

**课 程 设 计 报 告**

**题目： 基于AVL树的集合ADT及其应用**

**课程名称： 数据结构课程设计**

**专业班级： CS1607**

**学 号： U201614700**

**姓 名： 王亚宁**

**指导教师： 李剑军**

**报告日期： 2018年2月26日**

**计算机科学与技术学院**

# 任务书

## 设计内容

平衡二叉树(AVL)作为一种重要的查找表结构，能有效地支持数据的并行处理。本设计使学生牢固掌握AVL树及其实现方法，并应用该结构实现集合抽象数据类型，提升学生对数据结构与数据抽象的认识，提高综合实践与应用能力。

## 设计要求

本设计分为三个层次：（1）以二叉链表为存储结构，设计与实现AVL树-动态查找表及其6种基本运算；（2）以AVL树表示集合，实现集合抽象数据类型及其10种基本运算；（3）以集合表示个人微博或社交网络中好友集、粉丝集、关注人集，实现共同关注、共同喜好、二度好友等查询功能。

目录

[任务书 3](#_Toc507612474)

[设计内容 3](#_Toc507612475)

[设计要求 3](#_Toc507612476)

[1．引言 6](#_Toc507612477)

[1.1课题背景与意义 6](#_Toc507612478)

[1.2课程设计的主要研究工作 6](#_Toc507612479)

[2．系统需求分析与总体设计 7](#_Toc507612480)

[2.1系统需求分析 7](#_Toc507612481)

[2.2系统总体设计 7](#_Toc507612482)

[3．系统详细设计 8](#_Toc507612483)

[3.1有关数据结构的定义 8](#_Toc507612484)

[3.2主要算法设计 8](#_Toc507612485)

[4．系统实现与测试 9](#_Toc507612486)

[4.1系统实现 9](#_Toc507612487)

[4.2系统测试 9](#_Toc507612488)

[5．总结与期望 10](#_Toc507612489)

[5.1全文总结 10](#_Toc507612490)

[5.2工作展望 10](#_Toc507612491)

[6．体会 11](#_Toc507612492)

[附录 12](#_Toc507612493)

[参考文献 12](#_Toc507612494)

# 1．引言

## 1.1课题背景与意义

“社交网络”是近些年最受关注的互联网名词，它的英文缩写是SNS，第一个S是Social社会化，第二个N代表Networking网络，第三个S是Services服务。SNS的概念起源于社会网络研究者提出的“六度理论”，即最多通过六个人你就能够认识任何一个陌生人。SNS将现实中的人际关系搬到了互联网上，让世界上的任何一个人都能联络彼此。

学习了一个学期的数据结构课程，到了要使用自己学到的技能来完成实际的项目需求的时候。课题我选择了基于AVL的集合及其应用，在课堂上了解到了二叉搜索树这个数据结构，同时也知道了二叉搜索树的查找效率受输入数据的影响很大，以此为契机了解到了AVL即平衡二叉搜索树，它最大优点就是不会因为输入的数据而产生极端的失衡。基于搜索二叉树的特性可以得到另一个常用的数据结构——集合。

## 1.2课程设计的主要研究工作

饰演的主要研究内容是分析二叉树的基本操作、集合基本操作、数据的收集和微博上面社交关系的研究。其中比较重要的是二叉树的基本操作和数据收集，二叉树的操作的学习主要来自网上的博客和文献，数据收集就是万能的某度和某歌，总体来说难度还是比较大的。

# 2．系统需求分析与总体设计

## 2.1系统需求分析

用户需求主要是对各种事物关系的查询和修改，主要有：爱好的增删查、关注集的增删查、粉丝集的删和查、好友的相关操作、二度好友的查询。而这一些操作又是基于集合的交集、并集和差集等操作的，集合的底层是使用的平衡二叉树，二叉树也需要实现自己的基本操作插入、删除、销毁和查询等。

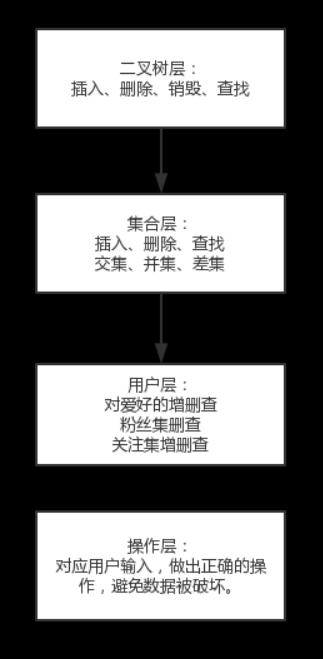
系统的主要实现目标是实现用户的各种合理的查询和修改需求，事务处理流程是用户输入相应的操作序号以及参数，程序进行对应的回应，合适的显示用户期望的结果。

## 2.2系统总体设计

根据上面分析的用户需求知道我们的主要目的就是实现二叉树、集合和用户的各个操作，所以其中的功能模块主要我概括为以下4个：

1. 二叉树模块：二叉树模块实现二叉树的插入、删除、平衡和查找等操作，保证底层的树始终属于有序和平衡状态。
2. 集合模块：集合模块基于二叉树对其进行适当的封装，将底层接口封装成用户可以识别的接口，并在其中实现集合的插入、删除、查找和关系操作，即交集、并集、差集等操作。
3. 用户模块：定义用户实体，包含用户的各种信息，用户会使用到的各种操作，比如关系的查找修改，爱好的添加修改等。
4. 操作层：这一层将用户对数据的操作进行隐藏，防止用户操作不当对数据造成破坏，在操作层会对用户的输入参数进行检查，对数据进行正确的操作。

系统的总体结构如图-1所示：

图-1

# 3．系统详细设计

## 3.1有关数据结构的定义

程序中出现的数据结构就是需求分析中的3个，二叉树、集合和用户实体。下面对其进行详细介绍：

二叉树AVLTree：其中定义了自己的结点类型，Node类型，Node包含了自己所保存的值，左右指针，当前结点高度。二叉树的定义基于Node，主要数据类型就是一个head指针，保存当前的操作的二叉树的根节点，当树为空树时head的值为nullptr。

集合Set：集合基于二叉树，数据域为一棵二叉树tree和一个记录当前结点数量的int数据set\_size。

用户Person：其中包含了集合的几个实体，爱好集合，关注集合，粉丝集合，还有用户id，用户名。

其中的关联关系简介：二叉树作为底层结构提供了最底层的数据存储功能，集合的底层就是对二叉树的一系列操作进行了封装，并且实现了集合的关系操作，用户层面就是对集合的直接应用，用户中含有好几个集合的实例化类，使用集合存储自己的数据，对外提供了操作用户数据的接口。

## 3.2主要算法设计

1）二叉树模块的各种操作实现流程：

插入操作：使用递归调用插入函数的方法，如果插入值大于当前结点值，插入到结点右侧的子树中，小于则插入到左侧的子树中，相等则返回插入失败，知道当前结点为空，直接插入即可，结点高度设置为1。插入完毕后递归向上时检测二叉树的平衡，不平衡的结点进行调整。判断为左左失衡直接将当前结点进行右旋，左右失衡先对左节点左旋，再对当前结点右旋，右侧失衡同样如此调整，依次递归向上，保证树的平衡。时间复杂度O(log(n))，空间复杂度O(log(n))。

删除操作：使用递归删除的方式，删除结点主要分为3种情况，左右结点均为空结点，有一侧为空结点，两侧结点都不为空。两侧都为空的时候直接调用delete函数释放当前空间，将指针值设置为nullptr即可。其中一侧为空结点，将当前结点空间释放，结点指针值设置为剩余非空结点的地址即可。两侧都不为空时，我选择的是寻找当前结点的前一结点，具体就是当前结点的左侧结点的最右侧结点称为交换结点，在寻找过程中将路过的结点使用栈存储起来，找到结点后将其前一节点的右结点替换为交换节点的左结点，将要删除结点释放后替换为要交换的结点，这时先将栈中的结点高度修改正确，进行对应的平衡判断、失衡情况判断和旋转处理，然后处理交换结点，由于左侧删除一节点后判断是否失衡，判断是右左失衡还是右右失衡，进行对应旋转，递归向上，进行平衡的调整和高度调整，保证树的平衡。时间复杂度O(log(n))，空间复杂度O(log(n))。

查找结点：使用类似与插入时的查找方式，递归查找，查找成功后返回true，失败则返回false。时间复杂度O(log(n))，空间复杂度O(log(n));

遍历二叉树：使用递归的中序遍历，保证输出的数据有序。时间复杂度O(n)，空间复杂度O(log(n))；

2）集合模块的实现操作流程：

插入、删除、查找均直接使用二叉树的接口，操作成功对大小进行适当调整返回true，失败就返回false。复杂度见二叉树部分。

交集操作：先获得传入集合的所有元素，遍历查找元素是否存在于此集合中，存在则插入到集合中，最后返回结果集合。时间复杂度O(nlog(n))，由于存储了所有的元素，空间复杂度为O(n)；

并集操作：先后获取两个集合中的元素，全部插入到结果集合中，返回结果即可。时间复杂度O(nlog(n))，空间复杂度O(n)；

差集操作：获取当前集合中的所有元素，遍历元素判断参数集合中是否包含此元素，不包含则插入到结果集合中，返回结果。时间复杂度O(nlog(n))，空间复杂度O(n)；

获取集合大小：直接返回数据域set\_size。时间复杂度O(1)，空间复杂度O(1)；

判断是否为子集：获取参数集合所有元素，遍历判断是否元素都在当前集合中，存在不在的返回false，不存在返回true。时间复杂度O(nlog(n))，空间复杂度O(n)；

判断是否相等：首先判断大小是否相等，不等直接返回false，然后获取其中一个集合的全部元素，遍历并判断另一集合中是否全部都有，都有返回true，存在不同返回false。时间复杂度O(nlog(n))，空间复杂度O(n)；

用户模块的实现流程：插入、删除、查找、交集、并集、差集等均直接调用Set的接口，复杂度见前面的Set部分；

# 4．系统实现与测试

## 4.1系统实现

本程序开发环境：

硬件：笔记本Core i5 6200U，8G DDR3L，256G samsung SSD。

软件：Windows10 1709专业版，开发环境Visual Studio 2017。

程序中的数据类型主要是AVLTree模板类，Set模板类还有Person类，其中的数据域在上面已经说明，各自包含的方法如图-3，图-4，图-5所示：

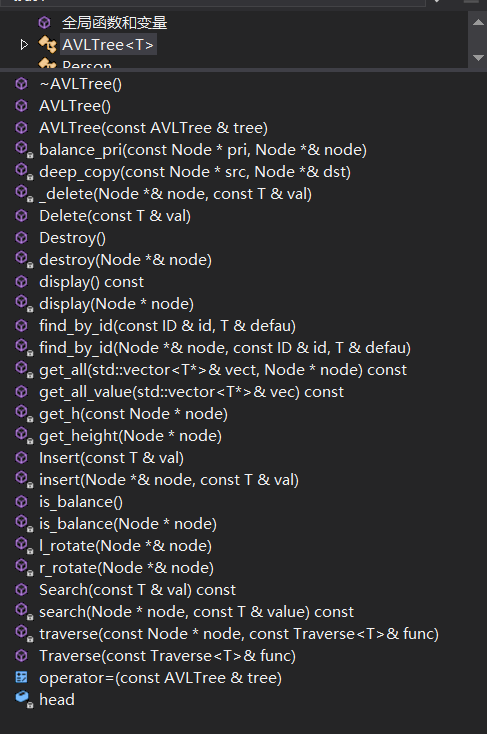


图-3

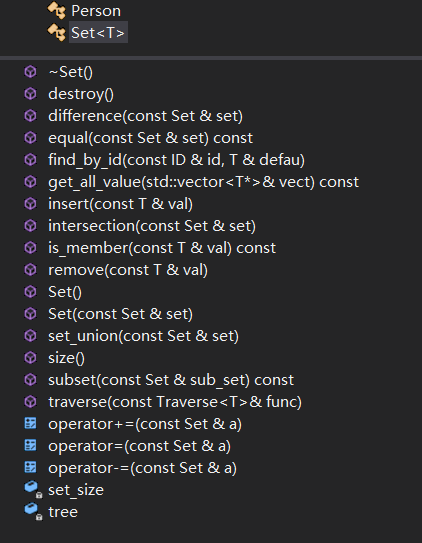


图-4

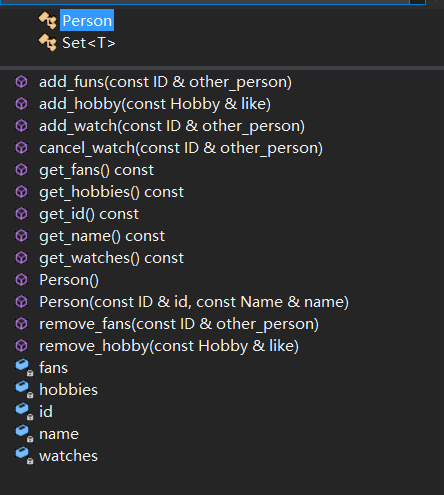


图-5

具体的代码可见附录文件或者网址[CourseDesign](https://github.com/hustr/CourseDesign)。

## 4.2系统测试

TODO首先叙述一下常用的软件测试方法，在选择几个主要的功能模块（自行掌握数量，关键要体现你的水平的一些模块）描述测试过程，（1）先明确模块的功能、设计目标等。（2）分析、叙述如何选取测试数据，要求有完整的测试大纲。（3）运行结果（这时可用截图）。（4）分析运行结果、确认程序满足该模块的设计目标。

常用的软件测试方法据我所知从测试设计方法设计上有白盒测试、黑盒测试灰盒测试，从测试的操作来看有手工测试和自动化测试，测试目的分类有单元测试、功能测试、集成测试、场景测试和系统测试。由于这次的课设由我一个人完成并且代码是我自己写的，所以我就只有针对代码可能出错的地方进行测试，这个程序的主要困难的地方就是AVLTree的插入删除时的平衡问题还有最终的用户操作时的数据操作问题。所以我决定直接写一个测试代码文件来测试AVLTree，而用户操作必须由我手动输入数据并且输入序号进行操作。

二叉树的测试流程：首先随机生成10000个int数据插入到std的set里面，再将set中数据插入到二叉树中，set可以保证数据是唯一的不会产生重合，如果插入失败或者插入后树失去平衡返回错误测试结束，完成插入后进行下一步，删除测试，使用set的迭代器删除set中一半数据，如果删除失败或者树失去平衡返回错误并结束测试，删除测试完毕测试查找，查找共进行20000次，如果二叉树的查找结果与set的查找结果不同则返回错误，最后若全部没有问题表示程序没有问题，输出测试完毕的指示并退出测试。

对二叉树的插入删除查找操作测试结果如图-6所示：

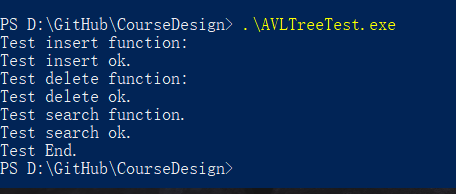


图-6

可见二叉树的操作没有问题。

用户操作的测试流程：这个测试时直接进行主程序的运行，检测程序的稳定性和是否可用，测试主要是测试各个功能的使用。

测试的数据集我是用Python脚本随机生成的，数据的数量可以根据程序的参数进行调整，其中的关系集合可以保证正确，文件和脚本见附带的文件。

下面进行测试：

首先从文件中加载数据，文件名为“data.txt”，加载结果如图-7所示：

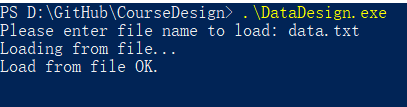


图-7

加载成功，按任意键进入菜单，菜单如图-8所示：

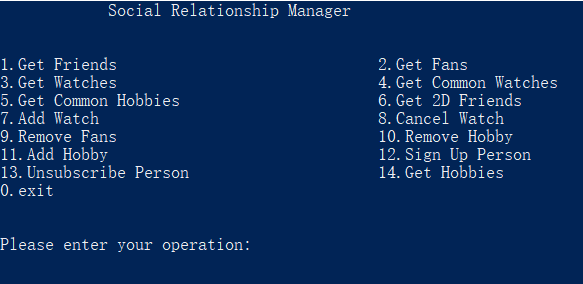


图-8

首先测试获取朋友，获取ID为1的人的所有朋友，结果如图-9所示：

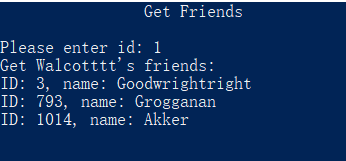


图-9

可知Walcotttt共有3名好友，ID分别为3、793和1014，测试Walcottt的粉丝集合并输出，如图-10所示：

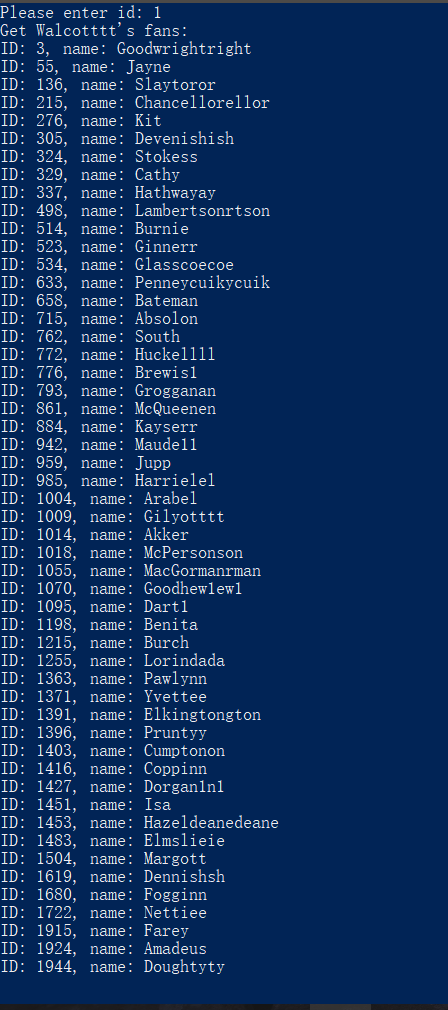


图-10

获取 1号的粉丝可见刚才的3、793和1014同样在粉丝集合中，由微博中好友的的定义可知正确，获取1号关注的人，结果如图-11所示：

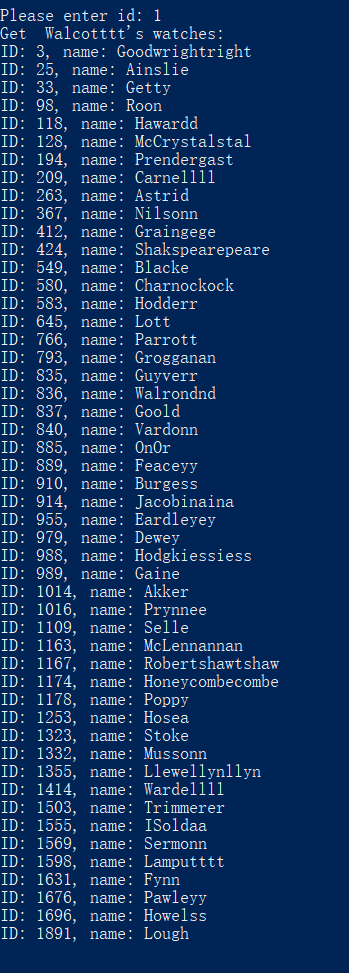


图-11

可见3、793和1014同样在1的关注集合中，关系成立，测试继续进行，获取1和2共同关注，结果如图-12所示：

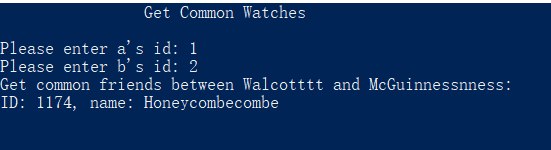


图-12

可见两者只有一个共同关注的人即ID为1174的Honeycombecombe，获取2的关注集合进行检查，获取2的关注集和如图-13所示：

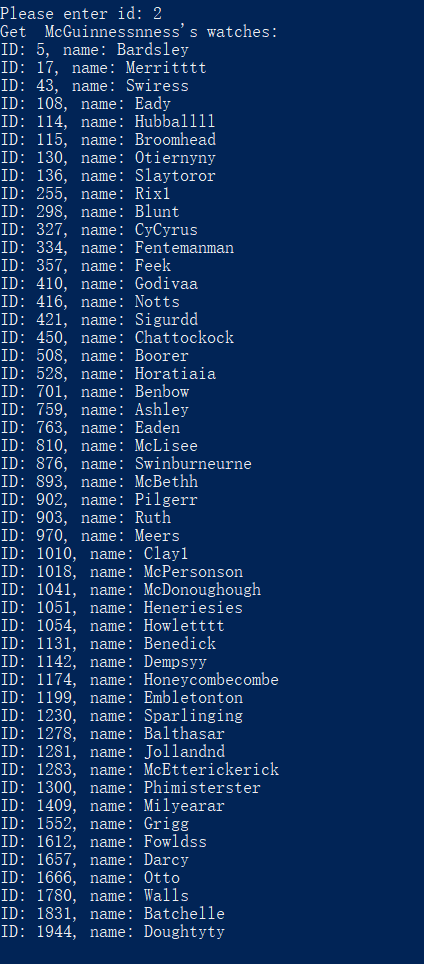


图-13

可见1174在2号的关注集合中，而且和1号的关注集合没有其它元素相同，所以共同关注功能暂且认为正确，下面测试获取共同爱好功能，获取1和2号的共同爱好，结果如图-14所示：

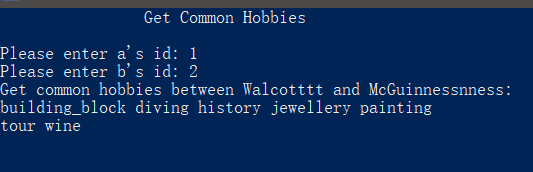


图-14

而在数据中的存储的1号与2号的各自的爱好如图-15和图-16所示：

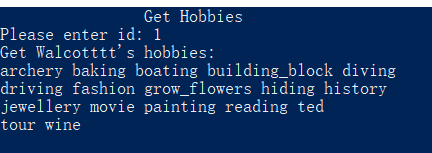


图-15

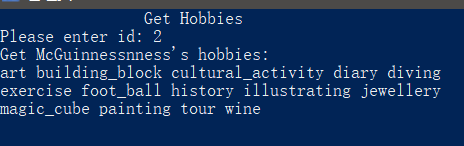


图-16

可见共同爱好获取的结果正确，测试二度好友功能，获取1号的二度好友，再分别获取1的好友3、793和1014的好友，结果如图-17，图-18，图-19，图-20所示：

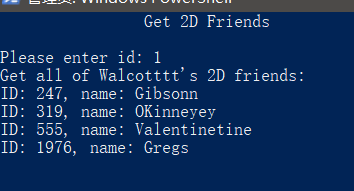


图-17

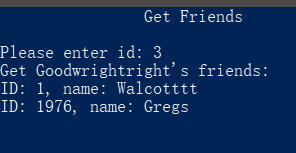


图-18

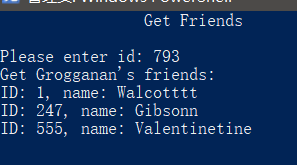


图-19

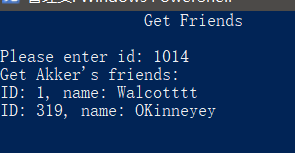


图-20

可见结果正确。接下来测试添加关注功能，为了测试方便可以先创建两个空帐号，添加结果如图-21和图-22所示：

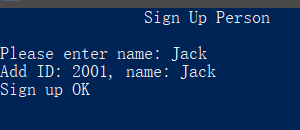


图-21

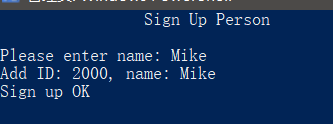


图-22

注册成功，下面进行添加关注测试，将ID2001加为ID2000的关注，插入结果如图-23，图-24，图-25所示：

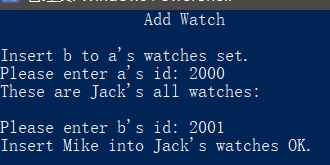


图-23

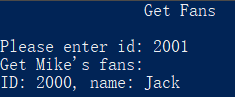


图-24

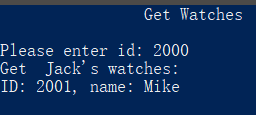


图-25

可知插入成功，下面测试取消关注，将2001从2000的关注集合中移除，结果如图-25，图-26，图-27所示：

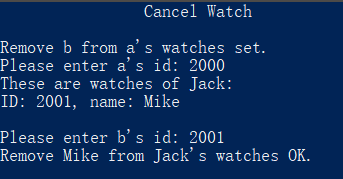


图-25

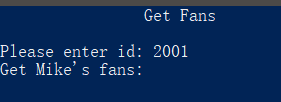


图-26

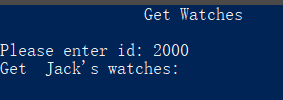


图-27

可知删除成功，下面测试移除粉丝功能，这个相当于粉丝在不知情的情况下取消关注，还是使用ID2000和ID2001测试，先添加ID2001到ID2000关注集中，然后测试ID2001移除粉丝ID2000，如图-28，图-29，图-30所示：

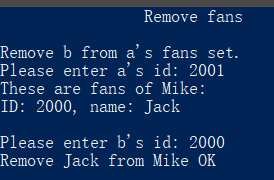


图2-28

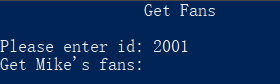


图2-29

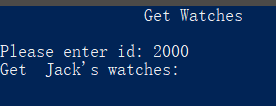


图2-30

可知粉丝移除成功，下面的爱好的操作和上面的粉丝集，关注集类似，在此不进行测试，下面测试注销账号功能，注销账号要求将与账号有关的所有信息删除，我的测试方式是将ID2001加入到ID2000的关注集合中，将ID2001账号注销，查看ID2000的关注集变化，测试时我已将ID2001加入ID2000关注集集合，测试结果如图-31，图-32所示：

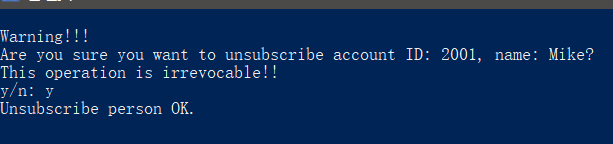


图-31

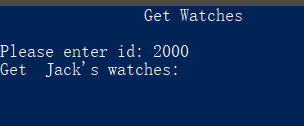


图-32

测试表示将ID2001注销账号后与其相关的信息也已经被清理干净了。

操作部分的测试到此结束。

测试完毕后未发现程序的错误，暂时可以认为程序正确。

# 5．总结与期望

## 5.1全文总结

实验报告的内容说多不多，最近在看软件工程的书，里面提倡的是文档引领程序，而实际上我们都是写完程序之后再考虑文档的事情，而且在编码的过程中没有考虑如何测试，实际工作中会使用的是TDD，先写测试再写具体代码，这样可以让程序开发中的陷阱少一些。

课程设计的代码量不是很大，而且由于我在课余时间学习了c++和Java等面向对象的语言，使用c++的模板类，写AVLTree和Set的时候不需要担心代码的重用性，所以还是很简单的。

## 5.2工作展望

希望自己以后能够对数据结构的掌握更加熟练，并将其应用到自己的各种程序中，虽然c++的stl提供的大量的实用类，但是自己掌握可以让它们的效率发挥的更好。还要学习一些软件工程的基本方法，测试的具体方式，为以后写程序打基础。

# 6．体会

课程设计中使用了二叉树的这个数据结构，根据实际中的测试结果可以看到当文件较大时会花费很多时间创建二叉树，推测原因时c++的new运算比较慢，而且c++是靠程序员自己把握内存的分配和释放，实际效率是肯定不如Java和Go一样有内存回收机制的语言，拥有极其先进的内存分配算法和释放策略。

通过这次的课程设计我也开始重视测试这一部分，拥有良好的测试代码编写习惯将会大大减少后期的工作量，bug也会更加容易的修复，debug对于不熟悉的人毕竟还是很痛苦的，在没有一个很好的调试工具的时候也是很难快速找到bug的。

还有就是代码可读性的重要性，一段优秀的代码应该想散文一样，让人有都下去的意愿，变量名需要有意义，函数名遵循自己的规则，不要随意改动，这样才能保证不会被变量名搞得云里雾里，我也看过有些人的代码，变量和操作符括号之间的空格，文件的及时关闭，变量类型的定义模糊，函数过大时的割分，注释的风马牛不相及，这些都是可读性的杀手，希望我的同学们在写代码过程中能够考虑到可读性的问题。

最后的就是各种类型的封装，类型之间的低耦合，类型自己的高内聚，这次课设的代码中我将AVLTree的最底层元素通过一个指针数列提取出来，这样做是非常危险的，可是现在还没有想到好的解决的方法，在AVLTree的查找中也用到了<和==在ID和T之间的重载，造成了T和ID之间的耦合，两者之间本来不应该有大小关系的，但是要通过ID查找一个Person的话就必须这么做，这个也很纠结。

本次的课设帮我极大的认识到了自己的知识薄弱点，希望以后自己可以通过不断的学习来增强自己的能力。

# 附录

## 参考文献

[1] 严蔚敏, 吴伟民. 数据结构（C语言版）. 北京: 清华大学出版社,1997

[2] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[3] Lin Chen. O(1) space complexity deletion for AVL trees, Information Processing Letters, 1986, 22(3)：147-149

[4] S.H. Zweben, M. A. McDonald.An optimal method for deletion in one-sided height-balanced trees, Communications of the ACM, 1978, 21(6): 441-445

[5] Guy Blelloch. Principles of Parallel Algorithms and Programming, CMU, 2014

## 程序代码

见附带的文件或者进入网址[CourseDesign](https://github.com/hustr/CourseDesign)。