项目说明文档

数据结构课程设计

——家谱管理系统

作 者 姓 名: 沈星宇

学　　号: 1951576

指 导 教 师: 张颖

学院、 专业: 软件学院 软件工程

同济大学

Tongji University

一、 分析

（1） 应用背景

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。

（2） 项目功能要求

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

```
template<class T>
class Stack {
private:
    T* elements;
    int top;
    int maxSize;
    void overflowProcess() { ... }
public:
    Stack() { ... }
    ~Stack() { ... }
    bool push(T& value) { ... }
    bool pop(T& value) { ... }
    bool getTop(T& topNum) { ... }
    bool isEmpty() { ... }
    bool isFull() { ... }
    int getSize() { ... }
    void makeEmpty() { ... }
};
```

```
class TreeNode {
private:
    char data[10];
    TreeNode* firstChild, * nextSibling;
public:
    TreeNode() { ... }
    TreeNode(char* value, TreeNode* fc, TreeNode* ns) { ... }
    char* getData() { ... }
    TreeNode* getFirstChild() { ... }
    TreeNode* getNextSibling() { ... }
    void changeData(char* value) { ... }
    void changeFirstChild(TreeNode* fc) { ... }
    void changeNextSibling(TreeNode* ns) { ... }
    bool insertSibling(TreeNode* newNode) { ... }
    bool insertChild(TreeNode* newNode) { ... }
};
```

```
class Tree {
private:
    TreeNode* root;
public:
    Tree() { root = NULL; }
    Tree(TreeNode* root) { ... }
    TreeNode* getRoot();
    bool firstChild(TreeNode* p);
    bool nextSibling(TreeNode* p);
    char* getData(TreeNode* p);
    bool insertChild(TreeNode* p, char* value);
    void deleteChild(TreeNode* pPre, TreeNode* p);
    void dfs(TreeNode* p, Stack<TreeNode*>& stack);
    bool isEmpty() { ... }
    bool find(TreeNode* p, char* value, TreeNode*& resultNode);
    bool find(TreeNode* p, char* value, TreeNode*& resultNode, TreeNode*& resultNodePre);
};
```

1、Stack 是栈，本项目利用栈后进先出的特性进行释放内存。本项目使用的是线性栈，用一个 top 指向栈顶，以此来对栈顶元素进行操作。其中最主要的函数 push()和 pop()是将元素压入和弹出。

2、TreeNode 是树的结点，本项目采用子女-兄弟树的存储方式，故每个结点内存储有 TreeNode* firstChild 和 TreeNode* nextSibling 两个指针，用来指向该节点的首子女和下一个兄弟结点的地址。

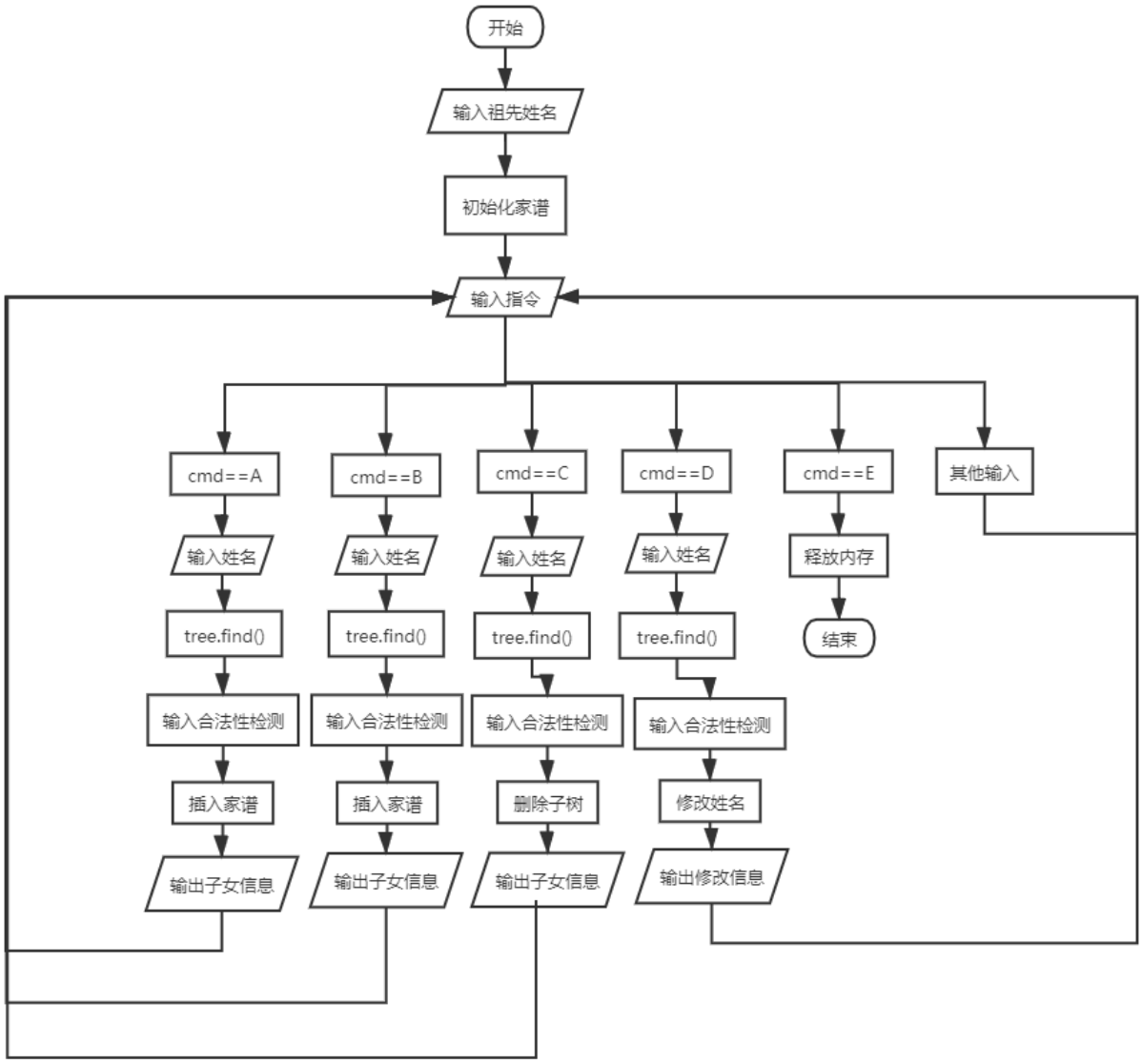
3、Tree 是树结构，基本的操作有判断当前节点 TreeNode* p 的首子女、下一兄弟节点是否存在，插入节点，删除子树，深搜遍历，查找结点等操作。

（2） 程序流程设计

1、首先输入的祖先的姓名，用它初始化树的根节点

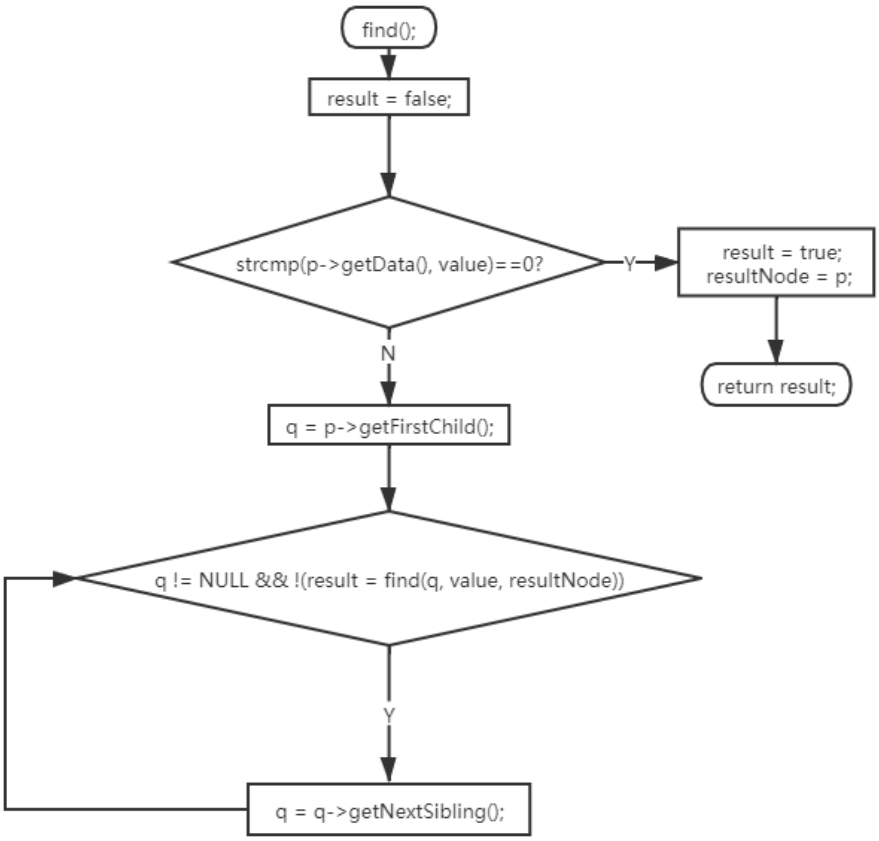
2、循环读入指令 cmd，根据指令的具体内容跳转具体的代码。其中，如果输入的指令不合法（即超出 A/a/B/b/C/c/D/d/E/e）,那么会要求重新输入。

3、对于每条分指令，会对输入的姓名进行搜索，若没搜索到，那么就会要求重新输入指令；如果搜索到了，就会对家谱进行具体的操作。



三、 实现

1、 Tree::find()的实现




```
bool Tree::find(TreeNode* p, char* value,TreeNode*& resultNode) {
    bool result = false;
    if (!strcmp(p->getData(), value)) {
        result = true;
        resultNode = p;
    }
    else {
        TreeNode* q = p->getFirstChild();
        while (q != NULL && !(result = find(q, value, resultNode))) {
            q = q->getNextSibling();
        }
    }
    return result;
}
```

(1) Tree::find()函数的三个参数 TreeNode* p, char* value,TreeNode*& resultNode, 分别代表着查找开始的根节点 root、查找的关键字姓名、和用于传出的结果节点地址。

(2) 用 strcmp(p->getData(), value)函数来比较是否查找到结果，如果查找到则将地址赋值给 resultNode；如果没有查找到，则用递归的办法去遍历搜索

2、 合法性检测部分

```
if (tempAdd == NULL) {
    std::cout << "名为" << tempName << "的成员不存在！请重新输入\n";
    return false;
}
//-----------------------------------------------------
std::cout << "请输入" << tempName << "的儿女人数:";
int num;
while (1) {
    std::cin >> num;
    if (std::cin.fail() || num<=0 ) {
        std::cout << "输入不合法，请输入一个正整数\n";
        return false;
    }
    else {
        break;
    }
}
//-----------------------------------------------------
```

(1) 对于输入的姓名没能在树当中的情况，则会通过返回 false 的情况来在 main 函数当中要求重新输入指令；对于输入的数字不是正整数的情况则会要求重新输入数字。

3、 TreeNode::insertSibling()和 TreeNode::insertChild()的实现

```
bool insertSibling(TreeNode* newNode) {
    if (this->nextSibling == NULL) {
        this->nextSibling = newNode;
        return true;
    }
    else {
        return false;
    }
}
bool insertChild(TreeNode* newNode) {
    if (this->firstChild == NULL) {
        this->firstChild = newNode;
        return true;
    }
    else {
        return false;
    }
}
```

由于是链式的队列，插入的实现其实就是改变该节点的 firstChild 或者 nextSibling 指针。因为事先无法得知现有节点是否含有子女结点，所以要通过这两个函数能够实现将插入是否成功的情况返回的功能。

4、 Tree::insetChild()的实现

Tree::insertChild()函数其实是按位后插，即在 TreeNode* p 的后面插入一个节点。

（1） 首先会对 TreeNode* p 进行判断是否含有子女，若无，则将 p 的 firstChild 链接到 newNode；若已有子女，则找到最后一个兄弟节点，将其 nextSibling 链接到 newNode。

```
bool Tree::insertChild(TreeNode* p, char* value) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->changeData(value);
    newNode->changeFirstChild(NULL);
    newNode->changeNextSibling(NULL);
    if (firstChild(p)) { //p结点已有子女结点
        TreeNode* temp = p->getFirstChild();
        while (1) {
            if (temp->getNextSibling() == NULL) {
                return temp->insertSibling(newNode);
            }
            temp = temp->getNextSibling();
        }
    }
    else {
        return p->insertChild(newNode);
    }
}
```

5、 **Tree::deleteChild()的实现**

（1） TreeNode* p 指向的是要释放的子树的根节点，TreeNode* pPre 指向的是要释放的子树的根节点的前一个节点（可能是父母节点或者是兄弟节点）

（2） 同时对传入的 TreeNode* p 进行判断，根节点无需进行调整，而非根节点需要调整 pPre 的指针（原本指向的是 p）

（3） 然后利用栈 freeStack，在用 dfs 深度优先搜索遍历 p 为根节点的子树时，将节点压入栈内，最后从栈内 pop()出元素进行释放。

```
void Tree::deleteChild(TreeNode*pPre,TreeNode* p) {
    if (p != getRoot()) {
        if (pPre->getNextSibling() == p) {
            pPre->changeNextSibling(p->getNextSibling());
        }
        else {
            pPre->changeFirstChild(NULL);
        }
    }
    TreeNode* deletePoint;
    Stack<TreeNode*> freeStack;
    dfs(p, freeStack);
    while (!freeStack.isEmpty()) {
        freeStack.pop(deletePoint);
        if (deletePoint != getRoot()) {
            free(deletePoint);
        }
    }
}
```

6、 **Tree:dfs()的实现**

用深度优先搜索遍历以 p 为根节点的树或子树，搜索到节点之后将每个节点压入栈中，每一层递归的退出条件是 p==NULL。

```
void Tree::dfs(TreeNode* p,Stack<TreeNode*>& stack) {
    if (p != NULL) {
        stack.push(p);
        for (p = p->getFirstChild(); p != NULL; p = p->getNextSibling()) {
            dfs(p, stack);
        }
    }
}
```

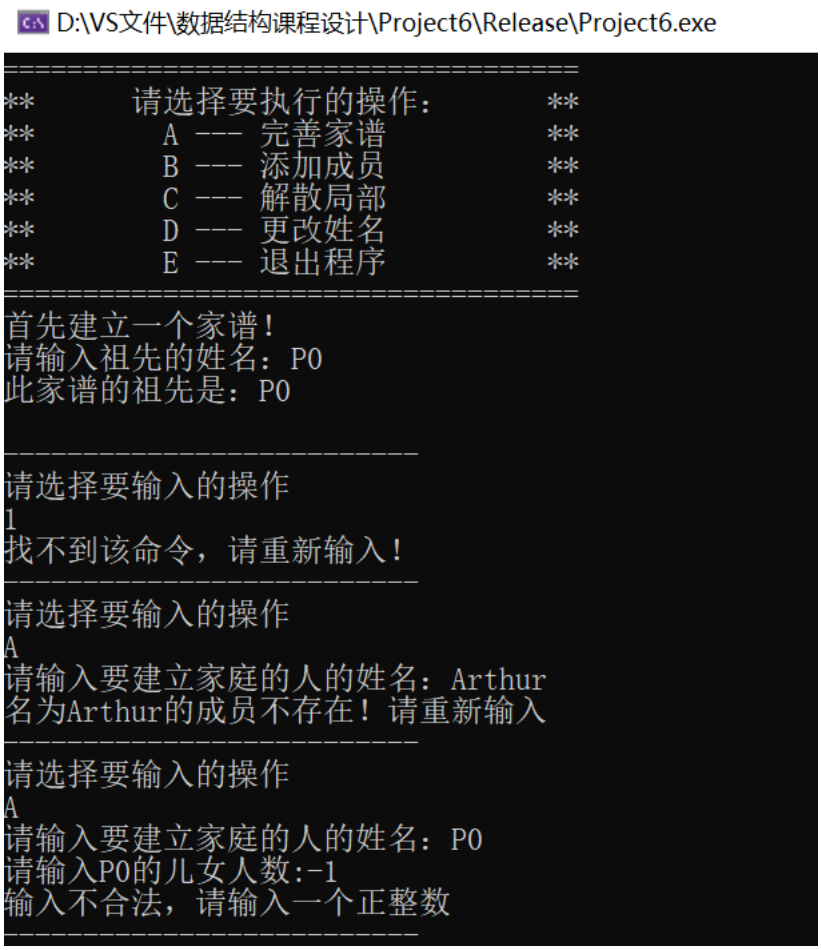
四、 测试

1、合法性检测

（1）输入的指令不合法（即超出 A/a/B/b/C/c/D/d/E/e）,会要求重新输入。

（2）如果命令指向的人不在家谱中，则会要求重新输入

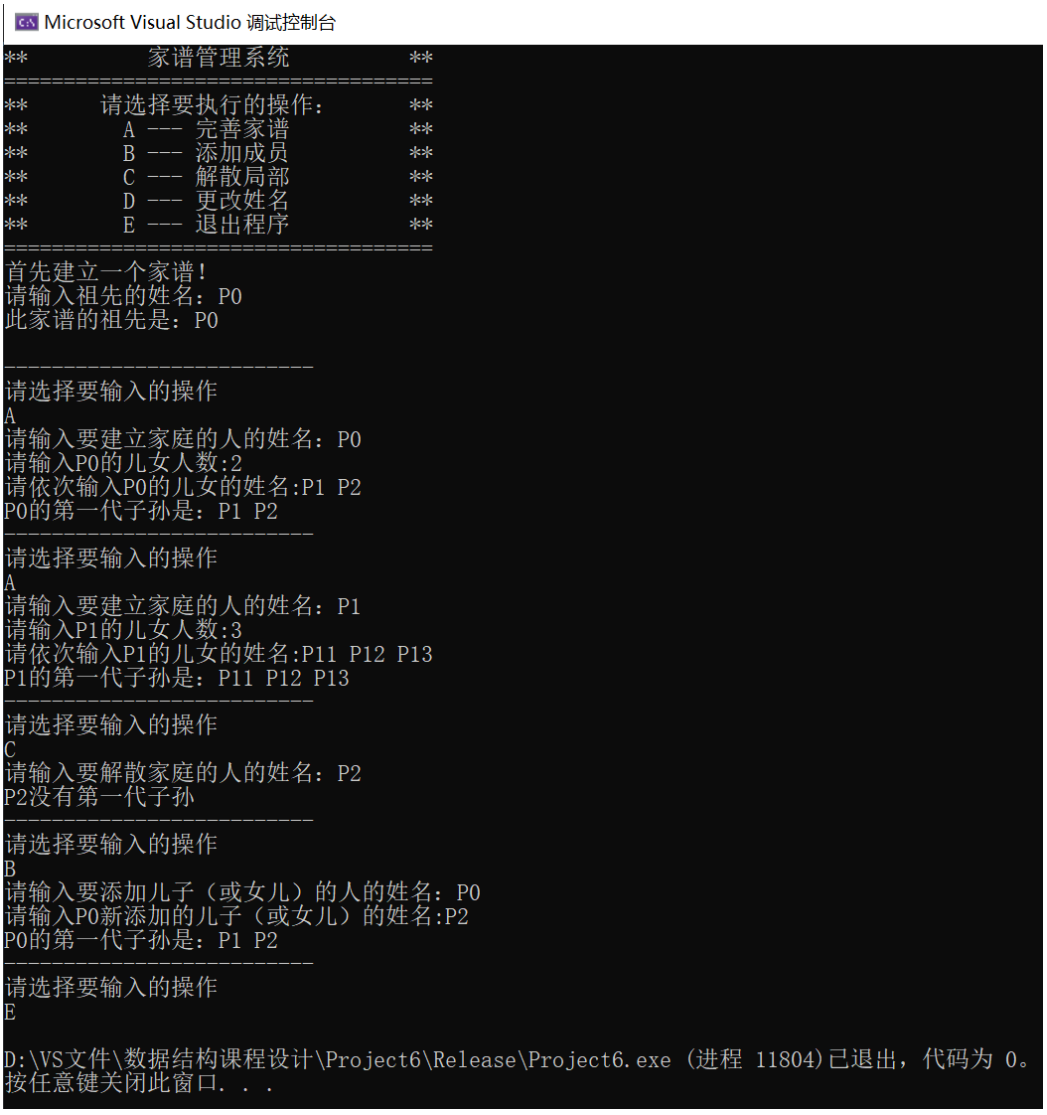
（3）如果输入的人数不为正整数，则会要求重新输入



2、一般情况的功能实现

(1) 测试的流程是首先建立以 P0 为根节点的家庭，子女为 P1 P2，其中 P1 的子女为 P11 P12 P13，解散 P2 的家庭，其中 P2 没有子女故输出相关信息（若有则输出子一代的信息）

(2)


Microsoft Visual Studio 调试控制台
**   家谱管理系统   **
==========
**   请选择要执行的操作: **
**   A --- 完善家谱   **
**   B --- 添加成员   **
**   C --- 解散局部   **
**   D --- 更改姓名   **
**   E --- 退出程序   **
==========
首先建立一个家谱!
请输入祖先的姓名：P0
此家谱的祖先是：P0

请选择要输入的操作
A
请输入要建立家庭的人的姓名：P0
请输入P0的儿女人数:2
请依次输入P0的儿女的姓名:P1 P2
P0的第一代子孙是：P1 P2

请选择要输入的操作
A
请输入要建立家庭的人的姓名：P1
请输入P1的儿女人数:3
请依次输入P1的儿女的姓名:P11 P12 P13
P1的第一代子孙是：P11 P12 P13

请选择要输入的操作
C
请输入要解散家庭的人的姓名：P2
P2没有第一代子孙

请选择要输入的操作
B
请输入要添加儿子（或女儿）的人的姓名：P0
请输入P0新添加的儿子（或女儿）的姓名:P2
P0的第一代子孙是：P1 P2

请选择要输入的操作
E
D:\VS文件\数据结构课程设计\Project6\Release\Project6.exe (进程 11804)已退出，代码为 0。
按任意键关闭此窗口. . .