STL: алгоритмы

Александр Смаль

CS центр
27 марта 2018
Санкт-Петербург

Функторы и min/max алгоритмы

- *Функтор* — класс, объекты которого ведут себя как функции, т.е. имеет перегруженные `operator()`.
- *Предикат* — функтор, возвращающий `bool`.

Функторы и min/max алгоритмы

- *Функтор* — класс, объекты которого ведут себя как функции, т.е. имеет перегруженные `operator()`.
- *Предикат* — функтор, возвращающий `bool`.

Функторы в стандартной библиотеке:

- `less`, `greater`, `less_equal`, `greater_equal`, `not_equal_to`, `equal_to`,
- `minus`, `plus`, `divides`, `modulus`, `multiplies`,
- `logical_not`, `logical_and`, `logical_or`
- `bit_and`, `bit_or`, `bit_xor`,
- `hash`.

Функторы и min/max алгоритмы

- *Функтор* — класс, объекты которого ведут себя как функции, т.е. имеет перегруженные `operator()`.
- *Предикат* — функтор, возвращающий `bool`.

Функторы в стандартной библиотеке:

- `less`, `greater`, `less_equal`, `greater_equal`, `not_equal_to`, `equal_to`,
- `minus`, `plus`, `divides`, `modulus`, `multiplies`,
- `logical_not`, `logical_and`, `logical_or`
- `bit_and`, `bit_or`, `bit_xor`,
- `hash`.

Алгоритмы min/max

- `min`, `max`, `minmax`,
- `min_element`, `max_element`, `minmax_element`.

Немодифицирующие алгоритмы

- `all_of`, `any_of`, `none_of`,
- `for_each`,
- `find`, `find_if`, `find_if_not`, `find_first_of`,
- `adjacent_find`,
- `count`, `count_if`,
- `equal`, `mismatch`,
- `is_permutation`,
- `lexicographical_compare`,
- `search`, `search_n`, `find_end`.

Для упорядоченных последовательностей

- `lower_bound`, `upper_bound`, `equal_range`,
- `set_intersection`, `set_difference`, `set_union`, `set_symmetric_difference`,
- `binary_search`, `includes`.

Примеры

```
vector<int> v = {2,3,5,7,13,17,19};
size_t c = count_if(v.begin(), v.end(),
                    [](int x) {return x % 2 == 0;});

auto it = lower_bound(v.begin(), v.end(), 11);

bool has7 = binary_search(v.begin(), v.end(), 7);
```

```
vector<string> & db = getNames();

for_each(db.begin(), db.begin() + db.size() / 2,
        [](string & s){cout << s << "\n";});

auto w = find(db.begin(), db.end(), "Waldo");

string agents[3] = {"Alice", "Bob", "Eve"};
auto it = find_first_of(db.begin(), db.end(),
                        agents, agents + 3);
```

Модифицирующие алгоритмы

- `fill`, `fill_n`, `generate`, `generate_n`,
- `random_shuffle`, `shuffle`,
- `copy`, `copy_n`, `copy_if`, `copy_backward`,
- `move`, `move_backward`,
- `remove`, `remove_if`, `remove_copy`, `remove_copy_if`,
- `replace`, `replace_if`, `replace_copy`, `replace_copy_if`,
- `reverse`, `reverse_copy`,
- `rotate`, `rotate_copy`,
- `swap_ranges`,
- `transform`,
- `unique`, `unique_copy`,
* `accumulate`, `adjacent_difference`, `inner_product`, `partial_sum`, `iota`.

Примеры

```
// случайные
vector<int> a(100);
generate(a.begin(), a.end(), [](){return rand() % 100;});

// 0,1,2,3,...
vector<int> b(a.size());
iota(b.begin(), b.end(), 0);

// c[i] = a[i] * b[i]
vector<int> c(b.size());
transform(a.begin(), a.end(), b.begin(),
          c.begin(), multiplies<int>());

// c[i] *= 2
transform(c.begin(), c.end(), c.begin(),
          [](int x) {return x * 2;});

// сумма c[i]
int sum = accumulate(c.begin(), c.end(), 0);
```

Удаление элементов из последовательности

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
remove(v.begin(), v.end(), 5);
```

Как изменится `v.size()`?

Какое содержимое вектора v?

Удаление элементов из последовательности

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
remove(v.begin(), v.end(), 5);
```

Как изменится `v.size()`? Не изменится.

Какое содержимое вектора v? `{2,1,8,2,8,5,2,5,8}`

Удаление элементов из последовательности

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
remove(v.begin(), v.end(), 5);
```

Как изменится `v.size()`? Не изменится.

Какое содержимое вектора v? `{2,1,8,2,8,5,2,5,8}`

Удаление элемента по значению:

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
v.erase(remove(v.begin(), v.end(), 5), v.end());
```

```
list<int> l = {2,5,1,5,8,5,2,5,8};
l.remove(5);
```

Удаление элементов из последовательности

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
remove(v.begin(), v.end(), 5);
```

Как изменится `v.size()`? Не изменится.

Какое содержимое вектора v? `{2,1,8,2,8,5,2,5,8}`

Удаление элемента по значению:

```
vector<int> v = {2,5,1,5,8,5,2,5,8};
v.erase(remove(v.begin(), v.end(), 5), v.end());
```

```
list<int> l = {2,5,1,5,8,5,2,5,8};
l.remove(5);
```

Удаление одинаковых элементов:

```
vector<int> v = {1,2,2,2,3,4,5,5,5,6,7,8,9};
v.erase(unique(v.begin(), v.end()), v.end());
```

```
list<int> l = {1,2,2,2,3,4,5,5,5,6,7,8,9};
l.unique();
```

Удаление из ассоциативных контейнеров

Неправильный вариант

```
map<string, int> m;
for (auto it = m.begin(); it != m.end(); ++it)
    if (it->second == 0)
        m.erase(it);
```

Удаление из ассоциативных контейнеров

Неправильный вариант

```
map<string, int> m;
for (auto it = m.begin(); it != m.end(); ++it)
    if (it->second == 0)
        m.erase(it);
```

Правильный вариант

```
for (auto it = m.begin() ; it != m.end(); )
    if (it->second == 0)
        it = m.erase(it);
    else
        ++it;
```

Удаление из ассоциативных контейнеров

Неправильный вариант

```
map<string, int> m;
for (auto it = m.begin(); it != m.end(); ++it)
    if (it->second == 0)
        m.erase(it);
```

Правильный вариант

```
for (auto it = m.begin() ; it != m.end(); )
    if (it->second == 0)
        it = m.erase(it);
    else
        ++it;
```

Альтернативный вариант (для старого стандарта)

```
for (map<string,int>::iterator it = m.begin(); it != m.end(); )
    if (it->second == 0)
        m.erase(it++);
    else
        ++it;
```

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};

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

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

Сортировка

- `is_sorted`, `is_sorted_until`,
- `sort`, `stable_sort`,
- `nth_element`, `partial_sort`,
- `merge`, `inplace_merge`,
- `partition`, `stable_partition`, `is_partitioned`, `partition_copy`, `partition_point`.

Сортировка

- is_sorted, is_sorted_until,
- sort, stable_sort,
- nth_element, partial_sort,
- merge, inplace_merge,
- partition, stable_partition, is_partitioned, partition_copy, partition_point.

```
vector<int> v = randomVector<int>();

auto med = v.begin() + v.size() / 2;
nth_element(v.begin(), med, v.end());
cout << "Median: " << *med;

auto m = partition(v.begin(), v.end(),
          [](int x){return x % 2 == 0;});
sort(v.begin(), m);
v.erase(m, v.end());
```

Что есть ещё?

- Операции с кучей:
  - `push_heap`,
  - `pop_heap`,
  - `make_heap`,
  - `sort_heap`
  - `is_heap`,
  - `is_heap_until`.
- Операции с неинициализированными интервалами:
  - `raw_storage_iterator`,
  - `uninitialized_copy`,
  - `uninitialized_fill`,
  - `uninitialized_fill_n`.
- Операции с перестановками
  - `next_permutation`,
  - `prev_permutation`.