数据结构课程设计

项目说明文档

家谱管理系统

|  |  |
| --- | --- |
| 作者姓名： | 高逸轩 |
| 学 号： | 2053385 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

# 1项目分析

## 项目需求分析

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，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。

* 功能完善

实现了建立、增加、查询、修改、删除的基本功能。

* 健壮性

对于错误的输入，如插入已经存在的成员、删除不存在的成员等，程序应该做到不崩溃并且给予一定的提示。

* 执行效率高

针对数据量比较大的情况，本系统也应该具有在较短时间内生成正确的家谱树的能力。

# 1.2 项目要求

### 1.2.1 功能要求

实现家族树的建立，查找，插入，修改，删除功能，并在主函数中完成测试。

### 1.2.2 输入格式

对应的操作序号、对应的家族成员信息

### 1.2.3 输出格式

展示对应操作结果，以及操作后家族树的内容

### 1.2.4项目示例

# 2 项目设计

## 2.1 数据结构设计

本项目需要实现存储一个家谱树，也即多叉树。多叉树的存储有多种方式，在这里，我们使用左儿子、右兄弟表示法：即采用二叉树的存储结构存储多叉树。其中，每一个节点的左节点是其长子，右节点则是其兄弟；

## 2.2 类设计

为了实现家谱树，本项目首先定义了家谱树节点，内部包括了当前节点的名字、所处位置、儿子和兄弟地址等信息。以下为部分代码：

### 2.2.1 节点

struct TreeNode

{

// 构造函数，根据所给数据初始化数据

TreeNode(string s = "", TreeNode\* fc = NULL, TreeNode\* ns = NULL) :name(s), firstChild(fc), nextSibling(ns) { father = NULL; };

TreeNode\* firstChild; // 指向第一个孩子

TreeNode\* nextSibling; // 指向下一个兄弟

TreeNode\* father; // 指向父节点

string name; // 名字

};

### 2.2.2 家族树

// 家族树

class Tree

{

public:

// 构造函数

Tree()

// 析构函数，释放空间

~Tree()

// 建立根节点，即家族的祖先

void BuildRoot()

// 建立某个子树

void BuildTree()

// 添加某个节点

void AddTreeNode()

// 删除某个子树

void DeleteTree()

// 更改某个节点信息

void ResetTreeNode()

// 展示某个节点子女

void ShowTree(const string& name)

// 搜索树中是否有名字为name的家族成员，查找成功返回对应地址的值，失败则返回NULL

TreeNode\* Search(const string& name)const

private:

TreeNode\* root; // 家族树根节点

// 查看当前结点是否为所需要查找的家族成员，查找成功返回对应地址的值，失败则返回NULL

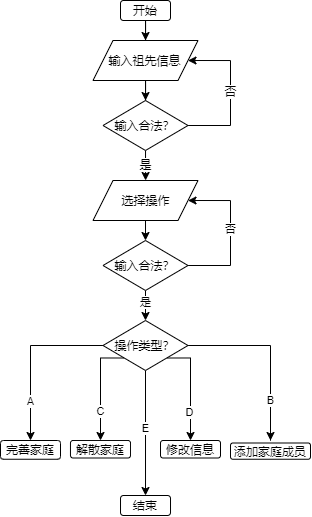
TreeNode\* Search(TreeNode\* current, const string& name)const

// 删除以当前结点为树根的子树

void DeleteTree(TreeNode\* current)

};

## 2.3 项目流程图



# 3 核心代码介绍

## 3.1 建立家谱

## 3.2 完善家谱

// 建立某个子树

    void BuildTree()

    {

        // 得到建立者姓名

        cout << "请输入要建立家庭的人的姓名：";

        string name;

        cin >> name;

        // 用parent来Search找到建立者的位置

        TreeNode\* parent = Search(name);

        // 错误输入处理

        if (parent == NULL)

        {

            cout << "本家谱内不存在该家庭成员！" << endl;

            return;

        }

     // 若当前人已经有了孩子，说明已经建立过家庭，则不能再次建立家庭，给出提示

        if (parent->firstChild != NULL)

        {

            cout << "该成员已经建立过家庭！" << endl;

            return;

        }

        // 利用getint函数得到儿女个数，保证健壮性

        cout << "请输入" << name << "的儿女个数：";

  int childNum = getint(0, INT\_MAX, "儿女个数必须为正整数，请重新输入！");

        // 为parent插入孩子

        cout << "请依次输入" << name << "的儿女的姓名:";

        // 为parent插入firstChild

        while (1)

        {

            cin >> name;

            // 该名称未被使用，可以插入

            if (Search(name) == NULL)

            {

                // 更新儿子

                parent->firstChild = new TreeNode(name);

                // 对内存管理进行错误处理，保证健壮性

                if (parent->firstChild == NULL)

                {

                    cerr << "内存分配错误！" << endl;

                    system("pause");

                    exit(1);

                }

                // 更新儿子节点的父亲为parent

                parent->firstChild->father = parent;

                break;

            }

            // 该名称已经使用过了，需要重新输入

            else

            {

          cout << "姓名为" << name << "的成员已存在！请重新输入！" << endl;

                cin.clear();

                // 清除缓存区，避免后续错误输入产生后效性

                cin.ignore(1024, '\n');

            }

        }

        // 更新其他子女信息

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

        {

            while (1)

            {

                cin >> name;

                // 该名称未被使用，可以插入

                if (Search(name) == NULL)

                {

                    TreeNode\* p = parent->firstChild;

                    // 将p指针移到parent的最后一个孩子处

                    while (p->nextSibling != NULL)

                        p = p->nextSibling;

                    // 为最后一个孩子更新兄弟

                    p->nextSibling = new TreeNode(name);

                    // 对内存管理进行错误处理，保证健壮性

                    if (p->nextSibling == NULL)

                    {

                        cerr << "内存分配错误！" << endl;

                        system("pause");

                        exit(1);

                    }

                    // 对最后一个孩子的father设置为parent

                    p->nextSibling->father = parent;

                    break;

                }

                // 该名称已经使用过了，需要重新输入

                else

                {

          cout<<"姓名为"<<name<<"的成员已存在！请重新输入！"<<endl;

                    cin.clear();

                    // 清除缓存区，避免后续错误输入产生后效性

                    cin.ignore(1024, '\n');

                }

            }

        }

        // 展示parent节点的孩子

        ShowTree(parent->name);

    }

## 3.3 添加子女

// 添加某个节点

    void AddTreeNode()

    {

        // 得到添加者姓名

        cout << "请输入要添加儿子（或女儿）的人的姓名：";

        string name;

        cin >> name;

        // 用parent记录添加者的位置

        TreeNode\* parent = Search(name);

        // 错误输入处理

        if (parent == NULL)

        {

            cout << "本家谱内不存在该家庭成员！" << endl;

            return;

        }

        // 得到新添加的节点的姓名

        cout << "请输入" << name << "新添加儿子（或女儿）的姓名：";

        // 更新firstChild

        if (parent->firstChild == NULL)

        {

            while (1)

            {

                cin >> name;

                // 该名称未被使用，可以插入

                if (Search(name) == NULL)

                {

                    // 更新firstChild

                    parent->firstChild = new TreeNode(name);

                    // 对内存进行处理，保证健壮性

                    if (parent->firstChild == NULL)

                    {

                        cerr << "内存分配错误！" << endl;

                        system("pause");

                        exit(1);

                    }

                    // 将firstChild的father设置为parent

                    parent->firstChild->father = parent;

                    break;

                }

                // 该名称已经使用过了，需要重新输入

                else

                {

                  cout<<"姓名为"<<name<<"的成员已存在！请重新输入！"<<endl;

                    cin.clear();

                    // 清除缓存区，避免后续错误输入产生后效性

                    cin.ignore(1024, '\n');

                }

            }

        }

        // 更新孩子节点的nextSibling

        else

        {

            while (1)

            {

                cin >> name;

                // 该名称未被使用，可以插入

                if (Search(name) == NULL)

                {

                    TreeNode\* p = parent->firstChild;

                    // 将p指针移到parent的最后一个孩子处

                    while (p->nextSibling != NULL)

                        p = p->nextSibling;

                    // 更新兄弟

                    p->nextSibling = new TreeNode(name);

                    // 对内存进行处理，保证健壮性

                    if (p->nextSibling == NULL)

                    {

                        cerr << "内存分配错误！" << endl;

                        system("pause");

                        exit(1);

                    }

                    // 设置当前最后一个孩子的father为parent

                    p->nextSibling->father = parent;

                    break;

                }

                // 该名称已经使用过了，需要重新输入

                else

                {

                  cout<<"姓名为"<<name<<"的成员已存在！请重新输入！"<<endl;

                    cin.clear();

                    // 清除缓存区，避免后续错误输入产生后效性

                    cin.ignore(1024, '\n');

                }

            }

        }

        // 展示parent节点的孩子

        ShowTree(parent->name);

    }

## 3.4 删除节点

    // 删除某个子树

    void DeleteTree()

    {

        // 得到解散者的姓名

        cout << "请输入要解散家庭的人的姓名:";

        string name;

        cin >> name;

        // 利用parent得到解散者的位置

        TreeNode\* parent = Search(name);

        if (parent == NULL)

        {

            cout << "本家谱内不存在该家庭成员！" << endl;

            return;

        }

        cout << "要解散家庭的人是:" << name << endl;

        // 在解散前首先展示当前解散者的孩子

        ShowTree(parent->name);

// 若解散者不是根节点，需要进行以下操作将以parent为根节点的树分离出，维护家族树仍然有序：

        if (parent != root)

        {

// 如果解散者是其父节点的第一个儿子，则其父节点的firstChild指针需要指向解散者的下一个兄弟

            if (parent->father->firstChild == parent) parent->father->firstChild = parent->nextSibling;

// 如果解散者不是其父节点的第一个儿子，则它的上一个兄弟节点的nextSibling指针需要指向解散者的下一个兄弟

            else

            {

                // 用p来寻找解散者的上一个兄弟节点

                TreeNode\* p = parent->father->firstChild;

                while (1)

                {

                    if (p->nextSibling == parent) break;

                    p = p->nextSibling;

                }

                //上一个兄弟节点的nextSibling指针需要指向解散者的下一个兄弟

                p->nextSibling = parent->nextSibling;

            }

            // 将以parent为根的树分离出后，再释放内存

            DeleteTree(parent);

        }

        else

            // 若解散者恰为根节点，不需要分离，直接释放内存即可

            DeleteTree(parent);

    }

    // 删除以当前结点为树根的子树

    void DeleteTree(TreeNode\* current)

    {

        // 遇到空节点回溯

        if (current == NULL) return;

        // 当前节点有子女，向下递归

        if (current->firstChild != NULL)

        {

            // 先删除第一个儿子

            DeleteTree(current->firstChild);

            TreeNode\* p = current->firstChild;

            // 再依次删除其兄弟

            while (p->nextSibling != NULL)

            {

                DeleteTree(p->nextSibling);

                p = p->nextSibling;

            }

            // 释放内存

            delete current;

        }

    }

在删除节点时，与之前操作不同的是，为了保证之前已经存在的节点保证有序，需要添加一些判断：若删除节点是其父节点的长子，则直接将删除节点的兄弟转换为父节点新的长子，同时删除这个节点；若删除节点不是其父节点的长子，则将其前置节点的后继节点标记为当前节点的后继节点，然后删除这个节点。另外定义了**重载函数**实现递归删除。

## 3.5 更改成员名称

    // 更改某个节点信息

    void ResetTreeNode()

    {

        // 得到改名人的信息

        cout << "请输入要更改姓名的人的目前姓名:";

        string name;

        cin >> name;

        // 用current记录改名者的位置

        TreeNode\* current = Search(name);

        // 对内存管理进行错误处理，保证健壮性

        if (current == NULL) {

            cout << "本家谱内不存在该家庭成员！" << endl;

            return;

        }

        // 得到更改后的姓名

        cout << "请输入更改后的姓名：";

        while (1)

        {

            cin >> name;

            // 若更改后的姓名没有出现过，可以改名

            if (Search(name) == NULL)

            {

                cout << current->name << "已更名为" << name << endl;

                current->name = name;

                break;

            }

            // 该名称已经使用过了，需要重新输入

            else

            {

                cout<<"姓名为"<<name<<"的成员已存在！请重新输入！" <<endl;

                cin.clear();

                // 清除缓存区，避免后续错误输入产生后效性

                cin.ignore(1024, '\n');

            }

        }

    }

## 3.6 展示节点信息

// 展示某个节点子女

    void ShowTree(const string& name)

    {

        // 用current定位name姓名人的位置

        TreeNode\* current = Search(name);

        if (current == NULL)

        {

            cout << endl;

            return;

        }

        // 没有子女处理

        current = current->firstChild;

        if (current == NULL)

        {

            cout << name << "没有子女" << endl;

            return;

        }

        // 有子女处理

        else

        {

            // 依次输出子女姓名

            cout << name << "的第一代子女有：";

            cout << current->name << " ";

            while (current->nextSibling != NULL)

            {

                current = current->nextSibling;

                cout << current->name << ' ';

            }

            cout << endl;

        }

}

## 3.7 搜索成员是否存在

    // 搜索树中是否有名字为name的家族成员，查找成功返回对应地址的值，失败则返回NULL

    TreeNode\* Search(const string& name)const

    {

        if (root == NULL)

            return NULL;

        else

            return Search(root, name);

    }

// 查看当前结点是否为所需要查找的家族成员，查找成功返回对应地址的值，失败则返回NULL

    TreeNode\* Search(TreeNode\* current, const string& name)const

    {

        // 如果当前节点符合

        if (current->name == name)

            return current;

        // 遍历当前节点的儿子节点进行递归搜索

        TreeNode\* findResult = NULL;

        for (TreeNode\* p = current->firstChild; p != NULL; p = p->nextSibling)

        {

            findResult = Search(p, name);

            // 儿子节点搜索成功

            if (findResult != NULL)

                return findResult;

        }

        // 儿子节点搜索失败

        return NULL;

    }

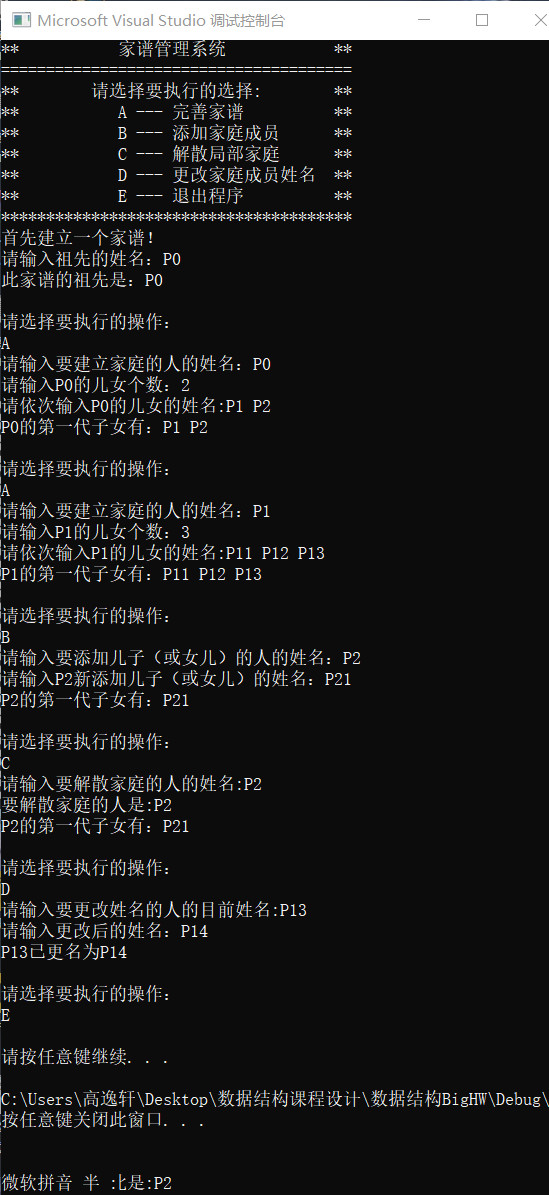
在插入、删除、建立操作中，为了保证健壮性，常常需要搜索用户输入的名字是否在家族树中存在，以防止插入已经存在的姓名、删除不存在的姓名等错误操作。为方便实现上述功能，编写了Search()函数来寻找家族树中是否已经存在某个姓名的节点。另外定义了重载函数，实现递归搜索。

# 4 项目测试

## 4.1 健壮性测试

以上为两种错误信息的处理，通过在各函数中调用Search()函数辅助实现。

## 4.2 功能测试

为方便老师测试，提供了文件6\_test.txt，内含一组数据，测试了本程序的全部功能。以下为一组测试数据：

# 5 心得体会

树是一种重要的数据结构，有多种表示方式。本题是最简单的树的功能的实现，其中节点与节点的关系互相表示采取了儿子兄弟表示法（二叉树方式）。

由于篇幅原因，报告内还有很多内容与解释没有展示，请老师和助教老师再移步源程序，在其中的注释写了每一步过程的详解。