**Использование итераторов в алгоритмах и контейнерах**

**ОБЩИЙ БАЛЛ 50**

**Вопрос 1**

**Дана функция f, принимающая число и возвращающая true, если для него выполнено некоторое условие, и false в противном случае. Кроме того, дано множество чисел s и пустой вектор v:**

int main() {

  set<int> s = { /\* какие-то числа \*/ };

  vector<int> v;

  return 0;

}

**Отметьте отрывки кода, которые заполнят вектор v всеми элементами множества s, не удовлетворяющими условию f.**

**28 баллов**



copy\_if(begin(s), end(s), back\_inserter(v),

        [](int x) { return !f(x); });



vector<int> garbage;

partition\_copy(

    begin(s), end(s),

    back\_inserter(v), back\_inserter(garbage), f);



v.assign(begin(s), end(s));

sort(begin(v), end(v),

     [](int lhs, int rhs) { return f(lhs) > f(rhs); });

auto it = partition\_point(begin(v), end(v), f);

v.erase(begin(v), it);



vector<int> garbage;

partition\_copy(

    begin(s), end(s),

    back\_inserter(garbage), back\_inserter(v), f);



v.assign(begin(s), end(s));

sort(begin(v), end(v),

     [](int lhs, int rhs) { return f(lhs) > f(rhs); });

auto it = partition\_point(begin(v), end(v), f);

v.erase(it, end(v));



v.assign(begin(s), end(s));

auto it = remove\_if(begin(v), end(v), f);

v.erase(it, end(v));



copy\_if(begin(s), end(s), back\_inserter(v), f);



auto it = remove\_if(begin(s), end(s),

                    [](int x) { return !f(x); });

copy(it, end(s), back\_inserter(v));



v.assign(begin(s), end(s));

auto it = partition(begin(v), end(v), f);

v.erase(it, end(v));



v.assign(begin(s), end(s));

auto it = partition(begin(v), end(v), f);

v.erase(begin(v), it)



v.assign(begin(s), end(s));

auto it = remove\_if(begin(v), end(v),

                    [](int x) { return !f(x); });

v.erase(it, end(v));



auto it = partition(begin(s), end(s), f);

copy(it, end(s), back\_inserter(v));



remove\_copy\_if(begin(s), end(s), back\_inserter(v),

               [](int x) { return !f(x); });



remove\_copy\_if(begin(s), end(s), back\_inserter(v), f);

Вопрос 2

Даны итераторы вектора, строки и множества:

vector<int> vec;

auto vector\_begin = begin(vec);

auto vector\_end = end(vec);

string str;

auto string\_begin = begin(str);

auto string\_end = end(str);

set<int> s;

auto set\_begin = begin(s);

auto set\_end = end(s);

**Отметьте вызовы алгоритмов, которые скомпилируются.**

**12 баллов**



partial\_sum(set\_begin, set\_end, vector\_begin);



res = is\_heap(string\_begin, string\_end);



partial\_sum(set\_begin, set\_end, back\_inserter(vec));



partial\_sum(vector\_begin, vector\_end, set\_begin);



auto res = is\_permutation(set\_begin, set\_end, back\_inserter(vec));



auto res = accumulate(vector\_begin, vector\_end, 0);

auto res = accumulate(set\_begin, set\_end, 0);



auto res = is\_heap(set\_begin, set\_end);



auto res = next\_permutation(set\_begin, set\_end);



auto res = is\_permutation(vector\_begin, vector\_end, vector\_begin);



auto res = is\_permutation(set\_begin, set\_end, vector\_begin);



auto res = next\_permutation(string\_begin, string\_end);

**Вопрос 3**

**Рассмотрим вектор v. Будучи вызванным от begin(v) и end(v), алгоритм is\_sorted\_until найдёт максимальный префикс (участок в начале) вектора, отсортированный по возрастанию. Что найдёт этот же алгоритм, если его вызвать от обратных итераторов — rbegin(v) и rend(v)?**

***Примечание.*** *Строго говоря, элементы могут быть равны, поэтому вместо возрастания здесь следует говорить о неубывании, а вместо убывания — о невозрастании. Тем не менее, для большей ясности мы будем использовать менее корректные термины «возрастание» и «убывание».*

10 баллов

Максимальный префикс вектора, отсортированный по возрастанию.

Максимальный префикс вектора, отсортированный по убыванию.

Максимальный суффикс (участок в конце) вектора, отсортированный по возрастанию.

Максимальный суффикс (участок в конце) вектора, отсортированный по убыванию.