

# Программирование на языке C++

## Лекция 9

### Последовательные контейнеры STL

Александр Смаль

# Общие сведения о контейнерах

Контейнеры библиотеки STL можно разделить на две категории:

- последовательные,
- ассоциативные.

## Общие сведения о контейнерах

Контейнеры библиотеки STL можно разделить на две категории:

- последовательные,
- ассоциативные.

## Общие вложенные типы

- `Container::value_type`
- `Container::iterator`
- `Container::const_iterator`

## Общие сведения о контейнерах

Контейнеры библиотеки STL можно разделить на две категории:

- последовательные,
- ассоциативные.

## Общие вложенные типы

- `Container::value_type`
- `Container::iterator`
- `Container::const_iterator`

## Общие методы контейнеров

- Все „особенные методы“ и `swap`.
- `size`, `max_size`, `empty`, `clear`.
- `begin`, `end`, `cbegin`, `cend`.
- Операторы сравнения: `==`, `!=`, `>`, `>=`, `<`, `<=`.

**Примечание:** вся STL определена в пространстве имён `std`.

## Шаблон array

Класс-обёртка над статическим массивом.

- operator[], at,
- back, front.
- fill,
- data.

Позволяет работать с массивом как с контейнером.

```
#include <array>
```

```
std::array<std::string, 3> a = {"One", "Two", "Three"};  
std::cout << a.size() << std::endl;  
std::cout << a[1] << std::endl;
```

```
// ошибка времени выполнения  
std::cout << a.at(3) << std::endl;
```

## Общие методы остальных последовательных контейнеров

- Конструктор от двух итераторов.
- Конструктор от `count` и `defVal`.
- Конструктор от `std::initializer_list<T>`.
- Методы `back`, `front`.
- Методы `push_back`, `emplace_back`
- Методы `assign`.
- Методы `insert`.
- Методы `emplace`.
- Методы `erase` от одного и двух итераторов.

## Шаблон vector

Динамический массив с автоматическим изменением размера при добавлении элементов.

- operator[], at,
- resize,
- capacity, reserve, shrink\_to\_fit,
- pop\_back,
- data.

Позволяет работать со старым кодом.

```
#include <vector>
```

```
std::vector<std::string> v = {"One", "Two"};  
v.reserve(100);  
v.push_back("Three");  
v.emplace_back("Four");  
legacy_function(v.data(), v.size());  
std::cout << v[2] << std::endl;
```

## Шаблон deque

Контейнер с возможностью быстрой вставки и удаления элементов на обоих концах за  $O(1)$ . Реализован как список указателей на массивы фиксированного размера.

- operator[], at,
- resize,
- push\_front, emplace\_front
- pop\_back, pop\_front,
- shrink\_to\_fit.

```
#include <deque>
```

```
std::deque<std::string> d = {"One", "Two"};  
d.emplace_back("Three");  
d.emplace_front("Zero");  
std::cout << d[1] << std::endl;
```



## Шаблон `list`

Двусвязный список. В любом месте контейнера вставка и удаление производятся за  $O(1)$ .

- `push_front`, `emplace_front`,
- `pop_back`, `pop_front`,
- `splice`,
- `merge`, `remove`, `remove_if`, `reverse`,  
`sort`, `unique`.

```
#include <list>
```

```
std::list<std::string> l = {"One", "Two"};  
l.emplace_back("Three");  
l.emplace_front("Zero");  
std::cout << l.front() << std::endl;
```

## Итерация по списку

У списка нет методов для доступа к элементам по индексу. Можно использовать range-based for:

```
using std::string;
std::list<string> l = {"One", "Two", "Three"};
for (string & s : l)
    std::cout << s << std::endl;
```

Для более сложных операций нужно использовать *итераторы*.

```
std::list<string>::iterator i = l.begin();
for ( ; i != l.end(); ++i) {
    if (*i == "Two")
        break;
}
l.erase(i);
```

Итератор списка можно перемещать в обоих направлениях:

```
auto last = l.end();
--last; // последний элемент
```

## Шаблон `forward_list`

Односвязный список. В любом месте контейнера вставка и удаление производятся за  $O(1)$ .

- `insert_after` и `emplace_after` вместо `insert` и `emplace`,
- `before_begin`, `cbefore_begin`,
- `push_front`, `emplace_front`, `pop_front`,
- `splice_after`,
- `merge`, `remove`, `remove_if`, `reverse`,  
`sort`, `unique`.

```
#include <forward_list>
using std::string;
```

```
std::forward_list<string> fl = {"One", "Two"};
fl.emplace_front("Zero");
fl.push_front("Minus one");
std::cout << fl.front() << std::endl;
```

## Шаблон `basic_string`

Контейнер для хранения символьных последовательностей.

```
typedef basic_string<char>      string;  
typedef basic_string<wchar_t>  wstring;  
typedef basic_string<char16_t> u16string;  
typedef basic_string<char32_t> u32string;
```

- Метод `c_str()` для совместимости со старым кодом,
- поддержка неявных преобразований со строками в стиле C,
- `operator[]`, `at`,
- `reserve`, `capacity`, `shrink_to_fit`,
- `append`, `operator+`, `operator+=`,
- `substr`, `replace`, `compare`,
- `find`, `rfind`, `find_first_of`,  
`find_first_not_of`, `find_last_of`,  
`find_last_not_of` (в терминах *индексов*)

# Адаптеры и псевдоконтейнеры

## Адаптеры:

- `stack` – реализация интерфейса стека.
- `queue` – реализация интерфейса очереди.
- `priority_queue` – очередь с приоритетом на куче.

## Псевдо-контейнеры:

- `vector<bool>`
  - ненастоящий контейнер (не хранит `bool`-ы),
  - использует проху-объекты.
- `bitset`

Служит для хранения битовых масок.  
Похож на `vector<bool>`.
- `valarray`

Шаблон служит для хранения числовых массивов и оптимизирован для достижения повышенной вычислительной производительности.

## Ещё о vector

- Самый универсальный последовательный контейнер.
- Во многих случаях самый эффективный.
- Предпочитайте vector другим контейнерам.
- Интерфейс вектора построен на итераторах, а не на индексах.
- Итераторы вектора ведут себя как указатели.

Использование reserve и capacity:

```
std::vector<int> v;  
v.reserve(N); // N – верхняя оценка на размер  
...  
if (v.capacity() == v.size()) // реаллокация
```

Сжатие и очистка в C++03:

```
std::vector<int> & v = getData();  
// shrink_to_fit  
std::vector<int>(v).swap(v);  
// clear + shrink_to_fit  
std::vector<int>().swap(v);
```