第31条：

STL中的排序算法的选择：

STL排序算法要求随机访问迭代器，所以只能被应用于vector、string、deque、array、C-style array。

1. 如果需要执行一次完全排序，则可以使用sort或者stable\_sort；
2. 如果只需要对等价性最前面的n个元素进行排序，可以使用partial\_sort；
3. 需要找到等价性最前面的n个元素，但不必对这n个元素，可以使用nth\_element（在Visual studio中它会把整个序列排序）。

第32条：

调用STL的remove算法后，调用erase来实现真正的删除。

STL的remove算法接受的参数为迭代器，它不清楚元素的所在的容器，所以无法删除元素。

remove的删除过程：

例：程序effective\_stl\_test12

*vector*<int> vec;

vec.*reserve*(10);

for (int i = 1; i <= 10; ++i)

{

vec.*push\_back*(i);

}

vec[3] = vec[5] = vec[9] = 99;

*vector*<int>::*iterator* new\_end = std::*remove*(vec.*begin*(), vec.*end*(), 99);

*cout* << "size = " << vec.*size*() << *endl*;

PrintElements(vec, "After remove : ");

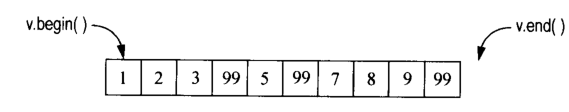
输出为：

size = 10

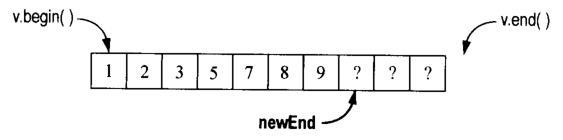
After remove : 1 2 3 5 7 8 9 8 9 99

从输出可以看出，remove之后，vector大小未发生改变。原因如下：

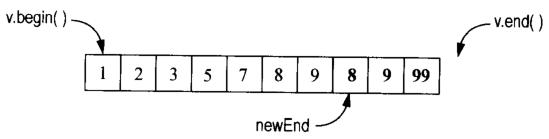
原始的vector：



remove之后：



一般的STL实现：



如果希望真正删除99，调用vector本身的erase区间函数来删除：从new\_end删除到容器本身的end。

例：程序effective\_stl\_test12

vec.*erase*(new\_end, vec.*end*());

*cout* << "size = " << vec.*size*() << *endl*;

PrintElements(vec, "After erase : ");

输出为：

size = 7

After erase : 1 2 3 5 7 8 9

第33条：

容器中存在指针元素，要尽量避免使用remove或remove\_if此类算法，如果避免不了，可以在调用erase\_remove算法之前，先把指针delete掉，并置为空。

第34条：

注意那些需要使用排序区间的算法，如果未使用排序区间有可能会产生无法预料的结果。

第35条：

通过lexicographical\_compare实行忽略大小写的字符串比较。

例：程序effective\_stl\_test13

bool CharLess(char c1, char c2)

{

return *tolower*(static\_cast<unsigned char>(c1))

< *tolower*(static\_cast<unsigned char>(c2));

}

bool StringCompare(const *string*& s1, const *string*& s2)

{

return *lexicographical\_compare*(s1.*begin*(), s1.*end*(),

s2.*begin*(), s2.*end*(),

CharLess);

}

*string* s1 = "lss";

*string* s2 = "Lss";

if (StringCompare(s1, s2))

{

*cout* << "Less" << *endl*;

}

else

{

*cout* << "Not less" << *endl*;

}

输出为：

Not less

第36条：

copy\_if算法的实现