项目说明文档

数据结构课程设计

——家谱管理系统

同济大学

Tongji University

目 录

[1 分析 1](#_Toc60273706)

[1.1 背景分析 1](#_Toc60273707)

[1.2 功能分析 1](#_Toc60273708)

[2 设计 1](#_Toc60273709)

[2.1 数据结构设计 1](#_Toc60273710)

[2.2 类结构设计 2](#_Toc60273711)

[2.3 成员与操作设计 2](#_Toc60273712)

[2.4 系统设计 5](#_Toc60273713)

[3 实现 6](#_Toc60273714)

[3.1 建立家谱功能的实现 6](#_Toc60273715)

[3.1.1 建立家谱功能流程图 6](#_Toc60273716)

[3.1.2 建立家谱功能核心代码 8](#_Toc60273717)

[3.1.3建立家谱功能截屏示例 9](#_Toc60273718)

[3.2 插入功能的实现 10](#_Toc60273719)

[3.2.1 插入功能流程图 10](#_Toc60273720)

[3.2.2 插入功能核心代码 12](#_Toc60273721)

[3.2.3 插入功能截屏示例 13](#_Toc60273722)

[3.3 删除功能的实现 14](#_Toc60273723)

[3.3.1 删除功能流程图 14](#_Toc60273724)

[3.3.2 删除功能核心代码 15](#_Toc60273725)

[3.3.3 删除功能截图示例 16](#_Toc60273726)

[3.4 修改成员姓名的实现 18](#_Toc60273727)

[3.4.1 修改成员姓名流程图 18](#_Toc60273728)

[3.4.2 修改姓名功能核心代码 20](#_Toc60273729)

[3.4.3 修改功能截屏示例 20](#_Toc60273730)

[3.5 查询成员的实现 21](#_Toc60273731)

[3.5.1 查询成员流程图 21](#_Toc60273732)

[3.5.2 查询功能核心代码 23](#_Toc60273733)

[3.5.3 查询功能截屏示例 23](#_Toc60273734)

[3.6 打印功能的实现 24](#_Toc60273735)

[3.6.1 打印功能流程图 24](#_Toc60273736)

[3.6.2 打印功能核心代码 26](#_Toc60273737)

[}3.6.3 打印部分截屏示例 26](#_Toc60273738)

[4 测试 28](#_Toc60273739)

[4.1 完善家谱测试 28](#_Toc60273740)

[4.1.1 完善家谱测试 28](#_Toc60273741)

[4.1.2 插入功能测试 30](#_Toc60273742)

[4.1.3 删除功能测试 32](#_Toc60273743)

[4.1.4 修改功能测试 33](#_Toc60273744)

[4.1.5 查询功能测试 36](#_Toc60273745)

[4.2 边界测试 37](#_Toc60273746)

[4.2.1 删除根节点 37](#_Toc60273747)

[4.2.2 在空表中修改成员姓名 38](#_Toc60273748)

[4.2.3 在已经有孩子的节点中完善家谱 39](#_Toc60273749)

[4.3 出错测试 40](#_Toc60273750)

[4.3.1 操作码错误 40](#_Toc60273751)

[4.3.2 输入成员存在错误 40](#_Toc60273752)

[4.3.3 添加成员时输入错误的数字 41](#_Toc60273753)

[4.3.4 添加成员时输入非法数字 42](#_Toc60273754)

# 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对家谱成员信息的建立，查找，插入，修改，删除等功能。

家谱结构里的信息，满足树结构——每一个parent对应着数量不定的child，但是同时不会有环，因为child不可能是其任意祖先的parent。所以应该用树结构来模拟实现家谱成员信息的建立，查找，插入，修改，删除等功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求实现家谱成员信息的建立，查找，插入，修改，删除等功能，而应该使用多叉树来进行模拟。同时，二叉树的建立，查找，插入等操作的实现较为容易，故本程序采取了孩子兄弟表示法来存储多叉树，将多叉树变成了一棵二叉树——每个节点的左孩子是家谱里这个人的兄弟姐妹，而右孩子是家谱里这个人的一个孩子。这样，就将多叉树转换成了二叉树，各种操作也便可以按照二叉树的方法实现了。

## 2.2 类结构设计

本程序主要使用的是CSTree类，里面包含着结构体CSNode的指针和家谱中元素的个数，CSNode里面包含着成员名字，兄弟，孩子等指针。

## 2.3 成员与操作设计

**孩子兄弟结构类（CSNode）**

struct CSNode

{

string name;

CSNode\* firstchild, \* lastchild;

CSNode\* nextsibling, \* parent;

};

**多叉树类（CSTree）**

**私有成员：**

CSNode\* root;

//存储根节点

int \_n;

//存储家谱中成员个数

//私有函数，不在类外调用

void print\_decsendents(CSNode\*)

//打印后代函数

void revise();

//修改家庭成员

CSNode\* search(CSNode\*,string);

//搜索姓名

void del(CSNode\*);

//删除节点

void deleteNodeFromSiblings(CSNode\*);

//从父母的孩子链表中删除节点

**公有操作：**

CSTree();

//构造函数，申请内存，给\_n赋初值

void build();

//完善家谱函数

void insert();

//添加家庭成员函数

void dismiss();

//解散局部成员函数

void query();

//询问家庭成员函数

void print();

//打印家谱函数

同时，为了能够通过层次遍历打印家谱，引入Queue类：

**Queue类：**

**私有成员：**

T\* e; //存储元素信息

int S; //当前元素个数

int MAXS; //最大能存储多少个元素

int \_front; //队头

int \_rear; //队尾

const int INIT\_SIZE = 100;

**ADT接口：**

Queue();

//构造函数，对私有成员进行赋值以及内存申请

Queue(Queue<T>& )

//赋值构造函数，由于涉及到变量的动态申请，所以不能使用默认构造函数

Queue<T>& operator=(Queue<T>& )

//复制构造函数，由于涉及到变量的动态申请，所以不能使用默认复制函数

bool empty()

//返回队列是否为空

int size()

//返回队列的大小

T front()

//返回队列的首元素，若队列为空，会报错

Status popQueue(T& );

//从队列中弹出元素，若队列为空，不会报错

T pop();

//从队列中弹出元素，若队列为空，会报错

void push(T);

//将元素插入到队列中

void printQueue();

//从队首到队尾打印队列

~Queue();

//析构函数

## 2.4 系统设计

系统首先调用printHead()函数实现对屏幕的初始化，完成对多叉树T的创建和输入数据工作，然后根据用户所输入的操作码（op）执行多叉树T对应的成员函数。

# 3 实现

## 3.1 建立家谱功能的实现

### 3.1.1 建立家谱功能流程图

开始

是否有祖先？

是否有孩子？

输入孩子数与孩子

尾插法插入p的孩子指针

输入要建立家庭的人的姓名

找到对应的指针p

结束

结束

是否存在？

y

n

y

y

n

n

创建祖先

### 3.1.2 建立家谱功能核心代码

void build()

{

string name;

if (!root)

{

...//根不存在的情况

}

printf("请输入要建立家庭的人的姓名：\t");

cin >> name;

CSNode\* parent = search(root, name);

//在家谱中用先序遍历递归查找目标节点

if (!parent) { cout << "在家谱里没有找到" << name << "!\n"; return; }

if (parent->firstchild) { cout << name << "已经有孩子了！\n"; return; }

int n;

cout << "请输入" << name << "的儿女人数（请输入正整数）：\t";

n = readInt(range);

\_n += n;

cout << "请输入" << name << "的儿女姓名：\n";

string child\_name;

//将n个孩子插入到家谱中

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

{

cin >> child\_name;

CSNode\* child = new CSNode{ child\_name,NULL,NULL,NULL,parent };

if (!parent->firstchild)parent->firstchild = parent->lastchild = child;

else parent->lastchild->nextsibling = child, parent->lastchild = child;

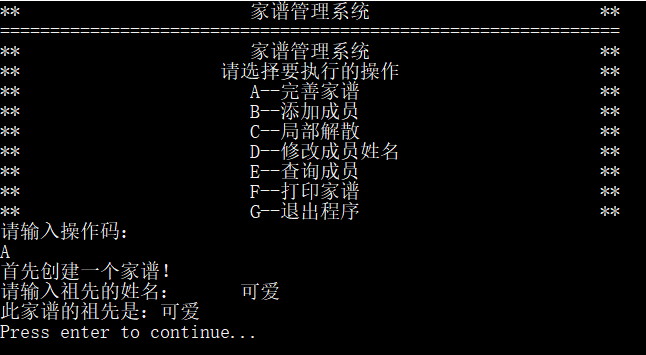
}

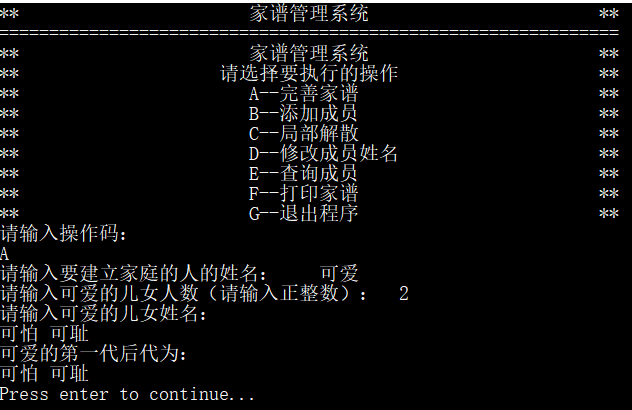
//打印后代

print\_decsendents(parent);

}

### 3.1.3建立家谱功能截屏示例





## 3.2 插入功能的实现

### 3.2.1 插入功能流程图

开始

是否有祖先？

输入孩子数与孩子

尾插法插入p的孩子指针

输入要建立家庭的人的姓名

找到对应的指针p

结束

结束

是否存在？

y

n

y

n

创建祖先

### 3.2.2 插入功能核心代码

void insert()

{

string name;

++\_n;

if (!root)

{

...//根不存在的情况

}

printf("请输入要添加后代的人的姓名：\t");

cin >> name;

//在家谱中用先序遍历递归查找目标节点

CSNode\* parent = search(root, name);

if (!parent) { cout << "在家谱里没有找到" << name << "!\n"; return; }

cout << "请输入" << name << "添加的后代的姓名：\n";

string child\_name;

cin >> child\_name;

CSNode\* child = new CSNode{ child\_name,NULL,NULL,NULL,parent };

if (!parent->firstchild)parent->firstchild = parent->lastchild = child;

else parent->lastchild->nextsibling = child, parent->lastchild = child;

//打印后代

print\_decsendents(parent);

}

//在家谱中用先序遍历递归查找目标节点

CSNode\* search(CSNode\* cur, string key)

{

if (!cur)return NULL;

if (cur->name == key)return cur;

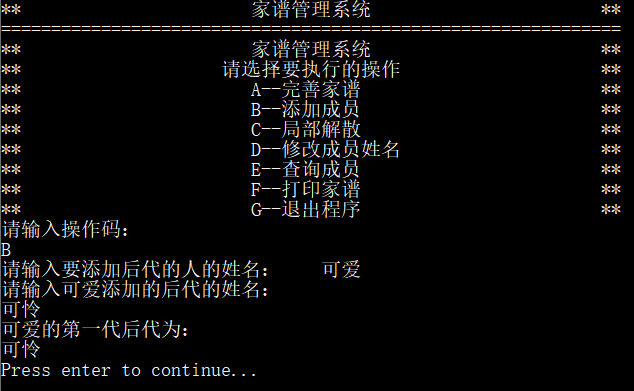
CSNode\* p = search(cur->firstchild, key);

if (p)return p;

return search(cur->nextsibling, key);

}

### 3.2.3 插入功能截屏示例



## 3.3 删除功能的实现

### 3.3.1 删除功能流程图

开始

是否有祖先？

输入要建立家庭的人的姓名

从其父亲的孩子链表里处删除它

结束

结束

是否存在？

y

n

y

n

n

后序遍历删除其后代

### 3.3.2 删除功能核心代码

void dismiss()

{

string name;

//家谱为空的情况

if (!root) { printf("家谱为空！\n"); return; }

printf("请输入要解散家庭的人的姓名：\t");

cin >> name;

//寻找目标节点

CSNode\* parent = search(root, name);

if (!parent) { cout << "在家谱里没有找到" << name << "!\n"; return; }

print\_decsendents(parent);

CSNode\* child = parent->firstchild;

//在其父辈节点中删除该节点

deleteNodeFromSiblings(parent);

//后序遍历删除其子嗣

del(child);

}

//在其父辈节点中删除该节点

void deleteNodeFromSiblings(CSNode\* p)

{

if (!p)return;

--\_n;

if (p == root) { delete p; root = NULL; return; }

CSNode\* parent = p->parent;

//p是第一个孩子的情况

if (parent->firstchild == p)

{

//p是其父母唯一的孩子的情况

if (parent->lastchild == p)parent->firstchild = parent->lastchild = NULL;

//p不是其父母唯一的孩子的情况

else parent->firstchild = p->nextsibling;

}

//p不是第一个孩子的情况

else

{

CSNode\* q;

for (q = parent->firstchild; q->nextsibling != p; q = q->nextsibling);

q->nextsibling = p->nextsibling;

}

delete p;

}

//后序遍历删除其子嗣

void del(CSNode\* cur)

{

if (!cur)return;

del(cur->nextsibling);

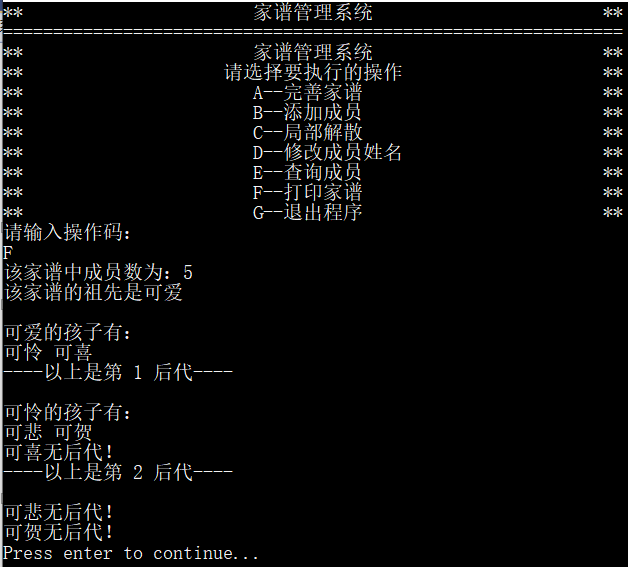
del(cur->firstchild);

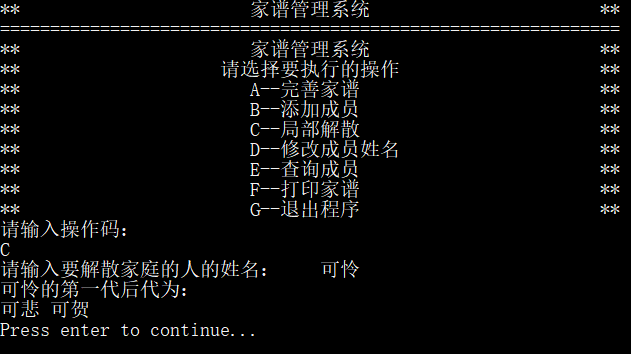
delete cur;

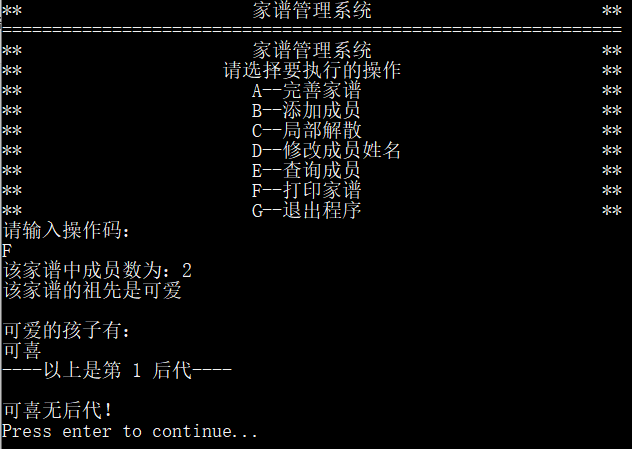
--\_n;

}

### 3.3.3 删除功能截图示例







## 3.4 修改成员姓名的实现

### 3.4.1 修改成员姓名流程图

开始

是否有祖先？

修改名字

输入要建立家庭的人的姓名

找到对应的指针p

结束

结束

是否存在？

y

n

y

n

### 3.4.2 修改姓名功能核心代码

void revise()

{

if (!root) { printf("家谱为空！\n"); return; }

string name;

printf("请输入要修改的人原先的姓名：\t");

cin >> name;

CSNode\* parent = search(root, name);

if (!parent) { cout << "在家谱里没有找到" << name << "!\n"; return; }

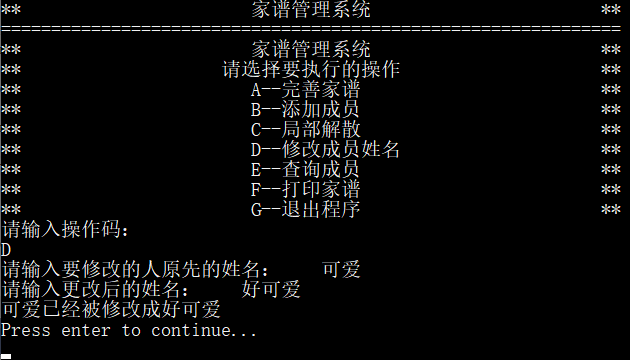
printf("请输入更改后的姓名：\t");

cin >> parent->name;

cout << name << "已经被修改成" << parent->name;

}

### 3.4.3 修改功能截屏示例



## 3.5 查询成员的实现

### 3.5.1 查询成员流程图

开始

是否有祖先？

输入要建立家庭的人的姓名

找到对应的指针p

结束

结束

是否存在？

y

n

y

n

输出其父母及后代

### 3.5.2 查询功能核心代码

void query()

{

string name;

if (!root) { printf("家谱为空！\n"); return; }

printf("请输入要查询的人的姓名：\t");

cin >> name;

CSNode\* parent = search(root, name);

if (!parent) { cout << "在家谱里没有找到" << name << "!\n"; return; }

if (parent->parent)cout << parent->parent->name << "为" << parent->name << "祖先\n";

else cout << parent->name << "无祖先\n";

if (parent->firstchild)

{

CSNode\* p;

cout << parent->name << "的第一代后代为：\n";

for (p = parent->firstchild; p; p = p->nextsibling)cout << p->name << ' ';

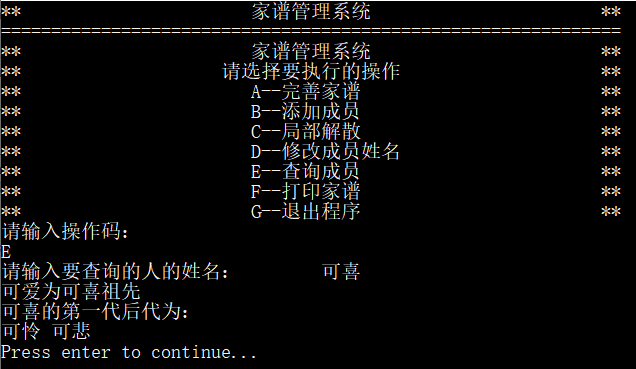
cout.put('\n');

}

else cout << parent->name << "无后辈！\n";

}

### 3.5.3 查询功能截屏示例



## 3.6 打印功能的实现

### 3.6.1 打印功能流程图

开始

是否有祖先？

将队首孩子依次输出，并且入队

取出队首，若队首无后嗣，则continue

结束

结束

队列为空？

y

n

n

y

将根节点入队列

### 3.6.2 打印功能核心代码

void print()

{

if (!root) { printf("家谱为空！\n"); return; }

printf("该家谱中成员数为：%d\n", \_n);

cout << "该家谱的祖先是" << root->name << endl << endl;

Queue<CSNode\*>Q;

Q.push(root);

int gene = 1, num[2];

num[0] = 1, num[1] = 0;

bool flag = 0;

while (Q.size())

{

CSNode\* p = Q.pop();

if (!p->firstchild) { cout << p->name << "无后代！\n"; }

else

{

cout << p->name << "的孩子有：\n";

for (CSNode\* q = p->firstchild; q; q = q->nextsibling)

{ cout << q->name << ' '; Q.push(q); ++num[gene % 2]; }

cout.put('\n');

flag = 1;

}

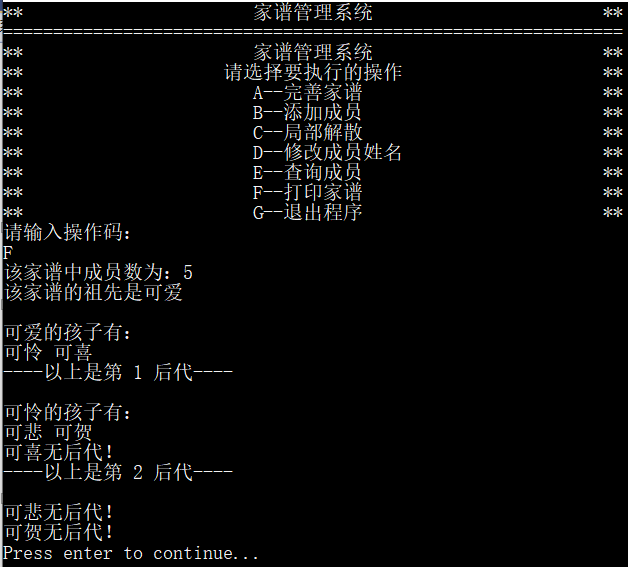
//用动态规划思想统计是第几后代

if (--num[(gene - 1) % 2] == 0 && num[gene % 2]&&flag)

{ cout << "----以上是第 " << gene++ << " 后代----\n\n"; flag = 0; }

}

### }3.6.3 打印部分截屏示例



# 4 测试

## 4.1 完善家谱测试

### 4.1.1 完善家谱测试

**测试用例**：

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

A h 2 l m

F

**预期结果**：

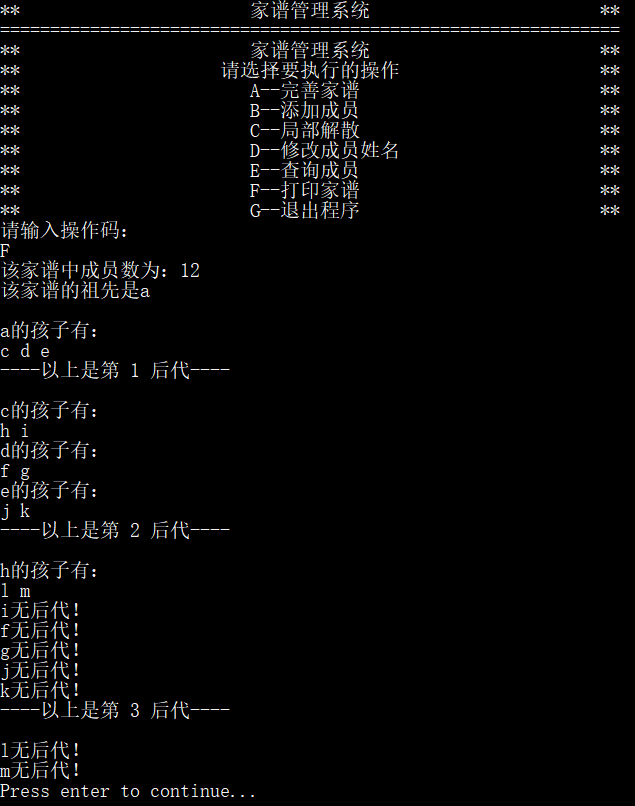
[a]

[c,d,e]

[h,i] [f,g] [j,k]

[l,m] [] [] [] [] []

**实验结果**



### 4.1.2 插入功能测试

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

A h 2 l m

B a n

B j o

B n p

F

**预期结果：**

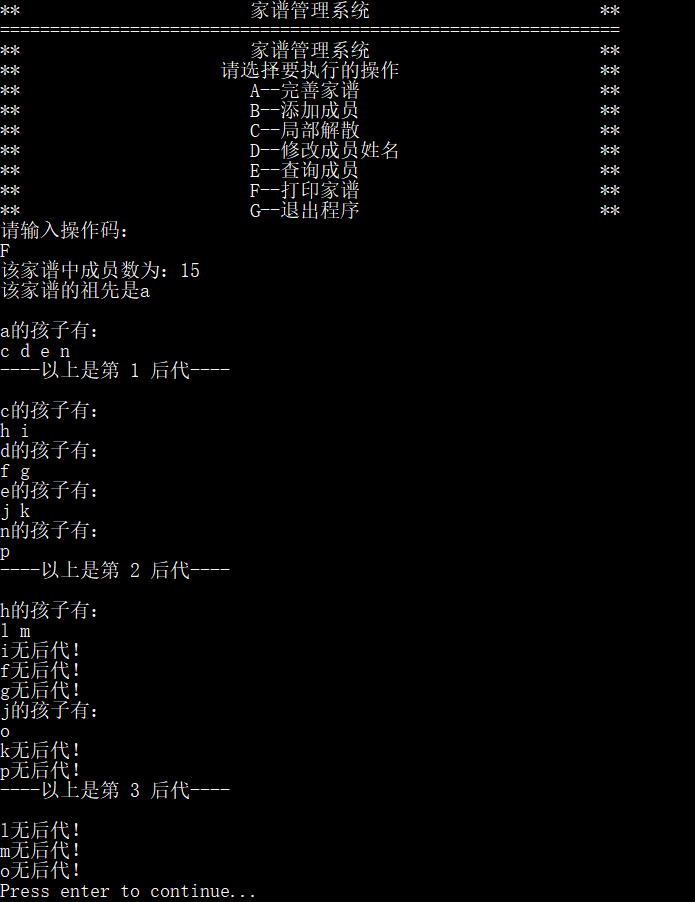
[a]

[c,d,e,n]

[h,i] [f,g] [j,k] [p]

[l,m] [] [] [] [o] [] []

**实验结果：**



### 4.1.3 删除功能测试

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

A h 2 l m

B a n

B j o

B n p

C c

F

**预期结果：**

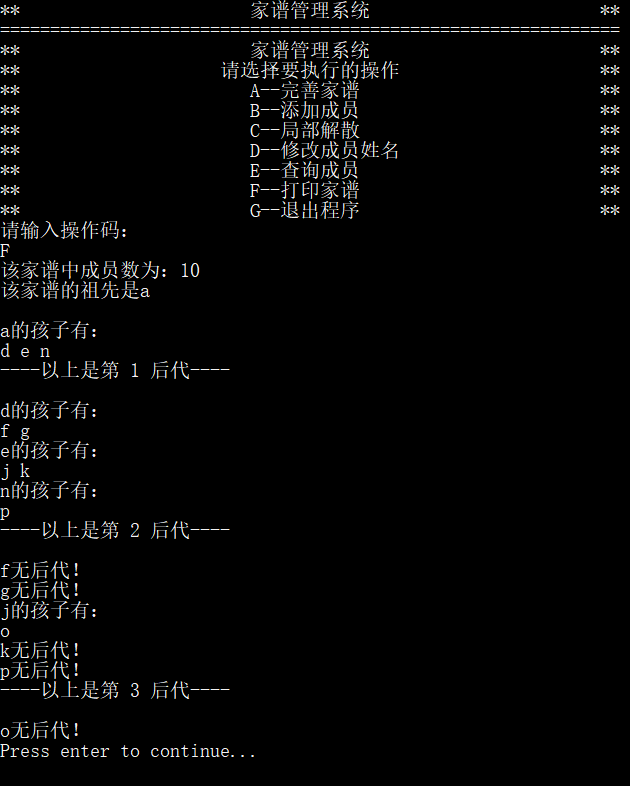
[a]

[d,e,n]

[f,g] [j,k] [p]

[] [] [o] [] []

**实验结果：**



### 4.1.4 修改功能测试

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

A h 2 l m

B a n

B j o

B n p

D a A

F

**预期结果：**

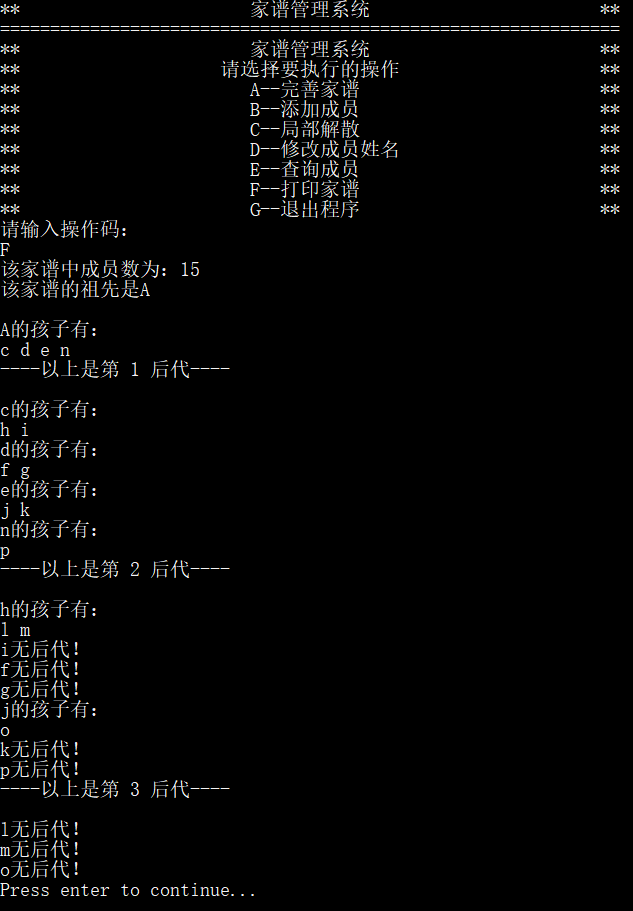
[A]

[c,d,e,n]

[h,i] [f,g] [j,k] [p]

[l,m] [] [] [] [o] [] []

**实验结果：**



### 4.1.5 查询功能测试

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

A h 2 l m

B a n

B j o

B n p

E h

E f

**预期结果：**

c为h祖先

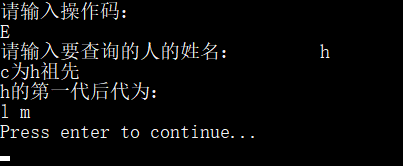
h的第一代后代为：

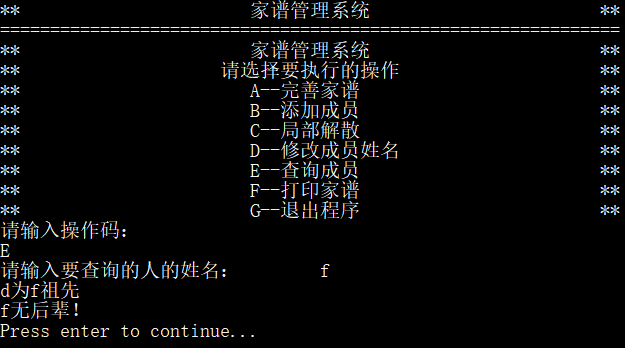
l m

d为f祖先

f无后辈

**实验结果：**

****

****

## 4.2 边界测试

### 4.2.1 删除根节点

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

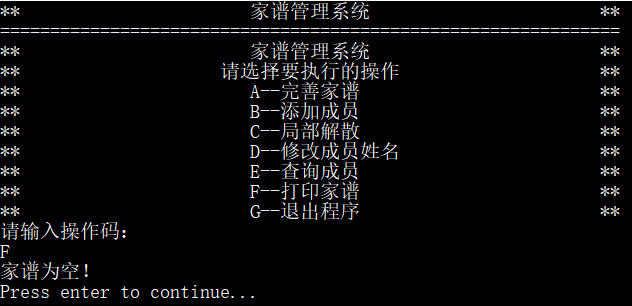
A h 2 l m

C a

F

**预期结果：**将根节点删除

**实验结果：**



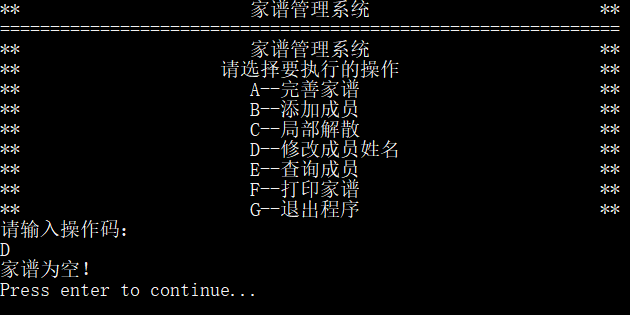
### 4.2.2 在空表中修改成员姓名

**测试用例：**

D

**预期结果：**程序给出提示信息。

**实验结果：**



### 4.2.3 在已经有孩子的节点中完善家谱

**测试用例：**

A a

A a 3 c d e

A d 2 f g

A c 2 h i

A e 2 j k

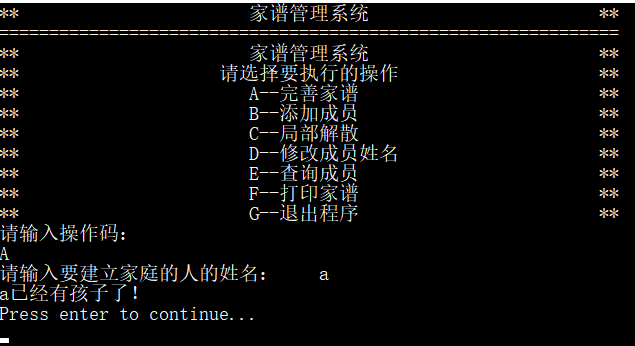
A h 2 l m

A a

**预期结果：**

给出提示信息

**实验结果：**



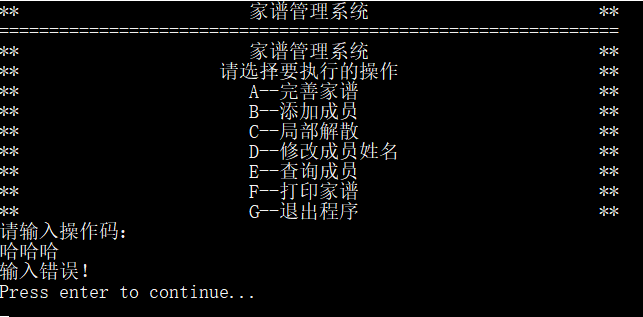
## 4.3 出错测试

### 4.3.1 操作码错误

**测试用例：**哈哈哈

**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 输入成员存在错误

**测试用例：**

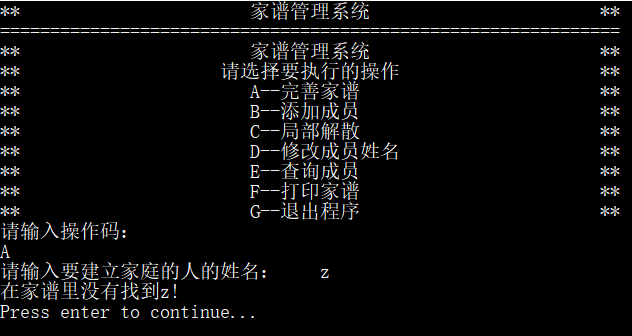
A a

A a 3 c d e

A z

**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：**

****

### 4.3.3 添加成员时输入错误的数字

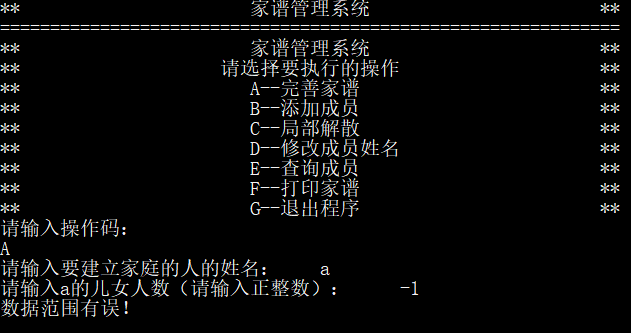
**测试用例：**

A a

A -1

**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：**



### 4.3.4 添加成员时输入非法数字

**测试用例：**

A a

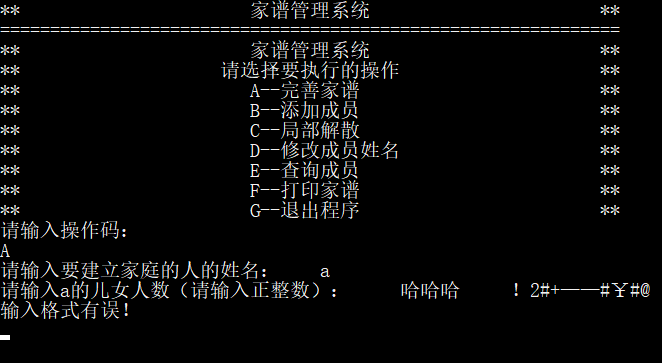
A

a

哈哈哈 ！2#+——#￥#@

**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：**

****