项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 郑柯凡

学 号： 1950072

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 2](#_Toc3558)

[1.1背景分析 2](#_Toc4839)

[1.2功能分析 3](#_Toc13527)

[2 设计 3](#_Toc21792)

[2.1思路设计 3](#_Toc22307)

[2.2数据结构的选择 3](#_Toc2292)

[2.3类的设计 3](#_Toc27305)

[3 功能实现 4](#_Toc17407)

[3.1 初始化及选择模块功能的实现 4](#_Toc20679)

[3.1.1初始化及函数选择模块功能流程图 4](#_Toc25473)

[3.1.2初始化及函数选择模块功能核心代码 5](#_Toc13774)

[3.1.3初始化及函数选择模块功能截屏示例 6](#_Toc15975)

[3.2 搜索函数功能的实现 6](#_Toc2384)

[3.2.1搜索函数功能流程图 6](#_Toc4331)

[3.2.2搜索函数功能核心代码 6](#_Toc15812)

[3.2.3搜索函数功能截屏示例 7](#_Toc13869)

[3.3 完善家庭功能的实现 7](#_Toc12874)

[3.3.1完善家庭功能流程图 7](#_Toc10076)

[3.3.2完善家庭功能核心代码  8](#_Toc32666)

[3.3.3完善家庭功能截屏示例 9](#_Toc13681)

[3.4 添加家族成员功能的实现 9](#_Toc26981)

[3.4.1添加家族成员功能流程图 9](#_Toc22929)

[3.4.2添加家族成员功能核心代码 10](#_Toc32680)

[3.4.3添加家族成员功能截屏示例 11](#_Toc6529)

[3.5 解散局部家庭功能的实现 11](#_Toc6090)

[3.5.1解散局部家庭功能流程图 11](#_Toc10464)

[3.5.2解散局部家庭功能核心代码 11](#_Toc17277)

[3.5.3解散局部家庭功能截屏示例 12](#_Toc29947)

[3.6 修改家庭成员姓名功能的实现 12](#_Toc23471)

[3.6.1修改家庭成员姓名功能流程图 12](#_Toc30841)

[3.6.2修改家庭成员姓名功能核心代码 13](#_Toc10265)

[3.6.3修改家庭成员姓名功能截屏示例 13](#_Toc23858)

[4 测试 14](#_Toc12329)

[4.1功能测试 14](#_Toc26092)

[4.1.1完善家庭功能测试 14](#_Toc26717)

[4.1.2添加家庭成员功能测试 14](#_Toc9939)

[4.1.3解散局部家庭功能测试 15](#_Toc18242)

[4.1.4修改家庭成员姓名功能测试 16](#_Toc2795)

[4.2报错测试 16](#_Toc26559)

[4.2.1 父母结点不存在 16](#_Toc21961)

[4.2.2 人名重复 17](#_Toc19020)

**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。

**1.2功能分析**

本项目对家谱管理进行简单的模拟，以实现建立家谱，查看祖先和子孙个人信息，插入家族成员，修改家族成员，删除家族成员等功能。

**2 设计**

**2.1思路设计**

首先定义存储家族成员数据结构，然后将每个功能作为一个成员函数来完成对数据的操作。（家谱中不支持重名，即默认每个家庭成员名字不重复，若重复即视为错误。）

**2.2数据结构的选择**

家谱可以抽象成一棵树的模型，由于一个祖先可能不止只有一个后代，因此一个根结点可能有多个子孙，不便储存。因此，可以采用由三叉链表实现的左长子，右兄弟类型的二叉树进行储存，采用三叉链表主要是为了存储父节点的信息，以便在删除家庭时将其他子女链接到祖先上。

**2.3类的设计**

家庭成员类：存储单个家庭成员的信息。其私有成员包括成员姓名，后代结点地址，兄弟结点地址以及父亲结点地址。其公有成员包括默认构造函数，复制构造函数以及析构函数。

1. //家庭成员类
2. **class** FamilyMember {
3. **friend** **class** FamilyTree;
4. **friend** **class** Queue;
5. **private**:
6. string name;
7. FamilyMember\* child;
8. FamilyMember\* brother;
9. FamilyMember\* parent;
10. **public**:
11. FamilyMember()
12. {
13. name = '#';
14. child = brother = parent=NULL;
15. }
16. FamilyMember(**const** string vname)
17. {
18. name = vname;
19. child = brother = NULL;
20. }
21. ~FamilyMember() {};
22. };

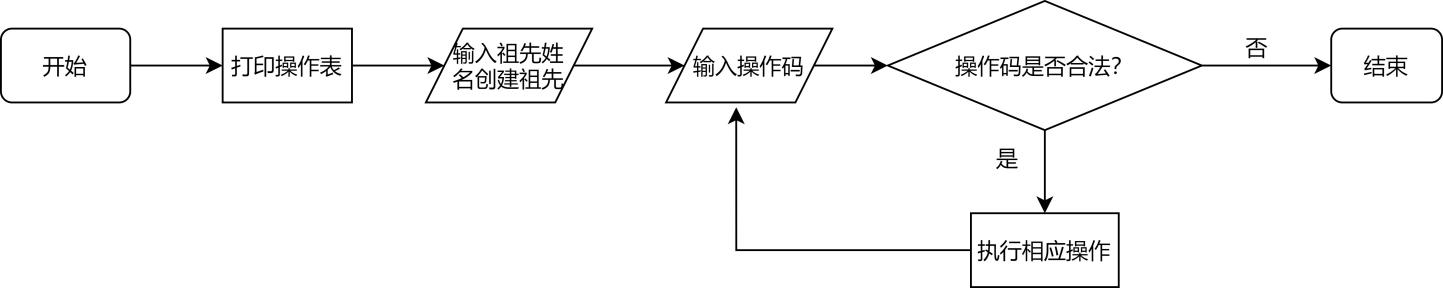
家谱树类：用来存储整个家谱。其私有成员包括家谱祖先的地址。其公有成员包括初始化，运行，搜索某个家族成员，创建家庭，添加家族成员，删除家庭，修改家族成员名字，以及打印（本题目未做要求，但可方便观察家谱树的改变，用于展示函数功能和Debug，通过MySTL头文件中的队列实现）等功能函数。

1. //家谱树类
2. **class** FamilyTree {
3. **private**:
4. FamilyMember\* ancestor;
5. **public**:
6. FamilyTree() { ancestor = NULL; };
7. ~FamilyTree() {};
8. **void** Init();
9. **bool** SelectOperation();
10. **bool** Find(string vname, FamilyMember\* start, FamilyMember\*& target);
11. **void** CreateFamily();
12. **void** AddMember();
13. **void** RemovepartFamily();
14. **void** Altername();
15. **void** PrintTree();
16. };

**3 功能实现**

**3.1 初始化及选择模块功能的实现**

**3.1.1初始化及函数选择模块功能流程图**



**3.1.2初始化及函数选择模块功能核心代码**

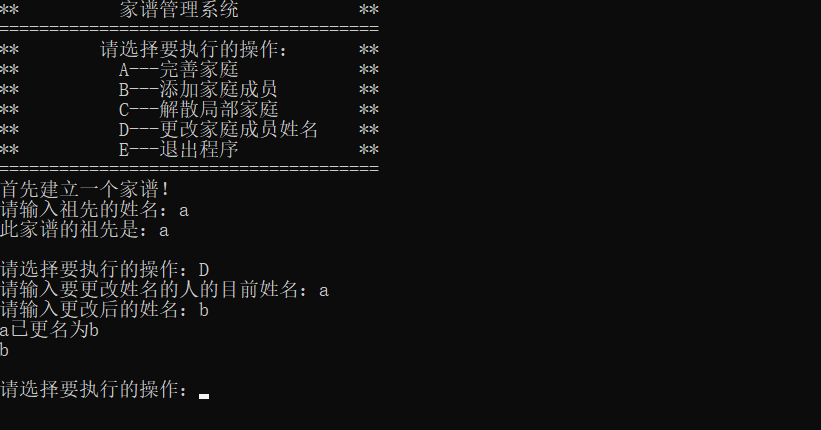
打印操作表并建立祖先，然后进入选择操作循环。

1. **void** FamilyTree::Init()
2. {
3. cout << "\*\*          家谱管理系统            \*\*" << endl;
4. cout << "======================================" << endl;
5. cout << "\*\*        请选择要执行的操作：      \*\*" << endl;
6. cout << "\*\*          A---完善家庭            \*\*" << endl;
7. cout << "\*\*          B---添加家庭成员        \*\*" << endl;
8. cout << "\*\*          C---解散局部家庭        \*\*" << endl;
9. cout << "\*\*          D---更改家庭成员姓名    \*\*" << endl;
10. cout << "\*\*          E---退出程序            \*\*" << endl;
11. cout << "======================================" << endl;
12. cout << "首先建立一个家谱！" << endl;
13. cout << "请输入祖先的姓名：";
14. string name; cin >> name;
15. cout << "此家谱的祖先是：" << name << endl;
16. cout << endl;
17. ancestor = **new** FamilyMember(name);
18. **while** (SelectOperation())
19. {
20. PrintTree();
21. cout << endl;
22. }
23. }

选择操作码，若操作码合法，则调用相应函数执行相应操作；若操作码不合法，则函数返回值为0，跳出上面的循环，结束程序。

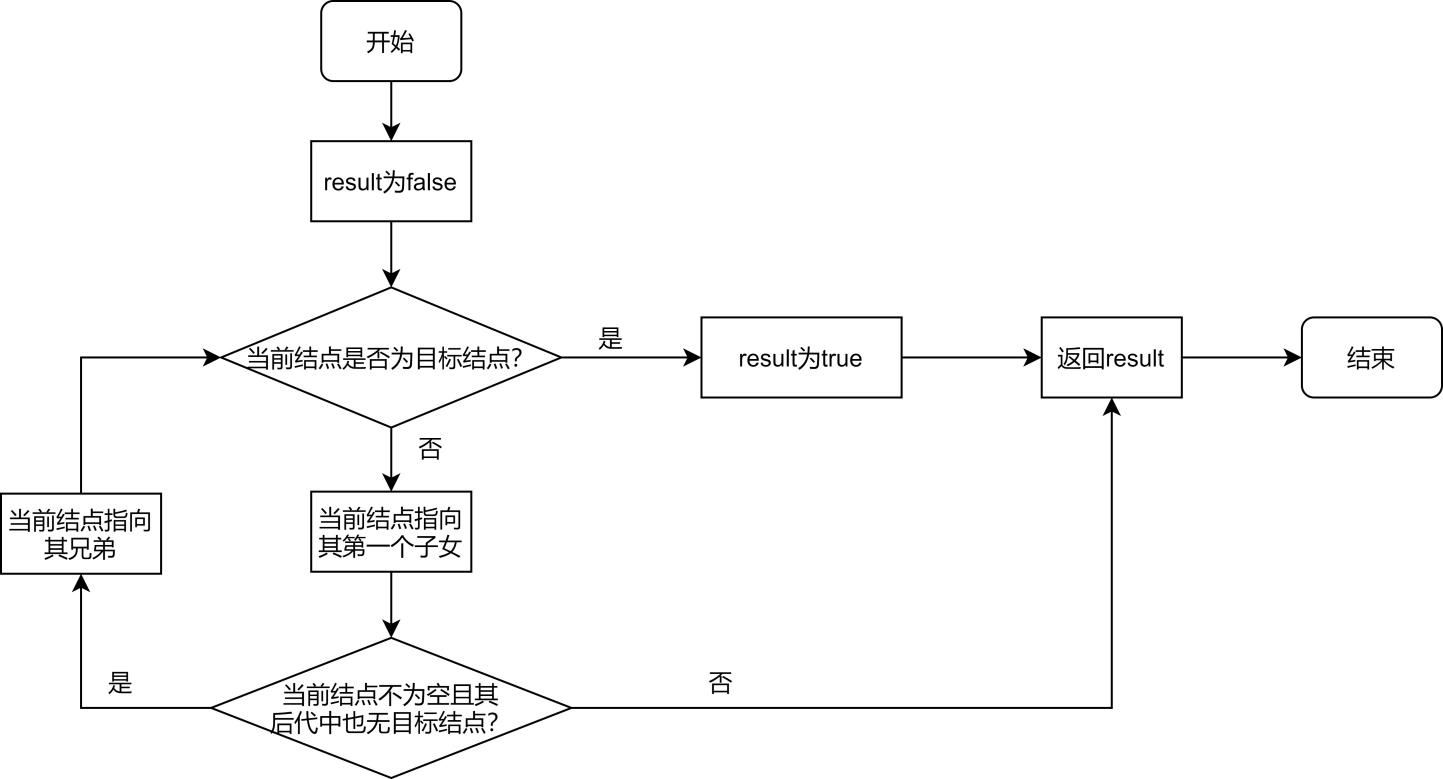
1. **bool** FamilyTree::SelectOperation()
2. {
3. **char** option;
4. cout << "请选择要执行的操作：";
5. cin >> option;
6. **switch** (option)
7. {
8. **case**'A':CreateFamily(); **return** 1;
9. **case**'B':AddMember(); **return** 1;
10. **case**'C':RemovepartFamily(); **return** 1;
11. **case**'D':Altername(); **return** 1;
12. **case**'E':**return** 0;
13. }
14. **return** 0;
15. }

**3.1.3初始化及函数选择模块功能截屏示例**



**3.2 搜索函数功能**的实现

**3.2.**1搜索函数功能流程图

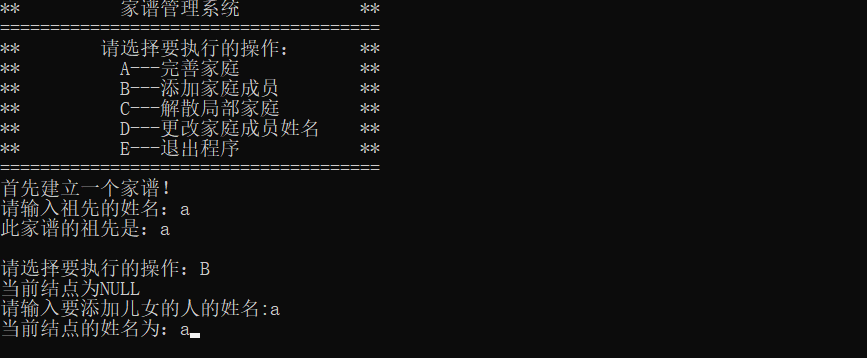


**3.2.2搜索函数功能核心代码**

通过递归实现搜索。一开始先令result为false，若当前结点为目标结点，即所要找的家族成员，就令result为true，并记录下结点的位置；若当前结点不为目标结点，就让当前结点指向其第一个子女，然后判断当前结点是否不为空且其后代也没有目标结点（通过递归返回值判断），若是，则当前结点指向其兄弟，若不是，不做任何操作。最后返回result。

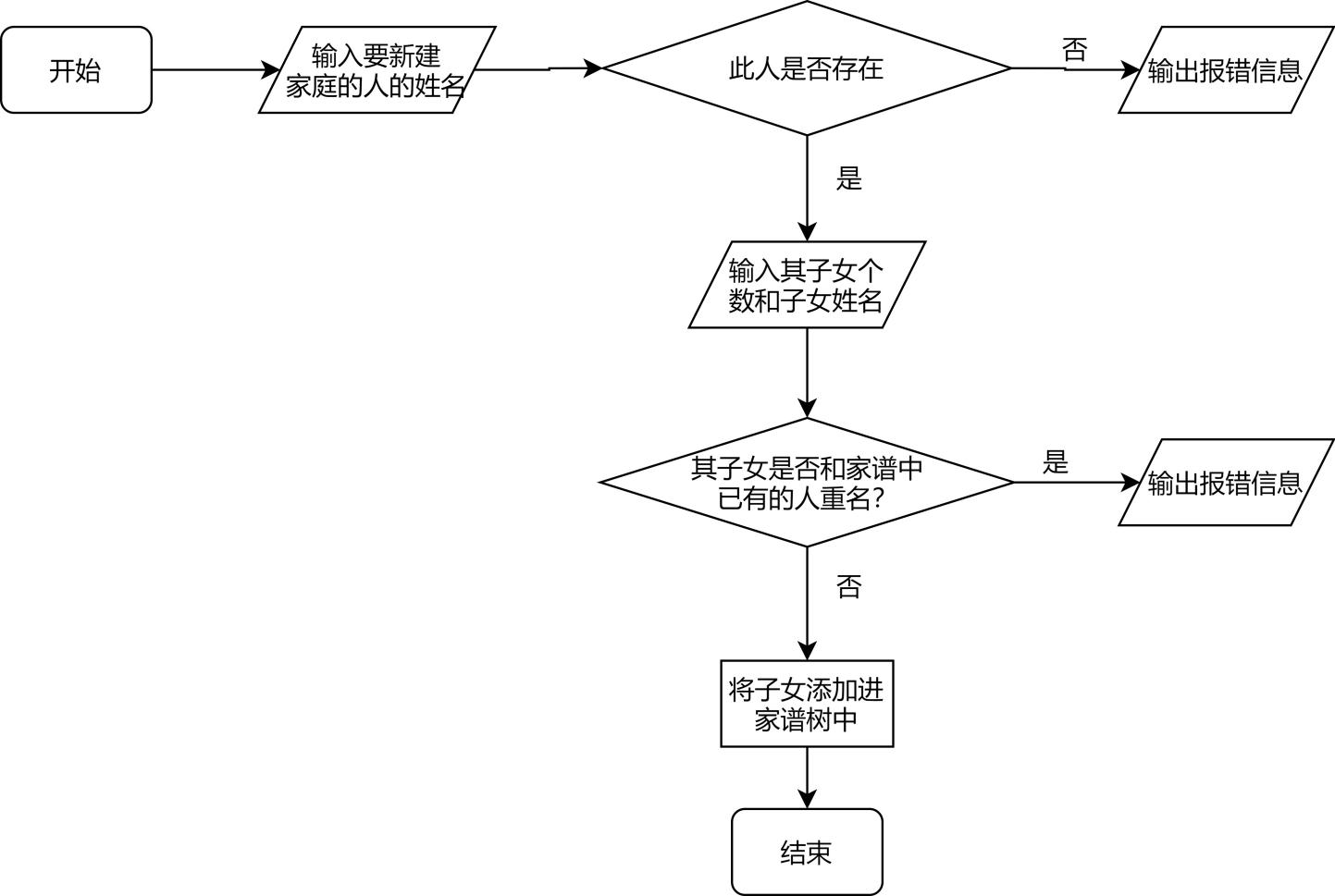
1. **bool** FamilyTree::Find(string vname,FamilyMember\* start,FamilyMember\*&target)
2. {
3. **bool** result = **false**;
4. **if** (start->name == vname)
5. {
6. result = **true**; target = start;
7. }
8. **else**
9. {
10. FamilyMember\* temp = start->child;
11. **while** (temp != NULL && !(result = Find(vname, temp,target)))
12. {
13. temp = temp->brother;
14. }
15. }
16. **return** result;
17. }

**3.2.3搜索函数功能截屏示例**



**3.3 完善家庭功能的实现**

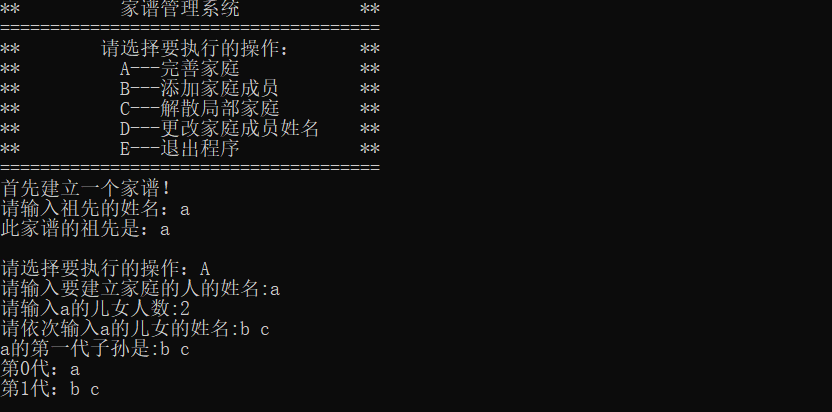
**3.3.1完善家庭功能流程图**

****

**3.3.2完善家庭功能核心代码**

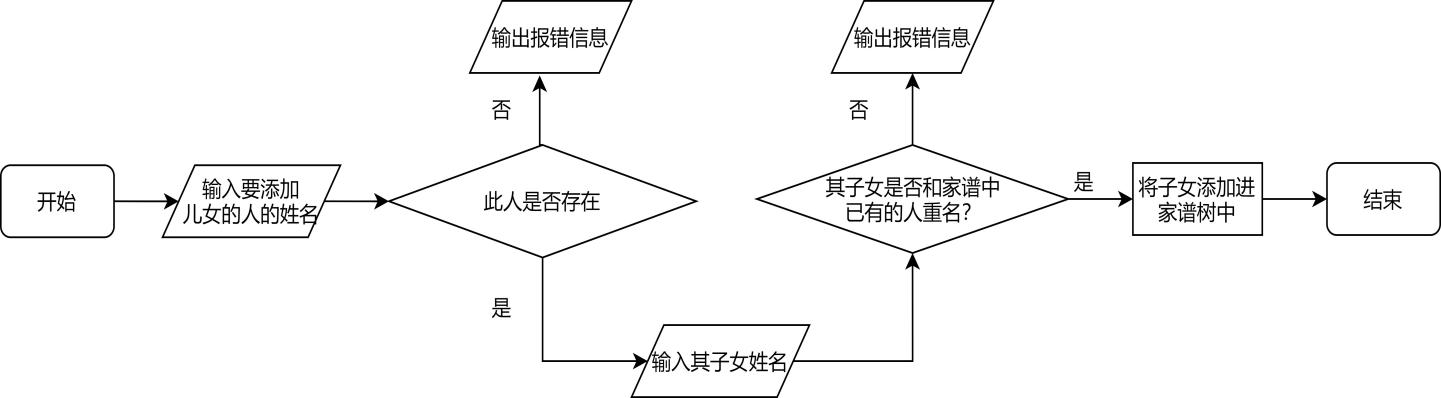
1. //完善家庭
2. **void** FamilyTree::CreateFamily()
3. {
4. **try**
5. {
6. string name; **int** childnum; string childname;
7. FamilyMember\* temp1 = NULL; FamilyMember\* temp2 = NULL;//变量
8. FamilyMember\* current = NULL;//锁定修改目标位置
9. FamilyMember\* check = NULL;//检测人名
10. cout << "请输入要建立家庭的人的姓名:";
11. cin >> name;
12. //遍历树找到要新建家庭的人
13. **if** (Find(name, ancestor,current) == 0)
14. {
15. **throw** exception("查无此人,请重新操作");
16. };
17. temp2 = current;
18. cout << "请输入" << name << "的儿女人数:";
19. cin >> childnum;
20. cout << "请依次输入" << name << "的儿女的姓名:";
21. **for** (**int** i = 0; i < childnum; i++)
22. {
23. cin >> childname;
24. **if** (Find(childname, ancestor, check) == 1)
25. {
26. **throw** exception("该人名已存在,请重新操作");
27. };
28. **if** (i == 0)
29. {
30. temp2->child = temp1;
31. temp2 = temp2->child;
32. }
33. **else**
34. {
35. temp2->brother = temp1;
36. temp2 = temp2->brother;
37. }
38. temp1 = **new** FamilyMember(childname);
39. temp1->parent = current;
40. }
41. cout << name << "的第一代子孙是:";
42. temp2 = current->child;
43. **while** (temp2 != NULL)
44. {
45. cout << temp2->name << " ";
46. temp2 = temp2->brother;
47. }
48. cout << endl;
49. }
50. **catch** (exception&error) { cout << error.what() << endl; }
51. }

**3.3.3完善家庭功能截屏示例**



**3.4 添加家族成员功能的实现**

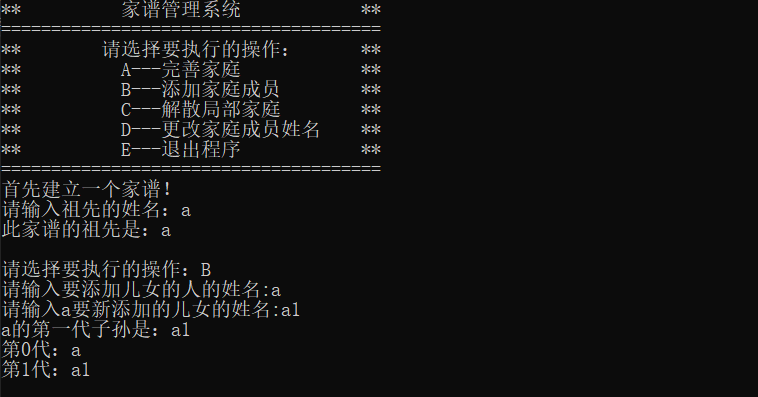
**3.4.1添加家族成员功能流程图**

****

**3.4.2添加家族成员功能核心代码**

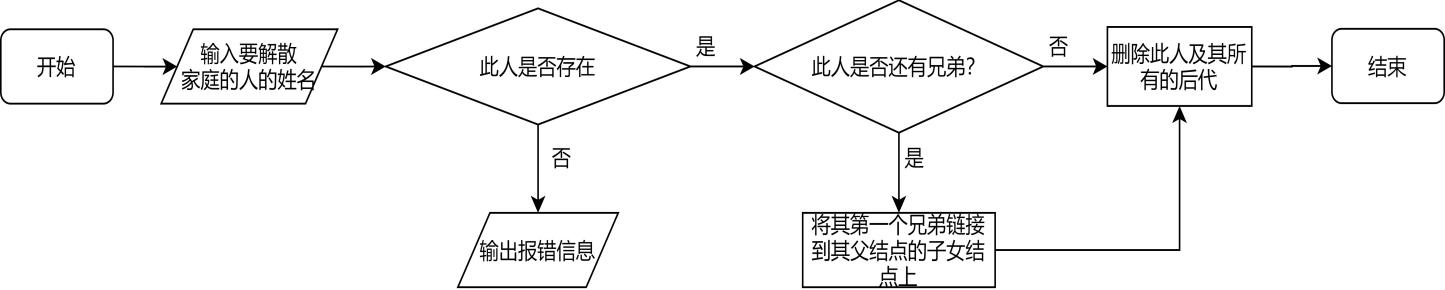
1. **void** FamilyTree::AddMember()
2. {
3. **try**
4. {
5. string name, childname;
6. FamilyMember\* current = NULL;
7. FamilyMember\* check = NULL;
8. cout << "请输入要添加儿女的人的姓名:";
9. cin >> name;
10. **if** (Find(name, ancestor,current) == 0)
11. {
12. **throw** exception("查无此人,请重新操作");
13. };
14. cout << "请输入" << name << "要新添加的儿女的姓名:";
15. cin >> childname;
16. **if** (Find(childname, ancestor, check) == 1)
17. {
18. **throw** exception("该人名已存在,请重新操作");
19. };
20. FamilyMember\* temp = **new** FamilyMember(childname);
21. temp->parent = current;
22. **if** (current->child != NULL)
23. {
24. current = current->child;
25. **while** (current->brother != NULL)
26. {
27. current = current->brother;
28. }
29. current->brother = temp;
30. }
31. **else**
32. {
33. current->child = temp;
34. }
35. cout << name << "的第一代子孙是：";
36. temp = temp->parent->child;
37. **while** (temp != NULL)
38. {
39. cout << temp->name << " ";
40. temp = temp->brother;
41. }
42. cout << endl;
43. }
44. **catch** (exception& error) { cout << error.what() << endl; }
45. }

**3.4.3添加家族成员功能截屏示例**



**3.5 解散局部家庭功能的实现**

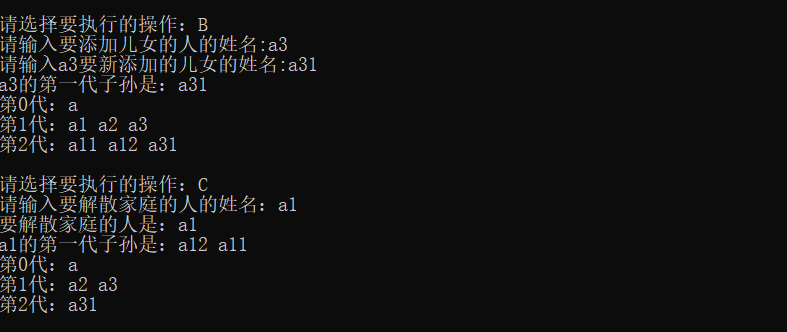
**3.5.1解散局部家庭功能流程图**

****

**3.5.2解散局部家庭功能核心代码**

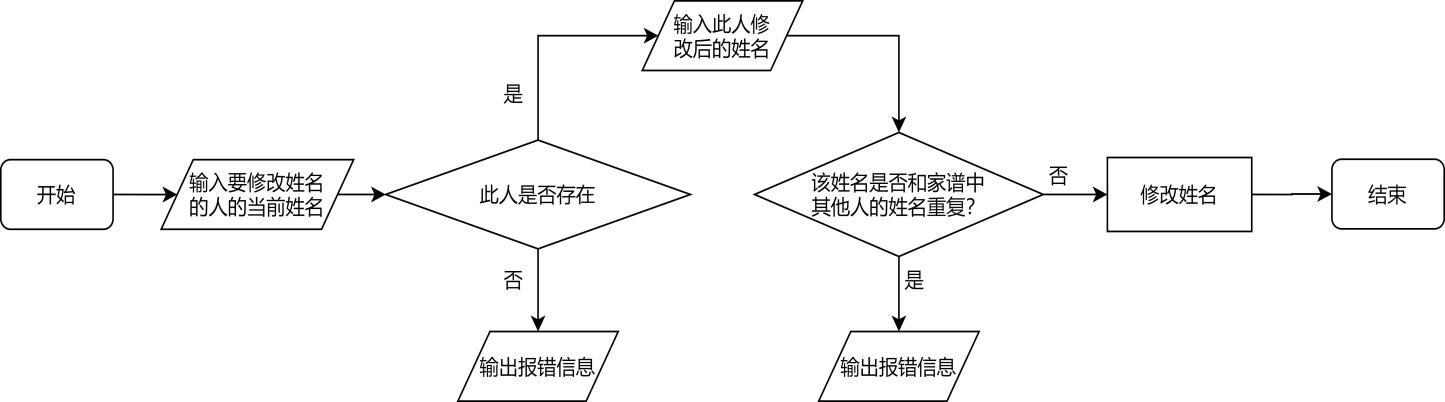
1. **void** FamilyTree::RemovepartFamily()
2. {
3. **try**
4. {
5. string name;
6. FamilyMember\* current = NULL;
7. cout << "请输入要解散家庭的人的姓名：";
8. cin >> name;
9. **if** (Find(name, ancestor,current) == 0)
10. {
11. **throw** exception("查无此人,请重新操作");
12. };
13. FamilyMember\* temp = current;
14. **if** (current->brother != NULL)
15. {
16. current->parent->child = current->brother;
17. }
18. cout << "要解散家庭的人是：" << name << endl;
19. cout << name << "的第一代子孙是：";
20. FamilyMember\* temp2;
21. FamilyMember\* temp3;
22. temp3 = temp;
23. temp = temp->child;
24. **delete** temp3;
25. **while** (temp != NULL)
26. {
27. **if** (temp->brother != NULL)
28. {
29. temp2 = temp->brother;
30. **while** (temp2 != NULL)
31. {
32. temp3 = temp2;
33. temp2 = temp2->brother;
34. cout << temp3->name << " ";
35. **delete** temp3;
36. }
37. }
38. temp3 = temp;
39. temp = temp->child;
40. cout << temp3->name << " ";
41. **delete** temp3;
42. }
43. cout << endl;
44. }
45. **catch** (exception& error) { cout << error.what() << endl; }
46. }

**3.5.3解散局部家庭功能截屏示例**



**3.6 修改家庭成员姓名功能的实现**

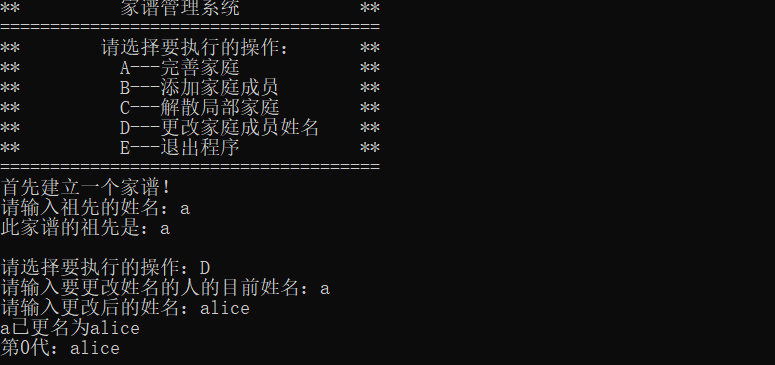
**3.6.1修改家庭成员姓名功能流程图**

****

**3.6.2修改家庭成员姓名功能核心代码**

1. **void** FamilyTree::Altername()
2. {
3. **try**
4. {
5. string name1, name2;
6. FamilyMember\* current = NULL;
7. FamilyMember\* check = NULL;
8. cout << "请输入要更改姓名的人的目前姓名：";
9. cin >> name1;
10. **if** (Find(name1, ancestor,current) == 0)
11. {
12. **throw** exception("查无此人,请重新操作");
13. };
14. cout << "请输入更改后的姓名：";
15. cin >> name2;
16. **if** (Find(name2, ancestor, check) == 1)
17. {
18. **throw** exception("该人名已存在,请重新操作");
19. };
20. current->name = name2;
21. cout << name1 << "已更名为" << name2;
22. cout << endl;
23. }
24. **catch** (exception& error) { cout << error.what() << endl; }
25. }

**3.6.3修改家庭成员姓名功能截屏示例**



**4 测试**

**4.1功能测试**

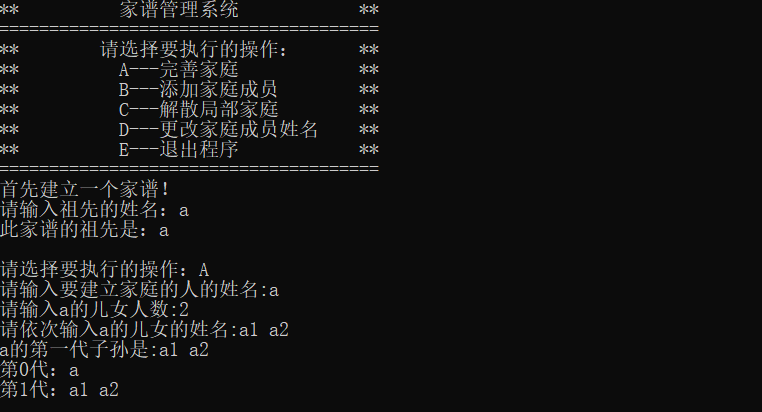
**4.1.1完善家庭功能测试**

**测试用例：**给a添加子女a1 a2

**预期结果：**a

a1 a2

**实验结果：**



**4.1.2添加家庭成员功能测试**

**测试用例：**a

a1 a2

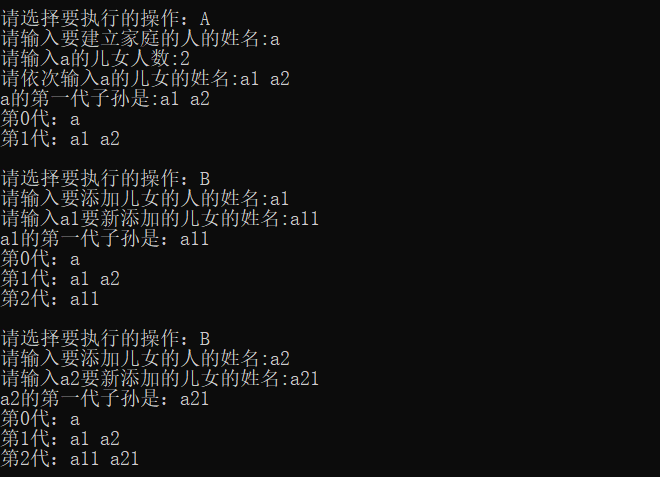
给a1添加子女a11，给a2添加子女a21

**预期结果：**a

a1 a2

a11 a21

**实验结果：**



**4.1.3解散局部家庭功能测试**

**测试用例：**a

a1 a2

a11 a21

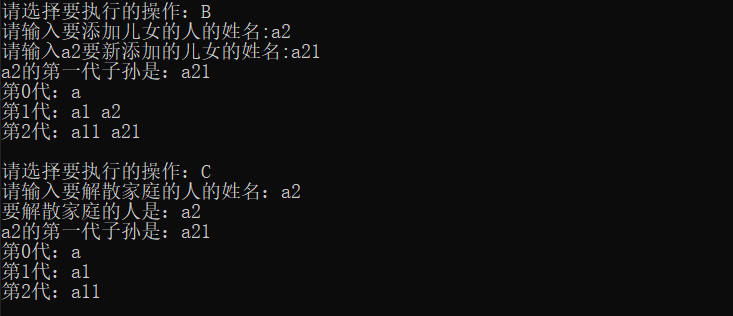
删除a2家庭

**预期结果：**a

a1

a11

**实验结果：**



**4.1.4修改家庭成员姓名功能测试**

**测试用例：**a

a1

a11

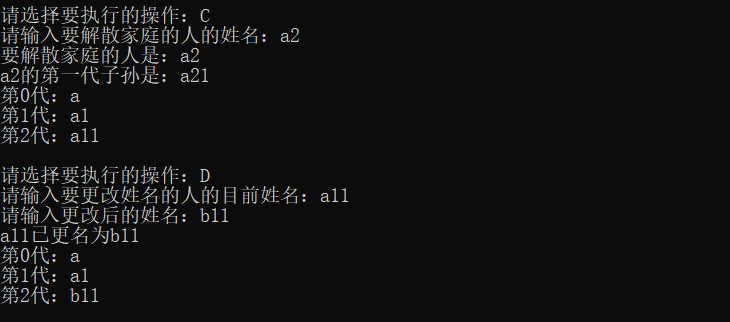
把a11修改成b11

**预期结果：**a

a1

b11

**实验结果：**



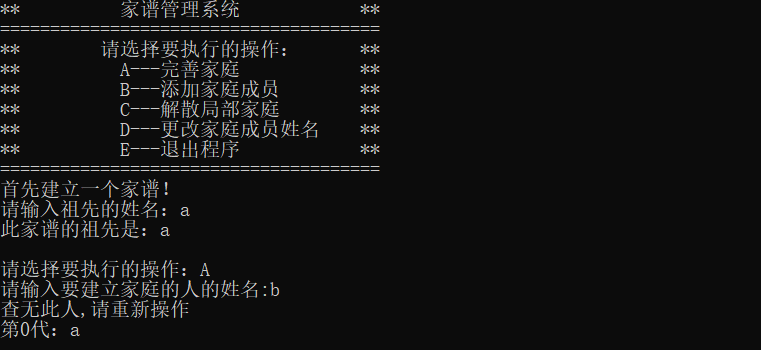
**4.2报错测试**

**4.2.1 父母结点不存在**

**测试用例：**a 查找b

**预期结果：**查无此人，请重新操作

**实验结果：**



**4.2.2 人名重复**

**测试用例：**a

a1 a2

把a2修改成a1

**预期结果：**该人名已存在，请重新操作

**实验结果：**

