3.29. Использование стандартных алгоритмов STL над контейнерами: std::sort, std:find, std::find_if, std::max_element, std::min_element, std::reverse, std::binary_search, std::lower_bou std::upper_bound, в том числе с произвольной функцией сравнения.

std::sort

std::sort применяется над контейнерами с итераторами категории Random Access Iterator для того, чтобы отсортировать их элементы на полуинтервале [first, last) за $O(nlogn)$, где n — количество элементов, first, last — итераторы. По умолчанию элементы сортируются по возрастанию, с применением произвольного компаратора этот порядок возможно изменить.

```
#include <iostream>
#include <algorithm>
#include <vector>

template<typename T>
struct myComp {
    bool operator()(T& a, T& b) {
        return a > b;
    }
};

int main() {
    std::vector<int> theormax = {10, 2, 3, 4};
    std::sort(theormax.begin(), theormax.end()); //[2, 3, 4, 10]
    std::sort(theormax.begin(), theormax.end(), myComp<int>()); //[10, 4, 3,
   2]
    return 0;
}
```

std::find, std::find_if

std::find используется для поиска конкретного элемента в контейнере: если он есть, то возвращается итератор на него, иначе на итератор end. std::find_if используется аналогичным образом для поиска элемента, отвечающего заданному условию.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>

int main() {
    std::vector<int> v{1, 2, 3, 4};
    int n1 = 3;
    int n2 = 5;
    auto is_even = [](int i){ return i%2 == 0; };

    auto result1 = std::find(begin(v), end(v), n1);
    auto result2 = std::find(begin(v), end(v), n2);
    auto result3 = std::find_if(begin(v), end(v), is_even);

    (result1 != std::end(v))
        ? std::cout << "v contains " << n1 << '\n'
        : std::cout << "v does not contain " << n1 << '\n';
    //v contains 3

```

```
    (result2 != std::end(v))
        ? std::cout << "v contains " << n2 << '\n'
        : std::cout << "v does not contain " << n2 << '\n';
    //v does not contain 5

    (result3 != std::end(v))
        ? std::cout << "v contains an even number: " << *result3 << '\n'
        : std::cout << "v does not contain even numbers\n";
    //v contains an even number: 2
}
```

std::max_element, std::min_element

std::max_element находит наибольшее значение в контейнере на полуинтервале [first, last), где first, last — итераторы категории не ниже Forward Iterator. Аналогичным образом находит наименьшее значение std::min_element.

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <cmath>

static bool abs_compare(int a, int b) {
    return (std::abs(a) < std::abs(b));
}

int main() {
    std::vector<int> v{ 3, 1, -14, 1, 5, 9 };
    std::vector<int>::iterator result;

    result = std::max_element(v.begin(), v.end());
    std::cout << "max element at: " <<
    std::distance(v.begin(), result) << '\n'; // 5

    result = std::max_element(v.begin(), v.end(), abs_compare);
    std::cout << "max element (absolute) at: " <<
    std::distance(v.begin(), result) << '\n'; // 2
}
```

std::reverse

std::reverse перемещает элементы в обратном порядке относительно их исходного расположения на полуинтервале [first, last), где first, last — итераторы категории не ниже Bidirectional Iterator.

std::binary_search, std::lower_bound, std::upper_bound

std::binary_search методом бинарного поиска (если поддерживается Random Access Iterator) пытается найти данный элемент на [first, last), если нашлось, возвращается true, иначе false.

```
#include <iostream>
#include <algorithm>
#include <vector>

int main() {
    std::vector<int> haystack {1, 3, 4, 5, 9};
    std::vector<int> needles {1, 2, 3};

    for (auto needle : needles) {
        std::cout << "Searching for " << needle << '\n';
        if (std::binary_search(haystack.begin(), haystack.end(), needle)) {
            std::cout << "Found " << needle << '\n';
        } else {
```

```
            std::cout << "no dice!\n";
        }
    }
}
//Searching for 1
//Found 1
//Searching for 2
//no dice!
//Searching for 3
//Found 3
```

std::lower_bound ищет первый элемент, не меньший данного, на [first, last), std::upper_bound ищет первый элемент, больший данного, на [first, last). В случае успеха возвращается итератор на найденный элемент, иначе возвращается итератор на end.