移除重复元素：还有remove和remove\_if(不能用于无序容器)

1. 移除连续重复元素，unique，线性复杂度

Iterator unique(Iterator beg, Iterator end)

将[beg,end)区间内的所有“与前一元素相等”的元素移除。如果希望移除所有重复元素，则必须先进行排序。

Iterator unique(Iterator beg, Iterator end, BinaryPredicate op)

将每一个“位于元素e之后并造成binary predicate

op(e, elem)

结果为true”的所有elem元素移除。上述的predicate并非拿前一元素比较，而是拿它和“经过处理后仍健在”的元素进行比较。

返回最后一个未被移除元素的下一个位置。

注意：不能应用在associatvie容器和unordered容器。

例：程序stl\_test74

// unique

int source[] = {1, 4, 4, 6, 1, 2, 2, 3, 1, 6, 6, 6, 5, 7, 5, 4, 4};

*list*<int> coll;

*copy*(*begin*(source), *end*(source), *back\_inserter*(coll));

PrintElements(coll, "original coll: ");

auto pos = *unique*(coll.*begin*(), coll.*end*());

// 使用unique后，应该

// 使用新的逻辑终点

// 因为unique不会改变

// 元素个数

//PrintElements(coll, "after unique: ");

*cout* << "after unique" << *endl*;

*copy*(coll.*begin*(), pos, *ostream\_iterator*<int>(*cout*, " "));

*cout* << "\n\n";

*copy*(*begin*(source), *end*(source), coll.*begin*());

PrintElements(coll, "original coll: ");

// 删除所有比重复元素小的元素

coll.*erase*(*unique*(coll.*begin*(), coll.*end*(), *greater*<int>()), coll.*end*());

PrintElements(coll, "after unique: ");

输出为：

original coll: 1 4 4 6 1 2 2 3 1 6 6 6 5 7 5 4 4

after unique

1 4 6 1 2 3 1 6 5 7 5 4

original coll: 1 4 4 6 1 2 2 3 1 6 6 6 5 7 5 4 4

after unique: 1 4 4 6 6 6 6 7

变序型算法：

1. 反序，reverse，线性复杂度

注意：random-access iterator也是bidirectional iterator

void reverse(BidirectionalIterator beg, BidirectionalIterator end)

将[beg, end)区间内的元素全部反转次序

Iterator reverse\_copy(BidirectonalIterator sourceBeg,

BidirectionalIterator sourceEnd, Iterator destBeg)

将源区间[sourceBeg, sourceEnd)内的元素复制到“以destBeg起始的目标区间”，并在复制过程中反转次序。

例：程序stl\_test75

// reverse

*vector*<int> coll;

InsertElements(coll, 1, 9);

PrintElements(coll, "original coll: ");

*reverse*(coll.*begin*(), coll.*end*());

PrintElements(coll, "reverse coll: ");

*reverse*(coll.*begin*() + 1, coll.*end*() - 1);

PrintElements(coll, "coll: ");

*reverse\_copy*(coll.*cbegin*(), coll.*cend*(), *ostream\_iterator*<int>(*cout*, " "));

输出为：

original coll: 1 2 3 4 5 6 7 8 9

reverse coll: 9 8 7 6 5 4 3 2 1

coll: 9 2 3 4 5 6 7 8 1

1 8 7 6 5 4 3 2 9

1. 旋转元素，rotate，线性复杂度

Iterator rotate(Iterator beg, Iterator newBeg, Iterator end)

将[beg, end)区间内的元素旋转，执行后，\*newBeg成为新的第一元素。

自C++11，返回beg+(end - newBeg)，那是原本的第一元素经旋转后的新位置。

注意：newBeg必须是[beg, end)区间内的一个位置。

例：程序stl\_test75

// rotate

*vector*<int> coll1 = {1, 2, 3, 4, 5, 6, 7, 8, 9};

PrintElements(coll1, "coll1: ");

// rotate one element to the left

*rotate*(coll1.*begin*(), *next*(coll1.*begin*()), coll1.*end*());

PrintElements(coll1, "one left: ");

//rotate two elements to the right

*vector*<int>::*iterator* pos = coll1.*end*();

*advance*(pos, -2);

*rotate*(coll1.*begin*(), pos, coll1.*end*());

PrintElements(coll1, "two right: ");

输出为：

coll1: 1 2 3 4 5 6 7 8 9

one left: 2 3 4 5 6 7 8 9 1

two right: 9 1 2 3 4 5 6 7 8

1. 排列元素，线性复杂度

bool

next\_permutation(BidirectionalIterator beg, BidirectionalIterator end)

bool

next\_permutation(BidirectionalIterator beg, BidirectionalIterator end,

BinaryPredicate op)

改变[beg, end)区间内的元素次序，使它们符合“下一排列次序”。

第一形式使用operator <比较元素

第二形式使用binary predicate

op(elem1, elem2)

比较元素，如果“elem1小于elem2”它应该返回true

bool

prev\_permutation(BidirectionalIterator beg, BidirectionalIterator end)

bool

prev\_permutation(BidirectionalIterator beg, BidirectionalIterator end,

BinaryPredicate op)

改变[beg, end)区间内的元素次序，使它们符合“上一排列次序”

第一形式使用operator <比较元素

第二形式使用binary predicate

op(elem1, elem2)

比较元素，如果“elem1小于elem2”它应该返回true

如果元素得以排列成正规次序（字典顺序），则返回false。对于next\_permutation()而言，字典顺序为升序，对于prev\_permutation()，字典顺序为降序。

例：程序stl\_test75

// next\_permutation, prev\_permutation

*vector*<int> coll2 = {1, 2, 3};

PrintElements(coll2, "on entry: ");

// permute elements until they ar sorted

while (*next\_permutation*(coll2.*begin*(), coll2.*end*()))

{

PrintElements(coll2, " ");

}

PrintElements(coll2, "afterward: ");

*prev\_permutation*(coll2.*begin*(), coll2.*end*());

PrintElements(coll2, "now: ");

// permute elements until they are

// sorted in descending order

while (*prev\_permutation*(coll2.*begin*(), coll2.*end*()))

{

PrintElements(coll2, " ");

}

PrintElements(coll2, "afterward: ");

输出为：

on entry: 1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

afterward: 1 2 3

now: 3 2 1

3 1 2

2 3 1

2 1 3

1 3 2

1 2 3

afterward: 3 2 1

1. 划分为两个子区间，partition\_copy，线性复杂度

pair<Iterator1, Iterator2>

partition\_copy(Iterator sourceBeg,Iterator sourceEnd,

Iterator1 destTrueBeg, Iterator2 destFalseBeg,

UnaryPredicate op)

以predicate op()为依据，拆分[beg, end)区间内的所有元素成为两个子区间。

所有“造成unary predicate

op(elem)

产生true”的元素都被复制到始于destTrueBeg的区间内。所有“造成该predicate产生false”的元素都被复制到始于destFalseBeg的区间内。

返回一个pair，两个iterator分别指向destTrueBeg区间和destFalseBeg区间内的最末元素的下一个位置。

例：程序stl\_test75

// partition\_copy

*vector*<int> coll3 = {1, 6, 33, 7, 22, 4, 11, 33, 2, 7, 0, 42, 5};

PrintElements(coll3, "coll3: ");

// destination collections

*vector*<int> evenColl;

*vector*<int> oddColl;

// copy all elements partitioned accordingly

// into even and odd elements

*partition\_copy*(coll3.*cbegin*(), coll3.*cend*(), // source range

*back\_inserter*(evenColl), // destination for even elements

*back\_inserter*(oddColl), // destination for odd elements

[](int elem) // predicate: check for even elements

{

return elem % 2 == 0;

});

PrintElements(evenColl, "even coll: ");

PrintElements(oddColl, "odd coll: ");

输出为：

coll3: 1 6 33 7 22 4 11 33 2 7 0 42 5

even coll: 6 22 4 2 0 42

odd coll: 1 33 7 11 33 7 5