

```
#include <algorithm>
```

У STL понад 100 алгоритмів. Ми будемо розглядати наступні алгоритми:

- мікроалгоритми
- алгоритми, що не модифікують послідовності
- алгоритми типу find
- алгоритми модифікації

C++11. for() to collections

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5  inline ostream& operator<<(ostream& out, const vector<int>& v) {
6      cout<< "<-";
7      for (auto& a : v) {
8          cout << a << " ";
9      }
10     cout << "->";
11     return out;
12 }
13 int main(void)
14 {
15     vector<int> v = { 13,43,63,857,13,76,134,6,99 };
16     cout << v << endl; // <- 13 43 63 857 13 76 134 6 99 ->
17     system("pause");
18     return (0);
19 }
```

//----- МІКРОАЛГОРИТМИ -----

```
vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
//*****1*****swap(T &a, T &b)
```

```
//змінює місцями значення двох елементів
```

```
swap(v[0], v[3]);
```

```
//виведення на екран v{ 4, 2, 3, 1, ... };
```

```
//*****2*****iter_swap(It p, It q)
```

```
//змінює місцями значення двох елементів, на які вказують ітератори
```

```
vector<int>::iterator it_a = v.begin() + 1;
```

```
vector<int>::iterator it_b = v.begin() + 2;
```

```
iter_swap(it_a, it_b);
```

```
//виведення на екран v{ 4, 3, 2, 1, ... };
```

```
//*****3*****max(const T &a, const T &b )
```

```
//повертає максимальний елемент з двох
```

```
cout<< max(v[3], v[v.size()-1])<<endl;//v[3]=1,v[v.size()-1] = 10
```

```
//виведення на екран : 10;
```

```
//*****4*****min(const T &a, const T &b )
```

```
//повертає мінімальний елемент з двох
```

```
cout << min(v[3], v[v.size() - 1]) << endl;//v[3]=1,v[v.size()-1] = 10
```

```
//виведення на екран : 1;
```

STL Algorithm. Алгоритми, що не модифікують послідовності

```
//----- АЛГОРИТМИ, ЩО НЕ МОДИФІКУЮТЬ ПОСЛІДОВНОСТІ -----  
vector<int> v{ 1, 2, 3, 2, 5, 2, 7, 8, 9, 10 };  
//****1****size_t count(It p, It q, const T &x)  
//повертає к-сть входжень елемента x у послідовність, задану ітераторами p та q  
vector<int>::iterator p = v.begin();  
vector<int>::iterator q = v.end();  
int target = 2;  
cout << "value " << target << " count " << count(p, q, target) << endl;  
//виведення на екран : value 2 count 3  
//****2****size_t count_if(It p, It q, Pr pred)  
//повертає к-сть true, які повертає предикат pred  
struct {  
    bool operator()(int target) {  
        return target % 2 == 0;  
    }  
}check_div_by_2; //перевіряє чи елемент парний  
cout << "check_div_by_2 = " << count_if(p, q, check_div_by_2) << endl; //check_div_by_2 = 5  
//lambda-вираз заміняє предикат check_div_by_2  
cout << "check_div_by_2 = " << count_if(p, q, [](int i) {return i % 2 == 0; }) << endl;
```


STL Algorithm. Алгоритми типу find

find(It p, It q, **const** T & x)

Повертає ітератор на перше входження елемента x в послідовність, задану ітераторами p і q.

find_if(It p, It q, Pr pred)

Повертає ітератор на перший елемент, для якого предикат pred повернув значення **true**.

find_first_of(It p, It q, Itr i, Itr j)

Повертає ітератор на перше входження будь-якого елементу з послідовності, заданої ітераторами i і j, в послідовність, задану ітераторами p і q. Послідовності можуть бути різних типів (наприклад std::vector і std::list).

```
vector<int> v{ 2, 3, 4, 7, 5 };
```

```
vector<int> v2{ 10, 99, 76, 5 };
```

```
vector<int>::iterator match =
```

```
    find_first_of(v.begin(), v.end(), v2.begin(), v2.end());
```

```
cout << "match found -> " << *match << endl; // match found -> 5
```

STL Algorithm. find()

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5  int main(void)
6  {
7      vector<int> v{ 22,1,9,10,88,0,7,4 };
8      int availability_check = 88;
9      if (find(v.begin(), v.end(), availability_check) != v.end())
10         cout << "num " << availability_check << " is present!\n";
11                                     //num 88 is present!
12     else
13         cout << "num " << availability_check << " is absent!\n";
14     system("pause");
15     return (0);
16 }
```

STL Algorithm. Алгоритми типу find

`min_element(It p, It q)`

Повертає ітератор на мінімальний елемент послідовності

`max_element(It p, It q)`

Повертає ітератор на максимальний елемент послідовності

`equal(It p, It q, Itr i)`

Порівнює дві послідовності на еквівалентність. Друга послідовність задається одним ітератором, так як послідовності повинні бути однакової довжини. Якщо друга коротша, то undefined behavior

```
vector<int> v1{ 2, 3, 4, 7, 5 };  
vector<int> v2{ 2, 3, 4, 7, 5 };  
bool check = equal(v1.begin(), v1.end(), v2.begin(), v2.end());  
(check) ? cout << "v1 & v2 are equals\n"  
          : cout << "v1 & v2 aren't equals\n";
```

STL Algorithm. min_element(), max_element()

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5  //пошук min i max значень елементів та їх індексів
6  int main(void)
7  {
8      vector<int> v{ 22, 1, 9, 10, 88, -1, 7, 4 };
9      int min = *min_element(v.begin(), v.end());
10     int min_index = min_element(v.begin(), v.end()) - v.begin();
11     cout << "min = " << min << "\tmin_index = " << min_index << endl;
12     int max = *max_element(v.begin(), v.end());
13     int max_index = max_element(v.begin(), v.end()) - v.begin();
14     cout << "max = " << max << "\tmax_index = " << max_index << endl;
15     system("pause");
16     return (0);
17 }
```

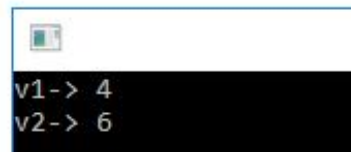
```
min = -1          min_index = 5
max = 88          max_index = 4
Для продолжения нажмите любую клавишу . . .
```


STL Algorithm. Алгоритми типу find

`pair <It, Itr> mismatch(It p, It q, Itr i)`

Повертає пару ітераторів, що вказує на першу розбіжність послідовностей

```
vector<int> v1{ 2, 3, 4, 7, 5 };  
vector<int> v2{ 2, 3, 6, 7, 5 };  
pair<vector<int>::iterator, vector<int>::iterator> result;  
result = mismatch(v1.begin(), v1.end(), v2.begin());  
cout << "v1-> " << *result.first << '\n'  
      << "v2-> " << *result.second << '\n'  
      << endl;
```



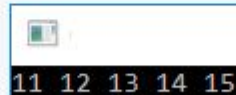
```
v1-> 4  
v2-> 6
```

STL Algorithm. Алгоритми типу find

F for_each(It p, It q, F func)

Для кожного елемента послідовності застосовує функтор func. Значення, що повертається функтором, після кожного застосування ігнорується. Повертає функтор func після його застосування до всіх елементів.

```
vector<int> v{ 1, 2, 3, 4, 5 };  
for_each(begin(v), end(v), [](int& n) { return n += 10; });  
for (auto& a : v) {  
    cout << a << " ";  
}
```



11 12 13 14 15

STL Algorithm. Алгоритми модифікації

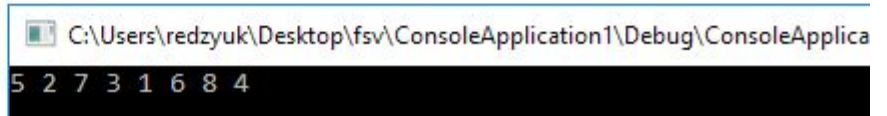
```
vector<int> v{ 1, 2, 3, 4, 5 };  
//*****1*****fill(It p, It q, const T &x),  
//*****fill_n(It p, Size n, const T &x)  
//заповнює колекцію значеннями, рівними x  
fill(begin(v), end(v), 20);  
for (auto& a : v) cout << a << " ";  
//виведення на екран : 20 20 20 20 20;  
fill_n(begin(v), 3, 10);  
for (auto& a : v) cout << a << " ";  
//виведення на екран : 10 10 10 20 20;
```

STL Algorithm. Алгоритми модифікації

```
vector<int> v(7,1);  
//*****2*****generate(It p, It q, F gen),  
//*****generate_n(It p, Size n, F gen)  
//заповнює колекцію значеннями, згенерованими функтором gen  
//(наприклад генератором випадкових чисел)  
srand(time(0));  
generate(begin(v), end(v), []() {return rand()%20-7 ; });  
    for (auto& a : v)  
        cout << a << " ";  
    cout << endl;  
//виведення на екран -4 11 7 -3 0 -1 8;  
vector<int> v2(7, 1);  
generate_n(v2.begin(), 4, []() { return rand() % 10 - 2; });  
for (auto& a : v2)  
    cout << a << " ";  
//виведення на екран 7 2 5 -2 1 1 1;
```


STL Algorithm. Алгоритми модифікації

```
vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8 };  
//*****3*****random_shuffle(It p, It q)  
//перемішує елементи колекції у випадковому порядку,  
//третім параметром може задаватись функтор-генератор  
random_shuffle(begin(v), end(v));  
    for (auto& a : v)  
        cout << a << " ";
```



STL Algorithm. Алгоритми модифікації

copy (It p, It q, ltr out)

Копіює значення елементів послідовності, заданої ітераторами p і q, в послідовність, що починається з ітератора out.

copy_backward (It p, It q, ltr out)

Копіює елементи послідовності, заданої ітераторами p і q, в послідовність, що закінчується ітератором out. Ітератори повинні бути BiDi.

```
vector<int> sour{ 1, 2, 3, 4, 5, 6, 7, 8 };
vector<int> dest(5); //якщо розмір цільової колекції недостатній,
                    //виникне помилка часу компіляції!!!
vector<int>::iterator d = dest.begin();
copy(begin(sour), begin(sour) + 5, begin(dest));
for (auto& a : dest)
    cout << a << " ";
cout << endl;
```



STL Algorithm. Алгоритми модифікації

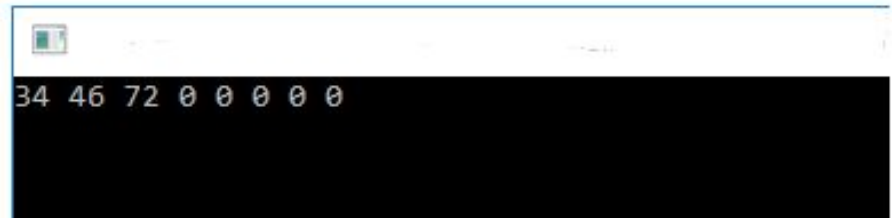
remove_copy (It p, It q, ltr out, const T & x)

Копіює значення елементів з послідовності, заданої ітераторами p і q, в послідовність, що починається з ітератора out, за винятком елементів, значення яких дорівнюють значенню x.

remove_copy_if (It p, It q, ltr out, Pr pred)

Копіює значення елементів з послідовності, заданої ітераторами p і q, в послідовність, що починається з ітератора out, за винятком елементів, для яких предикат pred повертає значення true.

```
vector<int> sour{ 11, 22, 34, 46, 55, 66, 72, 88 };
vector<int> dest(sour.size());
//якщо розмір цільової колекції недостатній,
//виникне помилка часу компіляції!!!
remove_copy_if(sour.begin(), sour.end() - 1,
               dest.begin(), [](int n) {return n % 11 == 0; });
for (auto& a : dest)
    cout << a << " ";
cout << endl;
```



```
34 46 72 0 0 0 0 0
```

STL Algorithm. Алгоритми модифікації

reverse (It p, It q)

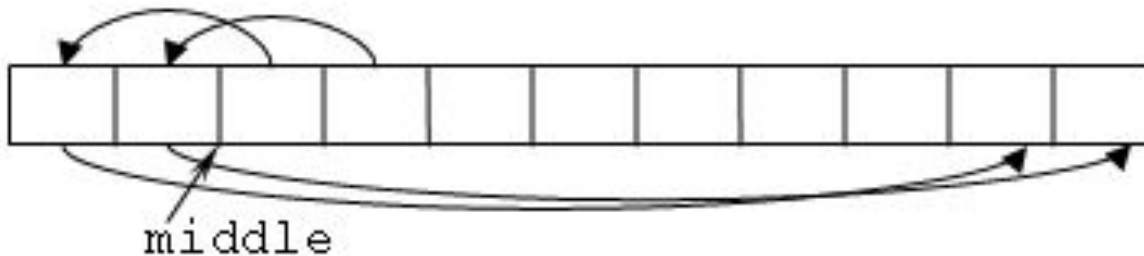
Переставляє елементи в зворотному порядку

reverse_copy (It p, It q, ltr out)

Копіює значення елементів в зворотному порядку

rotate (It p, It middle, It q)

Зрушує елементи послідовності так, що елемент, на який вказує ітератор middle стає першим.



```
vector<int> v{ 0, 10, 20, 30, 40, 50, 60 };  
rotate(begin(v), begin(v) + 2, end(v));  
for (auto& a : v )  
    cout << a << " ";  
cout << endl;
```

```
20 30 40 50 60 0 10
```


STL Algorithm. Алгоритми модифікації

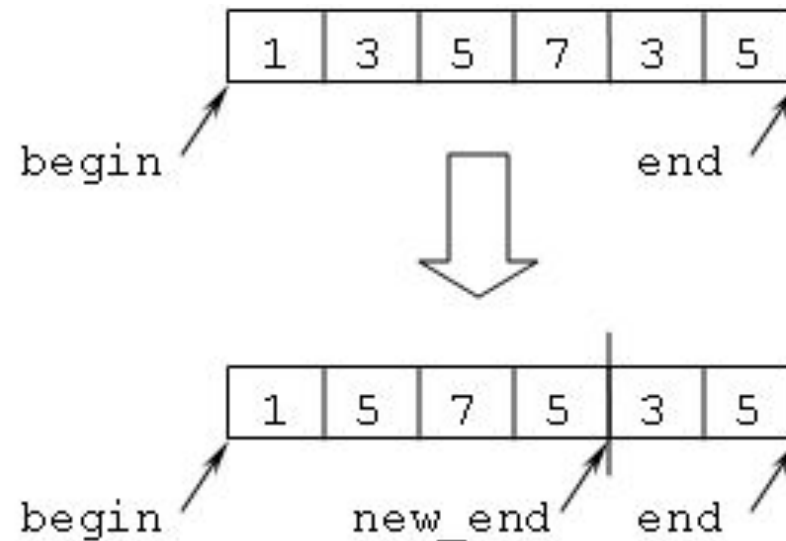
`remove(It p, It q, const T &x)`

Видаляє з послідовності елементи, значення яких збігаються за значенням з `x`. Повертає ітератор на новий кінець послідовності.

Насправді елементи не видаляються, оскільки алгоритму не передається колекція

//видалити всі трійки

```
vector<int>::iterator new_end = std::remove(v.begin(), v.end(), 3);
```



Правильний варіант видалення:

```
v.erase(std::remove(v.begin(), v.end(), 3), v.end());
```

Видаляємо все, що знаходиться між ітератором, який повернув `remove`, і кінцем послідовності. Це називається **remove-erase-idiom**

STL Algorithm. sort()

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5  //пряме та зворотне сортування колекції
6  ostream& operator<<(ostream& out, vector<int>& v) {
7      for (auto& a : v) {
8          out << a << " ";
9      }
10     out << endl;
11     return out;
12 }
13 int main(void)
14 {
15     vector<int> v{ 22, 1, 9, 10, 88, -1, 7, 4 };
16     cout << "non sorted vector\t : " << v << endl;
17     sort(begin(v), end(v));
18     cout << "Forward iterator sorting : " << v << endl;
19     sort(rbegin(v), rend(v));
20     cout << "Back iterator sorting : " << v << endl;
21     system("pause");
22     return (0);
23 }
```

```
non sorted vector      : 22 1 9 10 88 -1 7 4
Forward iterator sorting : -1 1 4 7 9 10 22 88
Back iterator sorting   : 88 22 10 9 7 4 1 -1
```

STL Algorithm. Інтервальний конструктор

```
4 using namespace std;
5 //інтервальний конструктор
6 ostream& operator<<(ostream& out, vector<int>& v) {
7     for (auto& a : v) {
8         out << a << " ";
9     }
10    out << endl;
11    return out;
12 }
13 int main(void)
14 {
15     vector<int> v{ 22, 1, 9, 10, 88, -1, 7, 4 };
16     cout <<"v\t : " << v <<endl;
17     sort(begin(v), end(v));
18     cout << "v sorted : " << v << endl;
19     //копіювати 5 перших значень колекції v у колекцію v2
20     //за допомогою інтервального конструктора
21     vector<int> v2{ v.begin(), v.begin() + 5 };
22     cout << "v2 content : " << v2 << endl;
23     system("pause");
```

**THANK YOU
FOR
YOUR
ATTENTION
ANY QUESTIONS?**

