项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名： 曹晓慈

学 号： 2052844

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

## 分析

### 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。

### 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。

## 二．设计

### 1.数据结构设计

考虑到家谱成员信息的建立，查找，插入，修改，删除等功能，本题目中家谱是一个比较明显的树形结构，因此我们可以用树的数据结构进行功能的实现，其中可以树可以设计为左孩子右兄弟的二叉树形式，这样便于功能的实现。

### 2.类结构设计

二叉树包括两个抽象数据类型分别是树结点类（BiNode）与树（BinaryTree）类，，而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

### 成员与操作设计

template<typename T>

class BiNode

{

public:

T data; //结点存储的数据，该题目为名字

BiNode\* lchild, \* rchild; //lchild指向左孩子，rchild指向右兄弟

};

template<typename T>

class BinaryTree

{

private:

BiNode<T>\* root;

public:

BinaryTree() { root = NULL; } //构造函数

BiNode<T>\* Search(T name, BiNode<T> \*p); //查找存储数据为name的结点并返回

BiNode<T>\* Find\_Parent(BiNode<T>\* child,BiNode<T>\* p); //以p为根节点查找结点child

void CreatBinaryTree(T name); //创建二叉树

void Perfect(T name); //功能A的完善name的家谱

void AddMember(); //功能B的添加成员

void Delete\_Family(T name); //功能C的解散局部家庭

void Delete\_Child(BiNode<T>\* p); //删除p结点对应的所有孩子

void Change(); //功能D的修改姓名

void Show\_child(BiNode<T>\* p); //展示p结点对应的所有孩子

};

## 实现

### 查找二叉树中某成员

因为是树的数据结构，所以对于某成员的查找可以采用递归的思想，将当前指针指向结点存储的数据等于所要找的成员作为递归终止条件，在兄弟指针不断递归直至搜索完后再搜索他们的孩子，未找到则返回NULL。

BiNode<T>\* BinaryTree<T>::Search(T name, BiNode<T>\* p)

{

BiNode<T>\* ans = NULL, \* temp = p;

if (p->data == name) //恰好传入函数的家庭成员key即是所找的成员名字

return p;

while (temp->rchild) { //在兄弟指针中不断递归寻找

ans = Search(name, temp->rchild);

if (ans) //找到该成员

return ans;

temp = temp->rchild;//未找到则继续搜寻其兄弟指针直至都搜寻完后搜寻孩子

}

temp = p;

if (temp->lchild)

ans = Search(name, temp->lchild);

return ans;

}

### 查找父母的功能实现

此处依然可以用递归的思想，找到一个根节点先对其左孩子进行判断，若不是则再循长子的兄弟链，递归在子树中查找。

if (p == NULL || child == NULL)

return NULL;

BiNode<T>\* q = p->lchild;

BiNode<T>\* s=NULL; //循长子的兄弟链，递归在子树中搜索

while (q != NULL && q != child) {

if (Find\_Parent(child, q) != NULL)

{

s = Find\_Parent(child, q);

return s; //找到双亲，返回

}

q = q->rchild;

}

if (q != NULL && q == child) return p; //找到双亲

else return NULL; //未找到

### 3.完善家谱功能实现

在得到家庭成员的姓名后，用查找函数在家庭树中查找该成员。在对其子指针进行判断，若不空则已建立家庭不进行操作。若为空则需要输入子女的个数和姓名，之后用循环添加新子女到家庭树中。

核心代码：

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

{

if (i == 0)

{

BiNode<T>\* lchild = p->lchild = new BiNode<T>;

cin >> tmp;

lchild->data = tmp;

q = lchild;

}

else

{

BiNode<T>\* rchild = q->rchild = new BiNode<T>;

cin >> tmp;

rchild->data = tmp;

q = rchild;

}

}

### 4.添加家庭成员功能实现

与完善家谱功能类似，得到家庭成员姓名后先在家庭树中找到该人，之后输入该成员的姓名，在家庭树中建立新的节点，方法与操作A相同。

核心代码：

cout << "请输入要添加儿子（或女儿）的人的姓名:";

cin >> parent;

BiNode<T>\* p = Search(parent, root);

BiNode<T>\* tmp = p;

cout << "请输入" << p->data << "新添加的儿子（或女儿）的姓名:";

cin >> child;

if (p->lchild == NULL)

{

BiNode<T>\* kid = p->lchild = new BiNode<T>;

kid->data = child;

}

else

{

p = p->lchild;

while (p->rchild != NULL)

p = p->rchild;

BiNode<T>\* kid = p->rchild = new BiNode<T>;

kid->data = child;

}

### 5.解散局部家庭功能实现

首先找到要删除的家庭成员的人，之后进行分情况讨论，要判断该人是否有兄弟指针，以及是否直接连接父结点，可以分为四种情况讨论。（黄色为要删除的结点）

核心代码：

BiNode<T>\* q = Find\_Parent(p, root);

if (q->lchild == p) //父节点的长子就为要删除的家庭

{

if (p->rchild != NULL) //长子有兄弟

{

q->lchild = p->rchild;

Delete\_Child(p); //删除其孩子

delete p;

}

else //长子无兄弟

{

q->lchild = NULL;

Delete\_Child(p); //删除其孩子

delete p;

}

cout << endl;

return;

}

else //要删除的家庭不为长子

{

q = q->lchild;

while (q->rchild != p)

q = q->rchild; //q指针的右指针指向要删除的家庭

cout << q->data;

if (p->rchild != NULL) //有弟弟

{

q->rchild = p->rchild; //重新进行结点的连接

Delete\_Child(p); //删除其孩子

delete p;

}

else //无弟弟

{

q->rchild = NULL;

Delete\_Child(p); //删除其孩子

delete p;

}

cout << endl;

return;

}

### 6.修改名字的实现

找到该家庭成员后更改其存储数据即可。

核心代码：

T name;

cin >> name;

BiNode<T> \*p = Search(name, root);

cout << "请输入更改后的姓名：";

cin >> p->data;

### 7.主要系统的实现

先输入家谱祖先之后对输入的选择进行判断，利用switch和while的循环使程序正确运行。

while (1)

{

cout << "请选择要执行的操作:";

cin >> choice;

if (choice == 'E')

break;

else

{

string name;

switch (choice)

{

case 'A':

cout << "请输入要建立家庭的人的姓名:";

cin >> name;

map.Perfect(name); //功能A完善家谱

break;

case 'B':

map.AddMember(); //功能B添加成员

break;

case 'C':

cout << "请输入要删除家庭的人的姓名:";

cin >> name;

map.Delete\_Family(name); //功能C解散局部家庭

break;

case 'D':

map.Change(); //功能D改变姓名

break;

default:

cout << "请输入正确的命令!" << endl;

break;

}

}

}

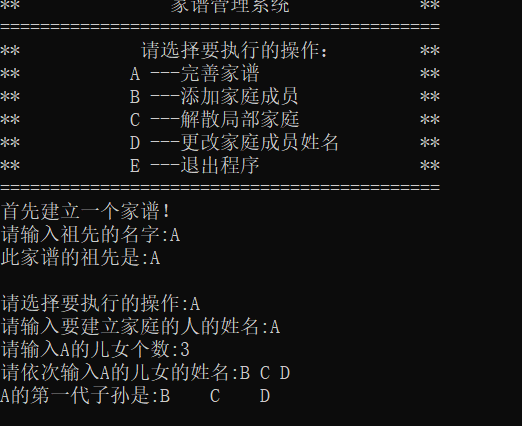
## 测试

### 功能A测试

**测试用例：3个子女分别为**B C D

**预期结果：**正确建立二叉树

**实验结果：**

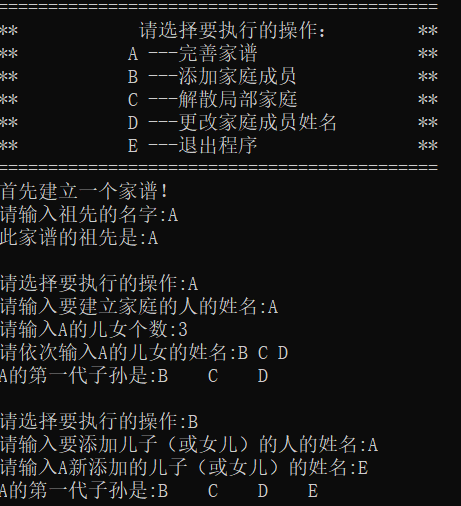


### 功能B的测试

**测试用例：3个子女分别为**B C D ，在添加一个E

**预期结果：**给出正确输出

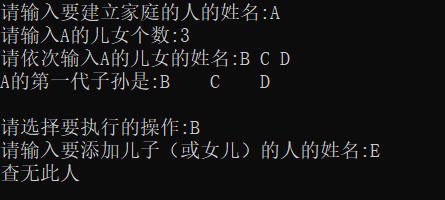
**实验结果：**



**测试用例：3个子女分别为**B C D ，添加的人的父母不存在

**预期结果：**给出错误提示信息

**实验结果：**

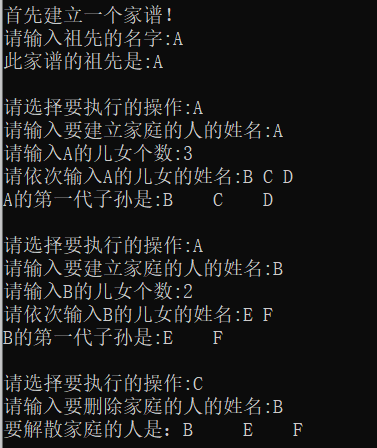


### 功能C的测试

**测试用例：3个子女分别为**B C D ，B再添加子女E F删除B所在家庭

**预期结果：**正确输出

**实验结果：**



### 功能D的测试

**测试用例：3个子女分别为**B C D ，将A改名为E

**预期结果：**正确输出

**实验结果：**

