

程序设计实习

第二十一讲 标准模板库（三）

<http://ai.pku.edu.cn/cpp2007>

内容提要 – 算法

1. fill、remove、replace 系列算法
2. 数学算法
3. 排序和查找算法
4. 交换、合并等算法
5. 集合操作算法
6. 位操作算法
7. 作业

1. fill、remove、replace 系列算法

1.1 fill, fill_n, generate 与 generate_n

```
template<class FwdIt, class T>
void fill(FwdIt first, FwdIt last, const T& x);
template<class OutIt, class Size, class T>
void fill_n(OutIt first, Size n, const T& x);

template<class FwdIt, class Gen>
void generate(FwdIt first, FwdIt last, Gen g);
```

fill, fill_n: 将容器中某一区间的元素全部设为某值
比如:

```
vector<int> v(100);
fill(v.begin(), v.end(), 5); //将全部元素设为5

vector<char> vc(100);
fill_n(vc.begin(), 5, 'A'); //将 begin() 及其后5个元素设为 'A'
```

generate 和 generate_n

依次将容器中某一区间的值，设为 g()

```
template<class FwdIt, class Gen>
void generate(FwdIt first, FwdIt last, Gen g);

template<class OutIt, class Pred, class Gen>
void generate_n(OutIt first, Dist n, Gen g);
```

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
char nextLetter() {
    static char letter = 'A';
    return letter++;
}
```

```
main()
{
    vector<char> v(5);
    ostream_iterator<char> output(cout, "");
    generate(v.begin(), v.end(), nextLetter);
    copy(v.begin(), v.end(), output);
    cout << endl;
    generate_n(v.begin(), 5, nextLetter);
    copy(v.begin(), v.end(), output);
} // 输出:
A,B,C,D,E,
F,G,H,I,J,
```

1.2 remove, remove_if, remove_copy, remove_copy_if

template<class FwdIt, class T>

FwdIt **remove**(FwdIt first, FwdIt last, const T& val);

删除 [first,last) 中所有和 val 相等的元素，这里“删除”的意思是用该区间后面的元素替换，后面的元素往前移动的意思，因此该函数不会使容器里的元素真正减少。

返回值是迭代器，指向被修改后序列的终点，

即被修改后的序列是 [first, fwdIt)

如果被删除的是 [first,last) 中的最后一个元素，那么就等于什么操作都没做，因为 [first,last) 里没有元素可以往前移，来替换被删除元素了，此时返回值指向 last 的前一个元素。如果 [first,last) 中没有元素等于 val，那么 fwdIt 等于 last

例：

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
main(){
    ostream_iterator<int> output(cout, "");
    const int SIZE = 5;
    int a[SIZE] = { 1,2,3,4,5 };
    vector<int> v(a,a+SIZE);
    vector<int>::iterator i = remove(v.begin(), v.end(), 5);
```

```
    cout << "1) ";
    copy(v.begin(), i, output); cout << endl;
    i = remove(v.begin(), v.begin() + 1, 1);
    cout << "2) ";
    copy(v.begin(), i, output); cout << endl;
    cout << "3) ";
    copy(v.begin(), v.end(), output); cout << endl;
    i = remove(v.begin(), v.begin() + 2, 1);
    cout << "4) ";
    copy(v.begin(), v.end(), output); cout << endl;
}
```

输出：

```
1) 1,2,3,4,
2)
3) 1,2,3,4,5,
4) 2,2,3,4,5,
```

remove_if:

template<class FwdIt, class Pred> FwdIt
remove_if(FwdIt first, FwdIt last, Pred pr);

remove_if 和 remove 类似，不同之处在于，被删除的元素

e 必须是满足 pr(e) == true 的。

pr 可以是函数对象或函数

```
remove_copy
template<class InIt, class OutIt, class T>
OutIt remove_copy(InIt first, InIt last, OutIt x, const T& val);
```

将 [first,last) 中所有不等于 val 的元素, 拷贝到另一容器中从 x 开始的位置(会覆盖另一容器中已有的元素, 须确保另一容器足够长, 否则会出错)

返回值是另一容器的迭代器, 指向被拷贝序列的最后一个元素的后面。

如果操作在同一容器上进行, 则 [x, x + (last - first)) 不能和 [first, last) 有重叠, 否则会出错。

remove_copy 示例:

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
main(){
    ostream_iterator<int> output(cout, ",");
    const int SIZE = 5;
    int a[SIZE] = { 1,2,3,4,5 };
    vector<int> v(a,a+SIZE);
    vector<int> v2(10);
    vector<int>::iterator i =
    remove_copy(v.begin(), v.end(), v2.begin(), 5);
```

```
cout << "1) ";
copy( v2.begin(), i, output); cout << endl;
i = remove_copy(v.begin(), v.begin() + 1, v2.begin(), 1);
cout << "2) ";
copy( v2.begin(), i, output); cout << endl;
cout << "3) ";
copy( v2.begin(), v2.end(), output); cout << endl;
i = remove_copy(v.begin(), v.begin() + 2, v2.begin(), 1);
cout << "4) ";
copy( v2.begin(), i, output); cout << endl;
cout << "5) ";
copy( v2.begin(), v2.end(), output); cout << endl;
}
```

输出:

- 1) 1,2,3,4,
- 2)
- 3) 1,2,3,4,0,0,0,0,0,0,
- 4) 2,
- 5) 2,2,3,4,0,0,0,0,0,0,

```
remove_copy_if
template<class InIt, class OutIt, class Pred>
OutIt remove_copy_if(InIt first, InIt last, OutIt x, Pred pr);
```

将 [first,last) 中所有 pr(e) 为 false 的元素 e, 拷贝到另一容器中从 x 开始的位置(会覆盖另一容器中已有的元素, 须确保另一容器足够长, 否则会出错)

返回值是另一容器的迭代器, 指向被拷贝序列的最后一个元素的后面。

如果操作在同一容器上进行, 则 [x, x + (last - first)) 不能和 [first, last) 有重叠, 否则会出错。

1.3 replace, replace_if, replace_copy, replace_copy_if

```
template<class FwdIt, class T>
void replace(FwdIt first, FwdIt last, const T& vold, const T& vnew);
```

将 [first,last) 区间中, 所有等于 vold 的元素, 替换为 vnew

```
template<class FwdIt, class Pred, class T>
```

```
void replace_if(FwdIt first, FwdIt last, Pred pr, const T& val);
```

将 [first,last) 区间中, 所有等于 pr(e) == true 的元素 e, 替换为 val

```
template<class InIt, class OutIt, class T>
```

```
OutIt replace_copy(InIt first, InIt last, OutIt x, const T& vold,  
const T& vnew);
```

将 [first,last) 区间中的所有元素，拷贝到另一容器的从 x 开始的地方。拷贝过程中碰到等于 vold 的元素，则不拷贝 vold,而是拷贝 vnew 到目标容器。

返回值是个迭代器，指向目标容器中拷入序列的最后一个元素的后面

如果操作在同一容器上进行，则 [x, x + (last - first)) 不能和 [first, last) 有重叠，否则会出错。

```
template<class InIt, class OutIt, class Pred, class T>
```

```
OutIt replace_copy_if(InIt first, InIt last, OutIt x, Pred pr, const T&  
val);
```

将 [first,last) 区间中的所有元素，拷贝到另一容器的从 x 开始的地方。拷贝过程中碰到 pr(e) == true 的元素 e, 则不拷贝 e, 而是拷贝 val 到目标容器。

返回值是个迭代器，指向目标容器中拷入序列的最后一个元素的后面

如果操作在同一容器上进行，则 [x, x + (last - first)) 不能和 [first, last) 有重叠，否则会出错。

2 数学算法

2.1 random_shuffle :

```
template<class RanIt>
```

```
void random_shuffle(RanIt first, RanIt last);
```

```
template<class RanIt, class Fun>
```

```
void random_shuffle(RanIt first, RanIt last, Fun& f);
```

随机打乱[first,last) 中的元素，适用于能随机访问的容器

2.2 count:

```
template<class InIt, class T>
```

```
size_t count(InIt first, InIt last, const T& val);
```

计算[first,last) 中等于 val 的元素个数

3) count_if

```
template<class InIt, class Pred, class Dist>
```

```
size_t count_if(InIt first, InIt last, Pred pr);
```

计算[first,last) 中符合pr(e) == true 的元素 e 的个数

2.3 min_element:

```
template<class FwdIt>
```

```
FwdIt min_element(FwdIt first, FwdIt last);
```

返回[first,last) 中最小元素的迭代器,以“<”作比较器

```
template<class FwdIt, class Pred>
```

```
FwdIt min_element(FwdIt first, FwdIt last, Pred pr);
```

返回[first,last) 中最小元素的迭代器,以 pr 作比较器

2.4 max_element:

```
template<class FwdIt>
```

```
FwdIt max_element(FwdIt first, FwdIt last);
```

返回[first,last) 中最大(不小) 元素的迭代器,以“<”作比较器

```
template<class FwdIt, class Pred>
```

```
FwdIt max_element(FwdIt first, FwdIt last, Pred pr);
```

返回[first,last) 中最大(不小)元素的迭代器,以 pr 作比较器

2.5 accumulate

```
template<class InIt, class T>
```

```
T accumulate(InIt first, InIt last, T val);
```

对 [first,last) 区间中的每个迭代器 l,
执行 val = val + *l;

返回 val

```
template<class InIt, class T, class Pred>
```

```
T accumulate(InIt first, InIt last, T val, Pred pr);
```

对 [first,last) 区间中的每个迭代器 l,
执行 val=pr(val, *l), 返回 val

2.6 for_each

```
template<class InIt, class Fun>
```

```
Fun for_each(InIt first, InIt last, Fun f);
```

对[first,last) 中的每个元素 e, 执行 f(e), 要求 f(e) 不能
改变 e

2.7 transform

```
template<class InIt, class OutIt, class Unop>
```

```
OutIt transform(InIt first, InIt last, OutIt x, Unop  
uop);
```

对[first,last) 中的每个迭代器 l, 执行

uop(*l); 要求 uop(*l) 不得改变 *l 的值

本模板返回值是个迭代器, 即 x + (last-first)

x 可以和 first 相等

```
template<class InIt1, class InIt2, class OutIt, class Binop>
```

```
OutIt transform(InIt1 first1, InIt1 last1, InIt2 first2, OutIt x,  
Binop bop);
```

The second template function evaluates

* (x + N) = bop(* (first1 + N), * (first2 + N)) once for each
N in

the range [0, last1 - first1).

It then returns x + (last1 - first1). The call bop(* (first1 + N),
* (first2 + N)) must not alter either * (first1 + N) or * (first2
+ N).

```
#include <vector>
#include <iostream>
#include <numeric>
#include <list>
#include <algorithm>
using namespace std;
class CLessThen9 {
public:
    bool operator()(int n) { return n < 9; }
};
void outputSquare(int value) { cout << value * value << " "; }
int calculateCube(int value) { return value * value * value; }
```

```
main() {
    const int SIZE = 10;
    int a1[] = { 1,2,3,4,5,6,7,8,9,10};
    int a2[] = { 100,2,8,1,50,3,8,9,10,2 };
    vector<int> v(a1,a1+SIZE);
    ostream_iterator<int> output(cout, " ");
    random_shuffle(v.begin(),v.end());
    cout << endl << "1" ";
    copy( v.begin(),v.end(),output);
    copy( a2,a2+SIZE,v.begin());
    cout << endl << "2" ";
    cout << count(v.begin(),v.end(),8);
    cout << endl << "3" ";
    cout << count_if(v.begin(),v.end(),CLessThen9());
```

```

cout << endl << "4) ";
cout << * (min_element(v.begin(),v.end()));
cout << endl << "5) ";
cout << * (max_element(v.begin(),v.end()));
cout << endl << "6) ";
cout << accumulate(v.begin(),v.end(),0);//求和
cout << endl << "7) ";
for_each(v.begin(),v.end(),outputSquare);
vector<int> cubes(SIZE);
transform(a1,a1+SIZE,cubes.begin(),calculateCube);
cout << endl << "8) ";
copy( cubes.begin(),cubes.end(),output);
}

```

输出:

```

1) 5 4 1 3 7 8 9 10 6 2
2) 2
3) 6
4) 1
5) 100
6) 193
7) 10000 4 64 1 2500 9 64 81 100 4
8) 1 8 27 64 125 216 343 512 729 1000

```

3 排序和查找算法

3.1 find

```
template<class InIt, class T>
```

```
InIt find(InIt first, InIt last, const T& val);
```

The template function determines the lowest value of N in the

range [0, last - first) for which the predicate $*(first + N) == val$

is true. It then returns first + N. If no such value exists, the function returns last. It evaluates the predicate once, at most, for each N.

3.2 find_if

```
template<class InIt, class Pred>
```

```
InIt find_if(InIt first, InIt last, Pred pr);
```

The template function determines the lowest value of N in

the range [0, last - first) for which the predicate

$pred(*(first + N))$ is true. It then returns first + N. If no such value exists, the function returns last. It evaluates the predicate once, at most, for each N.

3.3 binary_search 折半查找, 要求容器已经有序

```
template<class FwdIt, class T>
```

```
bool binary_search(FwdIt first, FwdIt last, const T& val);
```

```
template<class FwdIt, class T, class Pred>
```

```
bool binary_search(FwdIt first, FwdIt last, const T& val,
    Pred pr);
```

The first template function determines whether a value of N exists in the range [0, last - first) for which $*(first + N)$ has [equivalent ordering](#) to val, where the elements designated by iterators in the range [first, last) form a sequence [ordered by](#) operator<. If so, the function returns true. If no such value exists, it returns false.

If FwdIt is a random-access iterator type, the function evaluates the ordering predicate $X < Y$ at most $\text{ceil}(\log(\text{last} - \text{first})) + 2$ times. Otherwise, the function evaluates the predicate a number of times proportional to last - first.

The second template function behaves the same, except that it replaces operator<(X, Y) with pr(X, Y).

```
#include <vector>
#include <bitset>
#include <iostream>
#include <numeric>
#include <list>
#include <algorithm>
using namespace std;
bool Greater10(int n)
{
    return n > 10;
}
```

```
main() {
    const int SIZE = 10;
    int a1[] = { 2,8,1,50,3,100,8,9,10,2 };
    vector<int> v(a1,a1+SIZE);
    ostream_iterator<int> output(cout," ");
    vector<int>::iterator location;
    location = find(v.begin(),v.end(),10);
    if( location != v.end()) {
        cout << endl << "1) " << location - v.begin();
    }
    location = find_if( v.begin(),v.end(),Greater10);
    if( location != v.end()) {
        cout << endl << "2) " << location - v.begin();
    }
}
```

```
sort(v.begin(),v.end());
if( binary_search(v.begin(),v.end(),9)) {
    cout << endl << "3) " << "9 found";
}
}
```

输出:

- 1) 8
- 2) 3
- 3) 9 found

3.4 lower_bound, upper_bound, equal_range

lower_bound:

```
template<class FwdIt, class T>
```

```
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val);
```

```
template<class FwdIt, class T, class Pred>
```

```
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val,
    Pred pr);
```

要求[first,last)是有序的,

查找[first,last)中的,最小的位置 FwdIt,使得[first,FwdIt) 中所有的元素都比 val 小

upper_bound

```
template<class FwdIt, class T>
```

```
FwdIt upper_bound(FwdIt first, FwdIt last, const T&
    val);
```

```
template<class FwdIt, class T, class Pred>
```

```
FwdIt upper_bound(FwdIt first, FwdIt last, const T& val,
    Pred pr);
```

要求[first,last)是有序的,

查找[first,last)中的,最大的位置 FwdIt,使得[FwdIt,last) 中所有的元素都比 val 大

equal_range

```
template<class FwdIt, class T>
```

```
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const
    T& val);
```

```
template<class FwdIt, class T, class Pred>
```

```
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const
    T& val, Pred pr);
```

要求[first,last)是有序的,

返回值是一个pair, 假设为 p, 则

[first,p.first) 中的元素都比 val 小

[p.second,last)中的所有元素都比 val 大

3.5 sort

```
template<class RanIt>
```

```
void sort(RanIt first, RanIt last);
```

```
template<class RanIt, class Pred> void sort(RanIt first, RanIt last, Pred pr);
```

The first template function reorders the sequence designated by iterators in the range [first, last) to form a sequence **ordered by** operator<. Thus, the elements are sorted in ascending order.

The function evaluates the ordering predicate $X < Y$ at most $\text{ceil}((\text{last} - \text{first}) * \log(\text{last} - \text{first}))$ times.

The second template function behaves the same, except that it replaces operator<(X, Y) with pr(X, Y).



sort 实际上是快速排序，时间复杂度 $O(n \log(n))$;

平均性能最优。但是最坏的情况下，性能可能非常差。

如果要保证“最坏情况下”的性能，那么可以使用

stable_sort

stable_sort 实际上是归并排序，特点是能保持相等元素之间的先后次序

在有足够存储空间的情况下，复杂度为 $n \log(n)$ ，否则复杂度为 $n \log(n) \log(n)$

stable_sort 用法和 sort 相同

排序算法要求随机存取迭代器的支持，所以list 不能使用排序算法，要使用list::sort



此外还有其他排序算法：

partial_sort：部分排序，直到前n个元素就位即可

nth_element：排序，直到第n个元素就位，并保证比第n个元素小的元素都在第n个元素之前即可

partition：改变元素次序，使得符合某准则的元素放在前面

...



3.6 堆排序

堆：一种二叉树，最大元素总是在堆顶上，二叉树中任何节

点的子节点总是小于或等于父节点的值

◆ 什么是堆？

n个记录的序列，其所对应的关键字的序列为 $\{k_0, k_1, k_2, \dots, k_{n-1}\}$ ，若有如下关系成立时，则称该记录序列构成一个堆。

$k_i \geq k_{2i+1}$ 且 $k_i \geq k_{2i+2}$ ，其中 $i=0, 1, \dots$

◆ 例如，下面的关键字序列构成一个堆。

96 83 27 38 11 9
y r p d f b k a c

堆排序的各种算法，如make_heap等，需要随机访问迭代器的支持



make_heap 函数模板

```
template<class RanIt>
```

```
void make_heap(RanIt first, RanIt last);
```

```
template<class RanIt, class Pred>
```

```
void make_heap(RanIt first, RanIt last, Pred pr);
```

The first template function reorders the sequence designated by iterators in the range [first, last) to form a heap **ordered by** operator<.

The function evaluates the ordering predicate $X < Y$ at most $3 * (\text{last} - \text{first})$ times.

The second template function behaves the same, except that it replaces operator<(X, Y) with pr(X, Y).



push_heap 函数模板

```
template<class RanIt>
```

```
void push_heap(RanIt first, RanIt last);
```

```
template<class RanIt, class Pred>
```

```
void push_heap(RanIt first, RanIt last, Pred pr);
```

The first template function reorders the sequence designated by iterators in the range [first, last) to form a new heap **ordered by** operator<. Iterators in the range [first, last - 1) must designate an existing heap, also ordered by operator<. Thus, first != last must be true and *(last - 1) is the element to add to (push on) the heap.

复杂度： $O(\log(n))$



The function evaluates the ordering predicate $X < Y$ $\text{ceil}(\log(\text{last} - \text{first}))$ times, at most.

The second template function behaves the same, except that it replaces $\text{operator}<(X, Y)$ with $\text{pr}(X, Y)$.

往已经是堆的容器中添加元素，可以在每次 `push_back` 一个元素后，再调用 `push_heap` 算法

`pop_heap` 函数模板

```
template<class RanIt>
```

```
void pop_heap(RanIt first, RanIt last);
```

```
template<class RanIt, class Pred>
```

```
void pop_heap(RanIt first, RanIt last, Pred pr);
```

The first template function reorders the sequence designated by iterators in the range `[first, last)` to form a new heap, [ordered by](#) `operator<` and designated by iterators in the range `[first, last - 1)`, leaving the original element at `*first` subsequently at `*(last - 1)`. The original sequence must designate an existing heap, also ordered by `operator<`. Thus, `first != last` must be true and `*(last - 1)` is the element to remove from (pop off) the heap.

The function evaluates the ordering predicate $X < Y$ $\text{ceil}(2 * \log(\text{last} - \text{first}))$ times, at most.

The second template function behaves the same, except that it replaces $\text{operator}<(X, Y)$ with $\text{pr}(X, Y)$.

将堆中的最大元素，即 `*first`，移到 `last - 1` 位置，原 `*(last - 1)` 被移到前面某个位置，并且移动后 `[first, last - 1)` 仍然是个堆

要求原 `[first, last)` 就是个堆

复杂度 $O(n \log(n))$

4. `swap`, `iter_swap`, `swap_ranges`

```
template<class T>
```

```
void swap(T& x, T& y);
```

交换两个元素的值

```
template<class FwdIt1, class FwdIt2>
```

```
void iter_swap(FwdIt1 x, FwdIt2 y);
```

交换迭代器 `x` 和 `y` 所指向的元素的值

```
template<class FwdIt1, class FwdIt2>
```

```
FwdIt2 swap_ranges(FwdIt1 first, FwdIt1 last, FwdIt2 x);
```

The template function evaluates [swap](#)(`*(first + N)`, `*(x + N)`) once for each `N` in the range `[0, last - first)`. It then returns `x + (last - first)`. If `x` and `first` designate regions of storage, the range `[x, x + (last - first))` must not overlap the range `[first, last)`.

`copy_backward`, `merge`, `unique`, `reverse`

```
template<class BidIt1, class BidIt2>
```

```
BidIt2 copy_backward(BidIt1 first, BidIt1 last, BidIt2 x);
```

The template function evaluates `*(x - N - 1) = *(last - N - 1)` once for each `N` in the range `[0, last - first)`, for strictly decreasing values of `N` beginning with the highest value. It then returns `x - (last - first)`. If `x` and `first` designate regions of storage, `x` must not be in the range `[first, last)`.

merge

```
template<class Inlt1, class Inlt2, class Outlt>
```

```
Outlt merge(Inlt1 first1, Inlt1 last1, Inlt2 first2, Inlt2 last2,
            Outlt x);
```

```
template<class Inlt1, class Inlt2, class Outlt, class Pred>
```

```
Outlt merge(Inlt1 first1, Inlt1 last1, Inlt2 first2, Inlt2 last2,
            Outlt x, Pred pr);
```

把[first1,last1), [first2,last2) 两个升序序列合并，形成第3个升序序列，第3个升序序列以x开头

unique:

```
template<class Fwdlt>
```

```
Fwdlt unique(Fwdlt first, Fwdlt last);
```

```
template<class Fwdlt, class Pred>
```

```
Fwdlt unique(Fwdlt first, Fwdlt last, Pred pr);
```

去除[first,last) 这个升序序列中的重复元素

reverse:

```
template<class Bidlt>
```

```
void reverse(Bidlt first, Bidlt last);
```

The template function evaluates [swap](#)(*first + N), *(last - 1 - N) once for each N in the range [0, (last - first) / 2). Thus, the function reverses the order of elements in the sequence.

5. 集合操作:

includes, set_difference, set_intersection, set_union, set_symmetric_difference

用于操作排序值的集合,即它们处理的区间都应该
是升序的

includes:

```
template<class Inlt1, class Inlt2>
```

```
bool includes(Inlt1 first1, Inlt1 last1, Inlt2 first2,
             Inlt2 last2);
```

```
template<class Inlt1, class Inlt2, class Pred>
```

```
bool includes(Inlt1 first1, Inlt1 last1, Inlt2 first2,
             Inlt2 last2, Pred pr);
```

判断 [first2,last2) 中的每个元素，是否都在 [first1,last1) 中

第一个用 == 作比较器，

第二个用 pr 作比较器

set_difference:

```
template<class Inlt1, class Inlt2, class Outlt>
```

```
Outlt set_difference(Inlt1 first1, Inlt1 last1, Inlt2 first2, Inlt2
                    last2, Outlt x);
```

```
template<class Inlt1, class Inlt2, class Outlt, class Pred>
```

```
Outlt set_difference(Inlt1 first1, Inlt1 last1, Inlt2 first2, Inlt2
                    last2, Outlt x, Pred pr);
```

求出[first1,last1) 中，不在[first2,last2) 中的元素，放到从x开始的地方

如果 [first1,last1) 里有多个相等元素不在[first2,last2) 中，
则这多个元素也都会被放入x代表的目标区间里

set_intersection

```
template<class Inlt1, class Inlt2, class Outlt>
```

```
Outlt set_intersection(Inlt1 first1, Inlt1 last1, Inlt2 first2,
                      Inlt2 last2, Outlt x);
```

```
template<class Inlt1, class Inlt2, class Outlt, class Pred>
```

```
Outlt set_intersection(Inlt1 first1, Inlt1 last1, Inlt2 first2,
                      Inlt2 last2, Outlt x, Pred pr);
```

求出[first1,last1)和[first2,last2)中共有的元素，放到从x开始的地方

若某个元素e在[first1,last1)里出现n1次，在[first2,last2)里出现n2次，则该元素在目标区间里出现min(n1,n2)次

```
set_symmetric_difference
template<class InIt1, class InIt2, class OutIt>
OutIt set_symmetric_difference(InIt1 first1, InIt1 last1,
                               InIt2 first2, InIt2 last2, OutIt x);

template<class InIt1, class InIt2, class OutIt, class
Pred>
OutIt set_symmetric_difference(InIt1 first1, InIt1 last1,
                               InIt2 first2, InIt2 last2, OutIt x, Pred pr);
```

把两个区间里相互不在另一区间里的元素放入x开始的地方

```
set_union
template<class InIt1, class InIt2, class OutIt>
OutIt set_union(InIt1 first1, InIt1 last1, InIt2 first2,
                InIt2 last2, OutIt x);

template<class InIt1, class InIt2, class OutIt, class
Pred> OutIt set_union(InIt1 first1, InIt1 last1, InIt2
first2, InIt2 last2, OutIt x, Pred pr);
```

求两个区间的并，放到以x开始的位置
若某个元素e在[first1,last1)里出现n1次，在[first2,last2)里出现n2次，则该元素在目标区间里出现max(n1,n2)次

6. bitset

```
template<size_t N> class bitset
{
    ....
};
```

实际使用的时候，N是个整型常数

如：

```
bitset<40> bst;
```

bst是一个由40位组成的对象，用bitset的函数可以方便地访问任何一位

bitset的成员函数：

```
bitset<N>& operator&= (const bitset<N>& rhs);
bitset<N>& operator|= (const bitset<N>& rhs);
bitset<N>& operator^= (const bitset<N>& rhs);
bitset<N>& operator<= (const bitset<N>& pos);
bitset<N>& operator>= (const bitset<N>& pos);
bitset<N>& set(); //全部设成1
bitset<N>& set(size_t pos, bool val = true); //设置某位
bitset<N>& reset(); //全部设成0
bitset<N>& reset(size_t pos); //某位设成0
bitset<N>& flip(); //全部翻转
bitset<N>& flip(size_t pos); //翻转某位
```

```
reference operator[] (size_t pos); //返回对某位的引用
bool operator[] (size_t pos) const; //判断某位是否为1
reference at(size_t pos);
bool at(size_t pos) const;
unsigned long to_ulong() const; //转换成整数
template<class E, class T, class A>
string to_string() const; //转换成字符串
size_t count() const; //计算1的个数
size_t size() const;
bool operator== (const bitset<N>& rhs) const;
bool operator!= (const bitset<N>& rhs) const;
```

```
bool test(size_t pos) const; //测试某位是否为1
bool any() const; //是否有某位为1
bool none() const; //是否全部为0
bitset<N> operator<< (size_t pos) const;
bitset<N> operator>> (size_t pos) const;
bitset<N> operator~();
static const size_t bitset_size = N;
};
```

注意：第0位在最右边

7. 作业:

1) 20.15

2) 写一个自己的 CMyostream_iterator 模板, 使之能和 ostream_iterator 模板达到一样的效果, 即

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
main(){
    int a[5] = {1,2,3,4,5};
    CMyostream_iterator<int> output(cout,"");
    vector<int> v(a,a+5);
    copy(v.begin(),v.end(),output);
}
```

程序的输出结果是:

1*2*3*4*5*

注意, 编写 CMyostream_iterator 时不能使用 ostream_iterator

参考 copy 的 help

copy

template<class InIt, class OutIt> OutIt copy(InIt first, InIt last, OutIt x);

The template function evaluates $*(x + N) = *(first + N)$ once for each N in the range $[0, last - first)$, for strictly increasing values of N beginning with the lowest value. It then returns $x + N$. If x and first designate regions of storage, x must not be in the range $[first, last)$

本节要点难点

1. remove 的用法
2. transform 的用法
3. binary_search, upper_bound, lower_bound, equal_range 要求是升序序列
4. 堆的概念
5. merge, unique 要求是升序序列
6. 集合操作算法 要求是升序序列
7. 位操作算法 可以定义以二进制位为单位操作的变量