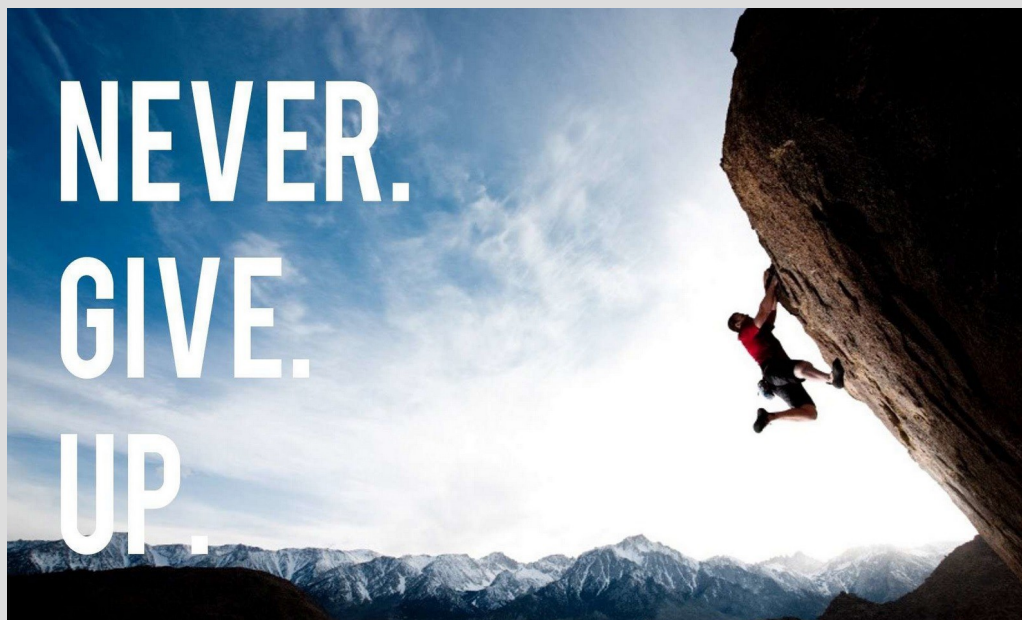


# Основы C++. Вебинар №4.

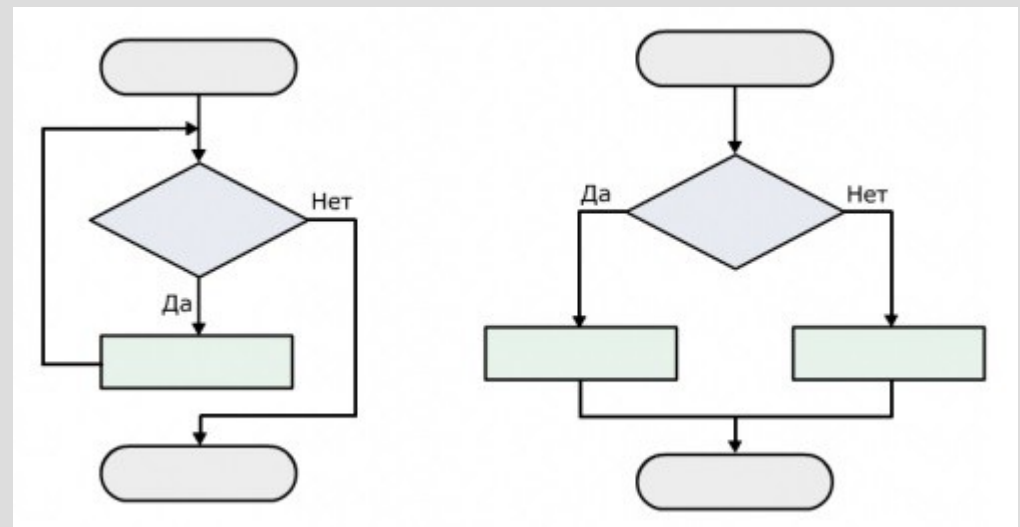
Длительность: 2 ч.



GeekBrains

## Что будет на уроке?

- Научимся писать условия (if, if else) и поймём принципы ветвления программы.
- Изучим особенности оператора множественного выбора (switch).
- Узнаем о циклах (for, while, do while) и областях видимости.
- Познакомимся с оператором безусловного перехода (goto);
- Узнаем о принципах программирования DRY, KISS, YAGNI.



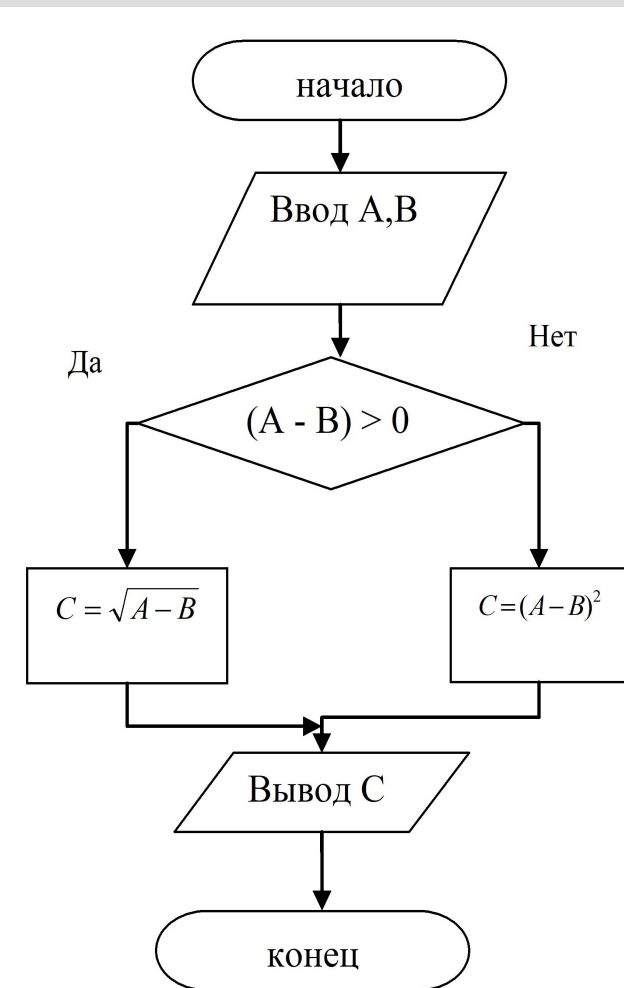
# Алгоритм

## ПОНЯТИЕ «АЛГОРИТМ»

Алгоритм — точное и понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи.

Название "алгоритм" произошло от латинской формы имени среднеазиатского математика **аль-Хорезми** — **Algorithmi**. Алгоритм — одно из основных понятий информатики и математики.

Пример блок-схемы алгоритма:



# Логическое условие в C++

Можно сказать что оно бывает 3х видов:

## if без else:

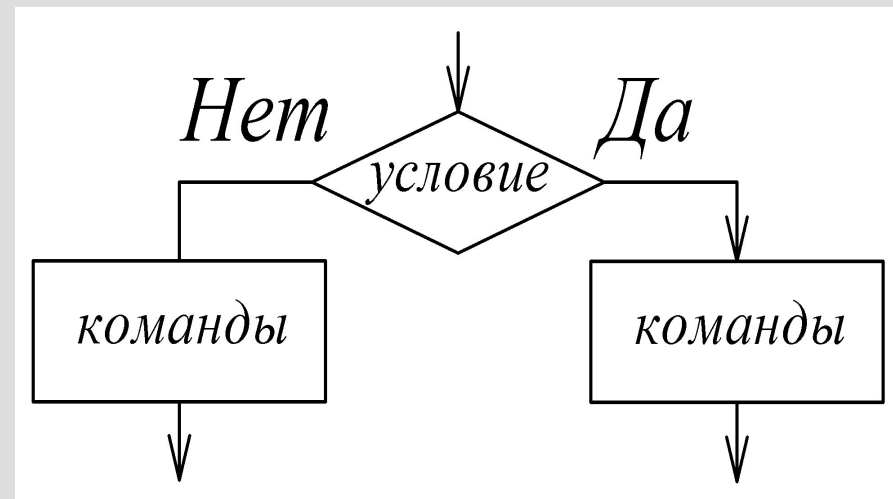
```
if (логическое условие)  
{ код; } // код если условие = true
```

## if с else (если - иначе):

```
if (логическое условие)  
{ код; } // код если условие = true  
else  
{ код; } // код если условие = false
```

## Каскадные if с else:

```
if (логическое условие)  
{ код; }  
else if (логическое условие)  
{ код; }  
else if (логическое условие)  
{ код; }
```



## Пример if - else

Пример:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    cout << "Enter random integer number: ";
```

```
    int a;
```

```
    cin >> a; // Читаем число с клавиатуры
```

```
    if (a > 0) // Если a > 0
```

```
    {
```

```
        cout << "Your number is greater zero";
```

```
    }
```

```
    else // Иначе
```

```
    {
```

```
        cout << "Your number is lower or equal zero";
```

```
    }
```

```
    return 0;
```

```
}
```

Также фигурные скобки могут отсутствовать если у вас только 1-а команда.

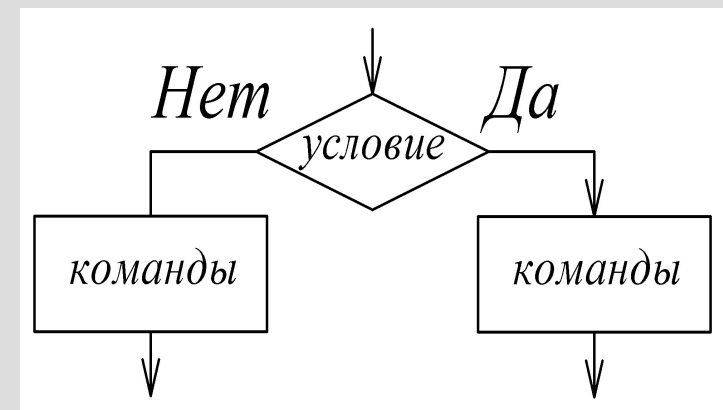
```
if (a > 0) // Если a > 0
```

```
    cout << "Your number is greater zero";
```

```
else // Иначе
```

```
    cout << "Your number is lower or equal zero";
```

Это верно и для циклов.



# Каскадные if - else

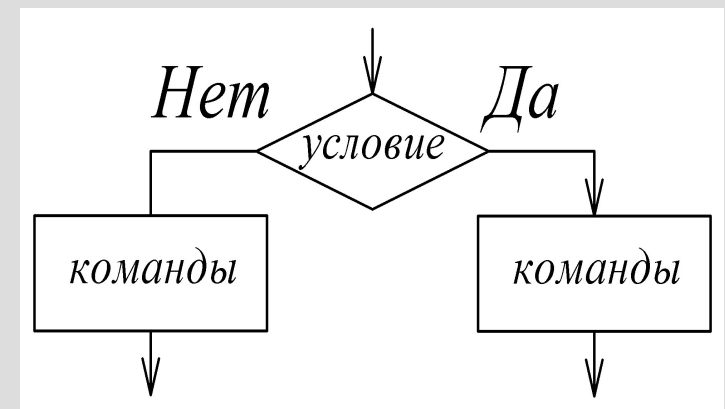
```
#include <iostream>

using namespace std;

int main()
{
    cout << "Enter random integer number: ";
    int a;
    cin >> a; // Читаем число с клавиатуры

    if (a > 0) // Если a > 0
    {
        cout << "Your number is greater zero";
    }
    else if (a < 0) // Иначе
    {
        cout << "Your number is lower than zero";
    }
    else if (a == 0)
    {
        cout << "Your number equal zero";
    }

    return 0;
}
```



# Оператор множественного выбора: switch - case

```
#include <iostream>
#include <locale> // для вызова setlocale

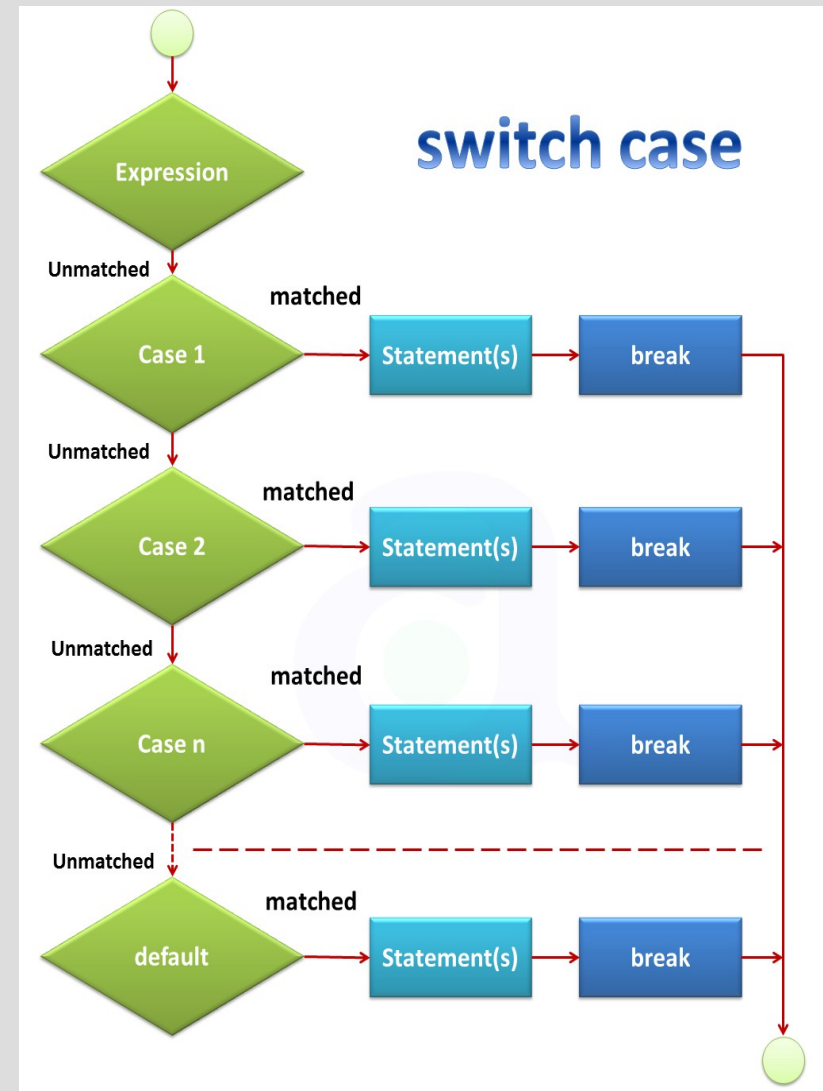
using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian"); // подключаем русский язык

    char op;
    cout << "Введите символ арифметической бинарной операции: ";
    cin >> op;

    switch (op) {
        case '+':
            cout << "Это символ операции сложения.";
            break;
        case '-':
            cout << "Это символ операции вычитания.";
            break;
        case '*':
            cout << "Это символ операции умножения.";
            break;
        case '/':
            cout << "Это символ операции деления.";
            break;
        default:
            cout << "Это что-то исключая * / + -";
    }
    return 0;
}
```

Обратите внимание на **break** и секцию **default**, которая может отсутствовать.



## 3 вида циклов в C++

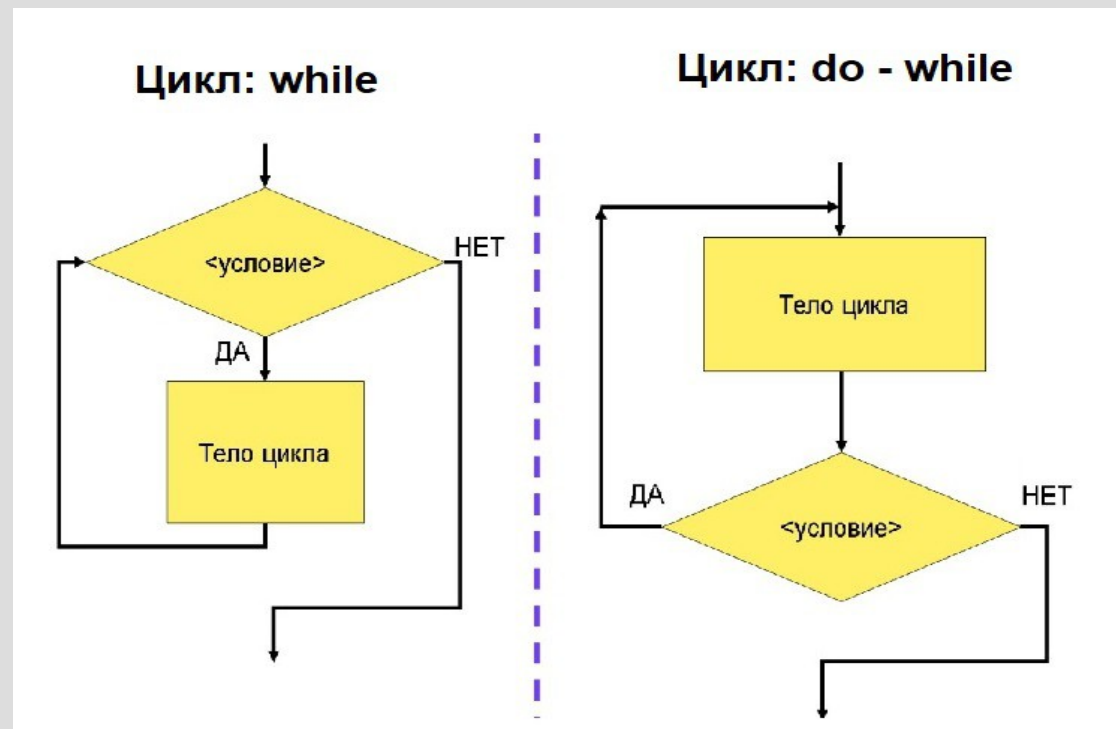
Циклы используются для повторения выполнения некоторого участка кода.  
В C++ 3-и вида циклов:

1. Цикл управляемый счетчиком:  
`for (инициализация; условие; изменение)`  
{  
    // some your code  
}

2. Циклы управляемые условием:

```
// Цикл while  
while (условие)  
{  
    // some your code  
}
```

```
// Цикл do - while  
do {  
    // some your code  
} while (условие);
```



Чтобы любой из 3-х циклов выполнялся **логическое условие должно быть true**. Но do while выполнится в любом случае хотя бы 1-н раз. Внутри циклов мы делаем отступ 3-4 пробела или Tab.



# Цикл управляемый счетчиком: for

Задача вывести на экран все числа от -10 до 10:

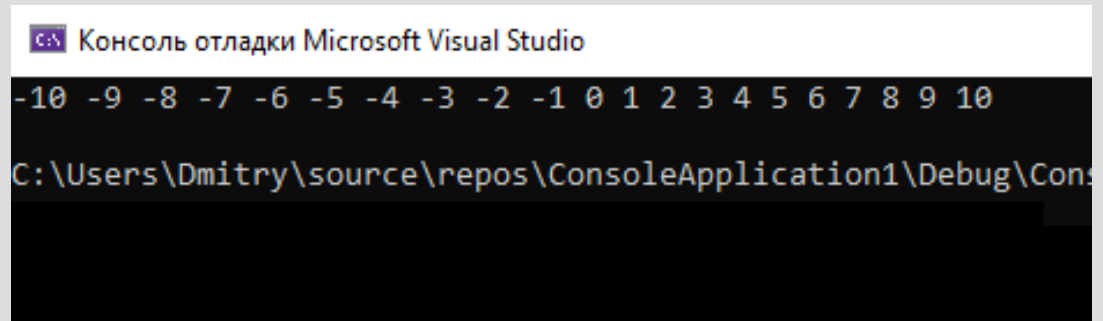
Цикл for:

```
/*  
for (инициализация; условие; изменение)  
{  
    // some your code  
}  
*/
```

```
for (int i = -10; i <= 10; i = i + 1)  
{  
    std::cout << i << " ";  
}
```

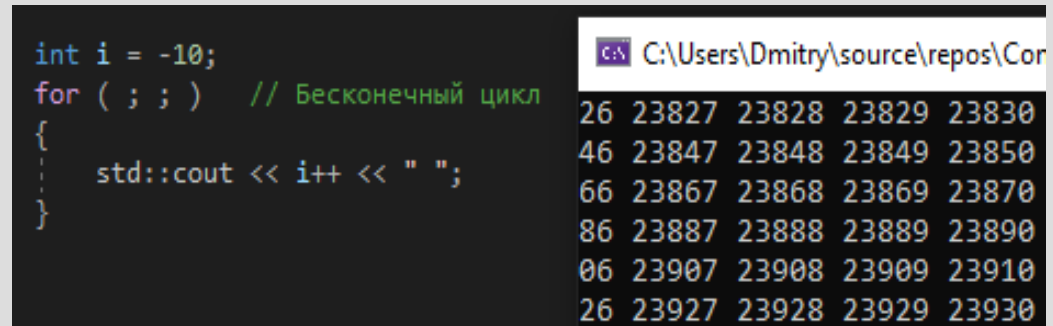
Любая из 3х секций: инициализация, условие, изменение **может отсутствовать**:

```
int i = -10;  
for (; i <= 10;)  
{  
    std::cout << i << " ";  
    i = i + 1;  
}
```



Консоль отладки Microsoft Visual Studio

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10  
C:\Users\Dmitry\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe
```



```
int i = -10;  
for ( ; ; ) // Бесконечный цикл  
{  
    std::cout << i++ << " ";  
}
```

C:\Users\Dmitry\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe

```
26 23827 23828 23829 23830  
46 23847 23848 23849 23850  
66 23867 23868 23869 23870  
86 23887 23888 23889 23890  
06 23907 23908 23909 23910  
26 23927 23928 23929 23930
```

# Циклы while и do while

Задача вывести на экран все числа от -10 до 10:

```
int i = -10;
while (i <= 10) // вывод списка чисел с помощью while
{
    std::cout << i << " ";
    i++;
}
```

```
std::cout << std::endl << i << std::endl;
```

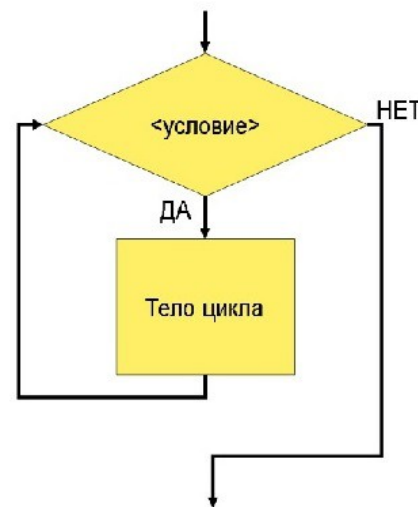
```
i = -10; // сбрасываем переменную в -10
```

```
do { // вывод списка чисел с помощью do while
    std::cout << i << " ";
    i++;
} while (i <= 10);
```

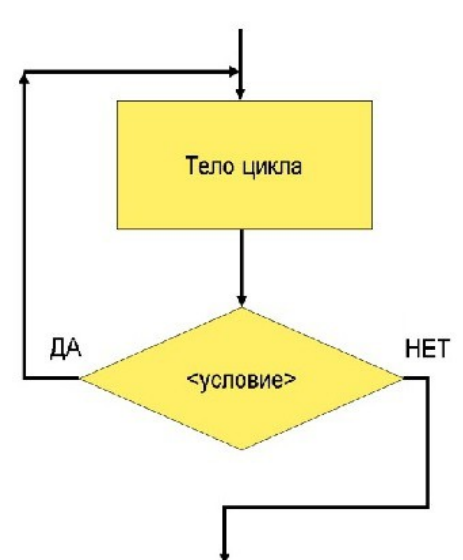
Консоль отладки Microsoft Visual Studio

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
11
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
```

Цикл: while



Цикл: do - while



## STL тип данных size\_t

В библиотеку STL был добавлен специальный тип данных для переменных счетчиков в циклах и индексов size\_t. Переменная этого типа может быть **0 или положительным целым числом**. А размер максимально большой на текущей платформе (4 байта для x86 (32) разрядной сборки и 8 байт для x64).

Пример использования:  
(задача просуммировать все элементы массива)

```
const size_t SIZE = 10;
int arr [SIZE] = { 10, 20, 30, 40, 50, 40, 30, 20, 10, -10 };
int sum = 0; // Здесь будем накапливать сумму

for (size_t index = 0; index < SIZE; index++)
{
    sum += arr [index];
    cout << arr [index] << " ";
}

cout << " sum = " << sum;
```



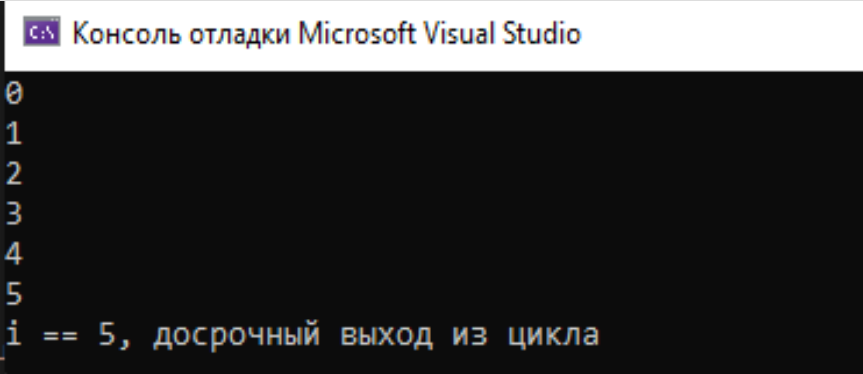
Консоль отладки Microsoft Visual Studio

```
10 20 30 40 50 40 30 20 10 -10 sum = 240
```

# Операторы break и continue

Эти операторы могут применяться внутри любого из циклов в C++. А также оператор break можно и нужно использовать в switch — case как мы уже видели.

```
for (size_t i = 0; i < 100; i++)  
{  
    cout << i << endl;  
  
    if (i == 5) // если i равна 5  
    {  
        cout << "i == 5, досрочный выход из цикла";  
        break;  
    }  
    else // иначе  
    {  
        continue; // переход на проверку условия в цикле  
        cout << "Этот текст не будет никогда напечатан";  
    }  
}
```



```
Консоль отладки Microsoft Visual Studio  
0  
1  
2  
3  
4  
5  
i == 5, досрочный выход из цикла
```

Обратите внимание на каскадные отступы в программе внутри цикла for и внутри условия if. Делайте их в своих программах подобным образом.

## Range-based for loop (since C++11)

Вывод массива на экран через for основанный на диапазоне:

```
int arr [ ] = { 0, 1, 2, 3, 4, 5, 6, 7 };
```

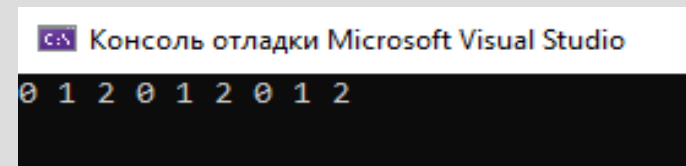
```
for (int n : arr) // Переменная n должна быть того же типа что и наш массив arr
{
    std::cout << n << ' ';
}
std::cout << "\n"; // перевод курсора на новую строку через символ \n
```

В переменную n копируется каждый элемент массива, шаг за шагом. Но нам не нужно управлять этим циклом как в обычном for. Также для переменной n можно использовать тип auto, что может упростить и ускорить программирование если тип данных массива очень длинный. Например:

```
enum VeryLongDataType { YES, NO, MAYBE };
VeryLongDataType array [ ] = { YES, NO, MAYBE, YES, NO, MAYBE, YES, NO, MAYBE };

for (auto n : array)
{
    std::cout << n << ' '; // при выводе элемент enum выводится как int
}
std::cout << "\n";
```

Иногда этот цикл называют for each но я бы не стал так делать так как в STL есть алгоритм с названием std::for\_each который появился гораздо раньше: [https://en.cppreference.com/w/cpp/algorithm/for\\_each](https://en.cppreference.com/w/cpp/algorithm/for_each)



## Циклы могут быть вложенными

Циклы могут иметь любую вложенность друг в друга которую позволяет ваш размер стека:

```
const int size = 3;
int array[size][size] = { { 10, 20, 30 }, { 30, 20, 10 }, { -10, 0, -30 } };

// Задача проверить есть ли 0 элемент в двумерном массиве

for (size_t i = 0; i < size; i++)
{
    for (size_t j = 0; j < size; j++)
    {
        if (array[i][j] == 0)
        {
            cout << "Zero found! array[" << i << "][" << j << "] = 0" << endl;
        }
    }
}
```

Консоль отладки Microsoft Visual Studio

Zero found! array[2][1] = 0

C:\Users\Dmitry\source\repos\...  
ом 0.

Чтобы автоматически закрывать  
томатически закрыть консоль п  
Нажмите любую клавишу, чтобы

Можно добавить 2-а break чтоб повысить эффективность алгоритма.

## 3 вложенных цикла for

Три вложенных массива for, со счетчиками i, j, k:

```
const size_t size = 3;
int array[size][size][size] = { { {1,2,3}, {4,5,6}, {7,8,9} },
                                  { {1,2,3}, {4,5,6}, {7,8,9} },
                                  { {7,8,1}, {3,4,5}, {6,7,8} } };

for (size_t i = 0; i < size; i++)
{
    for (size_t j = 0; j < size; j++)
    {
        for (size_t k = 0; k < size; k++)
        {
            cout << array[i][j][k] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

Консоль отладки Microsoft Visual Studio

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

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

```
7 8 1
3 4 5
6 7 8
```

## Область видимости

**Запомнить: переменные объявленные внутри цикла видны и существуют только внутри него.**

Программа с ошибкой:

```
for (size_t i = 0; i < 100; i++)  
{  
    int sum;  
    sum += i;    // считаем сумму всех i  
}
```

`cout << sum;` // Ошибка компиляции: идентификатор "sum" не определен.





# Структурное программирование и оператор goto

**Структурное программирование** — парадигма программирования, в основе которой лежит представление программы в виде **иерархической структуры блоков**. Концептуализирована в конце 1960-х — начале 1970-х годов на фундаменте теоремы Бёма — Якопини, математически обосновывающей возможность структурной организации программ, и работы Эдсгера Дейкстры «О вреде оператора goto».

В соответствии с этой парадигмой, **любая программа**, которая строится без использования оператора goto, состоит из **трёх базовых управляющих конструкций**: **последовательность**, **ветвление**, **цикл**; **кроме того, используются подпрограммы**. При этом разработка программы ведётся пошагово, методом «сверху вниз».

Программа с goto:

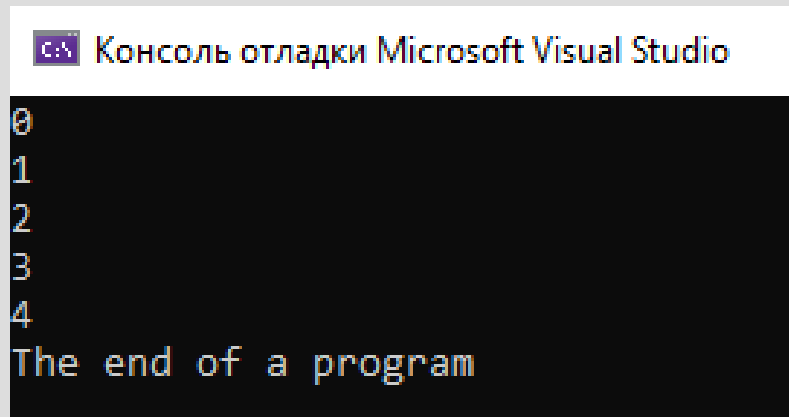
```
int main()
{
    int i = 0;

    label1: // метка для перехода с помощью goto

    cout << i++ << endl;

    if (i < 5) // если i < 5 идем на метку label1
        goto label1;

    cout << "The end of a program";
}
```



```
0
1
2
3
4
The end of a program
```

1. Оператор goto **усложняет программу, повышает вероятность ошибок**, превращает ваш код в «спагетти» и не рекомендуется к использованию.
2. Оператор goto работает только внутри одной функции. Также вы не можете перепрыгнуть вперед через переменную, которая инициализирована в том же блоке, что и goto.

## «Магические» числа

«Магическими числами» называют плохую практику программирования, когда в исходном тексте встречается числовое значение и неочевиден его смысл. Например, такой фрагмент, будет плохим:

```
drawSprite(53, 320, 240);
```

Человеку, который не является автором программы, трудно сказать, что такое 53, 320 или 240. Но если этот код переписать, всё становится на свои места.

```
const int SCREEN_WIDTH = 640;  
const int SCREEN_HEIGHT = 480;  
const int SCREEN_X_CENTER = SCREEN_WIDTH / 2;  
const int SCREEN_Y_CENTER = SCREEN_HEIGHT / 2;  
const int SPRITE_ALPHA = 53;
```

...

```
drawSprite(SPRITE_ALPHA, SCREEN_X_CENTER, SCREEN_Y_CENTER);
```

Есть практика называть константы в программах заглавными буквами.



# Аргументы командной строки

```
#include <iostream>

using namespace std;

int main(int n, char** args) // Параметры запуска программы из командной строки
{                             // Первый параметр всегда имя самой программы (exe)
    if (n > 0)
    {
        for (int i = 0; i < n; i++)
        {
            cout << "Param " << i + 1 << ": " << args[i] << endl;
        }
    }

    return 0;
}
```

`char** args` — это у нас массив строк  
(указатель на указатель)  
Тоже самое что и: `char* args [ ]`  
массив указателей на `char`

```
c:\Users\Dmitry\source\repos\App1\Debug>dir
Том в устройстве C имеет метку Windows-SSD
Серийный номер тома: BE06-4A14

Содержимое папки c:\Users\Dmitry\source\repos\App1\Debug

09.02.2021  10:32    <DIR>          .
09.02.2021  10:32    <DIR>          ..
09.02.2021  10:32                49 152 App1.exe
09.02.2021  10:32                414 736 App1.ilc
09.02.2021  10:32                462 848 App1.pdb
                                3 файлов          926 736 байт
                                2 папок    245 841 952 768 байт свободно
```

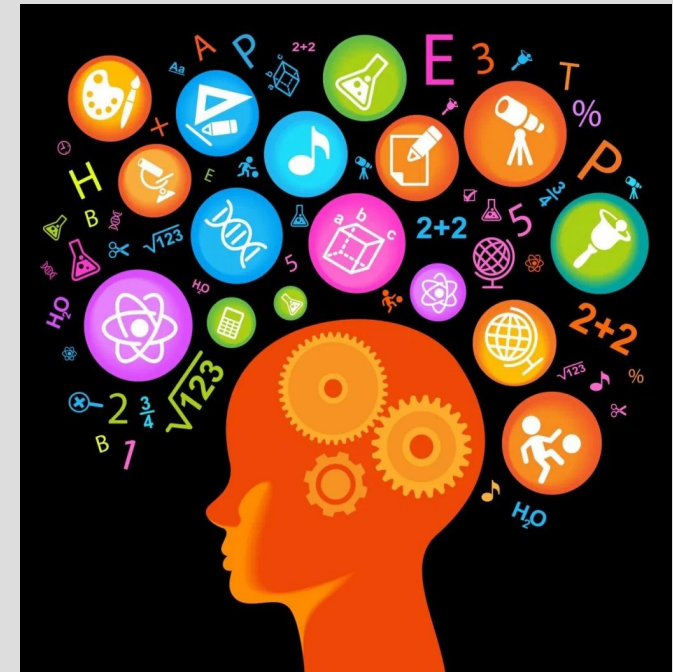
```
c:\Users\Dmitry\source\repos\App1\Debug>App1.exe -aaa -bbb MyParam SomeParam -kk
Param 1: App1.exe
Param 2: -aaa
Param 3: -bbb
Param 4: MyParam
Param 5: SomeParam
Param 6: -kk
```

# Принцип программирования DRY

**DRY — (англ. don't repeat yourself)** — не повторяйся. Принцип разработки ПО, нацеленный на снижение повторения информации, особенно в сложных системах.

Любая информация должна иметь единственное, непротиворечивое и авторитетное представление в рамках информационной системы.

Нарушения принципа DRY называют WET — «Write Everything Twice» Пиши все по два раза.



# Принцип программирования KISS

**KISS — (англ. keep it simple) — не усложняй.**

Принцип проектирования, появившийся в ВМС США.

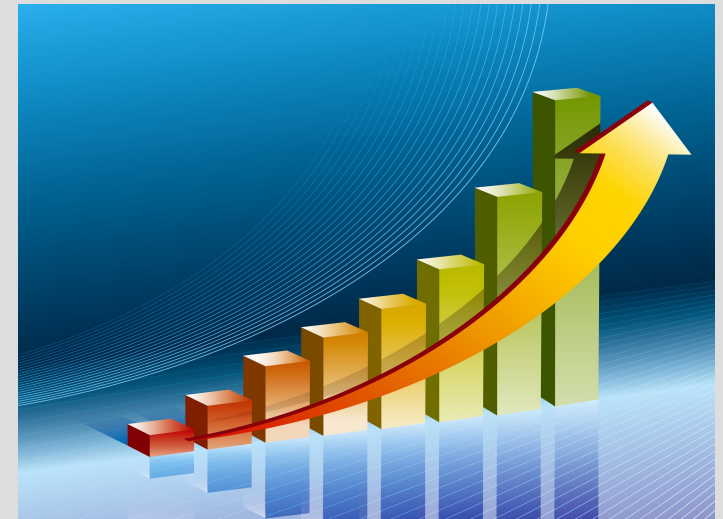
Принцип KISS утверждает, что большинство систем **работают лучше**, если они **остаются простыми**, а не усложняются. Поэтому в области проектирования **простота** должна быть **одной из ключевых целей**, и следует избегать ненужной сложности.



# Принцип программирования YAGNI

**YAGNI** — (англ. *you aren't gonna need it*) — вам это не понадобится.

Принцип проектирования ПО, при котором в качестве основной цели и ценности декларируется отказ от **избыточной** функциональности (в которой нет непосредственной надобности).



# Больше практики программирования

Для тех, кто уверенно себя чувствует с C++ и **успевает решать ДЗ** =) и хочет больше тренироваться в программировании можно порекомендовать сайты для тренировок:

Я, например, использую: <https://www.hackerrank.com>

28 сайтов, на которых можно решать задачи по программированию:  
<https://tproger.ru/digest/competitive-programming-practice/>

**HackerRank**

### Your Skills

INTERVIEW PREPARATION

**Interview Preparation Kit**

Curated challenges and tips based on learnings from 1000+ companies to help you prepare for your upcoming interviews.

View

LANGUAGE PROFICIENCY

**C++**

55% (24/44 challenges solved)

Continue Practice

### Skills Available For Practice

Algorithms	Data Structures	Mathematics
C	C++	Java
Python	Ruby	Linux Shell
Functional Programming	Artificial Intelligence	SQL
Databases	Regex	



## Домашнее задание

1. Написать программу, проверяющую что сумма двух (введенных с клавиатуры) чисел лежит в пределах от 10 до 20 (включительно), если да – вывести строку "true", в противном случае – "false";
2. Написать программу, выводящую на экран строку "true", если две целочисленные константы, объявленные в её начале либо обе равны десяти сами по себе, либо их сумма равна десяти. Иначе "false".
3. Написать программу которая выводит на экран список всех **нечетных** чисел от 1 до 50. Например: "Your numbers: 1 3 5 7 9 11 13 ...". Для решения используйте любой C++ цикл.
4. **Со звездочкой.** Написать программу, проверяющую, является ли некоторое число - простым. Простое число — это целое положительное число, которое делится без остатка только на единицу и себя само.
5. **Со звездочкой.** Пользователь вводит с клавиатуры число (год): от 1 до 3000. Написать программу, которая определяет является ли этот год високосным. Каждый 4-й год является високосным, кроме каждого 100-го, при этом каждый 400-й – високосный. Вывести результаты работы программы в консоль.

**Замечание:** Можно сделать **в одном проекте** (например разместить разные задания в разных функциях). Или в разных проектах если это кажется удобнее.





## Основы C++. Вебинар №4.

Успеха с домашним заданием!



**GeekBrains**