Predicate检验区间：

1. 检验是否排序（或局部排序），线性复杂度。

bool is\_sorted(Iterator beg, Iterator end)

bool is\_sorted(Iterator beg, Iterator end, BinaryPredicate op)

检验[beg, end)区间内的元素是否已经排序

Iterator is\_sorted\_until(Iterator beg, Iterator end)

Iterator is\_sorted\_until(Iterator beg, Iterator end, BinaryPredicate op)

返回[beg, end)区间内第1个破坏排序的元素。如果没有返回end。

例：程序stl\_test69

// is\_sorted, is\_sorted\_until

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

PrintElements(coll1, "coll1: ");

if (*is\_sorted*(coll1.*begin*(), coll1.*end*()))

{

*cout* << "coll1 is sorted" << *endl*;

}

else

{

*cout* << "coll1 is not sorted" << *endl*;

}

*map*<int, *string*> coll2 = { {1, "Bill"}, {2, "Jim"}, {3, "Nico"}, {4, "Liu"}, {5, "Ai"} };

PrintMappedElements(coll2, "coll2: ");

auto CompareName = [](const *pair*<int, *string*>& e1,

const *pair*<int, *string*>& e2) {

return e1.*second* < e2.*second*;

};

if (*is\_sorted*(coll2.*cbegin*(), coll2.*cend*(), CompareName))

{

*cout* << "names in coll2 are sorted" << *endl*;

}

else

{

*cout* << "names in coll2 are not sorted" << *endl*;

}

auto pos = *is\_sorted\_until*(coll2.*cbegin*(), coll2.*cend*(), CompareName);

if (pos != coll2.*end*())

{

*cout* << "first unsorted name: " << pos->*second* << *endl*;

}

输出为：

coll1: 1 1 2 3 4 5 6 7 8 9

coll1 is sorted

coll2: [1,Bill] [2,Jim] [3,Nico] [4,Liu] [5,Ai]

names in coll2 are not sorted

first unsorted name: Liu

1. 检验是否被分割：

bool is\_partitioned(Iterator beg, Iterator end, UnaryPredicate op)

判断[beg, end)区间内的元素是否被分割（所有符合predicate op()的元素被置于所有不符合的元素之前）

线性复杂度

Iterator partition\_point(Iterator beg, Iterator end, UnaryPredicate op)

返回[beg, end)第1个分割点的位置，如果没有，返回end。

例：程序stl\_test69

// is\_partitioned, partition\_point

*vector*<int> coll3 = {5, 3, 9, 1, 3, 4, 8};

PrintElements(coll3, "coll3: ");

auto IsOdd = [](int elem) {

return elem % 2 == 1;

};

if (*is\_partitioned*(coll3.*cbegin*(), coll3.*cend*(), IsOdd))

{

*cout* << "coll3 is partitioned" << *endl*;

// find first even element

auto pos1 = *partition\_point*(coll3.*cbegin*(), coll3.*cend*(), IsOdd);

*cout* << "first even element: " << \*pos1 << *endl*;

}

else

{

*cout* << "coll3 is not partitioned" << *endl*;

}

*cout* << *endl*;

*vector*<int> coll4 = {6, 4, 5, 3, 9, 1, 4, 8 };

PrintElements(coll4, "coll4: ");

auto IsEven = [](int elem) {

return elem % 2 == 0;

};

if (*is\_partitioned*(coll4.*cbegin*(), coll4.*cend*(), IsEven))

{

*cout* << "coll4 is partitioned" << *endl*;

}

else

{

*cout* << "coll4 is not partitioned" << *endl*;

}

// 区间不为分割的情况下，返回第1个分割点的位置

auto pos2 = *partition\_point*(coll4.*cbegin*(), coll4.*cend*(), IsEven);

*cout* << "first partitioned element: " << \*pos2 << *endl*;

输出为：

coll3: 5 3 9 1 3 4 8

coll3 is partitioned

first even element: 4

coll4: 6 4 5 3 9 1 4 8

coll4 is not partitioned

first partitioned element: 5

1. 检验是否形成Heap（最大值在第1位），线性复杂度

bool is\_heap(RandomAccessIterator beg, RandomAccessIterator end)

bool is\_heap(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

判断[beg, end)区间内的元素是否形成一个heap

RandomAccessIterator is\_heap\_until(RandomAccessIterator beg, RandomAccessIterator end)

RandomAccessIterator is\_heap\_until(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

返回区间[beg, end)内第1个“破坏排序是无法形成heap”的元素位置

注意：以上函数只能适用用随机访问迭代器，由vector, deque, array, string, c-style array提供。

例：程序stl\_test69

// is\_heap, is\_heap\_until

*vector*<int> coll5 = {9, 8, 7, 7, 7, 5, 4, 2, 1};

*vector*<int> coll6 = {5, 3, 2, 1, 4, 7, 9, 8, 6};

PrintElements(coll5, "coll5: ");

PrintElements(coll6, "coll6: ");

*cout* << *boolalpha* << "coll5 is heap: "

<< *is\_heap*(coll5.*cbegin*(), coll5.*cend*()) << *endl*;

*cout* << *boolalpha* << "coll6 is heap: "

<< *is\_heap*(coll6.*cbegin*(), coll6.*cend*()) << *endl*;

auto pos3 = *is\_heap\_until*(coll6.*cbegin*(), coll6.*cend*());

if (pos3 != coll6.*end*())

{

*cout* << "first non-heap element: " << \*pos3 << *endl*;

}

输出为：

coll5: 9 8 7 7 7 5 4 2 1

coll6: 5 3 2 1 4 7 9 8 6

coll5 is heap: true

coll6 is heap: false

first non-heap element: 4

1. 检验All、Any或None，线性复杂度

bool all\_of(Iterator beg, Iterator end, UnaryPredicate op)

判断[beg, end)区间内全部元素使unary predicate

op(elem)

返回true

bool any\_of(Iterator beg, Iterator end, UnaryPredicate op)

判断[beg, end)区间内至少1个元素使unary predicate

op(elem)

返回true

bool none\_of(Iterator beg, Iterator end, UnaryPredicate op)

判断[beg, end)区间内没有任何元素使unary predicate

op(elem)

返回true

例：程序stl\_test70

// all\_of, any\_of, none\_of

*vector*<int> coll;

InsertElements(coll, 1, 9);

PrintElements(coll, "coll: ");

auto IsEven = [](int elem) {

return elem % 2 == 0;

};

*cout* << *boolalpha* << "all even?: "

<< *all\_of*(coll.*cbegin*(), coll.*cend*(), IsEven) << *endl*;

*cout* << *boolalpha* << "any even?: "

<< *any\_of*(coll.*cbegin*(), coll.*cend*(), IsEven) << *endl*;

*cout* << "none even?: "

<< *none\_of*(coll.*cbegin*(), coll.*cend*(), IsEven) << *endl*;

输出为：

coll: 1 2 3 4 5 6 7 8 9

all even?: false

any even?: true

none even?: false