Курс-интенсив

Программирование на C++

Стандартная библиотека
Часть 1. STL

academy.rubius.com
konstantin.dobrychev@rubius.com
Константин Добрычев

Функциональные объекты

```
constexpr int sum(int a, int b)
{
    return a + b;
}

sum(1, 2);
```

```
struct Sum {
    constexpr int operator()(int a, int b) const {
        return a + b;
    }
};

Sum sum;
sum(1, 2);

Sum()(1, 2);
```

Функциональные объекты

```
struct CsvReader {
  char delimiter;
  mutable std::vector<std::vector<std::string>> data;

  explicit CsvReader(char delimiter = ',')
    : delimiter(delimiter)
  {}

  std::string operator()(std::string line) const {
    data.push_back(split(line, delimiter));
    return line;
  }
};
```

Лямбда-функции

```
auto sum = [](int a, int b) {
    return a + b;
};

sum(1, 2);
```

```
int init = 42;

auto sum = [init](int a, int b) {
    return init + a + b;
};

sum(1, 2);
```

```
template<typename C, typename T>
bool containsAll(const C& container, list<T> values)
{
    return std::all_of(values.begin(), values.end()),
    [&container](const T& value) {
        return contains(container, value);
    });
}
```

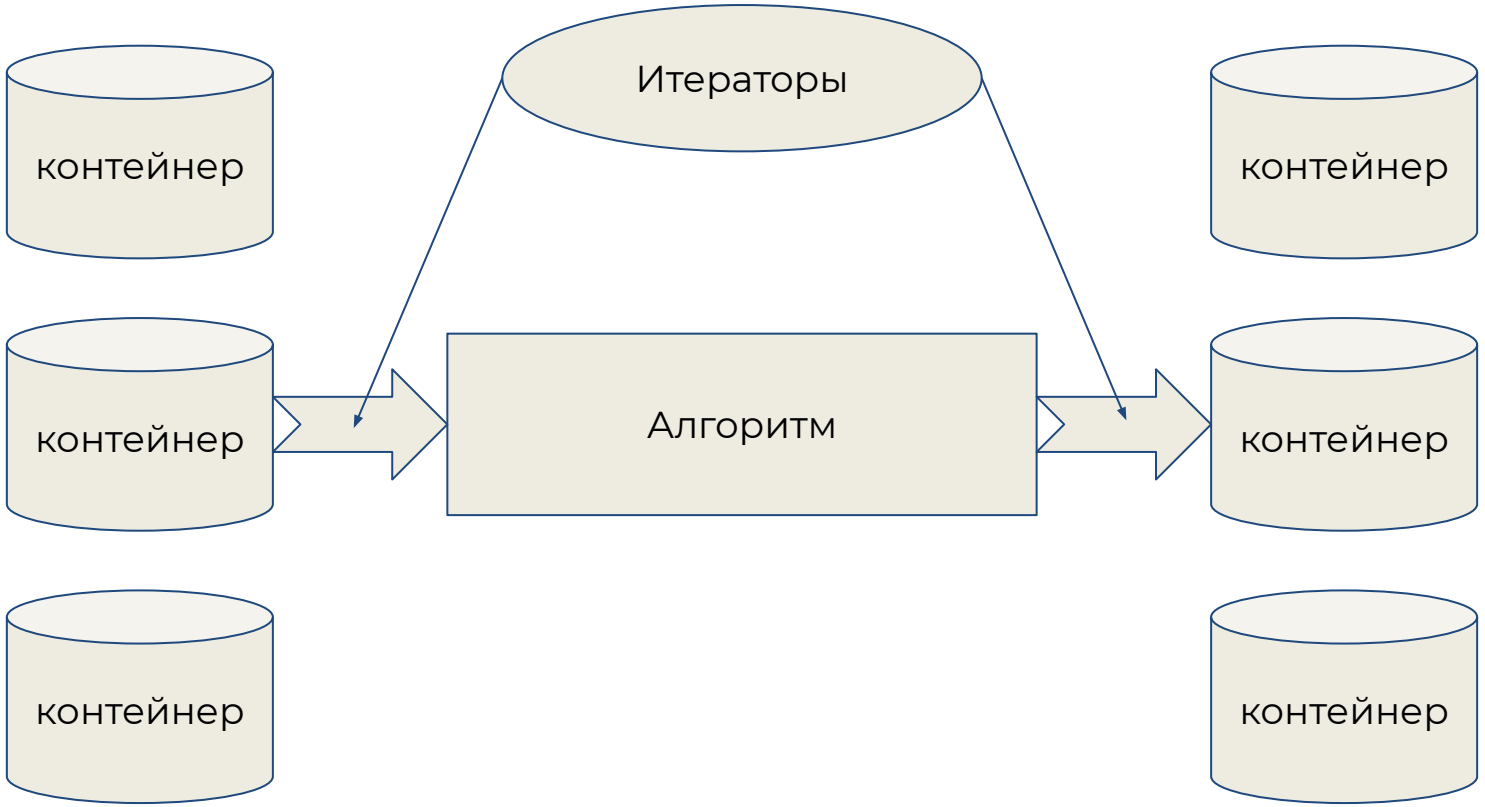
Компоненты STL

Контейнеры используются для управления коллекциями объектов различного типа.

Итераторы используются для обхода элементов в коллекциях объектов

Алгоритмы предназначены для обработки элементов коллекций.

Компоненты STL



Контейнеры

Последовательные контейнеры — упорядоченные коллекции, в которых каждый элемент занимает определённую позицию .

Ассоциативные контейнеры — упорядоченные коллекции, в которых позиция элементов зависит от его значения (или ключа) в соответствии с указанным критерием сортировки.

Неупорядоченные (ассоциативные) контейнеры — коллекции, позиция элемента в которых не имеет значения.

Контейнеры

Последовательные

- std::array
- std::vector
- std::deque
- std::list
- std::forward_list

Ещё контейнеры?

- строки
- plain C массивы
- пользовательские

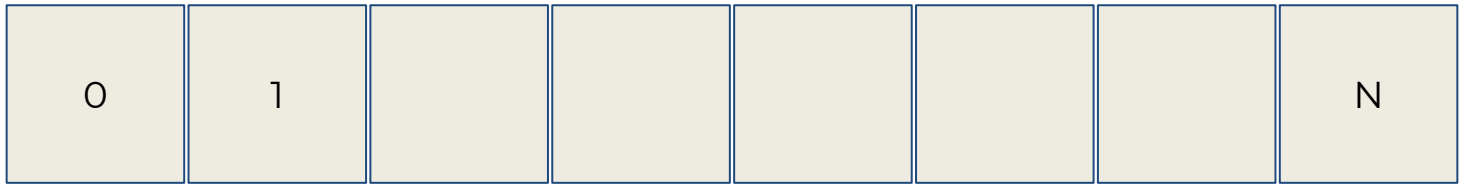
Ассоциативные

- std::set
- std::multiset
- std::map
- std::multimap

Неупорядоченные

- std::unordered_set
- std::unordered_multiset
- std::unordered_map
- std::unordered_multimap

Массивы




```
#include <array>

namespace std {
    template<typename T, size_t N>
    class array;
}
```

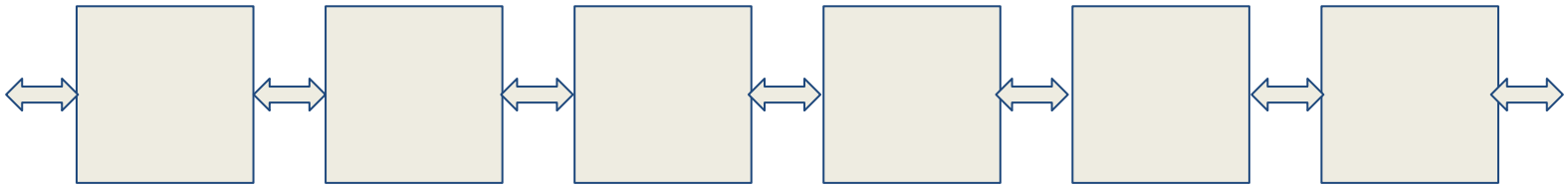
Векторы


```
#include <vector>

namespace std {
    template<
        typename T,
        typename Allocator = allocator<T>>
    class vector;
}
```

Деки

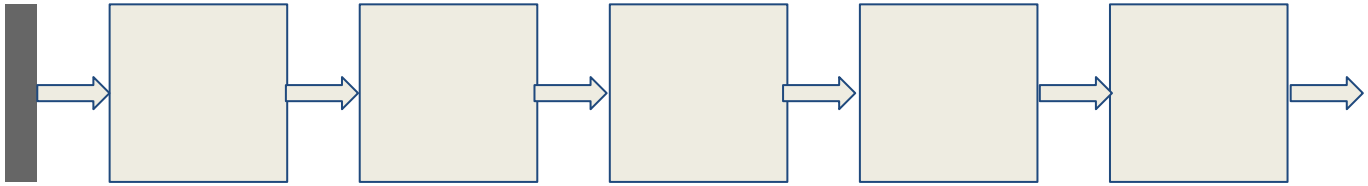
```
#include <deque>

namespace std {
    template<
        typename T,
        typename Allocator = allocator<T>>
    class deque;
}
```

Списки



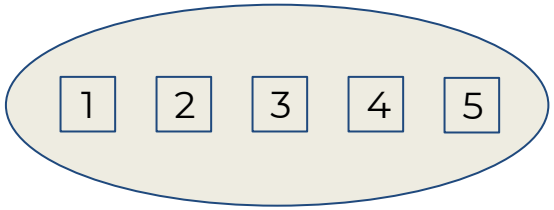
```
#include <list>

namespace std {
    template<
        typename T,
        typename Allocator = allocator<T>>
    class list;
}
```

Последовательные списки

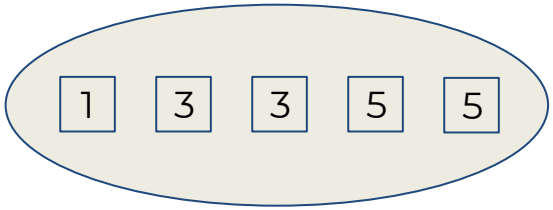


```
#include <forward_list>

namespace std {
    template<
        typename T,
        typename Allocator = allocator<T>>
    class forward_list;
}
```

Множества



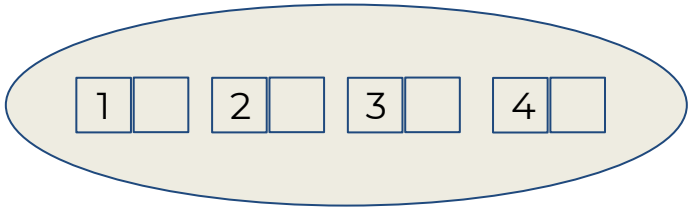
Множество



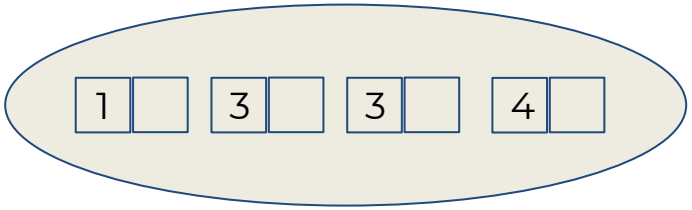
Мультимножество

```
#include <set>

namespace std {
    template<
        typename T,
        typename Compare = less<T>
        typename Allocator = allocator<T>>
    class set; // class multiset
}
```

Отображения



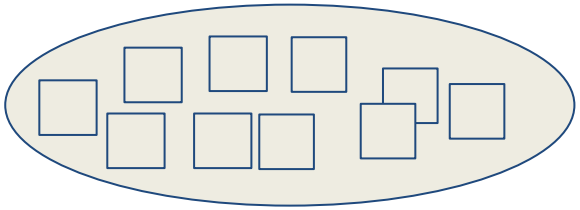
Отображение



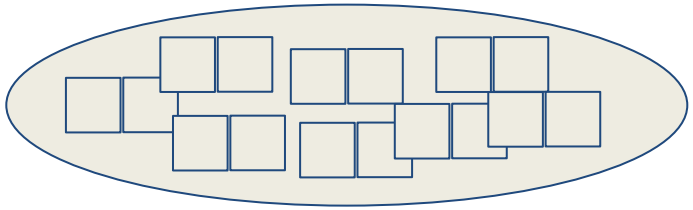
Мультитображение

```
#include <map>

namespace std {
    template<
        typename Key, typename T
        typename Compare = less<Key>
        typename Allocator = allocator<pair<const Key, T>>
    class map; // class multimap
}
```

Неупорядоченные контейнеры



Неупорядоченное множество/мультимножество



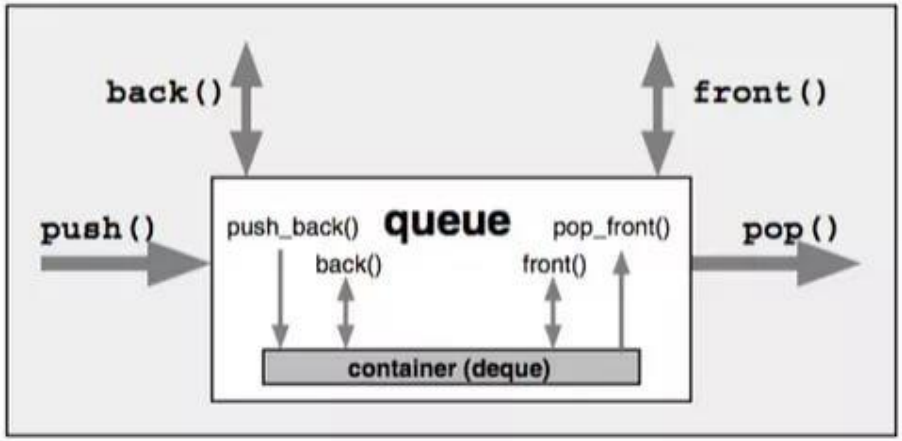
Неупорядоченное отображение/мультиотображение

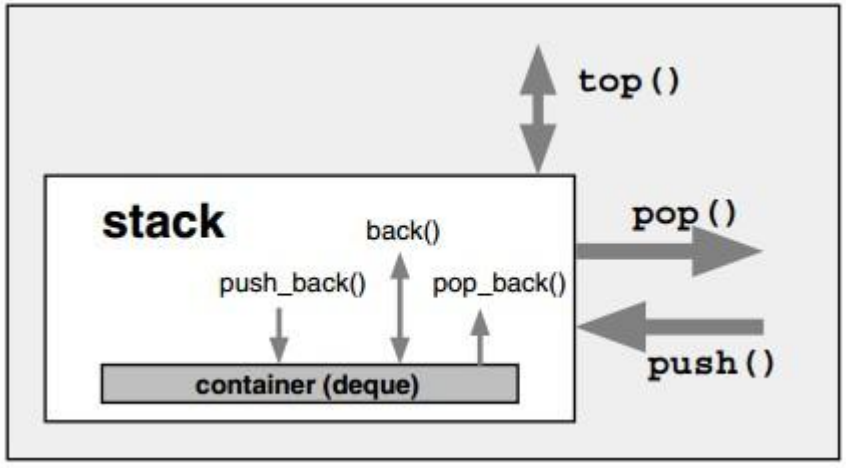
```
#include <unordered_set>

namespace std {
  template<
    typename T,
    typename Hash = hash<T>
    typename EqPred = equal_to<T>
    typename Allocator = allocator<T>>
  class unordered_set;
}
```

```
#include <unordered_map>

namespace std {
  template<
    typename Key, typename T
    typename Hash = hash<Key>
    typename EqPred = equal_to<Key>
    typename Allocator =
allocator<pair<const Key, T>>
  class unordered_map;
}
```

Адаптеры контейнеров

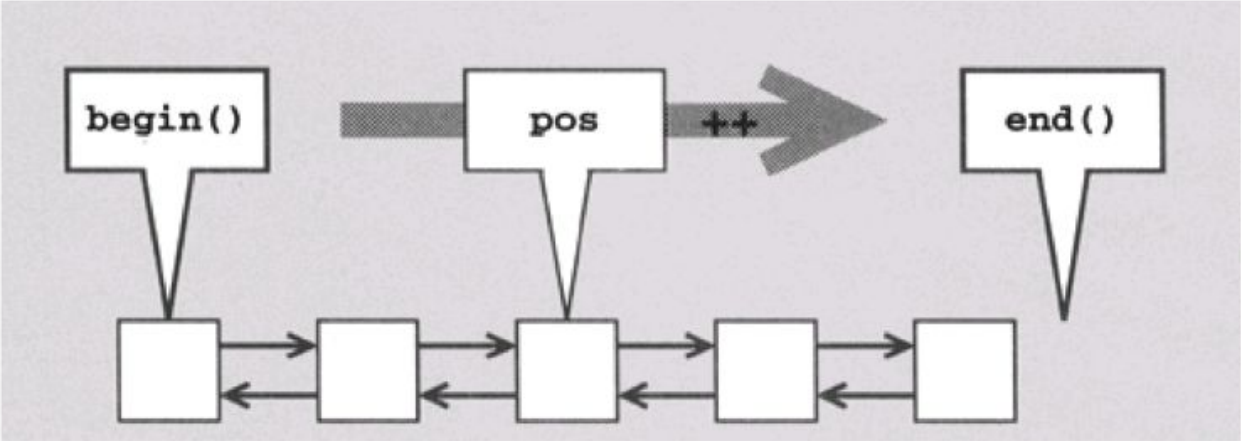
- std::stack
- std::queue
- std::priority_queue






```
template<
  class T,
  class Container = std::deque<T>
> class stack;
```

Итераторы



- Предоставляет доступ к элементу в текущей позиции
- Выполняет переход к следующей позиции

Категории итераторов

Категория	Многопереходность	Операции
Итератор вывода	Нет	`*it = value` `++it и it++`

Итератор ввода	Нет	`*it и it->member`
		`++it и it++`
		`it1 == it2 и it1 != it 2`

Однонаправленный	Да	
Двунаправленный	Да	`--it и it--`

С произвольным доступом	Да	`it[n]`
		`it += n`
		`it1 < it2`

Непрерывный	Да	`std::memmove, ...`

Адаптеры итераторов

- std::reverse_iterator
- std::move_iterator
- std::back_insert_iterator
- std::front_insert_iterator
- std::insert_iterator

Ещё немного итераторов

- std::istream_iterator
- std::ostream_iterator
- std::regex_iterator

Полезные функции

- std::advance
- std::distance
- std::next
- std::prev

Свойства итераторов

```
#include <iterator>

namespace std {
  template<typename T>
  struct iterator_traits {
    using iterator_category = typename T::iterator_category;
    using value_type = typename T::value_type;
    using difference_type = typename T::difference_type;
    using pointer = typename T::pointer;
    using reference = typename T::reference;
  };
}
```

Алгоритмы

https://en.cppreference.com/w/cpp/algorithm

- Немодифицирующие алгоритмы
- Модифицирующие алгоритмы
- Алгоритмы удаления
- Перестановочные алгоритмы
- Алгоритмы сортировки
- Алгоритмы для упорядоченных диапазонов
- Численные алгоритмы

std::find_if

```
Item firstItem = *items.first();

const auto it = std::find_if(items.cbegin(), items.cend(),
[&firstItem](Item* item) {
    return (item->font() != firstItem.font())
        || (item->background() != firstItem.background())
        || (item->text() != firstItem.text());
});

it->animate();
```

std::transform

```
template<typename Map>
std::vector<typename Map::key_type> keys(const Map& map)
{
    std::vector<typename Map::key_type> keys(map.size());

    std::transform(map.cbegin(), map.cend(), keys.begin(),
    [](const typename Map::value_type &pair) {
        return pair.first;
    });

    return keys;
}
```

std::accumulate

```
template<typename Container>
double averageValue(const Container& container) {
    int counter = 0;

    return std::accumulate(container.begin(), container.end(), 0.0,
    [&counter](double middle, const Point& point) {
        const double delta = point.y() - middle;
       return middle + (delta / ++counter);
    });
}
```

Диапазоны (Ranges)

https://en.cppreference.com/w/cpp/ranges

https://github.com/ericniebler/range-v3

```
int sum = accumulate(
    views::ints(1)
    | views::transform([](int i) { return i * i; })
    | views::take(10), 0
);
```

Курс-интенсив

Программирование на C++

Стандартная библиотека
Часть 1. STL

academy.rubius.com
konstantin.dobrychev@rubius.com
Константин Добрычев