

# Контейнеры, итераторы, представления, алгоритмы

2021



# Массив

```
std::array<float,3> x{1,2,3}, y{3,4,5};
std::cout << x.empty() << '\n'; // 0
std::cout << x.size() << '\n'; // 3
std::cout << (x < y) << '\n'; // 1
for (float f : x) {
    std::cout << f << '\n';
}
float z[3] = {1,2,3};
for (float f : z) { // ошибка
    std::cout << f << '\n';
}
```

Требования к элементам:

- ▶ T(T&&) или T(const T&)
- ▶ T& operator=(T&&) или T& operator=(const T&)

# Вектор

```
std::vector<float> x{1,2,3}, y{3,4,5};
std::cout << x.empty() << '\n'; // 0
std::cout << x.size() << '\n'; // 3
std::cout << (x < y) << '\n'; // 1
for (float f : x) {
    std::cout << f << '\n';
}
```

Требования к элементам:

- ▶ T(T&&) или T(const T&)
- ▶ T& operator=(T&&) или T& operator=(const T&)

# Таблица

```
std::map<std::string, int> x{
    {"hello", 3},
    {"world", 4}
};
x["!!!"] = 5;
std::cout << x.empty() << '\n'; // 0
std::cout << x.size() << '\n'; // 3
for (const auto& pair : x) {
    std::cout << pair.first << "->" << pair.second << '\n';
}
```

Требования к элементам:

- ▶ T(T&&) или T(const T&)
- ▶ T& operator=(T&&) или T& operator=(const T&)
- ▶ bool operator<(const T&)

# Хэш-таблица

```
std::unordered_map<std::string, int> x{
    {"hello", 3},
    {"world", 4}
};
x["!!!"] = 5;
std::cout << x.empty() << '\n'; // 0
std::cout << x.size() << '\n'; // 3
for (const auto& pair : x) {
    std::cout << pair.first << "->" << pair.second << '\n';
}
```

Требования к элементам:

- ▶ T(T&&) или T(const T&)
- ▶ T& operator=(T&&) или T& operator=(const T&)
- ▶ специализация std::hash<T>

# Шаблон `std::hash`

```
namespace std {  
    // объявление шаблона  
    template <class T> struct hash;  
    // частичная специализация для примитивных типов  
    template <>  
    struct hash<int> {  
        using result_type = size_t;  
        using argument_type = int;  
        result_type operator()(argument_type x) const {  
            return static_cast<result_type>(x);  
        }  
    };  
}
```

# Очередь

```
std::queue<float> q;  
q.push(0);  
q.push(1);  
q.push(2);  
float f = q.front();  
q.pop(); // извлечение первого элемента
```

Требования к элементам:

- ▶ T(T&&) или T(const T&)
- ▶ T& operator=(T&&) или T& operator=(const T&)

Что будет, если положить контейнер  
в контейнер?



# Контейнер в контейнере

```
using namespace std;
vector<vector<float>> x; // ок
map<vector<string>,int> y; // ок
y = {
    {"hello", "world"}, 3},
    {"a", "b"}, 4}
};
x[{"!!!", "???"}] = 5;
unordered_map<vector<string>,int> z; // ошибка
map<map<string,string>,int> u; // ок
unordered_map<tuple<string,string>,int> v; // ок
```

# Все контейнеры

## Массивы:

- ▶ `std::array`
- ▶ `std::vector`

## Очереди:

- ▶ `std::queue`
- ▶ `std::deque`
- ▶ `std::stack`
- ▶ `std::priority_queue`

## Списки:

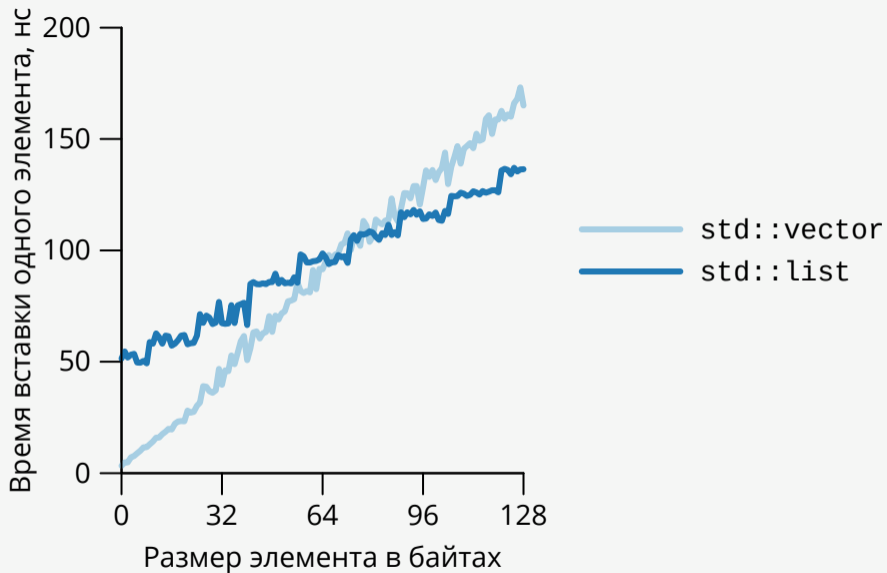
- ▶ `std::list`
- ▶ `std::forward_list`

## Упорядоченные ассоциативные массивы:

- ▶ `std::map`
- ▶ `std::set`
- ▶ `std::multimap`
- ▶ `std::multiset`

## Ассоциативные массивы с хэшем:

- ▶ `std::unordered_map`
- ▶ `std::unordered_set`
- ▶ `std::unordered_multimap`
- ▶ `std::unordered_multiset`



# Указатели

```
std::vector<float> x(10);  
float* first = x.data();           // указатель на первый элемент  
float* last = x.data() + x.size(); // указатель на элемент,  
                                   // идущего за последним  
  
while (first != last) {  
    std::cout << *first << '\n'; // "разыменование"  
    ++first;                       // инкремент  
}  
  
float* first2 = &x[0];            // получить адрес
```

# Итераторы

```
using iterator = std::vector<float>::iterator;
std::vector<float> x(10);
iterator first = x.begin();           // итератор для первого элемента
iterator last = x.end();              // итератор для элемента,
// идущего за последним
while (first != last) {
    std::cout << *first << '\n';     // "разыменование"
    ++first;                          // инкремент
}
```

```
std::map<std::string,int> x{ {"hello",3}, {"world",4} };
auto first = x.begin(), last = x.end();
while (first != last) {
    std::cout << first->first << "->" << first->second << '\n';
    ++first;
}
```

# Типы итераторов

- ▶ Ввод (`std::input_iterator_tag`).
- ▶ Вывод (`std::output_iterator_tag`).
- ▶ Однонаправленный (`std::forward_iterator_tag`).
- ▶ Двухнаправленный (`std::bidirectional_iterator_tag`).
- ▶ Произвольный доступ (`std::random_access_iterator_tag`).

# Пример: итератор для массива

```
template <class T> class array_iterator {
private:
    T* ptr;
public:
    using iterator_category = std::random_access_iterator_tag;
    using value_type = T;
    using reference = T&; using const_reference = const T&;
    using pointer = T*; using const_pointer = const T*;
    reference operator*() { return *ptr; } // разыменование
    const_reference operator*() const { return *ptr; }
    pointer operator->() { return ptr; } // индирекция
    const_pointer operator->() const { return ptr; }
    array_iterator& operator++() { return ++ptr; } // инкремент
    array_iterator operator++(int) { return ptr++; }
    bool operator==(const array_iterator&) const;
    bool operator!=(const array_iterator&) const;
};
```

# Пример: считывание чисел

```
std::istream_iterator<float> first(std::cin), last; // итератор
std::vector<float> x;                               // ввода
while (first != last) {
    x.push_back(*first);
    ++first;
}
```



# Пример: считывание строк

```
struct Line { std::string text; };

std::istream& operator>>(std::istream& in, Line& line) {
    return std::getline(in, line.text, '\n');
}

std::istream_iterator<Line> first(std::cin), last; // итератор
std::vector<Line> lines;                          // ввода
while (first != last) {
    lines.emplace_back(*first);
    ++first;
}
```

# Пример: запись строк

```
struct Line { std::string text; };

std::ostream& operator<<(std::ostream& out, const Line& line) {
    return out << line;
}

// итератор вывода
std::ostream_iterator<Line> result(std::cout, "\n");
std::vector<Line> lines;
// ... заполнение вектора строк ...
auto first = lines.begin();
auto last = lines.end();
while (first != last) {
    *result = *first;
    ++first; ++result;
}
```

## Пример: копирование строк

```
std::istream_iterator<Line> first(std::cin), last;  
std::ostream_iterator<Line> result(std::cout, "\n");  
while (first != last) {  
    *result = *first;  
    ++first; ++result;  
}
```

# Пример: копирование строк

```
std::istream_iterator<Line> first(std::cin), last;  
std::ostream_iterator<Line> result(std::cout, "\n");  
while (first != last) {  
    *result = *first;  
    ++first; ++result;  
}
```

Версия с алгоритмом:

```
std::copy(  
    std::istream_iterator<Line>(std::cin),  
    std::istream_iterator<Line>(),  
    std::ostream_iterator<Line>(std::cout, "\n")  
);
```

# Копирование

```
template <class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first, InputIterator last,
    OutputIterator result) {
    while (first != last) {
        *result++ = *first++;
    }
    return result;
}
```

# Копирование с условием

```
template <class InputIterator, class OutputIterator, class Pred>
OutputIterator copy_if(InputIterator first, InputIterator last,
    OutputIterator result, Pred pred) {
    while (first != last) {
        if (pred(*first)) { *result++ = *first; }
        ++first;
    }
    return result;
}
```

# Диспетчеризация

```
template <class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first, InputIterator last,
    OutputIterator result) {
    using V1 = typename iterator_traits<InputIterator>::value_type;
    using V2 = typename iterator_traits<OutputIterator>::value_type;
    constexpr bool simple = is_pointer<InputIterator>::value &&
        is_pointer<OutputIterator>::value &&
        is_same<V1,V2>::value &&
        is_trivial<V1>::value;
    if constexpr (simple) {
        std::memmove(result, first, (last-first)*sizeof(V1));
    } else {
        while (first != last) { *result++ = *first++; }
    }
    return result;
}
```

# Сумма

```
template <class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init) {
    while (first != last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```



# Некоторые алгоритмы

```
std::copy  
std::copy_if  
std::sort  
std::find  
std::find_if  
std::mismatch  
std::accumulate  
std::search
```

```
std::equal  
std::for_each  
std::copy_n  
std::generate  
std::generate_n  
std::fill  
std::transform  
std::reverse
```

Всего более 100 алгоритмов.

# Итератор вставки

```
std::vector<float> x;  
std::copy(  
    std::istream_iterator<float>(std::cin),  
    std::istream_iterator<float>(),  
    std::back_inserter(x));  
  
std::list<float> y;  
std::copy(  
    std::istream_iterator<float>(std::cin),  
    std::istream_iterator<float>(),  
    std::inserter(y, y.begin()));
```

# Итератор чередования

```
template <class T, class Delimiter=const char*>
class intersperse_iterator {
    std::ostream* ostr = nullptr;
    Delimiter delim = nullptr;
    bool first = true;
public:
    intersperse_iterator&
    operator=(const T& value) {
        if (ostr) {
            if (delim != 0 && !first) { *ostr << delim; }
            *ostr << value;
            if (first) { first = false; }
        }
        return *this;
    }
};
```

# Представление контейнера

```
std::vector<int> x{1,2,3,4};  
std::ostream_iterator<int> out(std::cout, "\n");  
// c++11  
std::copy(x.begin(), x.end(), out);  
// c++20  
std::ranges::copy(x, out);
```

# Представление контейнера

```
// c++20
using namespace std::views;
for (int i : iota(1,3)) { std::cout << i << ' '; } // 1 2 3
for (int i : iota(1,3) | take(2)) { std::cout << i << ' '; } // 1 2
std::vector<int> x{10,20,3,4};
for (int i : all(x) | take(2)) { std::cout << i << ' '; } // 10 20
```

# Итератор для директории

```
struct MyEntry: public dirent { /* ... */ };  
struct MyDirectory {  
    MyDirectory& operator>>(MyEntry& entry) { /* ... */ }  
    MyIstreamIterator<MyEntry> begin() { return {*this}; }  
    MyIstreamIterator<MyEntry> end() { return {}; }  
};  
  
MyDirectory home("/home/myuser");  
for (const MyEntry& entry : home) {  
    std::cout << entry.d_name << std::endl;  
}
```

# Ссылки

- ▶ [Все алгоритмы](#)

© 2019–2021 Ivan Gankevich [i.gankevich@spbu.ru](mailto:i.gankevich@spbu.ru)

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. The copy of the license is available at <https://creativecommons.org/licenses/by-sa/4.0/>.