STL: алгоритмы

Александр Смаль

CS центр
1 марта 2017
Санкт-Петербург

Advanced итераторы

Определены следующие специальные итераторы итераторов:

1. back_inserter, front_inserter, inserter
2. istream_iterator, ostream_iterator

```
vector<Person> db;
template<class OutIt>
void findByName(vector<Person> const& db,
                string name, OutIt out);
...
vector<Person> res; // size is not known
findByName(db, "Ivan", std::back_inserter(res));

ifstream file("input.txt");
vector<double> v((istream_iterator<double>(file)),
                 istream_iterator<double>());

copy(v.begin(), v.end(), ostream_iterator<double>(cout, '\n'));
```

Функторы и предикаты

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

Функторы в стандартной библиотеке:

1. `less, greater, less_equal, greater_equal, not_equal_to, equal_to`
2. `minus, plus, divides, modulus, multiplies`
3. `logical_not, logical_and, logical_or`
4. `mem_fun, mem_fun_ref, ptr_fun`
5. `bind1st, bind2nd, not1, not2`

```
map<int, string, greater<int>> m;


bind2nd(less<int>(),      10);
bind1st(modulus<size_t>, 1024);
not1(logical_not<bool>());
```

Как написать свой функтор

```
#include <functional>
template <class Arg, class Result>
  struct unary_function {
    typedef Arg argument_type;
    typedef Result result_type;
  };
template <class Arg1, class Arg2, class Result>
  struct binary_function {
    typedef Arg1 first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Result result_type;
  };

struct MyFunctor : binary_function<int, double, size_t>
{
    size_t operator()(int i, double d) {...}
};
```

Не модифицирующие алгоритмы

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

Для сортированных последовательностей.

1. `binary_search`
2. `includes`
3. `lower_bound, upper_bound, equal_range`

Общие принципы

Алгоритмы поиска возвращают итератор.

```
template <class ForwardIt1, class ForwardIt2>
ForwardIt1 search (ForwardIt1 first1, ForwardIt1 last1,
                   ForwardIt2 first2, ForwardIt2 last2);
```

Алгоритмы `*_if` принимают унарный предикат.

```
template<class InputIt, class UnaryPredicate>
Iterator find_if(InputIt i, InputIt j, UnaryPredicate p);
```

Алгоритмы подразумевающие отношения порядка или эквивалентности могут принимать соответствующий предикат.

```
template <class InputIt1, class InputIt2, class BinaryPredicate>
bool equal (InputIt1 first1, InputIt1 last1,
            InputIt2 first2, BinaryPredicate pred);
```

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);
```

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`

Общие принципы

Алгоритмы *_n принимают итератор и количество.

```
template <class OutputIt, class Size, class T>
OutputIt fill_n (OutputIt first, Size n, const T& val);
```

Алгоритмы *_copy не in-place.

```
template <class InputIt, class OutputIt, class T>
OutputIt remove_copy (InputIt first, InputIt last,
                      OutputIt result, const T& val);
```

Алгоритмы unique* удаляют только *последовательные* одинаковые элементы.

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>());
```

Модифицирующие алгоритмы: идиомы

Удаление элемента по значению

Модифицирующие алгоритмы: идиомы

Удаление элемента по значению

```
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;
}
```

Модифицирующие алгоритмы: идиомы

Удаление элемента по значению

```
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;
}
```

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

Модифицирующие алгоритмы: идиомы

Удаление одинаковых элементов

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

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

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`