

# Алгоритмы

# 1. Алгоритмы : основное понятие

---

**Алгоритм** – метод решения задачи, задающий последовательность выполнения операций над исходными данными (ввод) с целью получения результата (вывод) и обладающий следующими свойствами:

**Конечность**

**Определенность**

**Эффективность**

# Алгоритмы C++

# Algorithms

# Что есть в STL?

- Контейнеры
  - Хранят данные
- Алгоритмы (~100 в C++17)
  - Поиск и подсчет элементов
  - Модификация, копирование элементов, перестановки
  - Сортировка и разделение
  - Удаление элементов
  - Сравнение
  - Параллельная обработка (c++17)

# Алгоритмы сортировки

## Операции сортировки (на отсортированных диапазонах)

Заголовочный файл `<algorithm>`

|                                      |  |
|--------------------------------------|--|
| <code>is_sorted</code> (C++11)       | проверяет, является ли диапазон отсортированным в порядке возрастания<br>(шаблон функции) <a href="#">[править]</a>        |
| <code>is_sorted_until</code> (C++11) | находит наибольший отсортированный поддиапазон<br>(шаблон функции) <a href="#">[править]</a>                               |
| <code>sort</code>                    | сортирует диапазон в порядке возрастания<br>(шаблон функции) <a href="#">[править]</a>                                     |
| <code>partial_sort</code>            | сортирует первые N элементов в диапазоне<br>(шаблон функции) <a href="#">[править]</a>                                     |
| <code>partial_sort_copy</code>       | копирует и частично сортирует диапазон элементов<br>(шаблон функции) <a href="#">[править]</a>                             |
| <code>stable_sort</code>             | сортирует диапазон элементов при сохранении порядка между равными элементами<br>(шаблон функции) <a href="#">[править]</a> |
| <code>nth_element</code>             | частично сортирует диапазон относительно заданного элемента<br>(шаблон функции) <a href="#">[править]</a>                  |

## «O» большое

`std::sort`  $\Rightarrow O(N \log_2(N))$

`std::stable_sort`  $\Rightarrow O(N \log_2^2(N))$

`std::minmax_element`  $\Rightarrow O(N)$

`std::nth_element`  $\Rightarrow \sim O(N)$

`std::partial_sort`  $\Rightarrow O(N \log_2(S))$

# 1. nth\_element

Определён в заголовочном файле `<algorithm>`

```
template< class RandomIt >
void nth_element( RandomIt first, RandomIt nth, RandomIt last );
```

(1)

```
template< class RandomIt, class Compare >
void nth_element( RandomIt first, RandomIt nth, RandomIt last, Compare comp );
```

(2)

Частично виды диапазоне `[first, last)` в порядке возрастания так, чтобы все элементы в диапазоне `[first, nth)` являются 'меньше, чем в диапазоне `[nth, last)`. Первый вариант используется `operator<` для сравнения элементов, вторая версия использует данную `comp` функцию сравнения. Элемент, помещенный в `nth` позиции именно элемент, что будет происходить в этом положении, если диапазон был полностью отсортированный.

## Параметры

`first, last` — итераторы произвольного доступа определении диапазона рода  
`nth` — итератор произвольного доступа, определяющей точкой рода разделов  
`comp` — функция сравнения, возвращающая `true` если первый аргумент *меньше* второго.  
Сигнатура функции сравнения должна быть эквивалентна следующей:

```
bool cmp(const Type1 &a, const Type2 &b);
```

# `std::nth_element(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

- Если отсортировать `[beg, end)` то значение `mid` не изменится

- Слева от `mid` — значения *большие* или *равные* `mid`

- Справа от `mid` - значения *меньшие* или *равные* `mid`

|   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 3 | 1 | 2 | 5 | 9 | 8 | 7 | 6 | @ |
|---|---|---|---|---|---|---|---|---|---|---|



# `std::nth_element(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

Если отсортировать `[beg, end)` то значение `mid` не изменится

Слева от `mid` — значения *большие* или *равные* `mid`

Справа от `mid` - значения *меньшие* или *равные* `mid`

|   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 3 | 1 | 2 | 5 | 9 | 8 | 7 | 6 | @ |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ |

# std::nth\_element

Найти 5 людей с наименьшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end());
```

Найти 5 людей с наибольшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end(), std::greater<>{});
```

Найти 1001 позвонившего

```
std::nth_element(v.begin(), v.begin() + 1000, v.end());
```

# 2. partial\_sort

Определён в заголовочном файле `<algorithm>`

```
template< class RandomIt >
void partial_sort( RandomIt first, RandomIt middle, RandomIt last );
```

(1)

```
template< class RandomIt, class Compare >
void partial_sort( RandomIt first, RandomIt middle, RandomIt last, Compare comp );
```

(2)

Сортирует часть элементов в диапазоне `[first, last)` в порядке возрастания. Первые `middle - first` отсортированные элементы находятся в диапазоне `[first, middle)`. Не гарантируется сохранение порядка равных элементов. Порядок элементов в диапазоне `[middle, last)` не определен. Первый вариант использует `operator<` для сравнения элементов, вторая версия использует переданную функцию сравнения `comp`.

## Параметры

`first, last` — диапазон элементов для сортировки  
`comp` — функция сравнения, возвращающая `true` если первый аргумент *меньше* второго.

Сигнатура функции сравнения должна быть эквивалентна следующей:

```
bool cmp(const Type1 &a, const Type2 &b);
```

Сигнатура не обязана содержать `const &`, однако, функция не может изменять переданные объекты.

Типы `Type1` и `Type2` должны быть таковы, что объект типа `RandomIt` может быть разыменован и затем неявно преобразован в оба из них.

# `std::partial_sort(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

- `[beg, mid)` не изменятся, если отсортировать `[beg, end)`

- `[beg, mid)` - отсортированы

|   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 9 | 5 | 8 | 7 | 6 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::partial\_sort

Распределить 5 призовых мест по наименьшему кол-ву штрафных баллов

```
std::partial_sort(v.begin(), v.begin() + 5, v.end());
```

Покарать 5 школьников, пришедших последними на урок

```
std::partial_sort(v.begin(), v.begin() + 5, v.end(), std::greater<>{});
```

# Применение алгоритмов в конкретной задаче

1. Вывод (`copy`)
2. Проверка пограничных значений (`max_element`).
3. Выборка по условию (`using boost::copy`).
4. Проверка существования (`any-of`)

# Бектор Student

```
enum class Gender : uint8_t {Female, Male };
struct Student {
    string name;
    int age = 0;
    Gender gender = Gender::Male;
};
ostream& operator<<(ostream& out, const Student& p);

int main() {
    vector<Student> students = {
        { "Vasya", 20, Gender::Male }, { "Frosya", 18, Gender::Female },
        /* тысячи их */
    }
    ...
}
```


**Задание: рассмотреть работу некоторых алгоритмов библиотеки `algorithm`**

# 1. Вывод данных

## copy C++11

Плохо

```
for (size_t i = 0; i <= students.size(); ++i) {  
    cout << students[i] << "\n";  
}
```



Лучше!

```
for (auto& student : students) {  
    cout << student << "\n";  
}
```

Еще  
лучше!

```
copy(students, ostream_iterator<Student>(cout, "\n"));
```



## 2. Проверка пограничных значений (`max_element`).

Выведем самого взрослого студента в потоке

```
if (!students.empty()) {  
    size_t oldest = 0;  
    for (size_t i = 1; i < students.size(); ++i) {  
        if (students[i].age > students[oldest].age) {  
            oldest = i;  
        }  
    }  
    cout << "The oldest student is " << students[oldest] << "\n";  
}  
else {  
    cout << "No students\n";  
}
```

# max\_element

## C++17

```
auto orderedByAge = [](auto& lhs, auto& rhs) {  
    return lhs.age < rhs.age;  
};  
  
auto oldest = max_element(students.begin(), students.end(),  
orderedByAge);  
if (oldest != students.end())  
    cout << "The oldest student is " << *oldest << "\n";  
else  
    cout << "No students\n";
```

### 3. Выборка по условию.

**Вывести сначала всех студенток от 17 до 21,  
затем студентов от 18 до 25 лет**

```
for (size_t i = 0; i < students.size(); ++i) {  
    if (students[i].gender == Gender::Female &&  
        (students[i].age >= 17 && students[i].age <= 21)) {  
        cout << students[i] << "\n";  
    }  
}
```

```
for (size_t i = 0; i < students.size(); ++i) {  
    if (students[i].gender == Gender::Male &&  
        (students[i].age >= 18 && students[i].age <= 25)) {  
        cout << students[i] << "\n";  
    }  
}
```

# Range based for

```
for (auto& student : students) {  
    if ((student.gender == Gender::Female) &&  
        (student.age >= 17 && student.age <= 21)) {  
        cout << student << "\n";  
    }  
}
```

```
for (auto& student : students) {  
    if ((student.gender == Gender::Male) &&  
        (student.age >= 18 && student.age <= 25)) {  
        cout << student << "\n";  
    }  
}
```

# Copy Filtered-range C++17

```
using boost::copy;
using boost::adaptors::filtered;

auto ByGenderAndAge = [](Gender g, int minAge, int maxAge) {
    return [=](auto&& p) {
        return (p.gender == g) && p.age >= minAge && p.age <= maxAge;
    };
};

auto ToStdout = ostream_iterator<Student>(cout, "\n");

copy(students
    | filtered(ByGenderAndAge(Gender::Female, 17, 21)),
    ToStdout);
copy(students | filtered(ByGenderAndAge(Gender::Male, 18, 25)), ToStdout);
```

## 4. Проверка наличия по выборке.

Найдем студенток старше 20 лет

```
bool isWomenOlderThan20 = false;
for (size_t i = 0; i < students.size(); ++i) {
    if (students[i].age > 20 && students[i].gender == Gender::Female)
    {
        isWomenOlderThan20 = true;
        break;
    }
}

if (isWomenOlderThan20)
    cout << "There are student female older than 20\n";
else
    cout << "There are no student female not older than 20 \n";
```

# Range

```
bool isWomenOlderThan20 = false;
for (auto student : students) {
    if (student.age > 20 && student.gender == Gender::Female) {
        isWomenOlderThan20 = true;
        break;
    }
}

if (isWomenOlderThan20)
    cout << "There are student female older than 20\n";
else
    cout << "There are no student female not older than 20 \n";
```

# Алгоритм any\_of C++11

```
auto isFemaleOlderThan = [](int age) {  
    return [=](auto&& p) {  
        return p.gender == Gender::Female  
            && p.age > age;  
    };  
};  
  
if (any_of(students, isFemaleOlderThan(25)))  
    cout << "There are women older than 25\n";  
else  
    cout << "There are no women above 25\n";
```