实验报告

——project3

组长：王迎旭（16340226）
组员：王友坤 （16340228）汤万鹏（16340205）
班级：16 级软件工程教务三班
日期：2017.11.13

目录

第一部分

问题描述

【题目要求】

用树形结构实现家族成员信息管理（如建立、删除、查询、统计、打印等）。

【实现功能】

该程序采用二叉树结构进行家谱的各种操作。二叉树的结构为：男性的左子树为他的兄第姐妹，右子树为他的妻子。妻子节点的左子树为她的孩子，右子树为其他妻子（丈夫再婚）。

程序可以读入文件并依据文件内容建立家谱，或者通过手动输入一个节点一个节点的添加建立家谱。

家谱的操作分为 5 大类：

①添加成员：添加妻子（需知道丈夫姓名）、添加孩子（需知道母亲姓名）。

②删除成员查询成员：删除成员（需知道其姓名）并删除他的妻子和孩子。

③查询成员：查询单个成员并输出（需知道其姓名）、查询成员及其家庭并输出（妻子和孩子）（需知道其姓名）、查询成员父母并输出（需知道其姓名）、查询某代的所有成员并输出。

④统计成员：统计家谱图中的总人数

⑤打印成员信息：以图的形式直观显示家谱图

⑥修改成员信息：修改姓名、修改性别、修改生命状态、修改年龄、修改婚姻状态。

第二部分

编译细节的介绍

2.1 运行环境

①Windows10（部分特殊字符无法显示）

②Fedora 虚拟机或者 Ubuntu 虚拟机**（推荐）**

2.2编程语言

c++

2.3开发工具

（1）Windows，linux 环境下的 g++编译器与 Sublime Text3 编辑器

（2）Dev c++（注:由于本地的编写实在 Sublime 中完成，所以在 c++中编译时候会产生注释乱码情况）

2.4编码的规范

Sub 中编辑时候，已经严格按照首行缩进的方式进行编译。但美中不足，在代码进行复制粘贴时候有可能导致代码错位的问题。

第三部分

分析与设计

3.1 数据结构

二叉树：用于模拟家谱成员的关系。

3.2 设计思路

前期考虑：

家谱内部成员用二叉树的结构储存。但是由于二叉树中每个节点最多只有两个子树，而每个成员有可能有多个子女。因此不能用二叉树中结点的父子关系表示家谱成员的父子关系。左子树表示该家谱成员的孩子，右子树表示该家谱成员的兄弟姐妹。这样，就可以将该家谱中所有成员合理存储在二叉树中。

整体的思路介绍：

1. **总体对家谱成员的操作与二叉树的插入删除遍历等操作类似。**
2. **若增添成员，则需要由用户输入该成员的全部信息。**

为了实现如何建立中 从空家谱树开始建立的可能，我们必须在家谱树是空的情况下设定一个祖宗节点，也就是数的根节点

添加成员有两种可能：一种是添加妻子，一种是添加子女

①添加妻子需要知道丈夫是谁（name）

②添加子女需要知道母亲是谁（name）

③由于存在给祖先节点添加兄弟姐妹的可能，但是祖先节点的母亲是谁我们无法提供，所以有必要单单对根节点提供一个添加兄弟姐妹的操作

3. **若删除成员，则输入成员的名字等部分信息。**

删除节点是件想对来讲比较困难的事，对应着现实生活中的移出家谱，因为家谱树中每个节点之间都可能存在一定的关系，一旦一个节点被删除，其他节点如何重新进行连接是个非常复杂的问题。所以这里要用到节点的连接，如果是度为零的结点就无需拼接，如果度不为零，则需要进行指针的遍历后找到这个位置，对这个位置进行拼接。

4. **若查看家谱成员，则按照二叉树的遍历操作依次输出家谱成员。**

在打印成员信息这一步使用目录打印法进行输出，打印出所有的父亲【M】母亲【F】以及父亲是否有重婚，以及孩子的分布情况，目录打印法的具体实现参考上周完成的小实验 5.

5. **若更新成员，则输入需要更新的成员的名字，随后进行年龄，死亡信息的更新。**
6. **若查询家谱中的总人数，则可以统计出家谱中所有人数（即家族树上所有结点的个数）**

第四部分

代码展示

ManagerUI 类:

```
class ManagerUI
{
  public:
    ManagerUI(FamilyTree *tree) : m_familytree_ptr(tree) {}//引导主界面
    void PrintUI();//打印主界面
    void InputLoop();//输入的选择循环
    void insertAncestor();//设立祖先父节点
    void insertWife();//插入妻子
    void insertChild();//插入孩子
    void updateMember();//更新成员的信息
    void deleteMember();//删除成员
    void queryMember();//查询成员
    void printTree();//打印列表
    void statisticAnalyze();//查询总人数
    FamilyMember* handleMultiQueryResult(vector<FamilyMember*> &res);//同名同性别人数选择
}
```

FamilyTree 类：

```
class FamilyTree {
 public:
  enum Sex {
    boy,
    girl,
    unknown
  };

  struct FamilyMember {
    static int current_id;
    int id;
    string name;//名字
    Sex sex;//性别
    int age;//年龄
    bool dead;//是否死亡
    bool divorced;//是否离婚
    FamilyMember *left;//左结点
    FamilyMember *right;//右结点
    FamilyMember(string t_name, Sex t_sex, int t_age, bool t_die = false, bool t_divorced = false, FamilyMember *t_left = nullptr, Fam
  };//结点的属性设立
  FamilyTree();//构造函数
  ~FamilyTree();//析构函数
  void insertChild(FamilyMember *, FamilyMember *);//插入孩子
  void insertWife(FamilyMember *, FamilyMember *);//插入妻子
  bool deleteMember(function<bool(const FamilyMember *)> filter);//删除成员
  vector<FamilyMember*> queryMember(function<bool(const FamilyMember *)> filter);//确认成员
  bool updateMember(function<bool(const FamilyMember *)> filter, function<void(FamilyMember *)> switcher);//更新成员信息
  size_t countMembers(function<bool(const FamilyMember *)> filter);//计算成员数目
  static void PrintMembers(const FamilyMember* , int printLevel = 0, bool isLeft = false, vector<bool> blankIndex = {});//打印成员信息
  FamilyMember *root;

};
```

第五部分

测试程序

（1）开始页面（需要先设立一个祖先）：

```
Welcome to your Family-Tree management system!


Setting a ancestor
You should enter as this format:[name] [age] [pass/not pass](0/1)
Name:Mike
Age:69
Pass/not Pass:0
```

（2）主菜单：

```
Menu
1. insertWife
2. insertChild
3. updateMember
4. deleteMember
5. find
6. printMembers
7. count all member's amount
0. out
Your choice:
```

测试数据（A）

（1）为家谱中某成员增添妻子：

```
@ insertWife
You should enter as this format:
[name] [age] [divorced/not divorced](0/1) [pass/not pass](0/1)
Name:Amy
Age:55
Divorced/not Divorced:0
Pass/not Pass:1
Please cin her [husband's name]
Husband's name:Mike
```

（2）为家谱中某成员增添孩子：

```
@ insertChild
Sex choice [0: female 1: male 2: unknown]
You should enter as this format:
[name] [sex] [age] [pass/not pass](0/1)
Name:li
Sex:0
Age:30
Pass/not Pass:0
```

（3）打印家谱的具体分布图：

```
Mike[M] / Amy[F]*
└─ li[F]
```

（4）更新成员信息：

（更新前）

```
@ updateMember
Enter the member's name
Name:Amy
Current member information:
ID: 1
Name: Amy
Sex: Female
Age: 55
Pass/not Pass: Yes
Divorced: Not
```

（更新中）

```
Enter updated member information:
[name] [sex] [age] [pass/not pass](0/1)
Name:Amy
Sex:0
Age:59
Dead/not Dead:0
```

（更新后，更新成功）

```
@ updateMember
Enter the member's name
Name:Amy
Current member information:
ID: 1
Name: Amy
Sex: Female
Age: 59
Pass/not Pass: Not
Divorced: Not
```

（5）删除家谱中某成员：

```
@ deleteMember
Enter the member's name
Name:li
```

（删除前打印）

```
Mike[M] / Amy[F]*
└─ li[F]
```

（删除后再次打印）

```
Mike[M] / MM[F]*
```

（6）计算家谱中成员数目：

```
The total number is: 2
```

测试数据（B）

```
Mike[M] / Jane[F]*
├─ Jack[M] / Tay[F]*
│   ├─ TOm[M] / Joke[F]*
│   │   └─ Mano[M]
│   └─ Jofe[M]
├─ Lily[F]
└─ TIme[M]
```

（家谱中多个附属关系的展示，此时 Mike 与 Jane 为夫妻关系；Jack，Lily，Time 三者互为同等级兄弟关系且是 Mike 与 Jane 的后代；Jack 与 Tay 为夫妻关系；Tom 与 Jofe 为 Jack 与 Tay 的后代；Tom 与 Joke 为夫妻关系；Mano 为 Tom 与 Joke 的后代）

测试数据（C）

```
Mike[M](reMarriage)
├─ lili[F]*
└─ LO[F]--Divorced
```

（假如 Mike 重婚，则会有所标注，这时 lili 与 LO 均为 Mike 的妻子而非后代）

测试数据（D）

```
Mike[M](reMarriage)
├─ lili[F]*
├─ LO[F]--Divorced
│   └─ Wang[F]
├─ Joker[F]--Divorced
└─ Koke[F]*
```

（假如 Mike 有多个妻子，则多个妻子之间互为同等级关系，同时会显示已经与之离婚的妻子的离婚状态，同时假如某一个妻子有孩子，孩子则会成为该妻子的附属。在这里，Mike 重婚，lili，LO，Joker，Koke 均为 Mike 的妻子，四者等级相同；Wang 为 Mike 与 LO 的后代）

测试数据（E）

```
@ deleteMember
Enter the member's name
Name:Ko
The total number is 2
Your choice is 0
ID: 9
Name: Ko
Sex: Unknown
Age: 2
Pass/not Pass: Not
Your choice is 1
ID: 10
Name: Ko
Sex: Male
Age: 15
Pass/not Pass: Not
Please choose a member
```

（假如删除成员时有重名，会弹出所有符合要求的人，供你删除）

测试数据（F）

```
@ deleteMember
Enter the member's name
Name:LO
Current member information:
ID: 3
Name: LO
Sex: Female
Age: 68
Pass/not Pass: Not
Divorced: Yes
You can not delete wife
```

（假如要删除的成员是一位母亲且带有孩子，则不能被删除）

测试数据（G）

```
@ deleteMember
Enter the member's name
Name:Mike
Current member information:
ID: 0
Name: Mike
Sex: Male
Age: 25
Pass/not Pass: Not
He/She has been deleted
```

```
The ancestor had gone ! The process has been over!
```

（如若删除祖先，则主程序会提示祖先小时程序则会结束）

第六部分

心得体会

用二叉树实现家谱成员的存储其实并不复杂。使用的基本是二叉树的插入数据、删除数据、查找数据和遍历数据操作。其代码架构与普通二叉树类的成员函数大同小异。

正所谓，越接近人的程序越复杂。基于现实情况，二叉树中结点所代表的家谱成员拥有很多属性和行为。每个家谱成员中、家谱成员之间的行为也相当复杂，需要考虑到二叉树中不同结点之间的复杂关系。在构思家谱程序的整体架构时必须考虑到很多平时在二叉树中不会出现的情况。

此外，面向对象的编程思想也在该家谱程序中普遍运用。家谱成员封装在名为 FamilyTree 的类中，面向用户的操作也封装在名为 ManagerUI 的类中。其实也算是对上学期在初级实训所学到的程序架构知识的应用。如此，程序的整体架构更为清晰，也大大提高了程序的可读性和可维护性。

还有，在集体开发程序中，最考验的还是团队合作能力。团体合作在某种程度上也运用到面向对象的思想。每个人分工明确——整体架构设计、代码实现、实验报告设计等等分别由不同的队员负责。每个人都不用关心其他人的工作是如何实现的，只需要知道其他人能够按要求完成，并能直接把其他人的工作成果投入实用。每个人的工作看似互不干

涉、内部却相辅相成，密不可分。每个人做好自己的模块，把各个模块拼接好就是一个完整的程序。这大概也是日后开发程序、乃至解决其他大型问题的基本方法吧。

该实验考察的是二叉树的基本操作与简单实际应用。因此，目前我们实现的是简单的家谱增减寻查操作。现实生活中可能发生的情况多种多样，若是要将该家谱程序运用到实际中，仍需不断完善。

第七部分

项目分工与自我评价

姓名	分工	自我评分	完成度
王迎旭	代码的设计，算法的实现	94	36%
王友坤	实验报告，代码的debug	95	34%
汤万鹏	主界面的设计，代码后期的细节处理	92	30%