

# Алгоритмизация и программирование

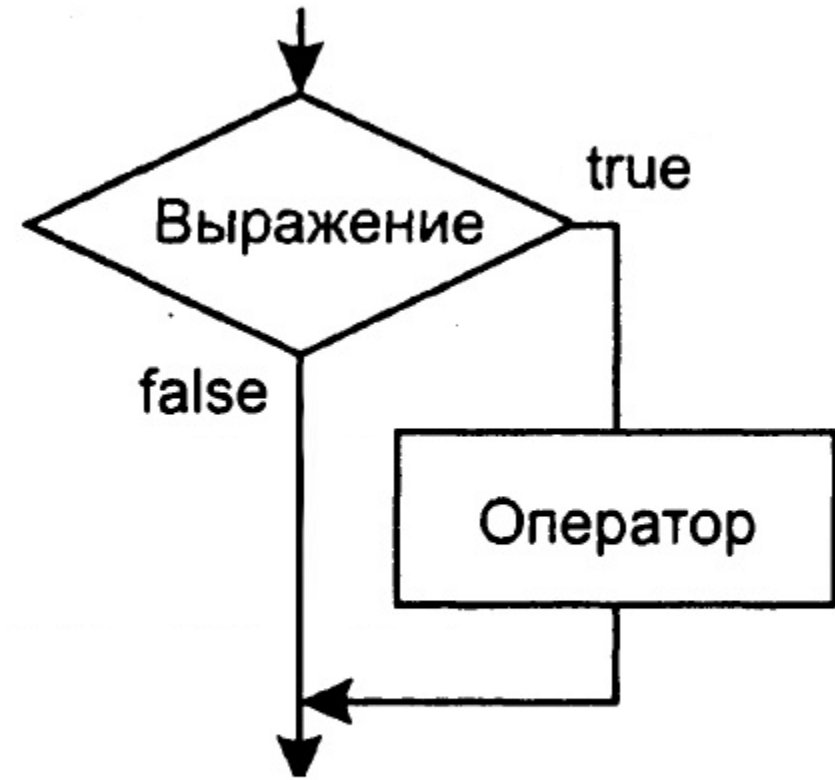
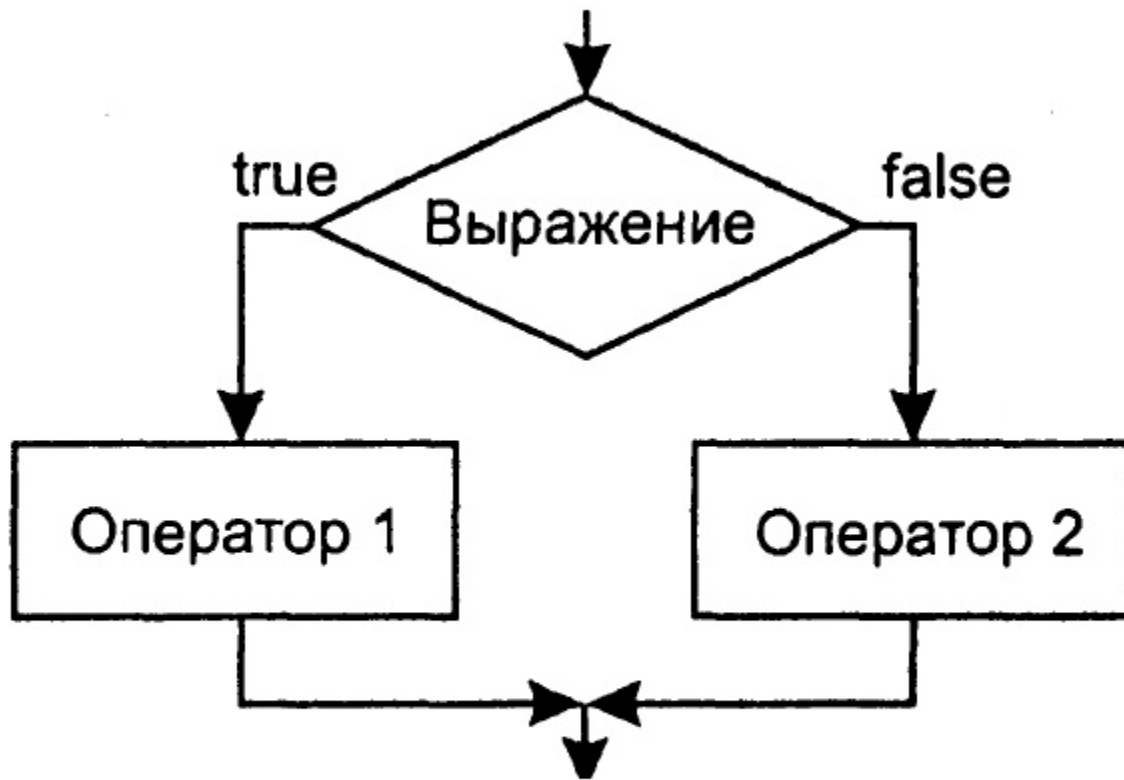
## Лекция 2

# Учебная задача 1

Даны 2 целых числа. Выведите на экран наибольшее из них. Если числа равны, то выведите любое из заданных чисел.

<https://wandbox.org/permlink/XvyRjmzdXqzuYK8J>

# Условный оператор



Структурная схема условного оператора

# if

Ключевое слово **если**

Выражение

Тело

The diagram shows the components of a C++ `if` statement. An arrow points from the label 'Ключевое слово **если**' to the word `if`. Another arrow points from 'Выражение' to the expression `(a > b)`. A third arrow points from 'Тело' to the body `std::cout << "Hello";`. The code is color-coded: `if` is purple, `(a > b)` is black, `std::cout` is blue, `<<` is black, `"Hello"` is red, and `;` is black.

```
if (a > b) std::cout << "Hello";
```

Ключевое слово **если**

Выражение

Тело

The diagram shows the components of a C++ `if` statement with a block body. An arrow points from the label 'Ключевое слово **если**' to the word `if`. Another arrow points from 'Выражение' to the expression `(a > b)`. A third arrow points from 'Тело' to the block body `{ std::cout << "Hello"; }`. The code is color-coded: `if` is purple, `(a > b)` is black, `{` is black, `std::cout` is blue, `<<` is black, `"Hello"` is red, `;` is black, and `}` is black.

```
if (a > b){  
    std::cout << "Hello";  
}
```

# if (выражение)

Ожидается, что выражение в скобках типа `bool`, поэтому будет попытка неявно [преобразовать его к `bool`](#).

Если преобразование не допустимо, то – ошибка.

if(инициализация; проверка)

init-statement  
↓  
if (int res = a > b; res) std::cout << "Hello";  
else std::cout << "Bye";

# if-else

Выражение



Ключевое слово **если** → **if** (a > b) **std::cout** << "Hello"; ← Тело **if**

Ключевое слово **иначе** → **else** **std::cout** << "Bye"; ← Тело **else**

Выражение



Ключевое слово **если** → **if** (a > b){  
    **std::cout** << "Hello"; ← Тело **if**

Ключевое слово **иначе** → } **else** {  
    **std::cout** << "Bye"; ← Тело **else**  
}

### if

```
if (/* условие */)
{
    /* true */
}
```

### If-else

```
if (/* условие */)
{
    /* true */
}
else
{
    /* false */
}
```

### If-else if

```
if (/* условие 1 */)
{
    /* true */
}
else if (/* условие 2 */)
{
    /* true */
}
else
{
    /* false */
}
```

### Вложенный if

```
if (/* условие */)
{
    if (/* условие */)
    {
        /* true */
    }
    else
    {
        /* false */
    }
}
else
{
    if (/* условие */)
    {
        /* true */
    }
    else
    {
        /* false */
    }
}
```



# If-else (ошибки)

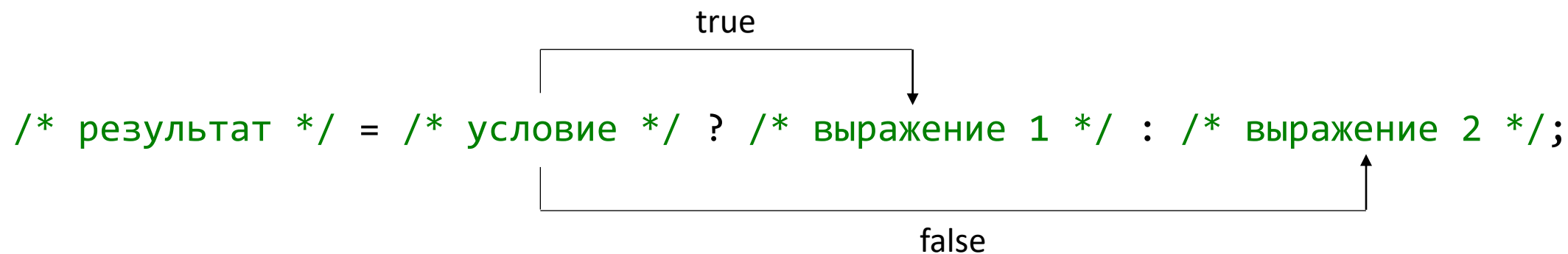
```
if (a > b);
```

```
if (a > b){/* код */};  
else {/* код */}
```

```
if (a > b)  
    if (a > c) std::cout << "Hello";  
else std::cout << "Bye";
```

# Тернарный оператор (?:)

```
variable = a > b ? a : b;
```



\* выражение 1 и выражение 2 должны быть одного или приводимого к одному типу

# Логические операторы

Название	Как выглядит		Как использовать	
И	&&	and	a && b;	a and b;
ИЛИ		or	a    b;	a or b;
НЕ	!	not	!a;	not a;

a	b	a and b	a or b	not a
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

Логические операторы применяются только к операндам типа `bool`, поэтому перед их применением будет попытка преобразовать операнды в `bool`. Если это не возможно, то получаем ошибку.

Операторы И и ИЛИ вычисляются по сокращённым правилам, т.к. если результат можно получить вычислив первый аргумент, второй не вычисляется:

```
false && std::cout << 1; // пусто
```

```
true && std::cout << 2; // 2
```

# Логические операторы

```
int a = 5;  
(a > 1) and (a < 10) // true  
(a == 5) or (a == 10) // true  
not (a == 10) // true  
-1 < a < 2 // true
```

Последнее выражение вычисляется последовательно:  $(-1 < a) < 2 \rightarrow \text{true} < 2$   
если хотите получить результат по математическим правилам пишите:  $(-1 < a) \text{ and } (a < 2)$

```
(a > 1) and (a < 3) // false  
(a == 1) or (a == 3) // false  
not (a == 5) // false
```

# Учебная задача 2

Даны 3 целых числа. Выведите на экран наибольшее из них. Если числа равны, то выведите любое из заданных чисел.

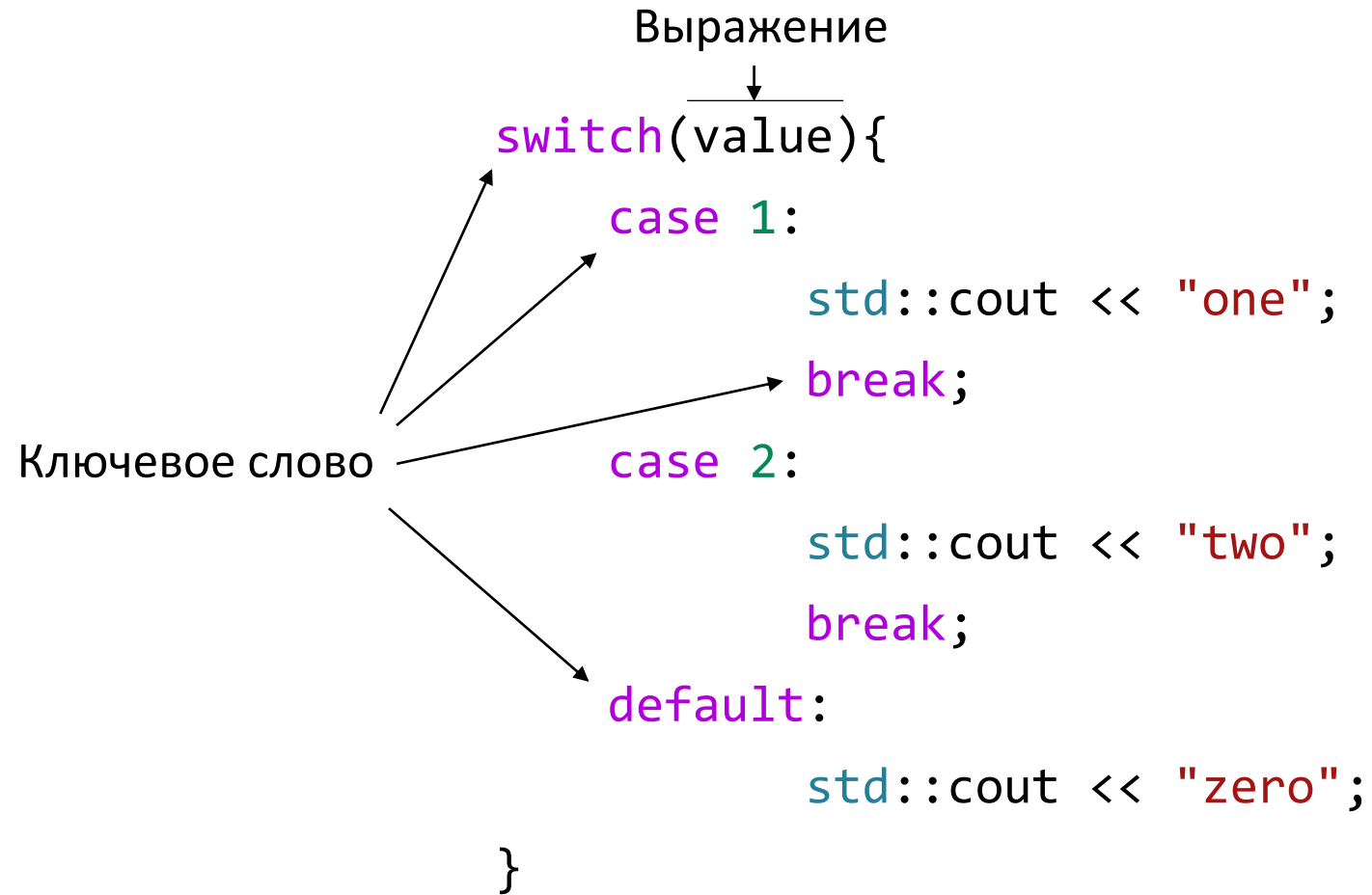
<https://wandbox.org/permlink/LbN9rAyUwpStKe6h>

# Учебная задача 3

Дано число, которое означает количество секунд на таймере. Выведите на экран это число и соответствующее количеству секунд слово, например: 1 секунда, 2 секунды, 5 секунд и т.д.

<https://wandbox.org/permlink/NHDaTZVTkHQp9FeC>

# switch



# switch (выражение)

Выражение перечислимого типа (целого), enum или что-то, что можно преобразовать в эти типы.



# switch(инициализация; выражение)

init-statement  
↓

```
switch (int res = a > b; res ){  
    case true: std::cout << ":)"; break;  
    default: std::cout << ":(  
}
```

# Учебная задача 4

Модифицируйте код предыдущей задачи, чтобы можно было вводить данные многократно без необходимости перезапускать программу после каждого ввода. Выход из программы должен осуществляться если пользователь ввёл отрицательное число.

<https://wandbox.org/permlink/cvJiXAjcTIFloXym>

# goto

```
label:  
    /* код */  
goto label;
```

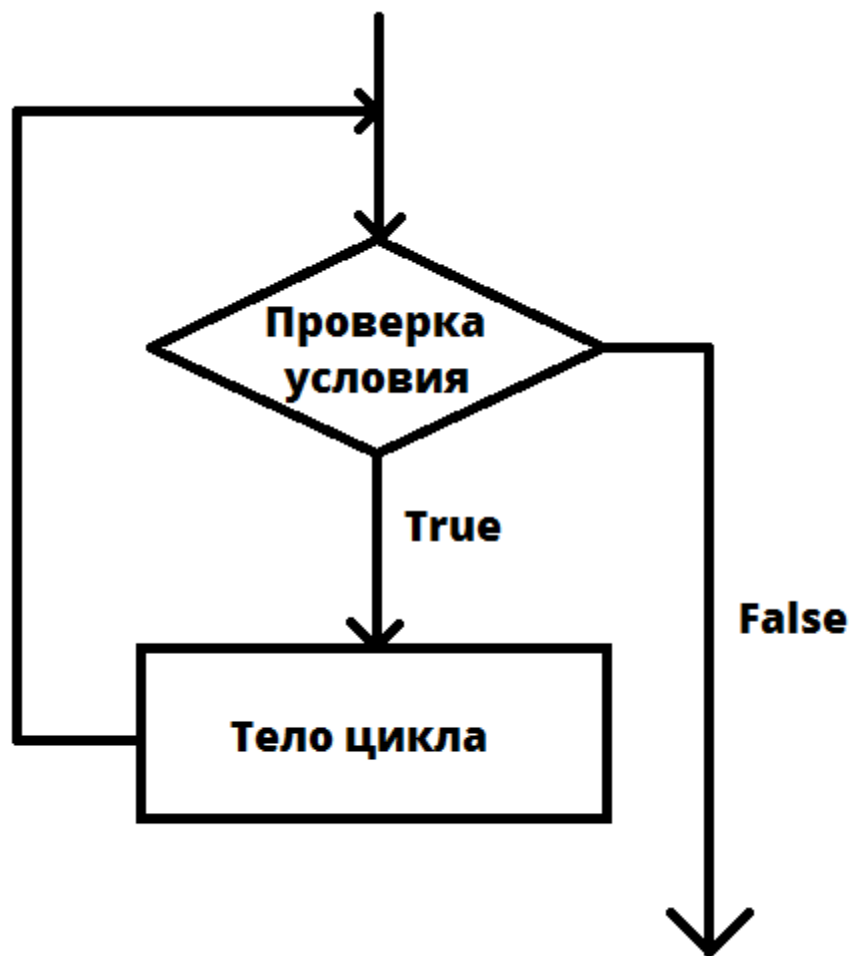
\* label – обычный идентификатор

# Учебная задача 5

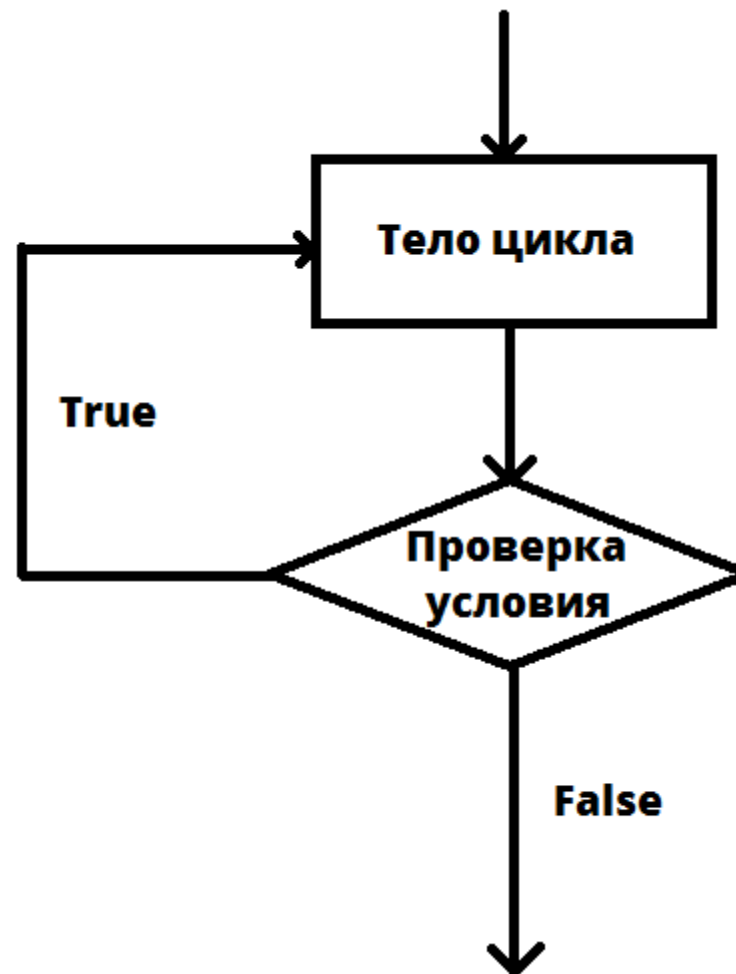
Пользователь вводит числа и после каждого числа нажимает Enter. После того, как пользователь введёт число 0, ввод считается завершённым и на экран нужно вывести получившуюся сумму.

<https://wandbox.org/permlink/95ir50qdVn3bPaQc>

# Оператор цикла



while



do-while

while

## Ключевое слово

## Выражение

## Тело

```
while (a > b) std::cin >> a;
```

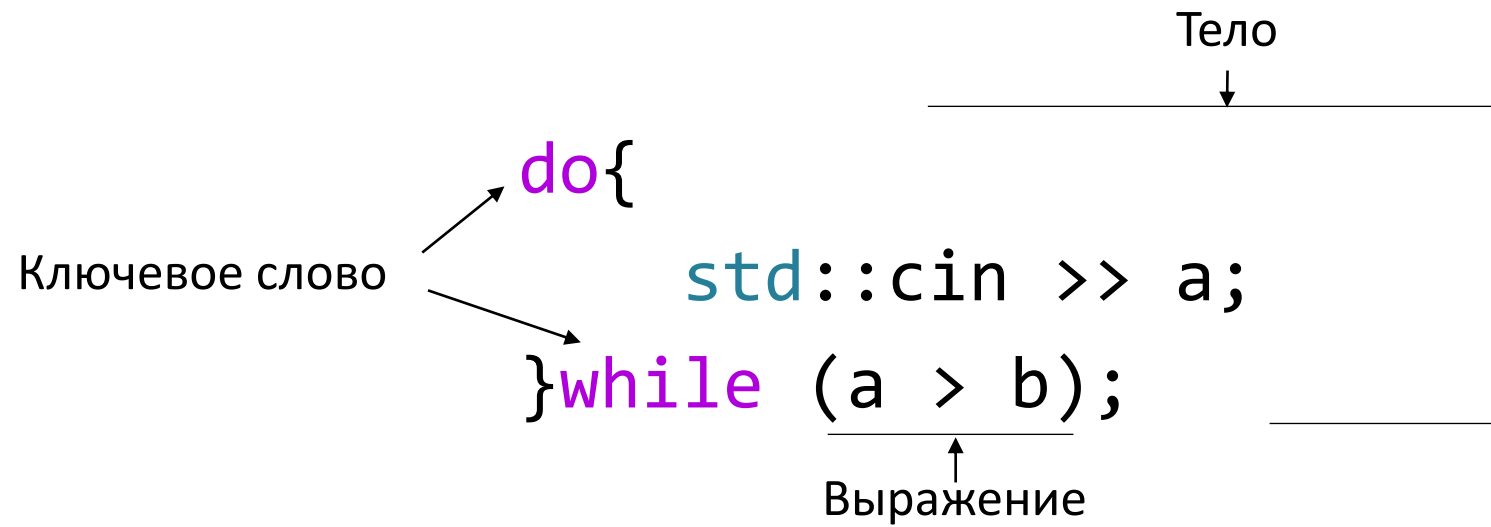
## Ключевое слово

## Выражение

# Тело

```
while (a > b){
    std::cin >> a;
}
```

# do-while



# while (выражение)

Ожидается, что выражение в скобках типа `bool`, поэтому будет попытка неявно [преобразовать его к `bool`](#).

Если преобразование не допустимо, то – ошибка.



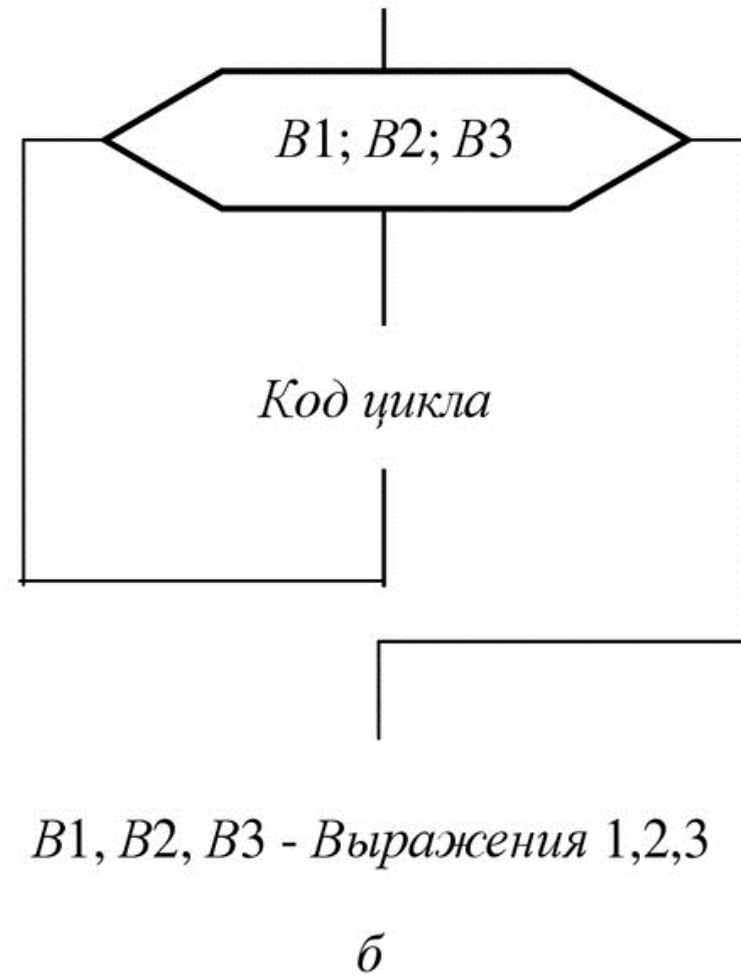
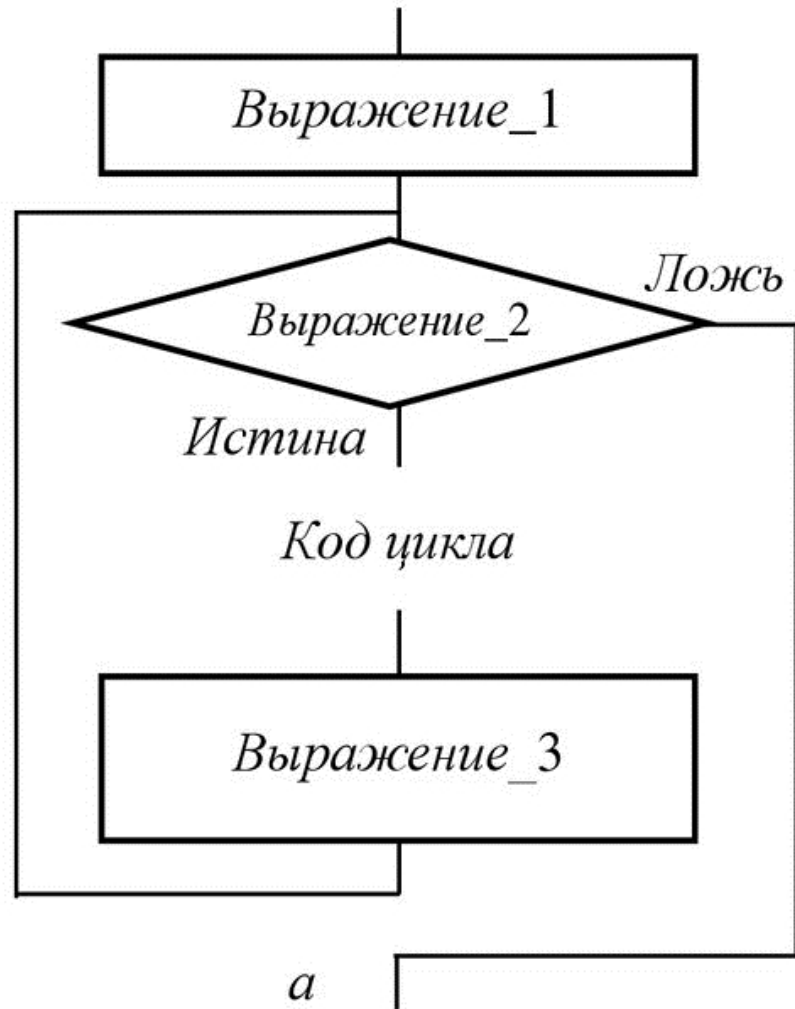
# Учебная задача 6

Пользователь вводит целое положительное число  $n$ . Затем в следующей строке, через пробел, вводятся  $n$  чисел. Посчитайте их среднее значение, минимум и максимум. Результат вывести на экран.

<https://wandbox.org/permlink/BUaCA5G6MCjDN71g>

<https://wandbox.org/permlink/3N9a0jlmHBcv8K3F>

# Оператор цикла for



# for

Ключевое слово

Выражение1

Выражение2

Выражение3

Тело



The diagram shows a C++ for loop: `for(int i=0; i < count; i++) std::cout << i;`. Above the code, four labels are positioned: 'Ключевое слово' (with an arrow pointing to 'for'), 'Выражение1' (with an arrow pointing to 'int i=0;'), 'Выражение2' (with an arrow pointing to 'i < count;'), 'Выражение3' (with an arrow pointing to 'i++'), and 'Тело' (with an arrow pointing to 'std::cout << i;'). Each label is underlined.

```
for(int i=0; i < count; i++) std::cout << i;
```

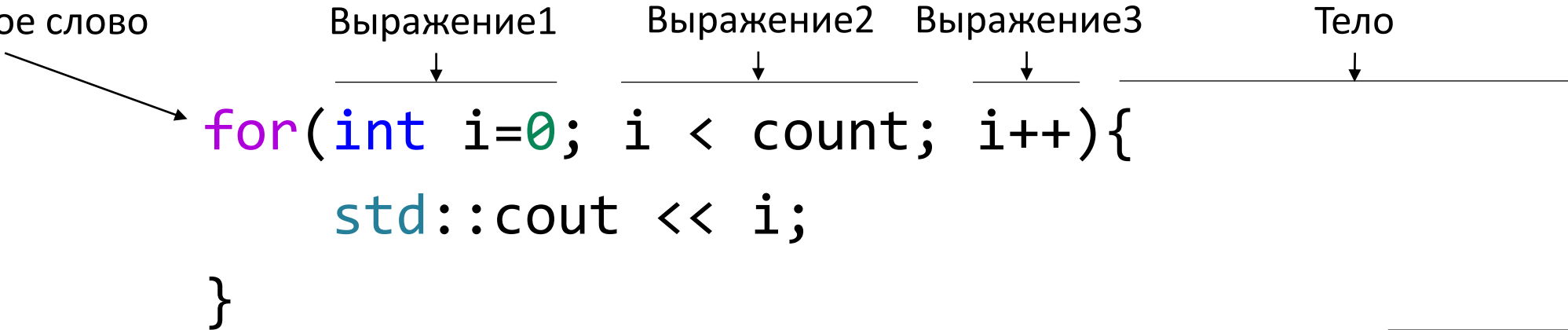
Ключевое слово

Выражение1

Выражение2

Выражение3

Тело



The diagram shows a C++ for loop with a block body: `for(int i=0; i < count; i++){ std::cout << i; }`. Above the code, four labels are positioned: 'Ключевое слово' (with an arrow pointing to 'for'), 'Выражение1' (with an arrow pointing to 'int i=0;'), 'Выражение2' (with an arrow pointing to 'i < count;'), and 'Выражение3' (with an arrow pointing to 'i++'). The label 'Тело' has an arrow pointing to the opening curly brace '{'. A line connects the closing curly brace '}' back to the 'Тело' label. Each label is underlined.

```
for(int i=0; i < count; i++){  
    std::cout << i;  
}
```

# for (выражение1; выражение2; выражение3)

Выражение1 – любое выражение или инициализация переменной. Обычно - инициализация переменной счётчика или нескольких;

Выражение2 – любое выражение или инициализация переменной. Обычно - выражение проверяющее условие работы цикла. Если выражение не указано, то считается, что оно равно true.

Выражение3 – выражение. Обычно инкремент/декремент счётчика(ов).

\* каждое из выражение не обязательное (можно не писать), но точки с запятой писать нужно.

# range-based for

Ключевое слово      Переменная      Контейнер      Тело

↓                  ↓                  ↓

```
for(auto i : array) std::cout << i;
```

Ключевое слово                      Переменный                      Выражение2                      Тело

↓                      ↓                      ↓                      ↓

for(auto [key, value] : mymap) std::cout << i;

# for (range-declaration : range-expression)

**range-expression** – любое выражение, представляющее последовательность элементов (либо массив, либо объект, для которого определены методы или функции `begin` и `end`) или список инициализации.

**range-declaration** – объявление именованной переменной, тип которой является типом элемента последовательности, представленного **range-expression**, или ссылкой на этот тип. Часто использует спецификатор `auto` для автоматического определения типа.

for(инициализация; range-declaration : range-expression)

```
for(auto list = {1,2,3}; auto i : list){  
    std::cout << i;  
}
```

# Учебная задача 7

Дана строка состоящая из круглых скобок, например: (), ((())), ()()(()), Проверьте баланс и правильность порядка скобок в строке. Если скобочная последовательность правильная, выведите ОК иначе NOT. Примеры не правильных скобочных последовательностей: ())(), )(, (.

[break и continue](#)

<https://wandbox.org/permlink/oFxagj2xXsl7dpH0>



```
string s = "I'm sorry, Dave.";
           0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 indices
```

non-mutating

s.size()	→ 16	(number of characters)
s[2]	→ 'm'	(character at index 2)
s.find("r")	→ 6	(first match from start)
s.rfind("r")	→ 7	(first match from end)
s.find("X")	→ string::npos	(not found, invalid index)
s.find(' ', 5)	→ 10	(first match after index ≥ 5)
s.substr(4, 6)	→ string{"sorry,"}	
s.contains("sorry")	→ true	(C++23)
s.starts_with('I')	→ true	(C++20)
s.ends_with("Dave.")	→ true	(C++20)
s.compare("I'm sorry, Dave.")	→ 0	(identical)
s.compare("I'm sorry, Anna.")	→ > 0	(same length, but 'D' > 'A')
s.compare("I'm sorry, Saul.")	→ < 0	(same length, but 'D' < 'S')

mutating

size  
index based

s += " I'm afraid I can't do that."	⇒ s = "I'm sorry, Dave. I'm afraid I can't do that."
s.append("...")	⇒ s = "I'm sorry, Dave..."
s.clear()	⇒ s = ""
s.resize(3)	⇒ s = "I'm"
s.resize(20, '?')	⇒ s = "I'm sorry, Dave.?????";
s.insert(4, "very ")	⇒ s = "I'm very sorry, Dave."
s.erase(5, 2)	⇒ s = "I'm srry, Dave."
s[15] = '!'	⇒ s = "I'm sorry, Dave!"
s.replace(11, 5, "Frank")	⇒ s = "I'm sorry, Frank"
s.insert(s.begin(), "HAL: ")	⇒ s = "HAL: I'm sorry, Dave."
s.insert(s.begin()+4, "very ")	⇒ s = "I'm very sorry, Dave."
s.erase(s.begin()+5)	⇒ s = "I'm srry, Dave."
s.erase(s.begin(), s.begin()+4)	⇒ s = "sorry, Dave."

iterator based

## Constructors

```
string('a', 'b', 'c') → a b c
string(4, '$') → $ $ $ $
string(@firstIn, @lastIn) → e f g h
                        source | iterator range
                        b c d e f g h i j
string( a b c d ) copy/move → a b c d
                        source string object
```

## Obtain Iterators or Reverse Iterators

```
.begin() → @first
           ↓
           a b c d e f

.rbegin() → reverse@last
           ↓
           a b c d e f

.end() → @one_behind_last
           ↓
           a b c d e f
           don't use to access elements!

.rend() → reverse@one_before_first
           ↓
           a b c d e f
           don't use to access elements!
```

## String → Number Conversion

```
int      stoi (●, ●, ●);
long     stol (●, ●, ●);
long long stoll(●, ●, ●);
const string&
input string
std::size_t* p = nullptr
output for
number of processed characters

unsigned long stoul (●, ●, ●);
unsigned long long stoull(●, ●, ●);
int base = 10
base of target system;
default: decimal

float      stof (●, ●, ●);
double    stod (●, ●, ●);
long double stold(●, ●, ●);
```

## Number → String Conversion

```
string to_string( ● );
           {
int | long | long long |
unsigned | unsigned long | unsigned long long |
float | double | long double
           }
```

# Массивы

Статические :

```
int arr[10];
```

Динамические:

```
int* arr = new int[10];  
delete[] arr;
```

STL:

```
std::array<int, 10> arr;  
std::vector<int> arr(10);
```

# C++ Standard Library Sequence Containers

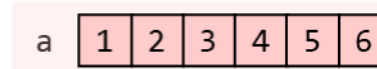
h/cpp/ hackingcpp.com

## `array<T, size>`

fixed-size array

```
#include <array>
```

```
std::array<int,6> a {1,2,3,4,5,6};  
cout << a.size();    // 6  
cout << a[2];        // 3  
a[0] = 7;             // 1st element ⇒ 7
```



contiguous memory; random access; fast linear traversal

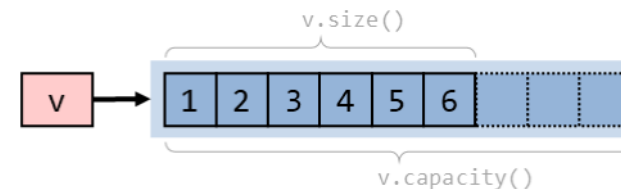
## `vector<T>`

dynamic array

C++'s "default" container

```
#include <vector>
```

```
std::vector<int> v {1,2,3,4,5,6};  
v.reserve(9);  
cout << v.capacity();    // 9  
cout << v.size();        // 6  
v.push_back(7);          // appends '7'  
v.insert(v.begin(), 0);  // prepends '0'  
v.pop_back();            // removes last  
v.erase(v.begin()+2);    // removes 3rd  
v.resize(20, 0);         // size ⇒ 20
```



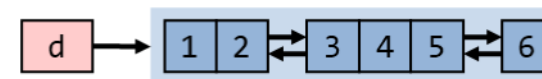
contiguous memory; random access;  
fast linear traversal; fast insertion/deletion at the ends

## `deque<T>`

double-ended queue

```
#include <deque>
```

```
std::deque<int> d {1,2,3,4,5,6};  
// same operations as vector  
// plus fast growth/deletion at front  
d.push_front(-1); // prepends '-1'  
d.pop_front();    // removes 1st
```



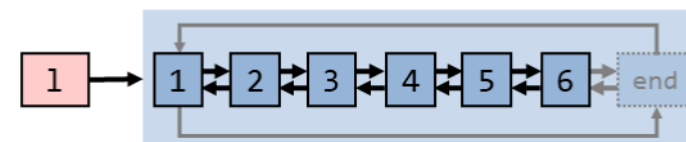
fast insertion/deletion at both ends

## `list<T>`

doubly-linked list

```
#include <list>
```

```
std::list<int> l {1,5,6};  
std::list<int> k {2,3,4};  
// O(1) splice of k into l:  
l.splice(l.begin()+1, std::move(k))  
// some special member function algorithms:  
l.reverse();  
l.sort();
```



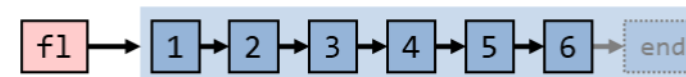
fast splicing; many operations without copy/move of elements

## `forward_list<T>`

singly-linked list

```
#include <forward_list>
```

```
std::forward_list<int> fl {2,2,4,5,6};  
fl.erase_after(begin(fl));  
fl.insert_after(begin(fl), 3);  
fl.insert_after(before_begin(fl), 1);
```



lower memory overhead than std::list; only forward traversal

# std::vector<ValueType>

C++'s "default"  
dynamic array

#include <vector>

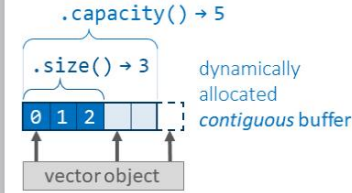
h/cpp hackingcpp.com

## Construct A New Vector Object

```
vector<int> v1 {2,9,1,8,5,4} → [2, 9, 1, 8, 5, 4]
vector<int> v2 (begin(v1)+3, end(v1)) → [8, 5, 4]
vector<int> v3 (5, 3) → [3, 3, 3, 3, 3]
vector<int> deep_copy_of_v1 (v1) → [2, 9, 1, 8, 5, 4]
```

C++17 value type deducible from argument type  
vector<int> w {7,4,2}; // vector<int>

## Typical Memory Layout



## Assign New Content To An Existing Vector

```
vector<int> v1 {8,5,3}; (deep copy from source)
vector<int> v2 {6,8,1,9};
v1 = v2;
new state of v1: [6, 8, 1, 9]
v1.assign({4,1,3,5}) → [4, 1, 3, 5]
v1.assign(2, 1) → [1, 1]
v1.assign(@InBeg, @InEnd) → [2, 1, 1, 2]
source container: [3, 2, 1, 1, 2, 3]
```

## Query/Change Size (= Number of Elements)

```
[8, 5, 3].empty() → false
[8, 5, 3].size() → 3
[8, 5, 3].resize(2) → [8, 5]
[8, 5, 3].resize(4, 1) → [8, 5, 3, 1]
[8, 5, 3].resize(6, 1) → [8, 5, 3, 1, 1, 1]
[8, 5, 3].clear() → [ ]
```

## Query/Grow Capacity (= Memory Buffer Size)

```
[8, 5, 3].capacity() → 4
[8, 5, 3].reserve(6) → [8, 5, 3, , , ]
```

## Get Element Values $O(1)$ Random Access

```
[2, 8, 5, 3][1] → 8
[2, 8, 5, 3].front() → 2
[2, 8, 5, 3].back() → 3
```

## Change Element Values

```
[2, 8, 5, 3][1] = 7 → [2, 7, 5, 3]
[2, 8, 5, 3].front() = 7 → [7, 8, 5, 3]
[2, 8, 5, 3].back() = 7 → [2, 8, 5, 7]
```

## Out of Bounds Access

```
[2, 8, 5, 3][6] → Undefined Behavior (Invalid Index!)
[2, 8, 5, 3].at(6) → Throws Exception (std::out_of_range)
```

## Erase Elements $O(n)$ Worst Case

```
vector<int> v {4,8,5,6};
[4, 8, 5, 6].pop_back() → [4, 8, 5]
[4, 8, 5, 6].erase(begin(v)+2) → [4, 8, 6]
[4, 8, 5, 6].erase(begin(v)+1, begin(v)+3) → [4, 6]
```

## Shrink The Capacity (might be inefficient)

Erasing, resizing or clearing will not shrink the capacity!

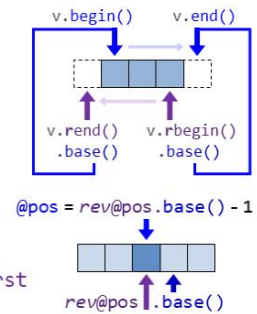
```
vector<int> v (1024, 0); // capacity is at least 1024
v.resize(40); // capacity unchanged!
v.shrink_to_fit(); // may shrink (not guaranteed)
v.swap(vector<int>(v)); // shrinks but has copy overhead
```

## Obtain Iterators

$O(1)$  Random Incrementing  
[0, 1, 2, 3].begin() → @first  
[0, 1, 2, 3].end() → @one\_behind\_last

## Obtain Reverse Iterators

```
[0, 1, 2, 3].rbegin() → rev@last
[0, 1, 2, 3].rend() → rev@one_before_first
```



```
[2, 8, 5, 3].data() → pointer_to_first
```

## Append Elements $O(1)$ Amortized Complexity

```
[8, 5, 3].push_back(7) → [8, 5, 3, 7]
```

Avoid expensive memory allocations:  
• **reserve** capacity before appending / inserting if you know the (approximate) number of elements to be stored in advance!

## Insert Elements at Arbitrary Positions $O(n)$ Worst Case

```
vector<int> v {8,5,3};
[8, 5, 3].insert(begin(v), 2) → [2, 8, 5, 3]
[8, 5, 3].insert(begin(v)+1, 7) → [8, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, 3, 7) → [8, 7, 7, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, {6,9,7}) → [8, 6, 9, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, @InBegin, @InEnd) → [8, 1, 8, 9, 5, 3]
source container: [3, 1, 8, 9, 2, 3]
```

## Insert & Construct Elements in Place $O(n)$ Worst Case

```
vector<pair<string,int>> v {"a",1}, {"w",7};
```

```
[a,1][w,7].emplace_back("b",4) → [a,1][w,7][b,4]
[a,1][w,7].emplace(begin(v)+1, "z",5) → [a,1][z,5][w,7]
```

# Учебная задача 8

Дана строка, состоящая только из строчных английских символов и цифр. Выведите ОК, если строка палиндром и NOT если не палиндром.

<https://wandbox.org/permlink/VYW2yq2t3ZNj7O2P>

<https://wandbox.org/permlink/Pp6g2vqi6zkAh8NB>

<https://wandbox.org/permlink/9yGKKuDanibsJsPb>

# Итераторы (начало)

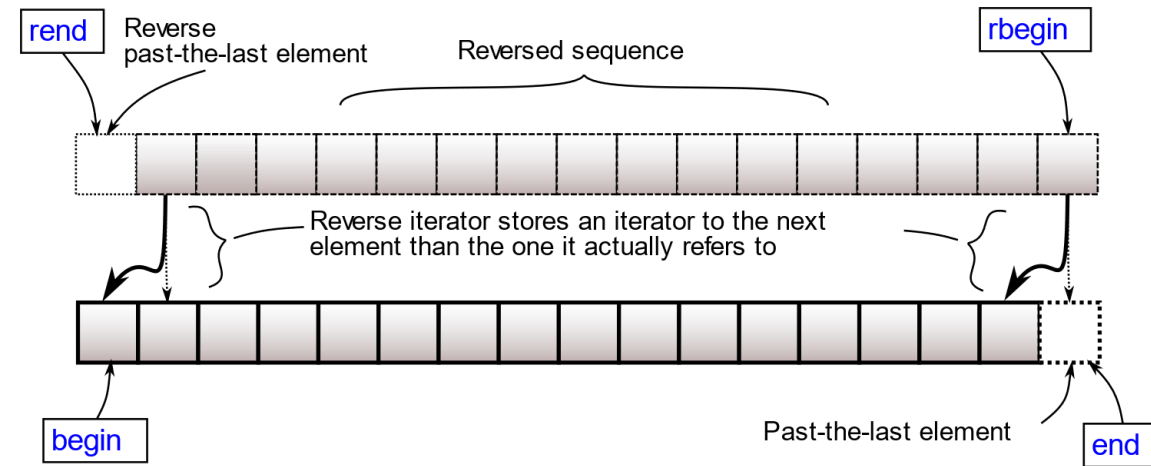
Итератор — это структура данных, предназначенная, для того чтобы перебирать элементы контейнера (последовательности), при этом не задумываясь, с каким именно контейнером происходит работа.

Получить итераторы у контейнера можно через:






- методы `begin()` и `end()`: `array.begin()`;
- одноимённые функции: `begin(array)`;

Чтобы ходить по контейнеру в обратном направлении используются реверс-итераторы: `rbegin()` и `rend()`.

# Итераторы (начало)



# Итераторы (начало)

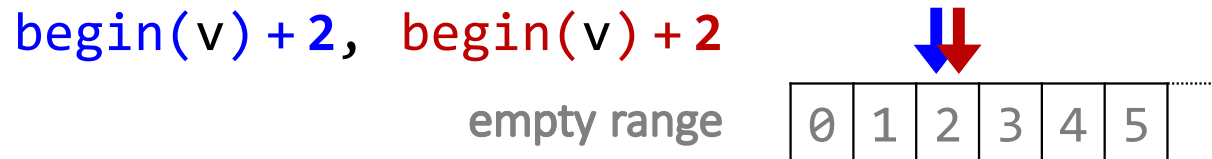
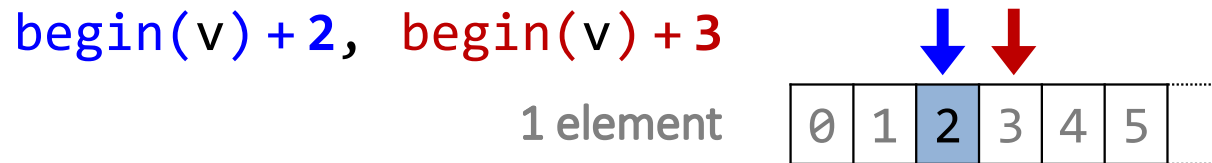
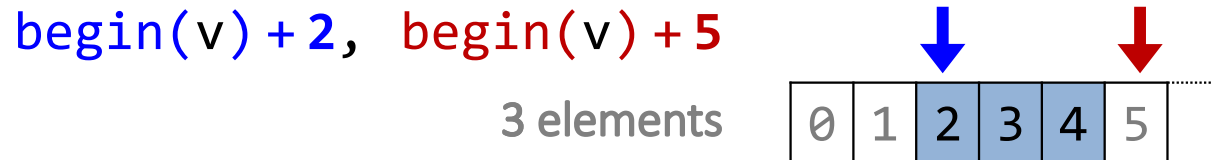
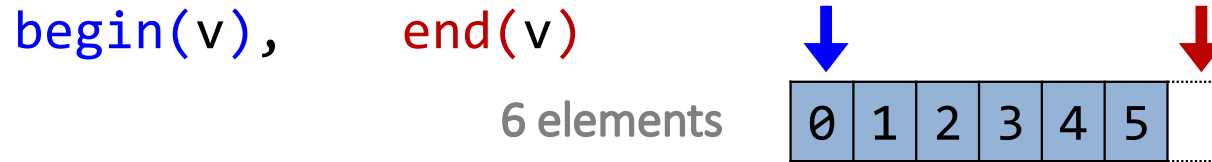
```
vector<int> v { 1, 2, 3, 4, 5, 6, 7, 8, 9,  };  
auto i = begin(v);   
int x = *i; // x: 1  
++i; // advance by 1   
auto j = begin(v) + 3;   
int y = *j; // y: 4  
auto e = end(v);   
*j = 47; // change element value: 4 → 47
```

**DO NOT ACCESS 'END' WITH '!!'**  
(does not refer to valid memory)  
**ONLY USE AS POSITION SPECIFIER!**



# Итераторы (начало)

```
std::vector<int> v {0, 1, 2, 3, 4, 5};
```



# Учебная задача 9

Дано целых  $n$  чисел. Определите и выведите на экран минимальное из них.

<https://wandbox.org/permlink/PtTpLNcMw1OdCe2O>

# Учебная задача 10

Дано  $n$  целых чисел. Выведите числа в порядке от меньшего к большему.

<https://wandbox.org/permlink/zF4ch6PDayqXEqt1>

# C++ Standard Library Algorithms

## Non-Modifying Sequence Operations

`min_element(@begin, @end) → @minimum`

7 9 3 5 3 2 6 1 8 0

`minmax_element(@begin, @end) → { @minimum, @maximum }`

7 1 3 5 3 8 6 2 9 0

`any_of(@begin, @end, f(o)→bool) →` true, if `f` yields true for any, all or none elements in the input range  
`all_of(@begin, @end, f(o)→bool) →`  
`none_of(@begin, @end, f(o)→bool) →` false otherwise

C++11 5 2 9 1 3 8 5 2 9 0

`find_if(@begin, @end, f(o)→bool) →` 1st match  
`find(@begin, @end, value) →` @end if no match

5 2 9 1 3 8 5 2 9 0

`count_if(@begin, @end, f(o)→bool) →` number of occurrences  
`count(@begin, @end, value) →`

5 2 9 1 3 2 5 2 0

`equal(@begin1, @end1, @begin2) → true` if all elements in both ranges are equal

0 1 2 3 4 5 6 7 8 9

`mismatch(@begin1, @end1, @begin2) → { @mismatch_in1, @mismatch_in2 }`

0 1 2 3 4 5 6 7 8 9

`search(@beg1, @end1, @beg2, @end2) →` 1st occurrence of sequence 2 inside sequence 1  
@end1 otherwise

0 1 2 3 4 5 6 7 8 9

## Binary Search On Sorted Sequences ⇒ $O(\log n)$

`lower_bound(@begin, @end, value) →` 1st element not < value  
@end if no such element exists

0 1 2 3 4 5 6 7 8 9

`upper_bound(@begin, @end, value) →` 1st element > value  
@end if no such element exists

0 1 2 3 4 5 6 7 8 9

`equal_range(@begin, @end, value) → { @1st item not < value or @end if none such found, @1st item > value or @end if none such found }`

1 1 2 3 4 5 5 5 6 6 7 7 8

## Reordering Elements

`reverse(@begin, @end)`

0 1 2 3 4 5 6 7 8 → 8 7 6 5 4 3 2 1 0

`sort(@begin, @end, f(o, o)→bool)`

`sort(@begin, @end, std::less)`

8 9 3 1 2 5 4 7 6 → 8 9 1 2 3 4 5 7 6

`stable_sort(@begin, @end, compare(o, o)→bool)`  
compare(o, o)→bool  
compare\_case\_insensitive  
"stable" preserves the relative order of equivalent elements

8 9 3 1 2 5 4 7 6

`nth_element(@begin, @nth, @end)`

4 2 5 6 3 7 1 8 → 4 2 3 1 5 7 6 8

element at `nth` position → element that would be in that position in a sorted sequence

`partition(@begin, @end, f(o)→bool) → @part2`

2 3 4 6 5 7 8 9 → 2 3 5 7 4 6 8 9

`rotate(@begin, @newfst, @end) → @old_begin`

0 1 2 3 4 5 6 7 8 → 0 3 4 5 6 7 1 2 8

`next_permutation(@begin, @end) → true` if new permutation is lexicographically greater

1 2 3 → 1 3 2

`shuffle(@begin, @end, random_engine)`

0 1 2 3 4 5 6 7 8 → 0 1 5 6 3 4 2 7 8

## Manipulate Sorted Sequences ⇒ $O(n)$

`merge(@1beg, @1end, @2beg, @2end, @out)`

sorted input! 0 2 4 6 7 9 0 1 3 5 8 9 sorted output 0 1 2 3 4 5 6 7 8 9

`set_union(@1beg, @1end, @2beg, @2end, @out)`

0 1 2 2 4 4 5 1 1 3 4 5 0 1 1 2 2 3 4 4 5

Iterator Ranges  
@begin+3 @begin+6  
0 1 2 3 4 5 6 7 8  
↑ @begin @end  
distance(@begin, @end) → element\_count  
1 2 3 4 5 6 7 8 9  
<iterator>

## Changing Values

`copy(@begin, @end, @out)`

1 2 3 4 5 6 7 8 9 0 0 4 5 6 7 8 0

`transform(@begin, @end, @out, f(o)→■)`

u v w x y z f(w) f(x) f(y)

`generate(@begin, @end, f())`

0 0 0 0 0 0 ascending\_step\_2 → 0 2 4 6 8 0 0

`replace(@begin, @end, old, new)`

1 2 3 2 4 2 2 0 2, 2, 0 → 1 2 3 0 4 0 0 0

`replace_if(@begin, @end, f(o)→bool, new)`

3 2 5 4 6, is\_even, 0 → 3 0 5 0 6

`remove(@begin, @end, value) → @end_of_remove`  
`remove_if(@begin, @end, f(o)→bool) → remaining`

0 1 4 3 5 8 2 7 8, is\_even → 0 1 3 5 7 8

`unique(@begin, @end) → @end_of_remaining`

0 4 1 3 3 3 0 3 1 3 → 0 4 1 3 3 1 3

`erase(container, value) → erased_count`

1 2 3 5 2 2 7, 2 → 3  
1 3 5 7

## Numeric Algorithms

`reduce(@begin, @end, w = 0, ⊕ = +)`  
`reduce(@begin, @end, w, ⊕(□, □)→■)`  
arbitrary evaluation order!

`transform_reduce(@begin, @end, @begin2, @end2, w, ⊕ = □ + □, ⊗ = □ × □)`  
`transform_reduce(@begin, @end, @begin2, @end2, w, ⊕(□, □)→■, ⊗(□, □)→■)`  
`transform_reduce(@begin, @end, @begin2, @end2, w, ⊕(□, □)→■, f(o)→■)`  
arbitrary evaluation order!

C++17  $R_f = w + f(o_0) + f(o_1) + \dots$   $R_\otimes = w + (o_0 \otimes o_1) + (o_1 \otimes o_2) + \dots$

`inclusive_scan(@begin, @end, @out, ⊕ = +)`  
`inclusive_scan(@begin, @end, @out, ⊕(□, □)→■, w)`  
→ @end

C++17 2 1 7 5 3 2 3 10 15 18  
a b c d e w=a w=a+b w=a+b+c w=a+b+c+d w=a+b+c+d+e

Sequence Queries	
all_of	(C++11)
any_of	(C++11)
none_of	(C++11)
count	count_if
find	find_if
find_if_not	(C++11)
find_end	
find_first_of	
adjacent_find	
for_each	(C++17)
for_each_n	(C++20)
sample	
equal	
mismatch	
search	
search_n	
lexicographical_compare	
lexicographical_compare_three_way	(C++20)
Reordering Elements	
reverse	reverse_copy
rotate	rotate_copy
shift_left	(C++20)
shift_right	(C++20)
shuffle	(C++11)
swap	
swap_ranges	
iter_swap	
Partitioning	
is_partitioned	
partition	
stable_partition	
partition_copy	
partition_point	
Permutations	
is_permutation	(C++11)
next_permutation	prev_permutation
Sorting	
sort	
stable_sort	
partial_sort	
partial_sort_copy	
is_sorted	
is_sorted_until	
nth_element	
Changing Elements	
copy	
copy_backward	
copy_if	(C++11)
copy_n	(C++11)
move	(C++11)
move_backward	(C++11)
fill	
fill_n	
generate	
generate_n	
transform	
replace	replace_if
replace_copy	replace_copy_if
remove	remove_if
remove_copy	remove_copy_if
unique	
unique_copy	
Binary Search on Sorted Ranges	
binary_search	
lower_bound	upper_bound
equal_range	
includes	
Merging of Sorted Ranges	
merge	
inplace_merge	
set_union	
set_intersection	
set_difference	
set_symmetric_difference	
Heaps	
make_heap	
sort_heap	
push_heap	
pop_heap	
is_heap	(C++11)
is_heap_until	(C++11)
Minimum/Maximum	
min	max
min_element	max_element
minmax	(C++11)
minmax_element	(C++11)
clamp	(C++17)
Numeric	
accumulate	#include <numeric>
adjacent_difference	
inner_product	
partial_sum	
iota	(C++11)
reduce	(C++17)
inclusive_scan	(C++17)
exclusive_scan	(C++17)
transform_reduce	(C++17)
transform_inclusive_scan	(C++17)
transform_exclusive_scan	(C++17)