Стандартная библиотека STL: алгоритмы

Александр Смаль

Академический университет
6 марта 2014
Санкт-Петербург

Не модифицирующие алгоритмы

#include <algorithm>

swap
iter_swap

max
min

1. count, count_if
2. for_each
3. equal, mismatch(p, q, p')
4. lexicographical_compare
5. min_element, max_element
6. find, find_if, find_first_of
7. search, search_n, find_end
8. adjacent_find

Для сортированных последовательностей.

1. binary_search → Bool
2. includes
3. lower_bound, upper_bound, equal_range

T const& max (T const& a, .. b)
max (T.. a, T.. b, Comp p)

NB: если не нашел => последний итератор.

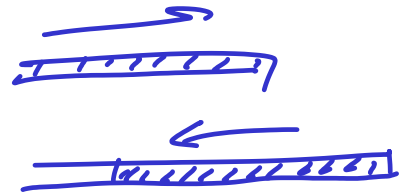
Не модифицирующие алгоритмы: примеры

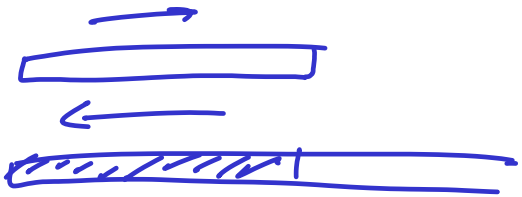
```
vector<int> v;
size_t c = count_if(v.begin(), v.end(),
                    bind1st(less<int>(), 0));

vector<string> w;
for_each(w.begin(), w.end(), mem_fun_ref(&string::clear));

bool bb = min_element(v.begin(), v.end()) ==
          max_element(v.begin(), v.end(), greater<int>());

vector<string>::iterator res =
    find(w.rbegin(), w.rend(), "Hello").base();

string names[3] = {"Jim", "Jon", "Joe"};
vector<string>::iterator it =
    find_first_of(w.begin(), w.end(),
                  names, names + 3);
```

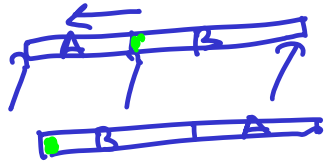
boost::bind

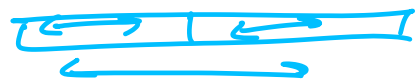
Модифицирующие алгоритмы

1. `fill, fill_n`
2. `generate, generate_n`
3. `random_shuffle`
4. `copy, copy_backwards`
5. `remove_copy, remove_copy_if`
6. `remove, remove_if`
7. `replace, replace_copy, replace_copy_if, replace_if`
8. `reverse, reverse_copy`
9. `rotate, rotate_copy`
10. `swap_ranges`
11. `transform`
12. `unique, unique_copy`
13. `accumulate, adjacent_difference, inner_product, partial_sum`









#include <numeric>

Модифицирующие алгоритмы: примеры

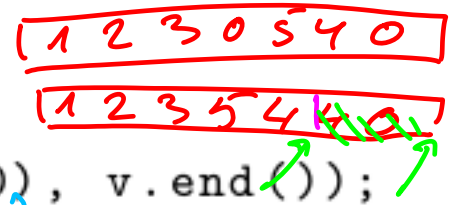
```
int RandomNumber () { return (std::rand()%100); }
vector<int> v(10);
generate(v.begin(), v.end(), &RandomNumber);

vector<int> w(v.size() / 2)
copy(v.begin(), v.begin() + w.size(), w.begin());

remove_copy(w.begin(), w.end(), v.rbegin(), 0);

replace(w.begin(), w.end(), 0, -1);

vector<int> r(w.size());
transform(w.begin(), w.end(), v.begin(), r.begin(),
          plus<int>());
transform(r.begin(), r.end(), r.begin(),
          bind2nd(multiplies<int>(), 2));

int sum = accumulate(r.begin(), r.end(), 10, plus<int>());
```

optional

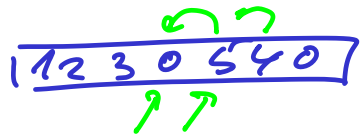
$10 + \sum_i 2(i)$

Модифицирующие алгоритмы: идиомы

Удаление элемента по значению



```
vector<int> v;
v.erase(remove(v.begin(), v.end(), 0), v.end());

template <class Iterator, class T>
Iterator remove (Iterator p, Iterator q, const T& val) {
    Iterator r = p;
    for (; p != q; ++p)
        if (!(*p == val))
            *r++ = *p;
    return r;
}
```



Удаление одинаковых элементов

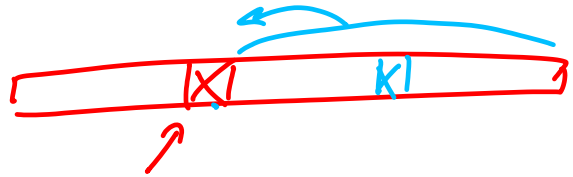
```
vector<int> v;
sort(v.begin(), v.end(), comp);
v.erase(unique(v.begin(), v.end(), comp), v.end());

list<int> l;
l.sort(comp);
l.erase(unique(l.begin(), l.end(), comp), l.end());
```

Модифицирующие алгоритмы: предикаты

```
struct ElementN {
    ElementN(size_t n) : n(n), i(0) {}

    template<class T>
    bool operator()(T const& t) { return (++i == n); }

    size_t n;
    size_t i;
};

vector<int> v = {0,1,2,3,4,5,6,7,8,9,10}; // c++11
v.erase(remove_if(v.begin(), v.end(), ElementN(3)),
        v.end());
```

Модифицирующие алгоритмы: предикаты

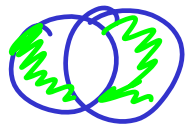
```
template<class Iterator, class Pred>
Iterator remove_if(Iterator p, Iterator q, Pred pred)
{
    Iterator s = find_if(p, q, pred);
    if (s == q)
        return q;

    Iterator out = s;
    ++s;
    return remove_copy_if(s, q, out, pred);
}
```



```
template<class Iterator, class OutIterator, class Pred>
Iterator remove_if(Iterator p, Iterator q,
                   OutIterator out, Pred pred) {
    for (; p != q; ++p)
        if (!pred(*p))
            *out++ = *p;
    return out;
}
```

Сортировка

1. sort, stable_sort
2. partition, stable_partition
3. nth_element
4. partial_sort
5. merge, inplace_merge
6. set_union, set_intersection, set_difference, set_symmetric_difference



```
vector<int> v;
sort(v.begin(), v.end(), greater<int>());

nth_element(v.begin(), v.begin() + v.size() / 2, v.end());
int med = *(v.begin() + v.size() / 2);

random_shuffle(v.begin(), v.end());

vector<int>::iterator m = partition(v.begin(), v.end(),
                                    bind2nd(less<int>(), med));
```

Что есть ещё?

1. Операции с кучей
   - `push_heap`,
   - `pop_heap`,
   - `make_heap`,
   - `sort_heap`.
2. Операции с неинициализированными интервалами
   - `raw_storage_iterator`,
   - `uninitialized_copy`,
   - `uninitialized_fill`,
   - `uninitialized_fill_n`
3. Операции с перестановками
   - `next_permutation`,
   - `prev_permutation`