STL面试

1.C++ STL 之所以得到广泛的赞誉，也被很多人使用，不只是提供了像vector, string, list等方便的容器，更重要的是STL封装了许多复杂的数据结构算法和大量常用数据结构操作。vector封装数组，list封装了链表，map和set封装了二叉树等

2.标准关联容器set, multiset, map, multimap内部采用的就是一种非常高效的平衡检索二叉树：红黑树，也成为RB树(Red-BlackTree)。RB树的统计性能要好于一般的[平衡二叉树](http://www.nist.gov/dads/HTML/avltree.html" \t "https://blog.csdn.net/u010002704/article/details/_blank)

3.STL map和set的使用虽不复杂，但也有一些不易理解的地方，如：

map: type [pair](http://dev.csdn.net/mk:@MSITStore:D:Program FilesMicrosoft Visual StudioMSDN2000JUL1033vclang.chm::/html/UTILITY_pair.htm" \l "pair" \t "https://blog.csdn.net/u010002704/article/details/_blank)<constKey, T>，很多不同的const Key对应的T对象的一个集合，所有的记录集中只要const Key 不一样就可以，T无关！  
set: type const Key. 只存单一的对const Key，没有map 的T对像！可以看成map的一个特例

（1）为何map和set的插入删除效率比用其他序列容器高？，树

答：因为对于关联容器来说，不需要做内存拷贝和内存移动。说对了，确实如此。map和set容器内所有元素都是以节点的方式来存储，其节点结构和链表差不多，指向父节点和子节点

（2）为何每次insert之后，以前保存的[iter](https://www.baidu.com/s?wd=iter&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/u010002704/article/details/_blank)ator不会失效？

答：iterator这里就相当于指向节点的指针，内存没有变，指向内存的指针怎么会失效呢(当然被删除的那个元素本身已经失效了)。相对于vector来说，每一次删除和插入，指针都有可能失效，调用push\_back在尾部插入也是如此。因为为了保证内部数据的连续存放，iterator指向的那块内存在删除和插入过程中可能已经被其他内存覆盖或者内存已经被释放了。即使时push\_back的时候，容器内部空间可能不够，需要一块新的更大的内存，只有把以前的内存释放，申请新的更大的内存，复制已有的数据元素到新的内存，最后把需要插入的元素放到最后，那么以前的内存指针自然就不可用了。特别时在和find等算法在一起使用的时候，牢记这个原则：不要使用过期的iterator。

（3）为何map和set不能像vector一样有个reserve函数来预分配数据？

答：我以前也这么问，究其原理来说时，引起它的原因在于在map和set内部存储的已经不是元素本身了，而是包含元素的节点。也就是说map内部使用的Alloc并不是map<Key, Data, Compare, Alloc>声明的时候从参数中传入的Alloc。例如：

****4.set, multiset****  
set和multiset会根据特定的排序准则自动将元素排序，set中元素不允许重复，multiset可以重复。

因为是排序的，所以set中的元素不能被修改，只能删除后再添加。  
向set中添加的元素类型必须重载<操作符用来排序。排序满足以下准则：  
1、非对称，若A<B为真，则B<A为假。  
2、可传递，若A<B,B<C，则A<C。  
3、A<A永远为假。

set中判断元素是否相等：  
if(!(A<B || B<A))，当A<B和B<A都为假时，它们相等。  
  
****5.map，multimap****  
map和multimap将key和value组成的pair作为元素，根据key的排序准则自动将元素排序，map中元素的key不允许重复，multimap可以重复。

map<key,value>

因为是排序的，所以map中元素的key不能被修改，只能删除后再添加。key对应的value可以修改。

向map中添加的元素的key类型必须重载<操作符用来排序。排序与set规则一致。

****6. List的功能方法****  
　　实际上有两种List: 一种是基本的ArrayList,其优点在于随机访问元素，另一种是更强大的LinkedList,它并不是为快速随机访问设计的，而是具有一套更通用的方法。   
　　List : 次序是List最重要的特点：它保证维护元素特定的顺序。List为Collection添加了许多方法，使得能够向List中间插入与移除元素(这只推荐LinkedList使用。)一个List可以生成ListIterator,使用它可以从两个方向遍历List,也可以从List中间插入和移除元素。

　　ArrayList : 由数组实现的List。允许对元素进行快速随机访问，但是向List中间插入与移除元素的速度很慢。ListIterator只应该用来由后向前遍历ArrayList,而不是用来插入和移除元素。因为那比LinkedList开销要大很多。   
　　LinkedList : 对顺序访问进行了优化，向List中间插入与删除的开销并不大。随机访问则相对较慢。(使用ArrayList代替。)还具有下列方法：addFirst(), addLast(), getFirst(),getLast(), removeFirst() 和 removeLast(), 这些方法 (没有在任何接口或基类中定义过)使得LinkedList可以当作堆栈、队列和双向队列使用

****7..1 hash\_map和map的区别在哪里？****

构造函数。hash\_map需要hash函数，等于函数；map只需要比较函数(小于函数).

存储结构。hash\_map采用hash表存储，map一般采用[红黑树(RB Tree)](http://stl.winterxy.com/html/000039.html" \t "https://blog.csdn.net/u010002704/article/details/_blank)实现。因此其memory数据结构是不一样的。

****7.2 什么时候需要用hash\_map，什么时候需要用map?****

总体来说，hash\_map 查找速度会比map快，而且查找速度基本和数据数据量大小，属于常数级别;而map的查找速度是log(n)级别。并不一定常数就比log(n)小，hash还有hash函数的耗时，明白了吧，如果你考虑效率，特别是在元素达到一定数量级时，考虑考虑hash\_map。但若你对内存使用特别严格，希望程序尽可能少消耗内存，那么一定要小心，hash\_map可能会让你陷入尴尬，特别是当你的hash\_map对象特别多时，你就更无法控制了，而且hash\_map的构造速度较慢。

现在知道如何选择了吗？权衡三个因素: 查找速度, 数据量, 内存使用。

****8.一些使用上的建议：****  
Level 1 - 仅仅作为Map使用：采用静态数组  
Level 2 - 保存定长数据，使用时也是全部遍历：采用动态数组（长度一开始就固定的话静态数组也行）  
Level 3 - 保存不定长数组，需要动态增加的能力，侧重于寻找数据的速度：采用vector  
Level 3 - 保存不定长数组，需要动态增加的能力，侧重于增加删除数据的速度：采用list  
Level 4 - 对数据有复杂操作，即需要前后增删数据的能力，又要良好的数据访问速度：采用deque  
Level 5 - 对数据中间的增删操作比较多：采用list，建议在排序的基础上，批量进行增删可以对运行效率提供最大的保证  
Level 6 - 上述中找不到适合的：组合STL容器或者自己建立特殊的数据结构来实现

9.

(1).vector - 会自动增长的数组

vector<int>vec(10) //一个有10个int元素的容器  
vector<float> vec(10, 0.5)//一个有10个float元素的容器，并且他们得值都是0.5  
vector<int>::iterator iter;  
for(iter = vec.begin(); iter != vec.end(); iter++)  
{  
　　　　//. . . . . . .   
}

vector由于数组的增长只能向前，所以也只提供了后端插入和后端删除，  
也就是push\_back和pop\_back。当然在前端和中间要操作数据也是可以的，  
用insert和erase，但是前端和中间对数据进行操作必然会引起数据块的移动，  
这对性能影响是非常大的。

最大的优势就是随机访问的能力。

vector<T1>::iterator相关的方法有：  
begin()：用来获得一个指向vector第一个元素的指针  
end()：用来获得一个指向vector最后一个元素之后的那个位置的指针，注意不是指向最后一个元素。  
erase(vector<T1>::iterator)：用来删除作为参数所传入的那个iterator所指向的那个元素。

(2).list - 擅长插入删除的链表

list<string>Milkshakes; list<int> Scores;

push\_back, pop\_backpush\_front. pop\_front

list是一个双向链表的实现。  
为了提供双向遍历的能力，list要比一般的数据单元多出两个指向前后的指针

一个使用iterator来删除元素的例子  
list<string> stringList;  
list<string>::iterator iter;  
advance(iter, 5); //将iterator指向stringList的第五个元素  
stringList.erase(iterator);//删除  
那么删除操作进行以后，传入erase()方法的iterator指向哪里了呢？它指向了删除操作前所指向的那个元素的后一个元素。

(3).deque - 拥有vector和list两者优点的双端队列

(4)．这三个模板的总结 比较和一般使用准则  
这三个模板都属于序列类模板，可以看到他们有一些通用方法  
size():得到容器大小  
begin():得到指向容器内第一个元素的指针（这个指针的类型依容器的不同而不同）  
end():得到指向容器内最后一个元素之后一个位置的指针  
erase():删除传入指针指向的那个元素  
clear():清除所有的元素  
＝＝运算符：判断两个容器是否相等  
＝运算符：用来给容器赋值  
上面的三个模板有各自的特点  
vector模板的数据在内存中连续的排列，所以随机存取元素（即通过[]运算符存取）的速度最快，这一点和数组是一致的。同样由于它的连续排列，所以它在除尾部以外的位置删除或添加元素的速度很慢，在使用vector时，要避免这种操作。  
list模板的数据是链式存储，所以不能随机存取元素。它的优势在于任意位置添加 删除元素的速度。  
deque模板是通过链接若干片连续的数据实现的，所以均衡了以上两个容器的特点