STL 中有用于操作迭代器的三个函数模板，它们是：

advance(p, n)：使迭代器 p 向前或向后移动 n 个元素。

distance(p, q)：计算两个迭代器之间的距离，即迭代器 p 经过多少次 + + 操作后和迭代器 q 相等。如果调用时 p 已经指向 q 的后面，则这个函数会陷入死循环。

iter\_swap(p, q)：用于交换两个迭代器 p、q 指向的值。

**迭代器的功能分类**

不同容器的迭代器，其功能强弱有所不同。容器的迭代器的功能强弱，决定了该容器是否支持 STL 中的某种算法。例如，排序算法需要通过随机访问迭代器来访问容器中的元素，因此有的容器就不支持排序算法。  
  
常用的迭代器按功能强弱分为输入、输出、正向、双向、随机访问五种，这里只介绍常用的三种。  
  
1) 正向迭代器。假设 p 是一个正向迭代器，则 p 支持以下操作：++p，p++，\*p。此外，两个正向迭代器可以互相赋值，还可以用==和!=运算符进行比较。  
  
2) 双向迭代器。双向迭代器具有正向迭代器的全部功能。除此之外，若 p 是一个双向迭代器，则--p和p--都是有定义的。--p使得 p 朝和++p相反的方向移动。  
  
3) 随机访问迭代器。随机访问迭代器具有双向迭代器的全部功能。若 p 是一个随机访问迭代器，i 是一个整型变量或常量，则 p 还支持以下操作：

* p+=i：使得 p 往后移动 i 个元素。
* p-=i：使得 p 往前移动 i 个元素。
* p+i：返回 p 后面第 i 个元素的迭代器。
* p-i：返回 p 前面第 i 个元素的迭代器。
* p[i]：返回 p 后面第 i 个元素的引用。

此外，两个随机访问迭代器 p1、p2 还可以用 <、>、<=、>= 运算符进行比较。p1<p2的含义是：p1 经过若干次（至少一次）++操作后，就会等于 p2。其他比较方式的含义与此类似。  
  
对于两个随机访问迭代器 p1、p2，表达式p2-p1也是有定义的，其返回值是 p2 所指向元素和 p1 所指向元素的序号之差（也可以说是 p2 和 p1 之间的元素个数减一）。  
  
表1所示为不同容器的迭代器的功能。

|  |  |
| --- | --- |
| 表1：不同容器的迭代器的功能 | |
| **容器** | **迭代器功能** |
| vector | 随机访问 |
| deque | 随机访问 |
| list | 双向 |
| set / multiset | 双向 |
| map / multimap | 双向 |
| stack | 不支持迭代器 |
| queue | 不支持迭代器 |
| priority\_queue | 不支持迭代器 |

|  |  |
| --- | --- |
| 表1：vector中常用的成员函数 | |
| **成员函数** | **作 用** |
| vector() | 无参构造函数，将容器初始化为空 |
| vector(int n) | 将容器初始化为有 n 个元素 |
| vector(int n, const T & val) | 假定元素的类型是 T，此构造函数将容器初始化为有 n 个元素，每 个元素的值都是 val |
| vector(iterator first, iterator last) | first 和 last 可以是其他容器的迭代器。一般来说，本构造函数初始化的结果就是将 vector 容器的内容变成与其他容器上的区间 [first, last) —致 |
| void clear() | 删除所有元素 |
| bool empty() | 判断容器是否为空 |
| void pop\_back() | 删除容器末尾的元素 |
| void push\_back( const T & val) | 将 val 添加到容器末尾 |
| int size() | 返回容器中元素的个数 |
| T & front() | 返回容器中第一个元素的引用 |
| T & back() | 返回容器中最后一个元素的引用 |
| iterator insert(iterator i, const T & val) | 将 val 插入迭代器 i 指向的位置，返回 i |
| iterator insert( iterator i, iterator first, iterator last) | 将其他容器上的区间 [first, last) 中的元素插入迭代器 i 指向的位置 |
| iterator erase(iterator i) | 删除迭代器 i 指向的元素，返回值是被删元素后面的元素的迭代器 |
| iterator erase(iterator first, iterator last) | 删除容器中的区间 [first, last) |
| void swap( vector <T> & v) | 将容器自身的内容和另一个同类型的容器 v 互换 |

|  |  |
| --- | --- |
| 表1：list 的成员函数 | |
| **成员函数或成员函数模板** | **作  用** |
| void push\_front(const T & val) | 将 val 插入链表最前面 |
| void pop\_front() | 删除链表最前面的元素 |
| void sort() | 将链表从小到大排序 |
| void remove (const T & val) | 删除和 val 相等的元素 |
| remove\_if | 删除符合某种条件的元素 |
| void unique() | 删除所有和前一个元素相等的元素 |
| void merge(list <T> & x) | 将链表 x 合并进来并清空 x。要求链表自身和 x 都是有序的 |
| void splice(iterator i, list <T> & x, iterator first, iterator last) | 在位置 i 前面插入链表 x 中的区间 [first, last)，并在链表 x 中删除该区间。链表自身和链表 x 可以是同一个链表，只要 i 不在 [first, last) 中即可 |

表1中列出的成员函数有些是重载的，如 unique、merge、splice 成员函数都不止一个， 这里不再一一列举并解释

deque 也是顺序容器的一种，同时也是一个可变长数组。要使用 deque，需要包含头文件 deque。所有适用于 vector 的操作都适用于 deque。  
  
deque 和 [vector](http://c.biancheng.net/view/348.html) 有很多类似的地方。在 deque 中，随机存取任何元素都能在常数时间内完成（但慢于vector）。它相比于 vector 的优点是，vector 在头部删除或添加元素的速度很慢，在尾部添加元素的性能较好，而 deque 在头尾增删元素都具有较好的性能（大多数情况下都能在常数时间内完成）。它有两种 vector 没有的成员函数：

void push\_front (const T & val);  //将 val 插入容器的头部  
void pop\_front();  //删除容器头部的元素

|  |  |
| --- | --- |
| 表1：multiset 的成员函数 | |
| **成员函数或成员函数模板** | **作  用** |
| iterator find (const T & val); | 在容器中查找值为 val 的元素，返回其迭代器。如果找不到，返 回 end() |
| iterator insert( const T & val); | 将 val 插入容器中并返回其迭代器 |
| void insert(iterator first, iterator last); | 将区间 [first, last) 中的元素插人容器 |
| int count( const T & val); | 统计有多少个元素的值和 val 相等 |
| iterator lower\_bound( const T & val); | 查找一个最大的位置 it，使得 [begin(), it) 中所有的元素者比 val 小 |
| iterator upper\_bound( const T & val); | 查找一个最小的位置 it，使得 [it, end()) 中所有的元素都比 val 大 |
| pair <iterator, iterator > equal\_range (const T & val); | 同时求得 lower\_bound 和 upper\_bound |
| iterator erase(iterator it); | 删除 it 指向的元素，返回其后面的元素的迭代器（Visual Studio 2010 中如此，但是在 [C++](http://c.biancheng.net/cplus/) 标准和 Dev C++ 中，返回值不是这样） |
| iterator erase(iterator first, iterator last); | 删除区间 [first, last)，返回 last（Visual Studio 2010 中如此，但是在 C++ 标准和 Dev C++ 中，返回值不是这样） |

multiset 及 set 中的 find 和 count 并不是用==运算符比较元素是否和待查找的值相等的。它们进行比较的原则是：如果x比y小和y比x小同时为假，就认为 x 和 y 相等。

multimap 的成员函数（未列出每个函数的所有版本）如表 1 所示。

|  |  |
| --- | --- |
| 表1：multimap 的成员函数 | |
| **成员函数或成员函数模板** | **作  用** |
| iterator find( const Key & val); | 在容器中查找关键字等于 val 的元素，返回其迭代器；如果找不到，返回 end() |
| iterator insert (pair <Key, T> const &p); | 将 pair 对象 p 插入容器中并返回其迭代器 |
| void insert(iterator first, iterator last); | 将区间 [first, last) 插入容器 |
| int count( const Key & val); | 统计有多少个元素的关键字和 val 相等 |
| iterator lower\_bound( const Key & val); | 查找一个最大的位置 it，使得 [begin( ), it) 中所有的元素的关键字都比 val 小 |
| iterator upper\_bound(const Key & val); | 查找一个最小的位置 it，使得 [it, end()) 中所有的元素的关键字都比 val 大 |
| pair < iterator, iterator > equal\_range (const Key & val); | 同时求得 lower\_bound 和 upper\_bound |
| iterator erase(iterator it); | 删除 it 指向的元素，返回其后面的元素的迭代器（Visual Studio 2010 中如此，但是在 C++ 标准和 Dev C++ 中，返回值不是这样） |
| iterator erase(iterator first, iterator last); | 删除区间 [first, last)，返回 last（Visual Studio 2010 中如此，但是在 C++ 标准和 Dev C++ 中，返回值不是这样） |

multimap 及 map 中的 find 和 count 不用==运算符比较两个关键字是否相等。如果x比y小和y比x小同时为假，就认为 x 和 y 相等。

虽然 stack 使用顺序容器实现，但它不提供顺序容器具有的成员函数。除了 size、 empty 这两个所有容器都有的成员函数外，stack 还有以下三个成员函数，如表 1 所示。

|  |  |
| --- | --- |
| 表1：stack的成员函数 | |
| **成员函数** | **功  能** |
| void pop(); | 弹出（即删除）栈顶元素 |
| T & top(); | 返回栈顶元素的引用。通过此函数可以读取栈顶元素的值，也可以修改栈顶元素 |
| void push (const T & x); | 将 x 压入栈顶 |

队列：先进先出（First in First out）

头文件必备：#include <queue>

接下来是各种常见的函数：定义一个queue的变量 queue<Type> M

查看是否为空 M.empty() 是的话返回1，不是返回0;

输出现有元素的个数 M.size()

显示第一个元素 M.front()

显示最后一个元素 M.back()

从已有元素后面增加元素 M.push()

清除第一个元素 M.pop()