



# Абстракции C++ часть 2

Иван Ганкевич

2018

# Раздел 1

## Контейнеры

# 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';
}
```

# 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';  
}
```

# 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';
}
```

# 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';
}
```

# Очередь

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

А что если положить контейнер  
в контейнер?



# Контейнер в контейнере

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

# Все контейнеры

```
std::array  
std::vector  
std::queue  
std::deque  
std::stack  
std::priority_queue  
std::list  
std::forward_list
```

```
std::map  
std::set  
std::multimap  
std::multiset  
std::unordered_map  
std::unordered_set  
std::unordered_multimap  
std::unordered_multiset
```

## Раздел 2

## Итераторы

# Указатели

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

# Итераторы

```
typedef std::vector<float>::iterator 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
}
```

# 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: 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::istream& operator<<(std::istream& out, const Line& line) {
    return out << line << '\n';
}

// 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")  
);
```

# Раздел 3

## Алгоритмы

# 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++;
        }
    }
    return result;
}
```

# Диспетчеризация

```
template <class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first, InputIterator last,
    OutputIterator result) {
    typedef typename iterator_traits<InputIterator>::value_type V1;
    typedef typename iterator_traits<OutputIterator>::value_type V2;
    bool simple = is_pointer<InputIterator>::value &&
        is_pointer<OutputIterator>::value &&
        is_same<V1,V2>::value &&
        is_trivial<V1>::value;
    if (simple) {
        std::memmove(result, first, last-first);
    } 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.

## Раздел 4

### Примеры итераторов

# 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

```
template <class Iterator> struct view {  
    Iterator first, last;  
    Iterator begin() { return first; }  
    Iterator end() { return last; }  
};
```

```
template <class Container> auto  
make_view(Container& cnt) -> view<decltype(cnt.begin())> {  
    return {cnt.begin(), cnt.end()};  
}
```

```
template <class X> view<std::istream_iterator<X>>  
make_view(std::istream& in) {  
    return { std::istream_iterator<X>(in),  
            std::istream_iterator<X>() };  
}
```

# Представление контейнера

```
for (float f : make_view<float>(std::cin)) {  
    std::cout << f << '\n';  
}
```

# 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;  
}
```