

数据结构课程设计

——家谱管理系统

项目说明文档

作 者 姓 名： 韦世贸

学 号： 2351131

指 导 老 师： 软件学院

专 业： 软件工程

同济大学

Tongji University

1 介绍

1.1 背景介绍

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 功能介绍

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

本项目可实现家谱的完善、添加成员和解散局部家庭，当然也可以对家庭成员进行一些修改，因为家谱成员之间存在一对多的层次关系，且家谱从形状上为一棵多叉树，实现这样的功能该项目以左子女，右兄弟的二叉链表来模拟多叉树。除此之外，还定义了一个父指针指向该结点的双亲结点。

2.2 类结构设计

在该项目中创建一个树的节点类和一个家谱成员类，当然也有在进行遍历时需要用到的链式栈，将家谱成员类定义为节点类的友元类，从而建立起一个二叉链表，因为该系统数据类型确定，无使用抽象数据类型的必要故直接定义了类型。

当然，栈的使用仍为抽象数据类型，方便使用。

2.3 成员与操作设计

2.3.1树结点类

树结点类以多叉树作为友元，既保证了封装性，又能够使得多叉树直接访问树结点。

//树节点定义

template <class T>

class TreeNode

{

template <class T>

friend class MultiTree;

private:

T data;

TreeNode<T>\* firstChild; //子结点

TreeNode<T>\* nextSibling; //兄弟结点

TreeNode<T>\* parent; //双亲结点

TreeNode(T &value) : data(value), firstChild(nullptr), nextSibling(nullptr), parent(nullptr) {}

};

2.3.2多叉树类

private:

TreeNode<T>\* root;

public:

MultiTree() : root(NULL) {}

~MultiTree()

{ }

void DeleteSubtree(TreeNode<T>\* node

//建立一棵树

void Creat()

//对多叉树进行中序遍历查找

TreeNode<T>\* Lookingfor(T data, TreeNode<T>\* p)

void print\_Child(TreeNode<T>\* f)

//完善家谱

bool Add()

//添加家庭成员

bool Insert()

//修改家庭成员的名字

bool ChangeName()

//解散家庭

bool Disband()

3 实现

以下功能的实现都需要遍历整棵树查找目标节点，再进行操作，这里我们使用栈对树进行中序遍历，以下是查找目标节点的实现

TreeNode<T>\* Lookingfor(T data, TreeNode<T>\* p)

{

TreeNode<T>\* res = NULL; //返回结果

LinkedStack<TreeNode<T>\*> TreeStack; //用栈存储结点

while (p || !TreeStack.isEmpty()) //遍历条件

{

if (p) {

TreeStack.Push(p);

p = p->firstChild;

}

else {

TreeStack.Pop(p);

if(p->data==data)

{

res = p;

return res;

}

p = p->nextSibling;

}

}

return res;

}

3.1 创建家谱功能的实现

3.1.1 创建家谱功能流程简介

创建家谱功能是家谱中没有人时执行的操作，是最初始的操作，用于创建家谱的一个祖先。

3.1.2 创建功能核心代码

//建立一棵树

void Creat()

{

cout << "请输入祖先的姓名：";

T rootname;

cin >> rootname;

TreeNode<T>\* New = new TreeNode<T>(rootname);

if (New == NULL) {

cout << "存储分配错误！" << endl;

return;

}

root = New;

cout << "此家谱的祖先是：" << New->data << endl;

}

3.2 完善家谱功能的实现

3.2.1 完善家谱功能流程简介

完善家谱是建立树的核心过程，可以以树根为基础重复完善家谱创建整个树（家谱），通过循环插入子节点，并将第一个节点作为树根的子节点，其余的节点作为第一个节点的兄弟节点，这样就能合理的完善家谱。具体代码如下所示：

3.2.2 完善家谱功能核心代码

//完善家谱

bool Add()

{

T parent\_data;

cin >> parent\_data;

TreeNode<T>\* f = Lookingfor(parent\_data, root);

if (f != NULL) {

TreeNode<T>\* p = f;

int num;

cin >> num;

for(int i = 1; i <= num; i++)

{

T child\_data;

cin >> child\_data;

TreeNode<T>\* New = new TreeNode<T>(child\_data);

if (i == 1) { //第一个子女

p->firstChild = New; //连接到p的子女结点

New->parent = p; //更新p的位置

p = p->firstChild;

}

else {

p->nextSibling = New;

New->parent = p->parent;

p = p->nextSibling;

}

}

return true;

}

else {

return false;

}

}

3.2 添加家庭成员功能的实现

3.2.1 添加家庭成员功能流程简介

添加家庭成员做法与完善家谱相似，两者的区别在于添加家庭成员功能只能每次添加一位成员，而完善家谱可以自定义添加的节点数量。具体代码如下所示：

3.2.2 添加家庭成员功能核心代码

//添加家庭成员

bool Insert() //只添加一个子女结点

{

T parent\_data;

cin >> parent\_data;

TreeNode<T>\* res = Lookingfor(parent\_data, root);

if (res != NULL)

{

TreeNode<T>\* temp = res;

T child\_data;

cin >> child\_data;

TreeNode<T>\* New = new TreeNode<T>(child\_data);

if (res->firstChild == NULL) //双亲结点的子女结点为空，直接将插入结点作为父节点的子女结点

{

res->firstChild = New;

New->parent = res;

}

else //否则放置在子女结点的兄弟结点位置

{

res = res->firstChild;

while (res->nextSibling != NULL)

res = res->nextSibling;

res->nextSibling = New;

New->parent = res->parent; //记录双亲

}

return true;

}

else {

return false;

}

}

3.3 解散家庭成员功能的实现

3.3.1 解散家庭成员功能流程简介

解散家庭成员对应于添加部分家庭成员，将其子女都进行解散，只留下根节点，解散时不能只单个解散。所以这里使用递归对该节点的所有子女进行删除。

3.3.2 解散家庭成员功能核心代码

void DeleteSubtree(TreeNode<T>\* node)

{

if (node != NULL)

{

DeleteSubtree(node->firstChild);

DeleteSubtree(node->nextSibling);

delete node;

}

}

3.4 更改家庭成员姓名功能的实现

3.4.1 更改家庭成员姓名功能流程介绍

更改家庭成员姓名功能进行家庭成员姓名的修改，可根据需要做任意修改，前提是通过遍历多叉树查找到目标家庭成员。

3.4.2 更改家庭成员姓名功能核心代码

bool ChangeName() //更改某个人的姓名

{

T now\_data;

cin >> now\_data;

TreeNode<T>\* res = Lookingfor(now\_data, root);

if (res != NULL)

{

cout << "请输入更改后的姓名：";

T new\_data;

cin >> new\_data;

res->data = new\_data;

cout << now\_data << "已经改名为" << res->data << endl;

return true;

}

else {

cout << "查无此人，请重新输入！" << endl;

return false;

}

}

3.5 总体系统的实现

3.5.1 总体系统简介

总体系统是分别对以上的调用和实现，除此之外还要打印初始界面和实现选项功能。

3.5.2 总体系统的核心代码

while (sel != 'E') {

cout << endl << "请选择要执行的操作：";

cin >> sel;

switch (sel)

{

//因为这些成员函数为布尔函数，所以可以直接使用返回值进行错误处理

case'A':

while(!FamilyTree.Add());

break;

case'B':

while (!FamilyTree.Insert());

break;

case'C':

while (!FamilyTree.Disband());

break;

case'D':

while (!FamilyTree.ChangeName());

break;

case'E':

cout << "操作结束，已退出系统！" << endl;

break;

default:

cout << "请输入正确的操作！" << endl;

break;

}

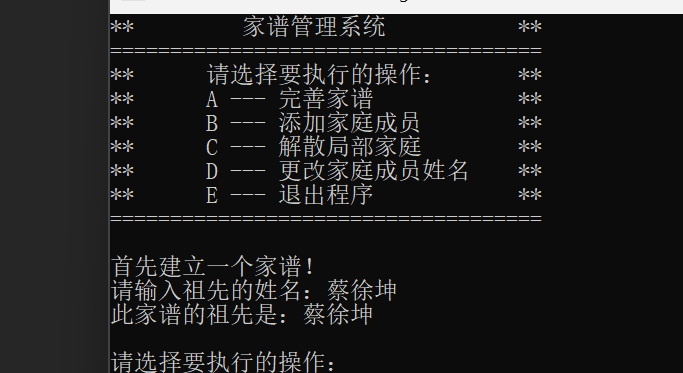
}

4 .功能检测

以下是对系统中A,B,C,D,E功能的检测

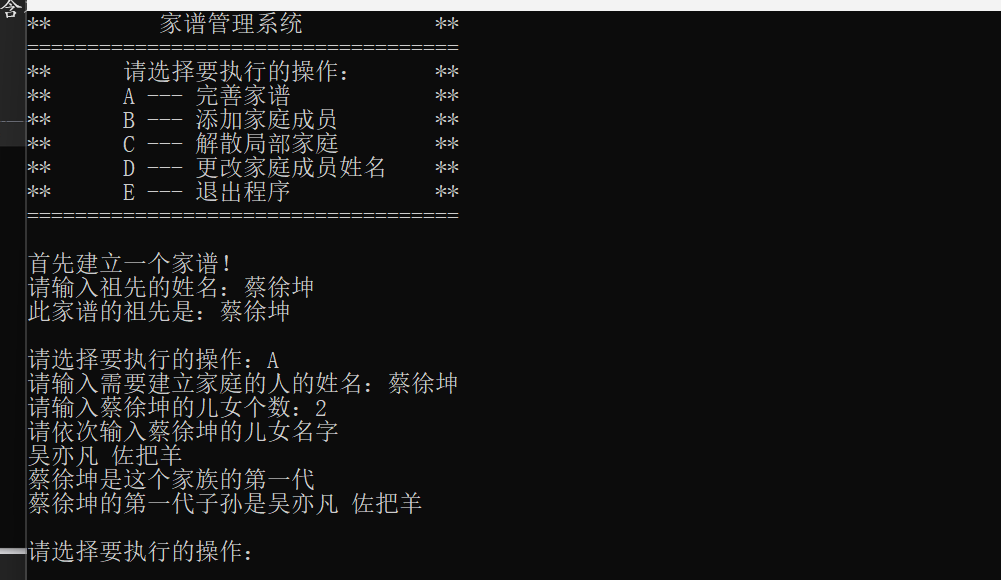
4.1创建家谱功能截屏示例

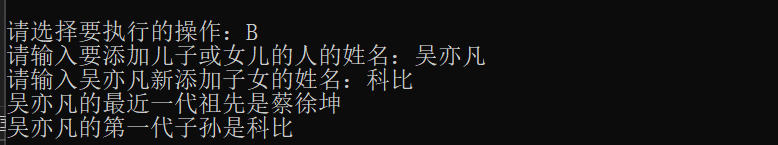
正常的插入功能展示，未涉及边界情况。创建家谱祖先展示。



4.2完善家谱和添加家庭成员功能截屏示例

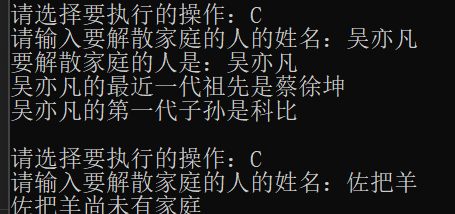
正常创建完善家谱和添加家庭成员的示例演示，其中也含有错误输入的示例，会进行错误操作进而达到提示目的。这两个功能都可以重复调用。



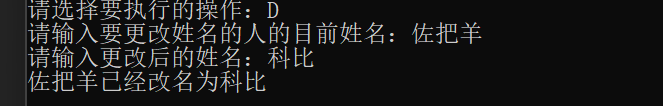


4.3解散家庭成员功能截图示例

解散家庭功能也可以进行多次，示例如下：

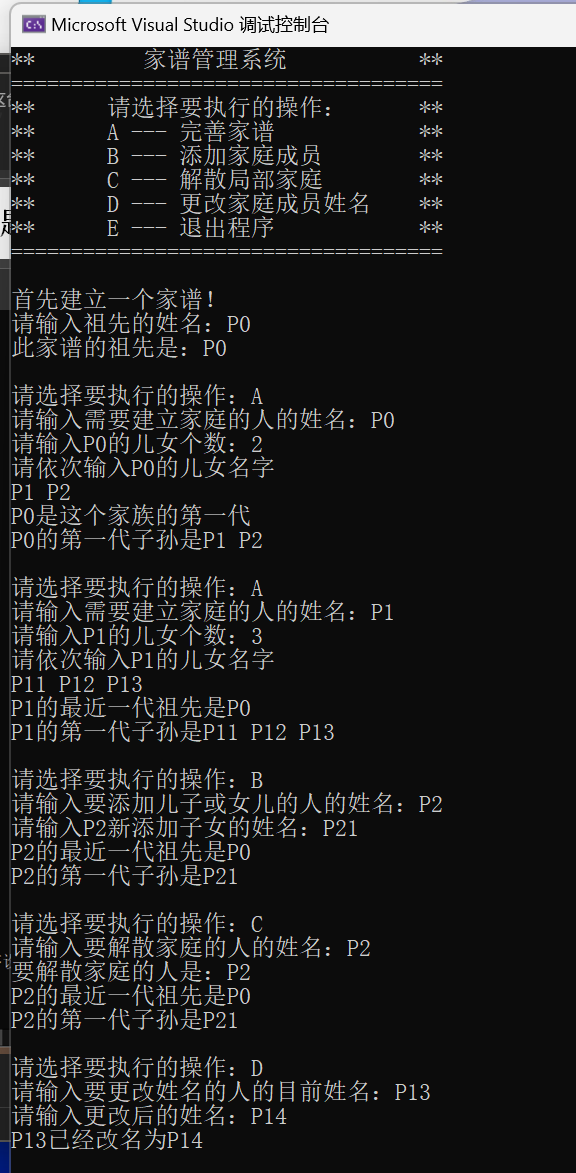


4.4更改家庭成员姓名功能截屏示例



5.设计小结

在这次的家谱管理系统的编程中我掌握了使用二叉树模拟多叉树，除此之外我更加熟练地掌握了栈的使用和二叉树的遍历以及递归的用法，在该程序完成之后，我也得到了样例的输出。



同时，我也再多叉树的定义中增加了一个双亲指针域。当双亲存在时，用于连接节点。在打印时，也会根据是否存在上一代祖先打印祖先，我认为这对于现实中的家谱管理还是很有帮助的。但是在家谱管理系统中需要很多细致的错误处理，在这个方面我做的不够，代码健壮性还可以加强，希望在下次的编程中可以改善这一点。