

# ЛЕКЦИЯ 4

- 13.09.2023
- Основные алгоритмические структуры
- Отладка программы

# ЦЕЛИ ЛЕКЦИИ

- Изучить реализации основных алгоритмических конструкций в С#



Это изображение, автор: Неизвестный автор, лицензия: [CC BY-NC](#)

# ОПРЕДЕЛЕНИЯ

- **Конструкция** [construction, structure] – структурный оператор, предназначенный для управления ходом выполнения программы

if

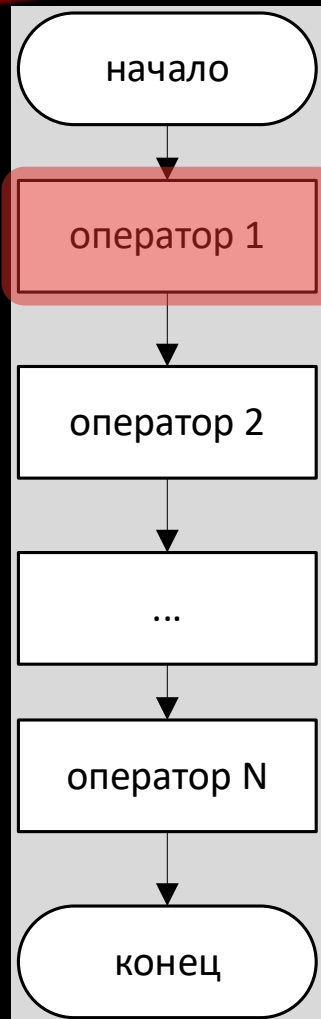
switch

do... while

while

for

# ЛИНЕЙНЫЙ АЛГОРИТМ



- Алгоритм с **линейной структурой** предполагает последовательное выполнение операторов один за другим

```
Console.Write("Enter a number: ");  
double x = double.Parse(Console.ReadLine());  
Console.WriteLine($"{x:f3}");
```

```
Console.Write("Enter first number: ");  
int x = int.Parse(Console.ReadLine());
```

```
Console.Write("Enter second number: ");  
int y = int.Parse(Console.ReadLine());
```

```
Console.WriteLine();  
Console.Write(x > y ? "First is greater": "Second is greater") ;
```

# РЕАЛИЗАЦИЯ ВЕТВЛЕНИЯ

Ветвящиеся алгоритмы

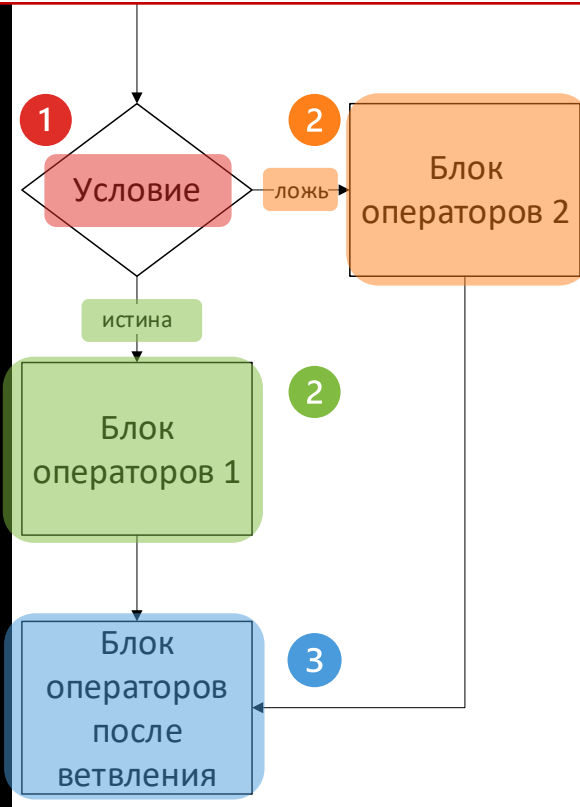
Структурный оператор if

Структурный оператор switch

Выражение switch

# РАЗВЕТВЛЯЮЩИЙСЯ АЛГОРИТМ

## Алгоритмическая конструкция «ветвление»



## Оператор ветвления языка С#

```
1 if (логическое_выражение)
{
2    // блок операторов 1
}
else
{
2    // блок операторов 2
}
3 // блок операторов после
```



# ПРИМЕР. ПОЛНАЯ ФОРМА КОНСТРУКЦИИ ВЕТВЛЕНИЯ

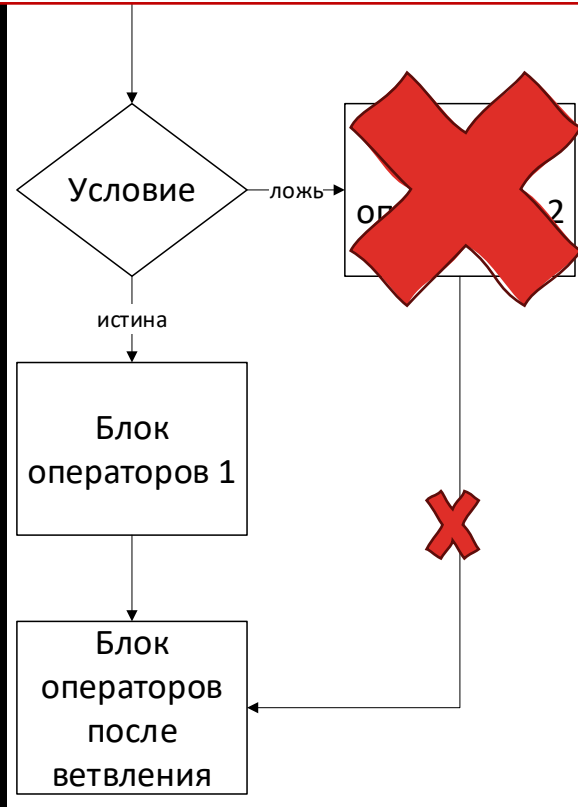
```
Console.Write("Enter a number: ");  
double x;  
if (double.TryParse(Console.ReadLine(), out x))  
{  
    Console.WriteLine($"{x:f3}");  
}  
else  
{  
    Console.WriteLine("Invalid data");  
}
```

## Модификации:

1. Замена `double.Parse()` на `double.TryParse()`
2. Добавлен оператор ветвления для анализа возвращаемого из `double.TryParse()` значения
3. Добавлен оператор печати при некорректных данных в блок оператор после `else`

# НЕПОЛНАЯ ФОРМА ВЕТВЛЕНИЯ

## Алгоритмическая конструкция «ветвление»



## Оператор ветвления языка С#

```
if (логическое_выражение)
{
    // блок операторов 1
}
```



# ПРИМЕР. НЕПОЛНАЯ ФОРМА КОНСТРУКЦИИ ВЕТВЛЕНИЯ

```
Console.Write("Enter a number: ");  
double x;  
if(!double.TryParse(Console.ReadLine(), out x))  
{  
    Console.WriteLine("Invalid data");  
    return;  
}  
Console.WriteLine($"{x:f3}");
```

## Модификации:

1. Замена `double.Parse()` на `double.TryParse()`
2. Добавлен оператор ветвления для анализа возвращаемого из `double.TryParse()` значения
3. Добавлен оператор печати при некорректных данных в блок оператор после `if`
4. Использован оператор безусловного перехода `return` для завершения программы при некорректных данных

# ВЛОЖЕННЫЕ ОПЕРАТОРЫ ВЕТВЛЕНИЯ

Внешний оператор  
ветвления

Полностью  
вложен в одну из  
секций  
внешнего if


```
int x, y;
Console.Write("Enter first number: ");
if (!int.TryParse(Console.ReadLine(), out x))
{
    Console.Write("Invalid first number!");
    return;
}
else
{
    Console.Write("Enter second number: ");
    if (!int.TryParse(Console.ReadLine(), out y))
    {
        Console.Write("Invalid second number!");
        return;
    }
    else
    {
        Console.WriteLine();
        Console.Write(x > y ? "First is greater" : "Second is greater");
    }
}
```

Вложенный  
оператор  
ветвления


# КУЛЬТУРА КОДИРОВАНИЯ (1)

- Обязательно используем `{ }` даже в случае с одним оператором в блоке
  - Это снизит количество ошибок
  - Сократит время при модификации кода

```
int a = 10, b = 22;  
if (a > 5)  
    Console.WriteLine(b / 2);  
Console.Write(b % 2);
```




```
int a = 10, b = 22;  
if (a > 5)  
{  
    Console.WriteLine(b / 2);  
}  
  
Console.Write(b % 2);
```




# КУЛЬТУРА КОДИРОВАНИЯ (2)

- Всё, что возвращает тип **bool** в условии **if ()** не сравнивается явно с **true/false**

```
int op;  
if(int.TryParse(Console.ReadLine(), out op)==true)  
{  
    Console.Write(++op);  
}  
else  
{  
    Console.Write("Not a number!");  
}
```



```
int op;  
if(int.TryParse(Console.ReadLine(),out op))  
{  
    Console.Write(++op);  
}  
else  
{  
    Console.Write("Not a number!");  
}
```



# ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

- Запустите код. Какие проблемы с оформлением кода вы видите?
  - Исправьте их при реализации
- Измените код программы, вычисляющей модуль целого числа, с использованием тернарной операции
  - **Требование:** Условный оператор if должен исчезнуть из программы!

```
1.  int x;  
2.  if (int.TryParse(Console.ReadLine(), out x))  
3.  {  
4.      if (x < 0)  
5.      { x = -x;  
6.          Console.Write("Модуль числа равен: {0}.", x);  
7.      }  
8.      else  
9.          Console.Write("Вы ввели не целое число!");  
10. }
```

# МНОЖЕСТВЕННЫЙ ВЫБОР



```
1 switch (Выражение)
{
  case label1: /* блок операторов 1 */ break;
  2 case label2: /* блок операторов 2 */ break;
  case label3: /* блок операторов 3 */ break;
  ...
  3 default: break;
}
```



# ПРИМЕР

```
int day;  
Console.WriteLine("Enter a number of a day: ");  
int.TryParse(Console.ReadLine(), out day);  
  
switch (day)  
{  
    case 1: Console.Write("Monday"); break;  
    case 2: Console.Write("Tuesday"); break;  
    case 3: Console.Write("Wednesday"); break;  
    case 4: Console.Write("Thursday"); break;  
    case 5: Console.Write("Friday"); break;  
    case 6: Console.Write("Saturday"); break;  
    case 7: Console.Write("Sunday"); break;  
    default: Console.Write("Invalid number of a day"); break;  
}
```

В этой задаче нам не важно, откуда 0 взялся после работы TryParse()

Нуль всегда попадёт сюда

# КУЛЬТУРА КОДИРОВАНИЯ

- Не рекомендуется создавать громоздкие переключатели
- Старайтесь делать тело блока, состоящее из одного выражения

# ВЫРАЖЕНИЕ-ПЕРЕКЛЮЧАТЕЛЬ

```
1 <выражение> switch {  
2     шаблон1 [охранное условие] => выражение1,  
3     шаблон2 [охранное условие] => выражение2,  
    ...  
    шаблонN [охранное условие] => выражениеN,  
    - => выражение  
};
```

Ветви выражения  
разделены запятыми

Шаблон пустой  
переменной

Если ни один шаблон не подходит, а шаблон  
пустой переменной не указан, происходит:  
`System.Runtime.CompilerServices.SwitchExpressionEx  
ception`

# ПРИМЕР ВЫРАЖЕНИЯ-ПЕРЕКЛЮЧАТЕЛЯ

```
1. int day;
2. Console.WriteLine("Enter a number of a day: ");
3. int.TryParse(Console.ReadLine(), out day);

4. string output = day switch
5. {
6.     1 => "Monday",
7.     2 => "Tuesday",
8.     3 => "Wednesday",
9.     4 => "Thirsdays",
10.    5 => "Friday",
11.    6 => "Saturday",
12.    7 => "Sunday",
13.    _ => "Invalid number of a day"
14. };
15. Console.WriteLine(output);
```

```
1. int day;
2. Console.WriteLine("Enter a number of a day: ");
3. int.TryParse(Console.ReadLine(), out day);

4. switch (day)
5. {
6.     case 1: Console.Write("Monday"); break;
7.     case 2: Console.Write("Tuesday"); break;
8.     case 3: Console.Write("Wednesday"); break;
9.     case 4: Console.Write("Thirsdays"); break;
10.    case 5: Console.Write("Friday"); break;
11.    case 6: Console.Write("Saturday"); break;
12.    case 7: Console.Write("Sunday"); break;
13.    default: Console.Write("Invalid number of a day"); break;
14. }
```

Найдите отличия


# ПРИМЕР РАЗНЫХ ШАБЛОНОВ

```
1. int day;
2. Console.WriteLine("Enter a number of a day:");
3. int.TryParse(Console.ReadLine(), out day);

4. string output = day switch
5. {
6.     <= 5 => "Workday",
7.     6 => "Weekend",
8.     7 => "Weekend",
9.     _ => "Invalid number of a day"
10. };
11. Console.WriteLine(output);
```

**В примере использованы:**

- Шаблон константы
- Реляционный шаблон
- Шаблон пустой переменной
- Логический дизъюнктивный шаблон



```
1. string output = day switch
2. {
3.     <= 5 => "Workday",
4.     6 or 7 => "Weekend",
5.     _ => "Invalid number of a day"
6. };
```

# ПЕРЕКЛЮЧАТЕЛЬ

```
1. int formNumber = 5;
2. switch(formNumber)
3. {
4.     case 1:
5.     case 2:
6.     case 3: Console.WriteLine("Less or equal to 3"); break;
7.     case 4:
8.     case 5:
9.     case 6: Console.WriteLine("More then 3 but less or equal to 7"); break;
10. }
```

## Как работает этот код?

Модифицируйте код с использованием выражения-переключателя.  
Добавьте возможность получения значения formNumber от пользователя



# РЕАЛИЗАЦИЯ ЦИКЛОВ

Циклические алгоритмы

Цикл с преусловием `do...while`

Цикл с постусловием `while`

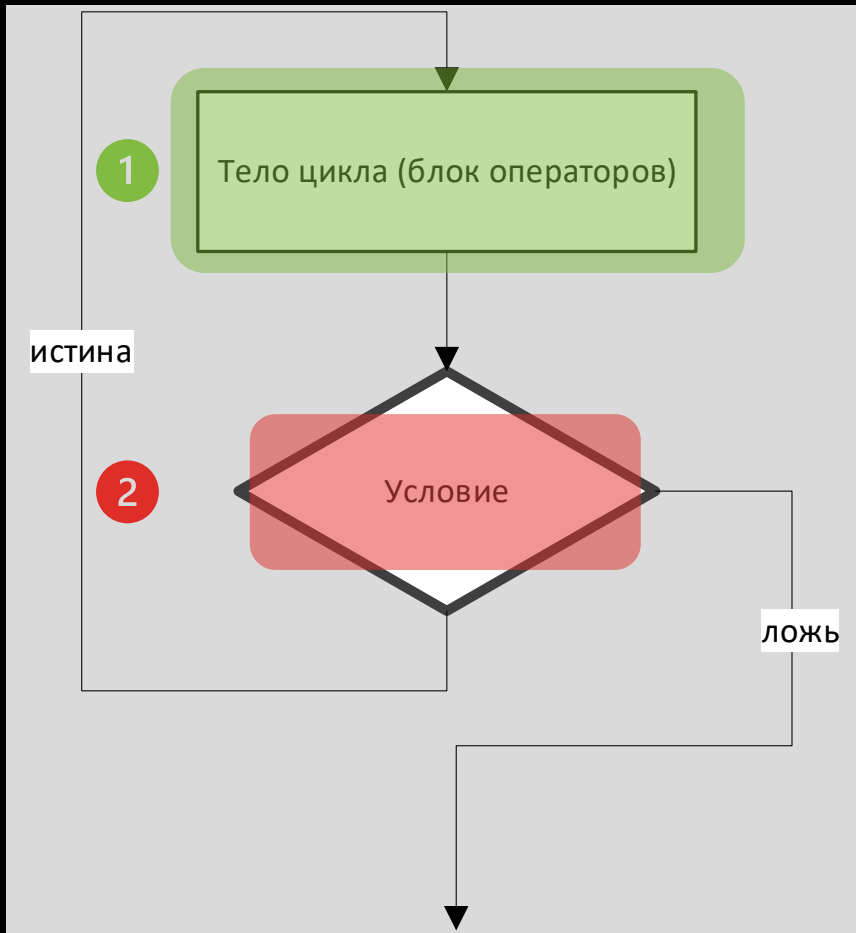
Универсальный цикл `for`

# ПОПУЛЯРНЫЕ ТИПЫ ЗАДАЧ С ЦИКЛАМИ

- **Поэлементная обработка данных**
  - Например, получаемых при вводе пользователем данных (чисел, строк и проч. объектов)
- **Вычисления конечных и бесконечных сумм, произведений, количества и другой агрегированной информации**
- **Формирование табличных данных и их вывод**

# ЦИКЛ С ПОСТУСЛОВИЕМ

Всегда выполняется хотя бы один раз



```
do
{
    // Тело цикла
}
while (Выражение);
```

# ПРИМЕР. ПОВТОРЕНИЕ ВВОДА ДАННЫХ

Использование цикла с постусловием для организации ввода данных

```
double x;  
  
do  
{  
    Console.Write("Enter a number: ");  
} while (!double.TryParse(Console.ReadLine(), out x));  
  
Console.WriteLine($"{x:f3}");
```

# ПРИМЕР. ВЫЧИСЛЕНИЕ ОБОБЩЁННЫХ ХАРАКТЕРИСТИК ДАННЫХ БЕЗ СОХРАНЕНИЯ

- Пользователь последовательно вводит с клавиатуры произвольные натуральные числа. Признак окончания ввода – ввод числа 0. Программа вычисляет и выводит на экран сумму введенных чисел и их среднее арифметическое. При некорректных данных запрашивать повторный ввод

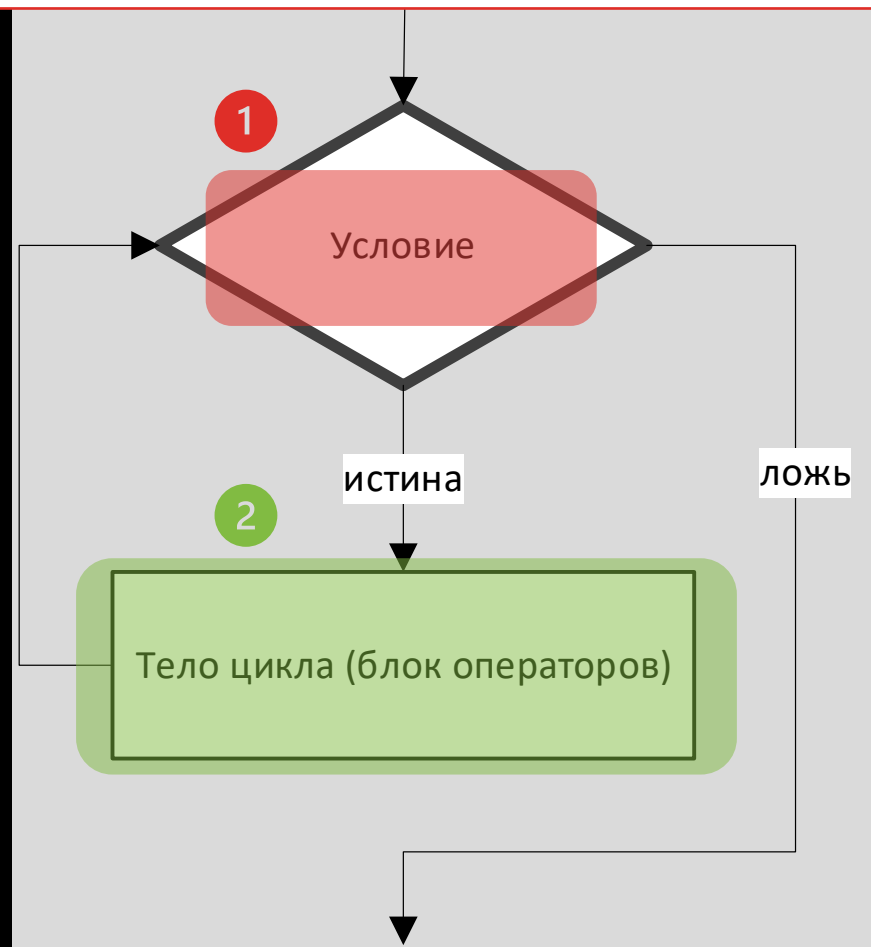
Внешний  
цикл

Вложенный  
цикл

```
1. int N;  
2. int sum = 0, counter = 0;  
3. do  
4. {  
5.     do  
6.     {  
7.         Console.WriteLine("Enter a number: ");  
8.     } while (!int.TryParse(Console.ReadLine(), out N) || N < 0);  
9.     sum += N;  
10.    counter++;  
11.} while (N != 0);  
12.Console.WriteLine($"The sum is: {sum}");  
13.Console.WriteLine($"The mean is: {sum / --counter}");
```

# ЦИКЛ С ПРЕДУСЛОВИЕМ

может не выполняться ни разу, если условие ложно



1

```
while (Выражение)
{
    // тело цикла
}
```

2

// тело цикла



# ПРИМЕР. ПЕЧАТЬ ЦИФР ЧИСЛА В ОБРАТНОМ ПОРЯДКЕ

```
int N;  
do  
{  
    Console.WriteLine("Enter N >= 0: ");  
} while (!int.TryParse(Console.ReadLine(), out N) || N < 0);  
  
int digit;  
while (N > 0)  
{  
    digit = N % 10;  
    Console.Write(digit);  
    N = (N - digit) / 10;  
}
```

# ВЫЧИСЛЕНИЕ КОНЕЧНЫХ СУММ И ПРОИЗВЕДЕНИЙ

- Пример записи сумм

$$\sum_{n=1}^K \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{K}$$

- Пример записи произведения

$$\prod_{n=2}^{\infty} \left( \frac{n^3 - 1}{n^3 + 1} \right) = \frac{7}{9} \cdot \frac{26}{29} \cdot \dots$$

# ПРИМЕР. ВЫЧИСЛЕНИЕ БЕСКОНЕЧНОЙ СУММЫ

```
int n = 1;
double sum = 1.0 / n;
double prevSum = 0;

while (sum - prevSum > double.Epsilon)
{
    prevSum = sum;
    sum += 1.0 / (++n * n);
}

Console.WriteLine($"From our program sum: {sum}");
Console.WriteLine($"Presice sum: {Math.PI * Math.PI / 6}");
```

ВЫЧИСЛИТЬ

$$\sum_{k=0}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

1,64491248698659
1,64493406684823

# ПРИМЕР. ВЫЧИСЛЕНИЕ БЕСКОНЕЧНОГО ПРОИЗВЕДЕНИЯ

```
int n = 2;
double prod = (Math.Pow(n, 3) - 1) / (Math.Pow(n, 3) + 1) ;
double prevProd = 0;

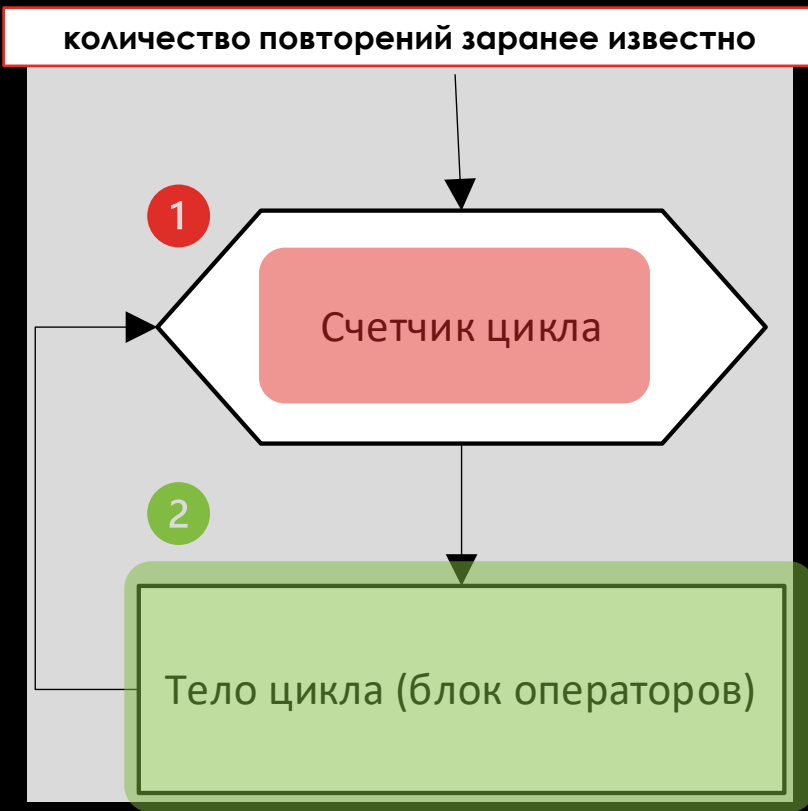
while (Math.Abs(prod - prevProd) > double.Epsilon)
{
    prevProd = prod;
    prod *= (Math.Pow(++n, 3) - 1) / (Math.Pow(n, 3) + 1)
}
Console.WriteLine($"From our program product: {prod}");
Console.WriteLine($"Presice product: {2 / 3D}");
```

ВЫЧИСЛИТЬ

$$\prod_{n=2}^{\infty} \left( \frac{n^3 - 1}{n^3 + 1} \right) = \frac{2}{3}$$

0.6666666666825042  
0.6666666666666666

# УНИВЕРСАЛЬНЫЙ ЦИКЛ FOR



1

```
for (инициатор счётчика; условие продолжения; изменение счётчика)
{
    // Блок операторов
    // тела цикла
}
2
```

# ЦИКЛ С ПАРАМЕТРОМ

1) Описываем и инициализируем переменную limit

2) Описываем и инициализируем переменную i

3) Проверяем условие продолжения

```
int limit = 15;  
for (int i = 0; i < limit; i++)  
{  
    Console.Write(i + " ");  
}
```

6) Снова проверяем условие продолжения

4) Выполняем тело цикла

5) Изменяем значение управляющей переменной



# ПРИМЕР. ВЫВОД НАПЕРЁД ЗАДАННОГО КОЛИЧЕСТВА ЧИСЕЛ

Пользователь задаёт отрезок  $[A, B]$ ,  $A$  и  $B$  – целые числа такие, что  $A < B$ .

Вывести на экран, разделяя пробелами все целые числа, принадлежащие этому отрезку.  
Сами числа  $A$  и  $B$  не выводить

```
int a;  
do Console.WriteLine("Введите A: ");  
while (!int.TryParse(Console.ReadLine(), out a));  
int b;  
do Console.WriteLine("Введите B: ");  
while (!int.TryParse(Console.ReadLine(), out b));  
  
for (int i = a + 1; i < b; i++)  
{  
    Console.Write($"{i} ");  
}
```

Исключаем  
левую границу

Исключаем  
правую границу



**Freya Holmér**  
@FreyaHolmer



btw these large scary math symbols are just for-loops

**Summation**  
(capital sigma)

$$\sum_{n=0}^4 3n$$

```
sum = 0;  
for( n=0; n<=4; n++ )  
    sum += 3*n;
```

**Product**  
(capital pi)

$$\prod_{n=1}^4 2n$$

```
prod = 1;  
for( n=1; n<=4; n++ )  
    prod *= 2*n;
```

# ПРИМЕР. ВЫЧИСЛЕНИЕ КОНЕЧНОЙ СУММЫ ЗНАКОПЕРЕМЕННОГО РЯДА

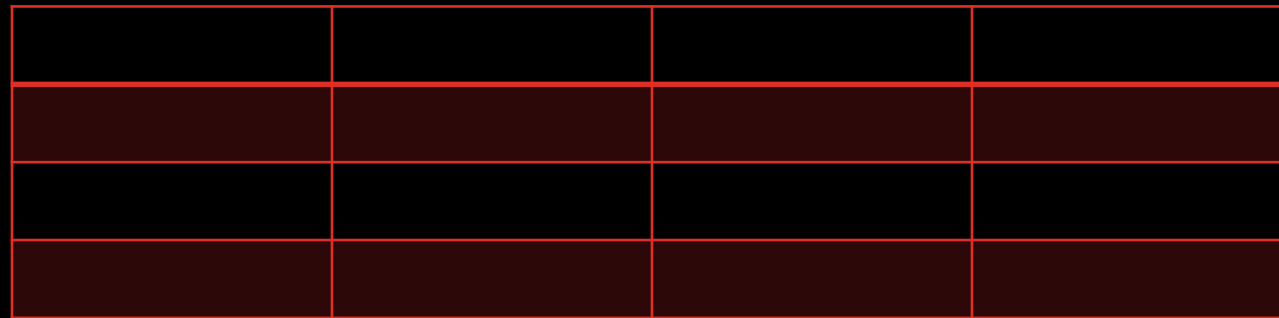
```
1. uint n;  
2. do Console.WriteLine("Введите n: ");  
3. while (!uint.TryParse(Console.ReadLine(), out n) || n == 0);  
  
4. int k = 1;  
5. double sum = 0.0;  
  
6. do  
7. {  
8.     sum += -1 / (double)((2 * k + 1) * k);  
9. } while (++k <= n);  
10. Console.WriteLine($"Sum = {sum:f5}");
```

$$\sum_{k=1}^n \frac{(-1)^k}{(2k+1)k}$$

# ВЫВОД ТАБЛИЧНЫХ ДАННЫХ

- Данные выводятся в табличном виде, то есть столбцы и строки
- Потребуется управление выводом данных в строку
- Потребуется механизмы старта новой строки
- Строки и столбцы понимаются в широком смысле

столбцы




строки

# ПРИМЕР. ВЫВОД ТАБЛИЧНЫХ ДАННЫХ

- Выведем на экран таблицу Пифагора

Внешний цикл управляет переводом строк

И данные

Внутренний цикл управляет выводом данных в одной строке

```
Console.BackgroundColor = ConsoleColor.Blue;  
Console.ForegroundColor = ConsoleColor.White;  
Console.Write("\t");  
for (int k = 1; k < 10; k++) {  
    Console.Write($"{k}\t");  
}
```

Выведем  
заголовок

```
Console.BackgroundColor = ConsoleColor.Black;  
for (int i = 1; i < 10; i++) {  
    Console.BackgroundColor = ConsoleColor.Blue;  
    Console.ForegroundColor = ConsoleColor.White;  
    Console.WriteLine();  
    Console.Write($"{i}\t");  
    Console.BackgroundColor = ConsoleColor.Black;  
    for (int j = 1; j < 10; j++) {  
        Console.Write($"{i * j}\t");  
    }  
}
```

# КОГДА КАКОЙ ЦИКЛ ИСПОЛЬЗОВАТЬ?

- Строгих правил нет и чаще всего использование циклов становится делом вкуса
- **НО есть рекомендации:**
  - Число итераций цикла и проверок условий нужно минимизировать
  - Если заранее известно количество повторений цикла – используем **for**
  - Если заранее число повторений не известно, используем **while** или **do...while**
  - Если требуется обязательно хотя бы раз выполнить какие-то действия, то используем **do...while**

# ОПЕРАТОРЫ БЕЗУСЛОВНОГО ПЕРЕХОДА

break

continue

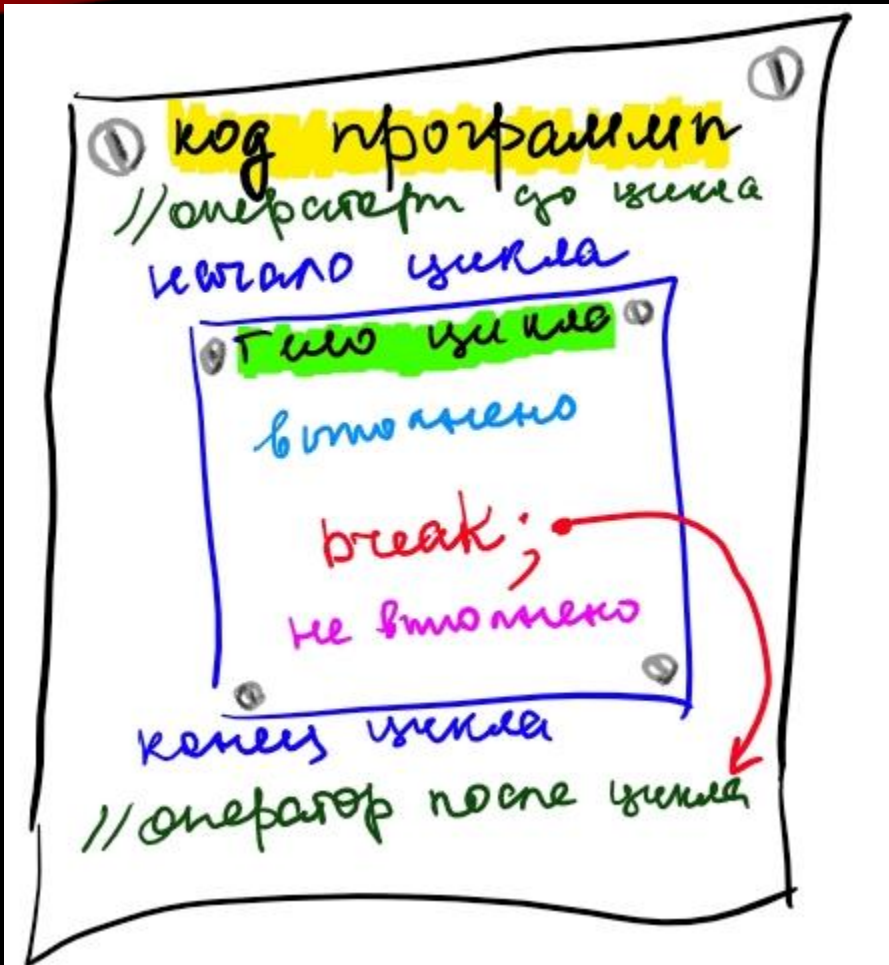




# ОПЕРАТОР BREAK

Завершает (ПОЛНОСТЬЮ) выполнение ближайшего оператора цикла и передаёт управление оператору, расположенному после цикла

```
for (int i = 0; i < 10; i++)
{
    if (i > 5)
    {
        break;
    }
    Console.WriteLine($"{i} ");
}
Console.WriteLine("Hello outside the for!!!");
```



# ОПЕРАТОР CONTINUE

Начинает **новую итерацию** ближайшего оператора цикла и передаёт управление оператору, расположенному после цикла

```
for (int i = 0; i < 10; i++)  
{  
    if (i % 2 == 0)  
    {  
        continue;  
    }  
    Console.WriteLine($"{i} ");  
}  
Console.WriteLine("Hello outside the for!!!");
```



# ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- Обзор среды CLR (<https://docs.microsoft.com/ru-ru/dotnet/standard/clr>)
- <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/switch-expression>
- [Selection statements - C# reference | Microsoft Docs](#)
- <https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/patterns>
- <https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/patterns#constant-pattern>
- [Операторы jump — break, continue, return и goto - C# | Microsoft Learn](#)
- Richter, J. CLR via C#. Fourth edition