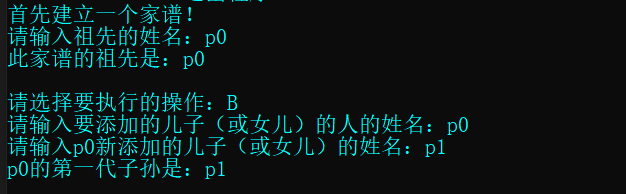


* + 1. 代码实现

1. **int** main()
2. {
3. familytree family;//建立一个家谱树
4. **char** judge;//定义操作码
5. cout << "请选择要执行的操作：";
6. **while** (cin>>judge)
7. {
8. **switch** (judge)
9. {
10. **case** 'A':
11. {
12. family.addfamily();
13. **break**;
14. }
15. **case** 'B':
16. {
17. family.addpeople();
18. **break**;
19. }
20. **case** 'C':
21. {
22. family.delfamily();
23. **break**;
24. }
25. **case** 'D':
26. {
27. family.changename();
28. **break**;
29. }
30. **case** 'E':
31. {
32. system("pause");
33. **return** 0;
34. }
35. **default**:
36. {
37. cout << "操作码有误！" << endl;
38. **break**;
39. }
40. }
41. cout << "请选择要执行的操作：";
42. }
43. }
44. 测试
    1. 边界情况测试
       1. 没有家庭成员时增添成员

目标：程序不崩溃，正常添加家庭成员

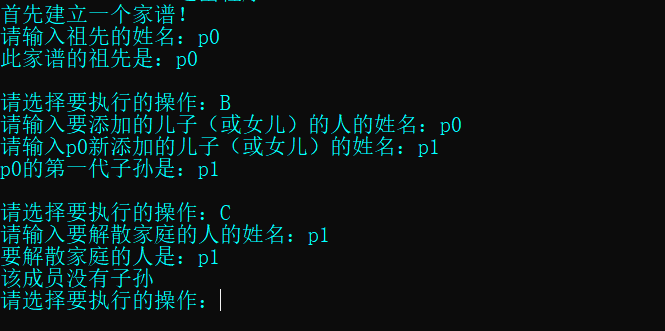
结果显示：



* + 1. 没有家庭成员时解散家庭

目标：程序不崩溃，提示该成员没有子女

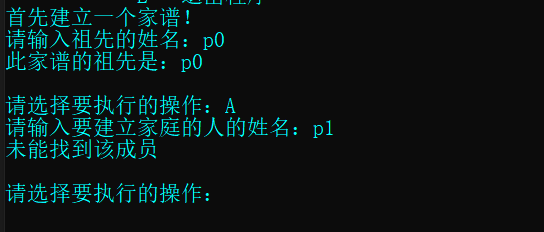
结果显示：



* 1. 错误测试
     1. 成员不存在

目标：因为成员遍历都是由函数traversal()来实现的，所以只需要测试一种操作情况下成员不存在即可。这里以新建家庭功能为例。

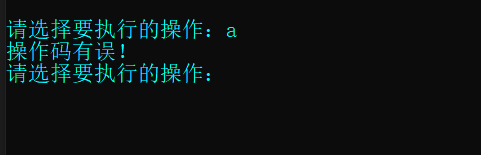
结果显示：



* + 1. 操作码错误

目标：程序不崩溃，显示错误信息

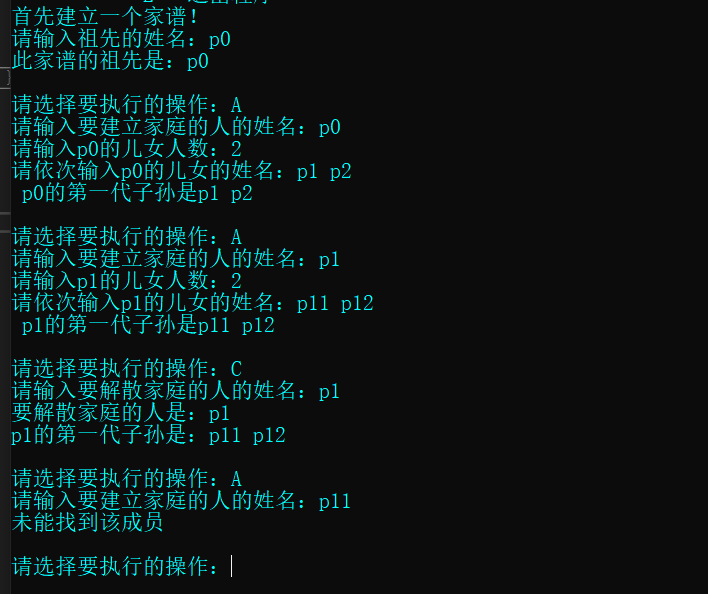
结果显示：



* + 1. 解散家庭的功能测试

在该段程序中，解散家庭使用的是destory()函数，destory()函数内部使用的是递归，这里进行测试解散家庭后原先的家庭成员是否在家谱树中出现。

结果显示：



1. 优化方向

对于这段程序，依然有值得优化的方向。主要是考虑到现实生活中可能会存在家谱中重名的情况，所以程序中并没有对此进行处理，这就导致在遍历树时只会返回第一个访问到的成员，其他重名成员会被自动忽略掉，这是不好的，如果进行优化，可以向这个方向进行。