**《数据结构》大作业报告**

一、项目功能

本大作业实现了一个简单的数据库，支持数据的单点以及区间的增、删、改、查等操作。实现的具体功能有：

1. 单点数据的操作

单点数据操作的操作对象仅为单条数据。这类操作包括：

（1）插入：插入一条数据至已有数据库中。操作指令中给出要插入的数据的具体信息。

（2）删除：删除一条数据

（3）数据值的设置：单独设置已有的某一条数据的某个属性的值

（4）数据值的增加：单独使某一条已有数据的值增加一个特定值。

（5）数据查询。支持一下三种查询方式：按ID查询、按name查询、按数据中的某个属性的值（范围）查询。

2. 区间数据的操作

区间数据操作的操作对象为某个ID区间的多条数据。这类操作包括：

（1）区间的删除：删除某个ID区间的数据

（2）区间数据值设置：设置某个ID区间的数据的某个属性的值

（3）区间数据值增加：增加某个ID区间的数据的某个属性的值

（4）区间数据查询：在某个ID区间上按某个属性的值的范围来查询数据：如QUERY 6921 27345 english = 4

（5）区间属性求和：在某个ID区间上对某个属性的值进行求和

3. 属性值排名  
 这类操作给出一个特定区间范围内某个属性的第k大（第k小）或前k大（前k小）的数据。

4. 集合操作。

该类操作包括：

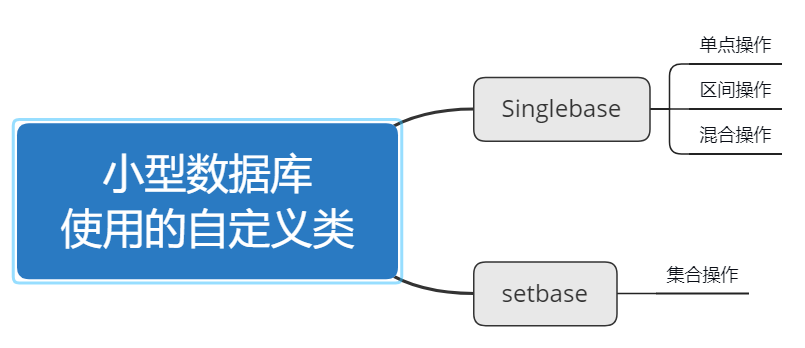
（1）并操作：给出两张数据表，输出它们的并。

（2）交操作：给出两张数据表，输出它们的交。

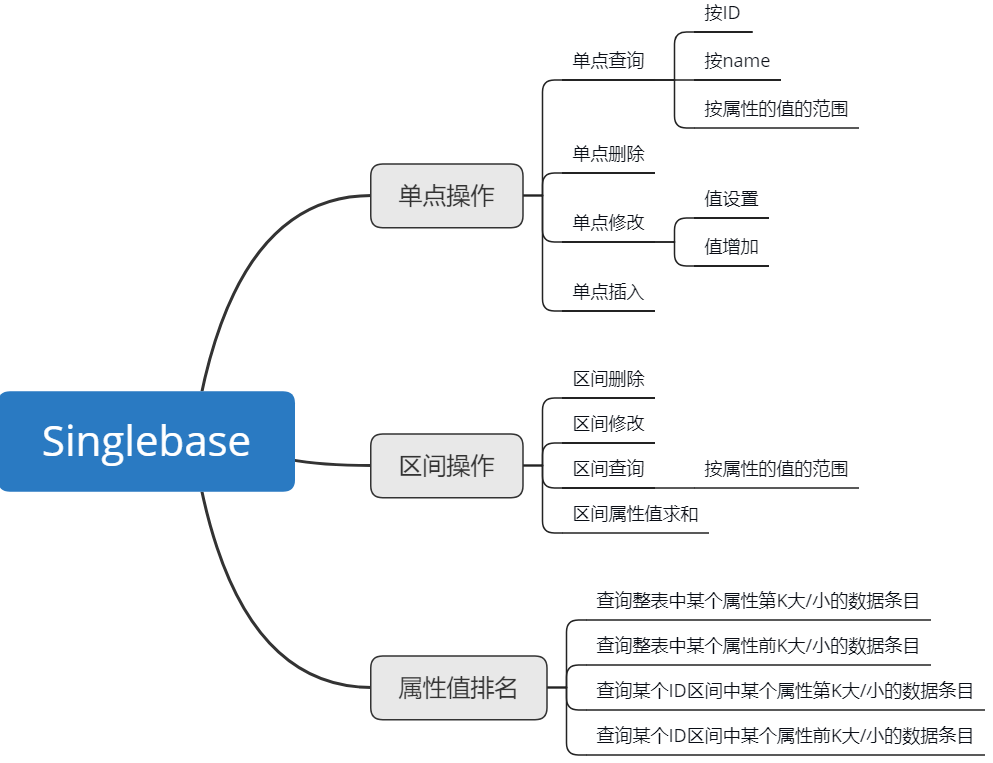
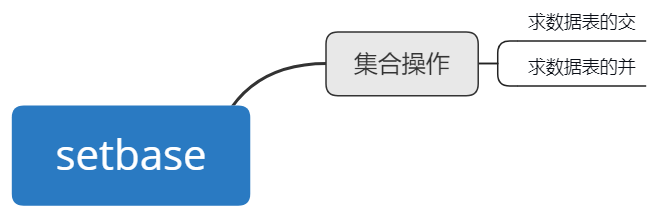
其它功能上的细节，如输出的格式，均依照实验要求来实现。

二、设计框架

各个操作使用的类：

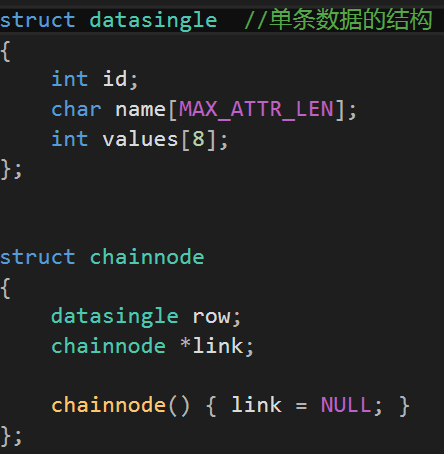


各个类完成的工作：



三、模块的详细设计

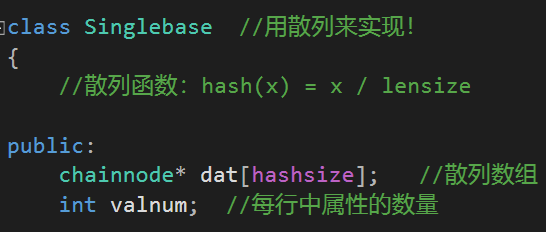
Singlebase类：采用散列表实现。



ID的范围是0~999999，共1000000个。将这1000000个ID哈希到一个散列表中，散列函数如下：



其中x为ID的值，lensize = 125. 这个散列函数将1000000个可能的ID值散列到hashsize(8000)个桶中。采用开散列法处理冲突。



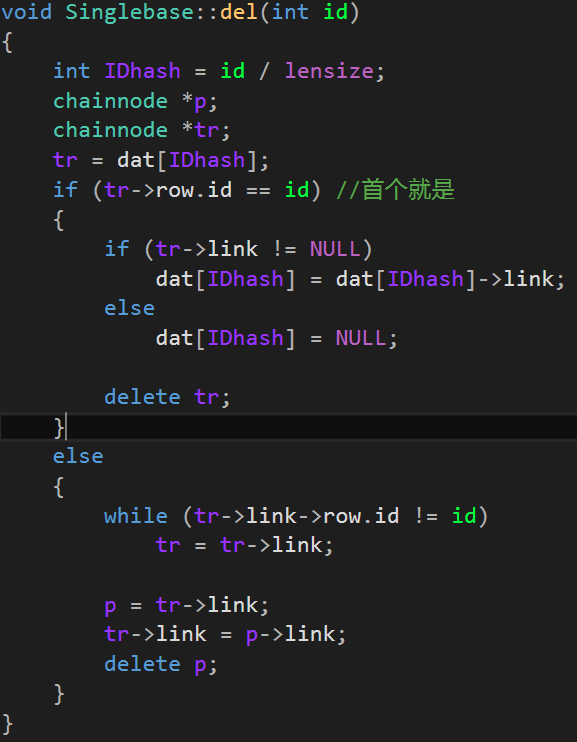
单点插入操作：计算得到的数据的ID的散列值，到相应的桶中进行寻找。若该桶为空，则直接插入该桶；否则沿该桶保存的链进行寻找，找到合适的位置插入。合适的位置指保持该链上各条数据的ID为升序的位置。

在这样的散列函数和插入操作下，从第0号桶到第7998号桶，每桶中的所有数据的ID都小于下一桶中所有数据的ID；每桶中各个数据的ID也保持升序。这样一来，通过在调用load\_data()函数时每读入一行数据（一个数据条目）就进行一次插入操作，便得到了一个已经排好序的散列表。这对后续的操作大有裨益。

**单点操作**：

（1）插入：过程见上。复杂度：最好为O(1)，最坏为O(125)(桶内最多125个元素)

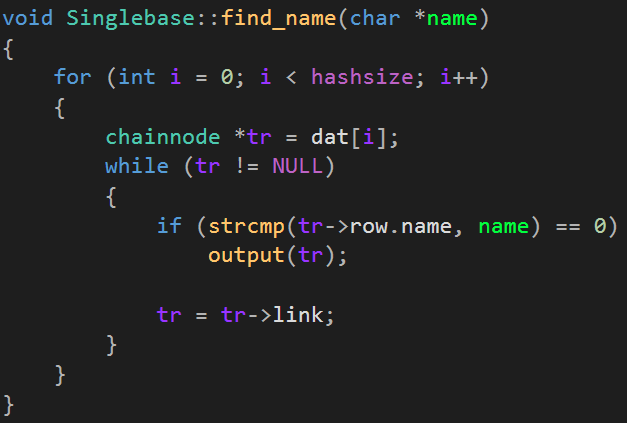
（2）删除：根据删除指令中的ID值，计算散列值，找到相应的桶，在桶中一一比对各条数据的ID，找到要删的数据条目后直接删除。复杂度：最好为O(1)，最坏为O(125)



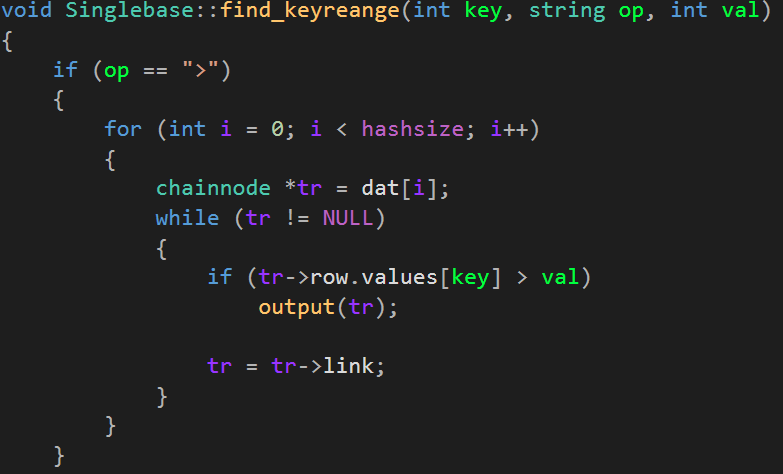
（3）查询：

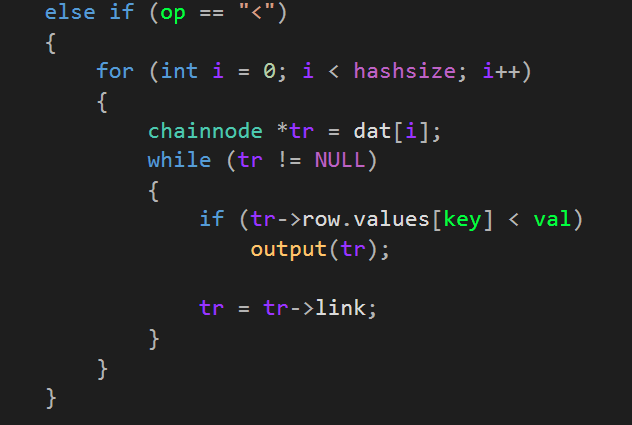
按ID：根据由ID得到的散列值直接查找。复杂度：最好为O(1)，最坏为O(125)

按name: 一一按顺序检索各个桶中的各个数据条目，发现其name属性与所给name相同时则输出该条目。复杂度：O(n)

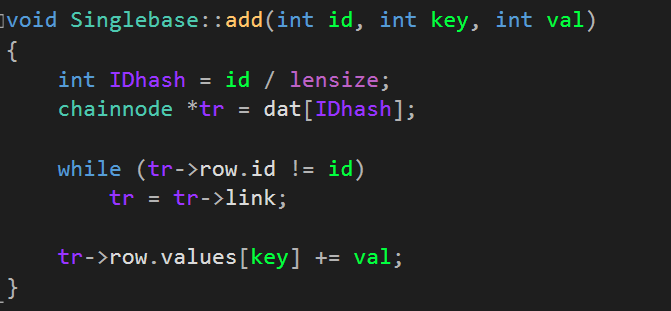


按特定属性的范围：提取出指令中的属性名、判断条件（大于、小于等）、值之后，一一按顺序检索各个桶中的各个条目，发现满足判断条件的条目后输出该条目。复杂度：O(n)





单点修改：由ID得到hash值后沿链查找到指定条目后，设置（set命令）或增加（add命令）该条目中相应属性的值。复杂度：最好为O(1)，最坏为O(125)



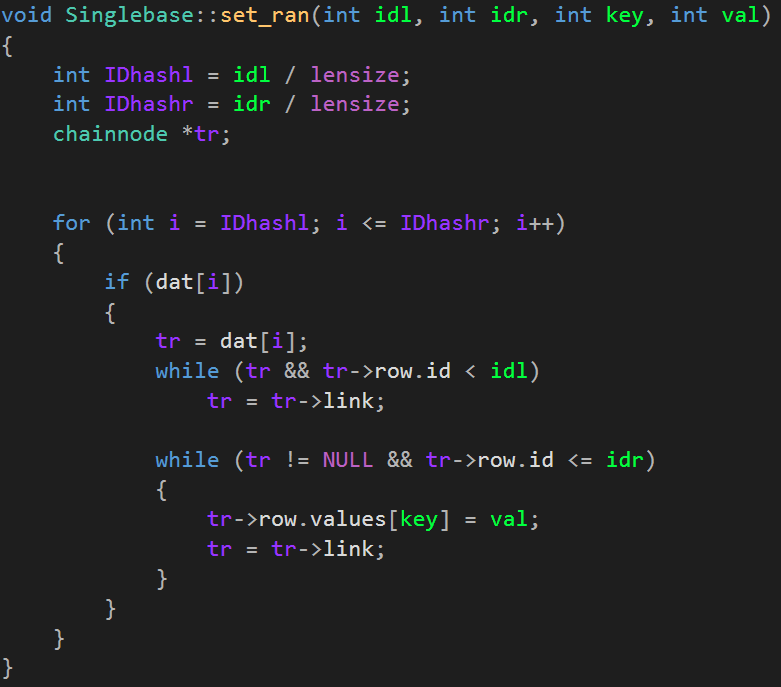
根据前面的分析，由于整个散列表在建立后呈现有序状态，且进行query检索时按桶的升序和桶内升序查找，找到指定的条目就立刻输出，故输出的数据是有序的（ID升序）。

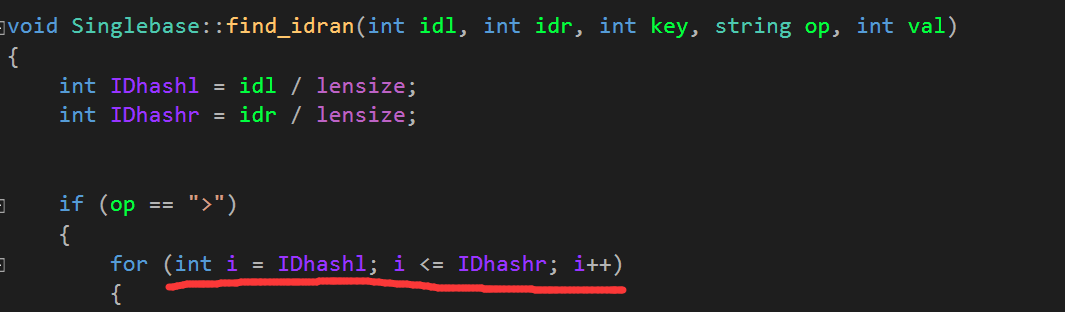
**区间操作：**

区间操作同样适用散列来完成，散列函数、散列结构与单点操作的完全一致。

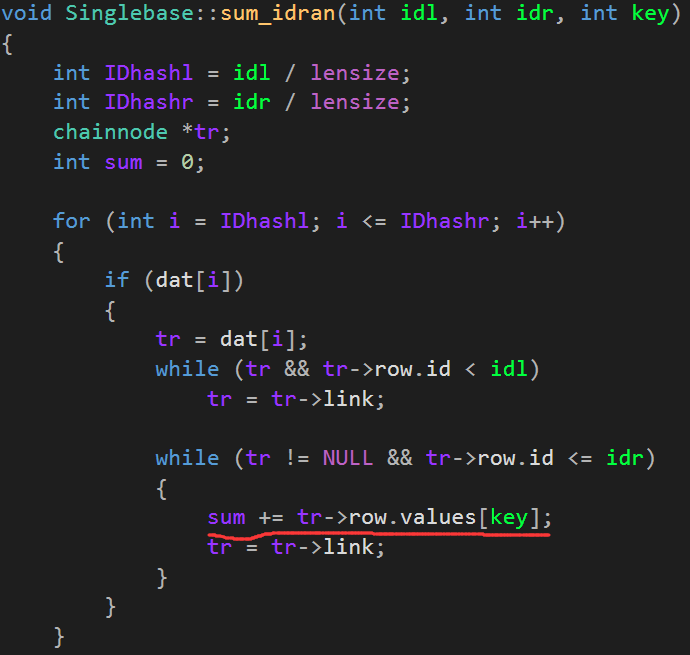
（1）区间删除：得到区间的2个端点idl和idr后，在散列表中逐一检索这个区间内的各个存在的表项，每检测到一项就删除一项。复杂度：O(idr - idl)

（2）区间修改：同区间删除，但每检测到一项就修改一项。修改操作可以是set或者是add. 复杂度：O(idr - idl)



（3）区间查询：与单点查询逻辑基本一致，但单点查询时要检索整个散列表，而区间查询只在某个区间内检索。复杂度：O(idr - idl)

（4）区间特定属性求和：思路与上述的区间操作一致，只要在散列表中遍历一遍区间内各个数据条目，每遍历到一个条目便把其内的某个属性的值加到某个sum变量，最后输出即可。复杂度：O(idr - idl)



**属性值排名：**

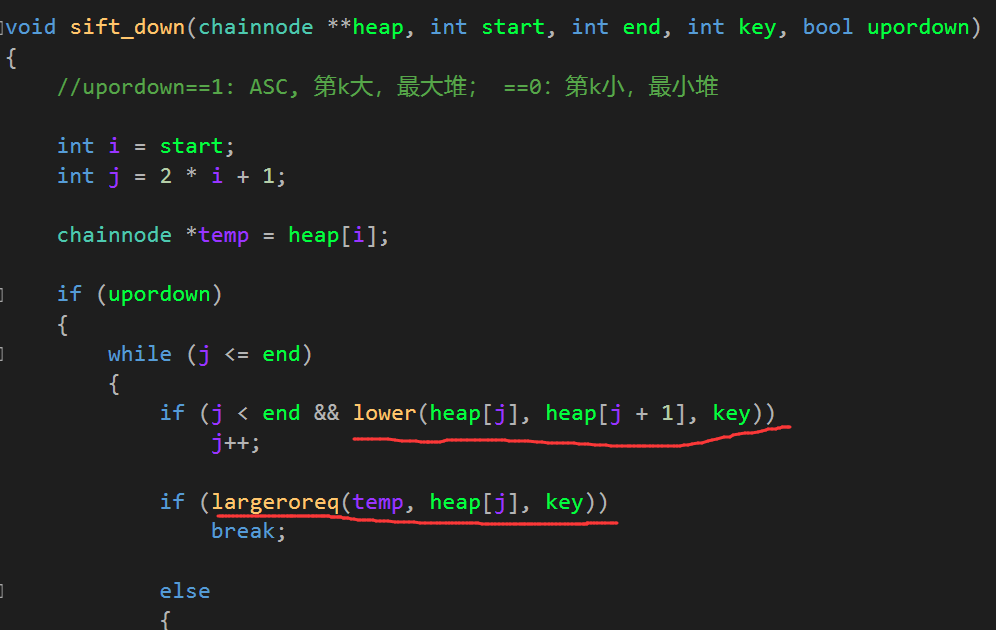
属性值排名使用了堆进行操作。把指向要排名的数据范围（若是给出了ID范围的指令，则数据范围是给出的ID范围，否则数据范围是整个散列表）中的各个数据条目的指针复制到一个指针数组中，然后把这个数组调整成堆。**根据求的是第/前k大还是第/前k小决定调整成最大堆还是最小堆。**调整时，由于数组中的元素是***指向数据条目结构体的指针***，且要求输出的数据条目按一定的规则进行排列，故元素之间的大于、小于等判断条件需自行定义，否则堆的调整过程中的元素交换会有问题。判断大于关系是否成立的基本过程是：首先比较两个数组元素指向的条目结构体中要排名的属性的值的大小，若满足大于关系则大于关系成立，若满足小于关系则大于关系不成立，若满足等于关系则再判断这两个条目结构体的ID值，满足一定条件则大于关系成立，否则不成立。根据这样的规则，可以判断出任意两个数组元素之间的大小关系，由此堆的调整便无障碍。

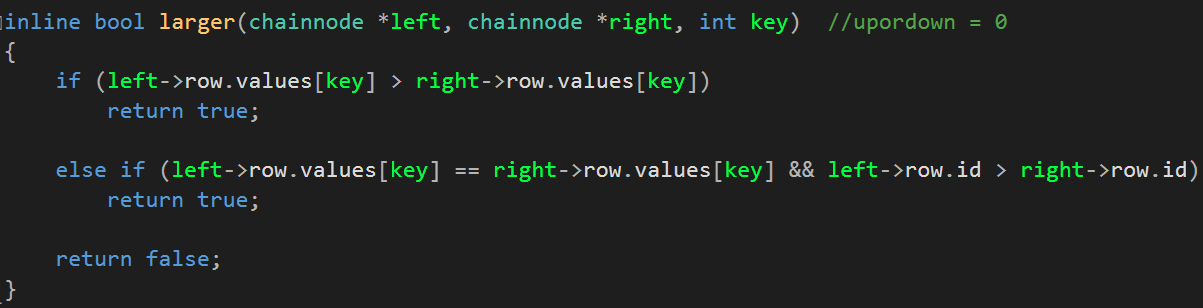
调整成堆后，进行“堆排序”。完整的堆排序需要进行n-1次调整（n为堆中元素个数），而这里不需要。我们只求第k大/小或前k大/小的条目，故只需进行K次堆排序中的调整即可。做完这k次调整之后，数组中倒数第k个元素便是指定范围内第k大/小的条目的指针，倒数前k个元素便是指定范围内前k大/小的条目的指针。此时根据这些指针输出即可。复杂度：复制：O(n); 调整成堆：O(n); k次调整：klog(n).

总复杂度：O(n + klog(n))

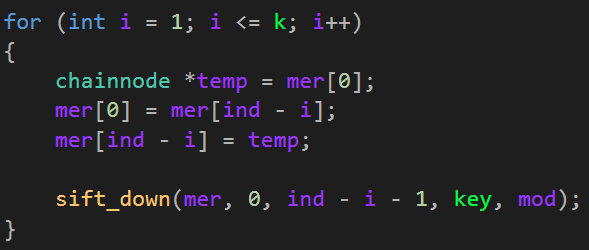
指针数组: 

调整函数sift\_down()和大小判断函数：

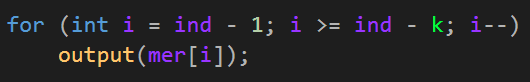


判断函数的判断细节：

进行k次堆排序中的调整：



输出：第k大/小：

输出：前k大/小：

**集合操作：**

集合操作采用散列表 + 单链表的结构来实现。在load\_data()函数中，将读到的数据表的条目同时存入一个散列表和一个单链表中。由于按name属性进行交、并操作，故散列函数应为字符串散列函数。这里我采用ELFHash函数作为字符串散列函数，将数据条目分配到10420个桶里面，采用开散列方式处理冲突。存入散列表的同时，将当前读入的条目链入一个单链表的末尾。注意，每读入一个新的条目时，都要检测该条目的name属性是否跟已在散列表中的某条目的name属性相同，若发现有则舍弃该条目，不加入散列表和单链表。也即散列表和单链表中不允许有name重复的条目。

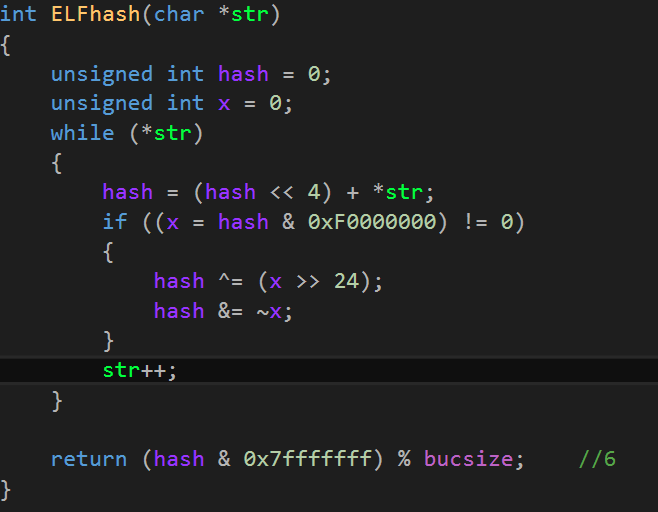
并操作：先沿保存table1的单链表逐一输出table1的全部条目。输出完后，沿保存table2的单链表逐一检查各元素的name是否在table1中出现过，若没有则输出该条目，否则舍弃。这样便按先table1后table2的顺序输出了table1与table2的并。复杂度：O(n1 + n2)

交操作：沿保存table1的单链表逐一检查各元素的name是否在table2中出现过，若有则输出该条目，否则舍弃。这样便按table1的编号顺序输出了table1与table2的交。复杂度：O(n1 + n2)

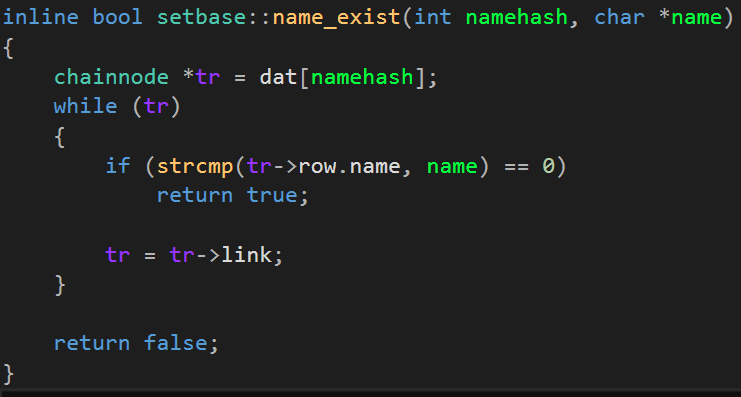
检查某个name是否在散列表中出现的过程运用了散列的快速性，以加快检查速度。

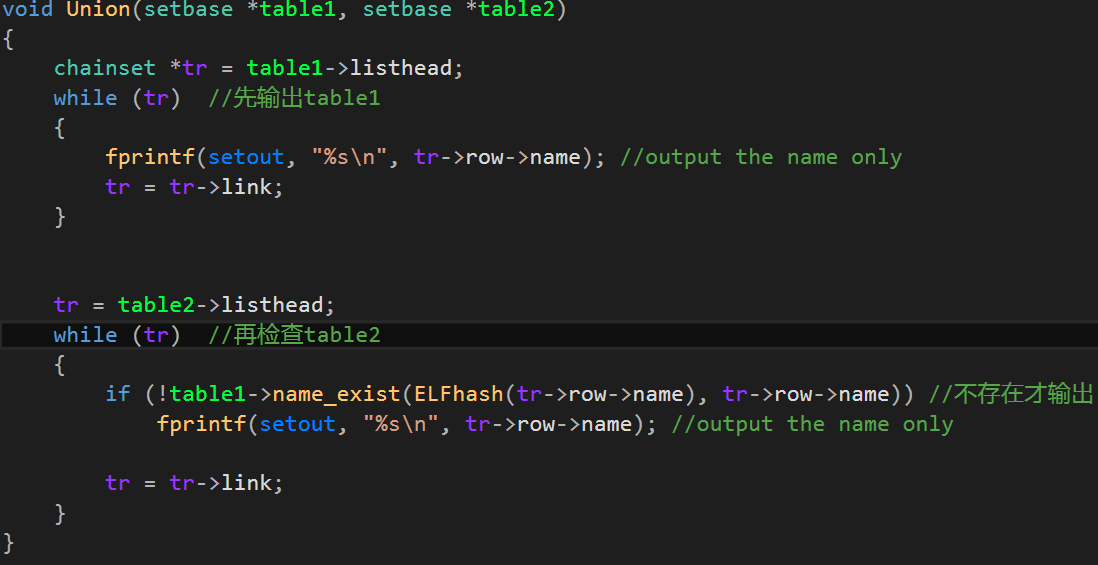
集合操作专用类和2个操作函数的定义：

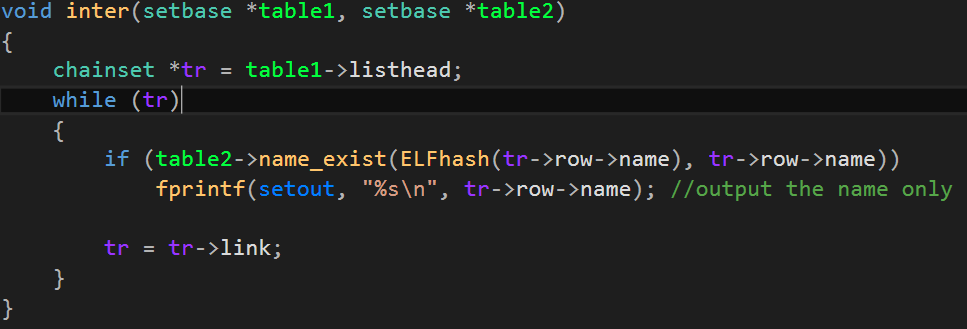
ELFHash()：bucsize为桶数，此处为10240.



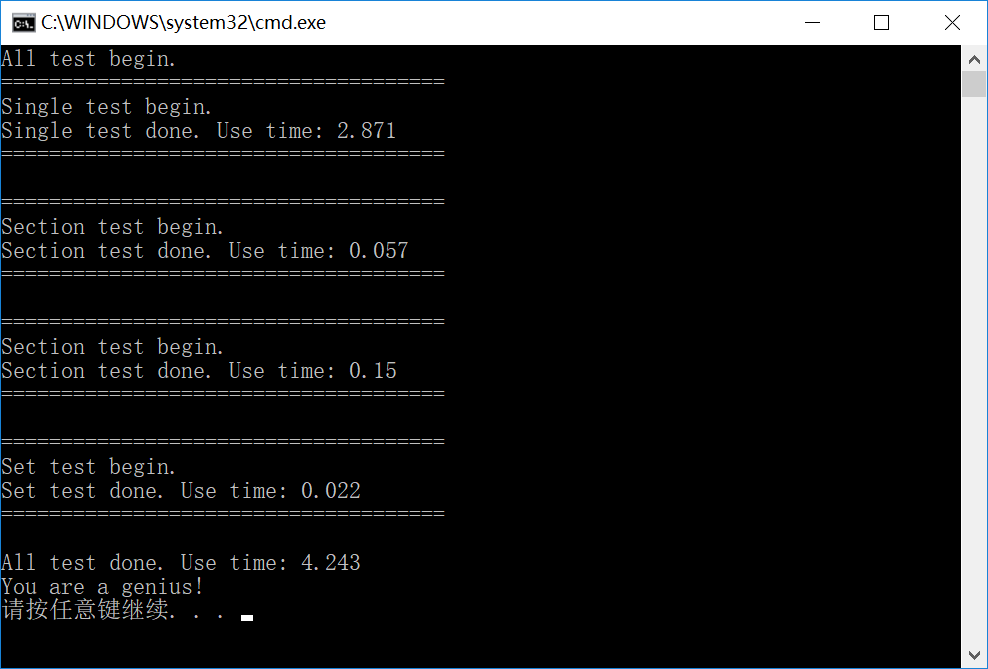
检查某个name是否在散列表中出现过的函数：namehash是根据name得到的hash值。

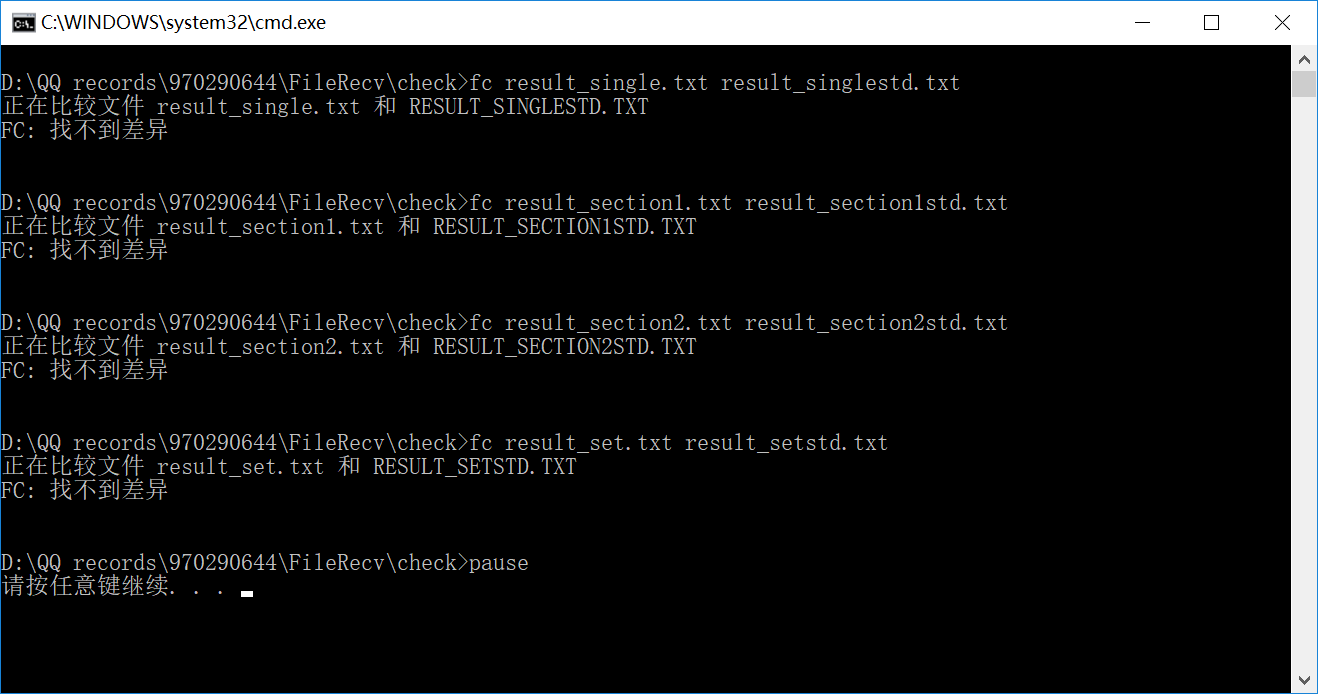


并操作：

交操作：

四、实现效果：



程序的输出文件与标准答案文件对比：

可见没有差异。

五、源代码清单：

common.h

common.cpp

database.h

database.cpp

test.h

test.cpp

main.cpp