# ЛЕКЦИЯ 3

- 13.09.2023
- Операторы и операции языка C#
- Выражения

# ЦЕЛИ ЛЕКЦИИ

- Получить сведения о группах операций языка C#
- Изучить некоторые особенности операций языка C#
- Получить представление об операторе присваивания, r-value и l-value

# ВСПОМНИМ ИДЕНТИФИКАТОРЫ

Укажите все верные идентификаторы в понимании языка C# (*рекомендации по культуре именования не учитывать*):

1) A
2) 2A
3) else
4) _if
5) Main_var

# ВСПОМНИМ ПЕРЕМЕННЫЕ И ТИПЫ ДАННЫХ

Укажите все верные варианты ответов, в кодах которых присутствуют корректные описания целочисленных беззнаковых переменных:

1) double x;
2) int a;
3) int b = 7;
4) byte c;
5) ushort 32bitVal

# ВСПОМНИМ ПРИВЕДЕНИЕ ТИПОВ

В коде программы использована переменная, объявленная с типом ushort. Укажите, в переменные с каким типом может быть приведено её значение неявно:

1) byte
2) double
3) int
4) sbyte
5) short

# ОПРЕДЕЛЕНИЯ

- **Фраза** [phrase] – любая последовательность символов, являющаяся правильной (валидной), то есть удовлетворяющая синтаксическим и семантическим правилам языка программирования

**Фразы:**

```
string[] fileData;
```

```
Console.WriteLine("Hello, World!");
```

**Не фразы:**

```
string[] 3fileData;
```

```
Console.WriteLine("Hello, World!"));
```

# ОПРЕДЕЛЕНИЯ

- **Выражение** [expression] – фраза на языке программирования, предназначенная для выполнения вычислений, которые позволяют получить некоторый результат (значение выражения)
  - Выражение обычно состоит из операндов (переменных, констант и др.), объединенных знаками операций (вызов подпрограммы тоже можно считать операцией)

# ОПРЕДЕЛЕНИЯ

- **Оператор** [statement, operator] – фраза на языке программирования, определяющая законченный этап обработки данных. В состав операторов входят ключевые слова, данные, выражения и др.
  - атомарные операторы, никакая часть которых не является самостоятельным оператором
  - структурные операторы, объединяющие набор операторов в единый (укрупненный) оператор
- **Конструкция** [construction, structure] – структурный оператор, предназначенный для управления ходом выполнения программы

# ГРУППЫ ОПЕРАЦИЙ C#

**Количество операндов**

- Унарные
- Бинарные
- Тернарная

**По типам операндов и цели**

- Арифметические
- Логические
- Битовые
- Сравнения
- Равенства

# УНАРНЫЕ ОПЕРАЦИИ

Унарные операции

Особенности префиксной и постфиксной формы инкремента и декремента

# УНАРНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ (1)

**для ulong не поддерживается**

\+

Знак плюс
(обозначение положительности)

```
int x = +10;  // x = 10
Console.Write(x);
```

\-

Знак минус
(обозначение отрицательности)

```
int x = +10;  // x = 10
Console.Write(-x); // Вывод: -10
```

Что выведет этот код?

```
int x = -10;
int y = +x;
Console.Write(y);
```

# УНАРНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ (2)

**Префиксный**

**Постфиксный**

++

Инкремент

Значение переменной изменяется **до** участия в выражениях

```
int x = 0; // x = 0;
int y = --x; // x = -1 y = -1
Console.WriteLine(x + y); // x = -1 y = -1
```

```
int x = 0; // x = 0;
int y = x++; // x = 0 y = 0
Console.WriteLine(x + y); // x = 1 y = 0
```

Значение переменной изменяется **после** участия в выражениях

--

Декремент

# ВЫРАЖЕНИЯ, ЭКВИВАЛЕНТНЫЕ ИНКРЕМЕНТУ И ДЕКРЕМЕНТУ

**Эквивалентные выражения:**

```
++x      x += 1      x = x + 1
```

**Не эквивалентные выражения:**

```
x++      x += 1 (x++ возвращает копию x)
```

```
int a = 1, b = 1, c = 1;
Console.WriteLine($"a+=1 = {a += 1}");  // a+=1 = 2
Console.WriteLine($"++b = {++b}");      // ++b = 2
Console.WriteLine($"c++ = {c++}");      // c++ = 1
```

# УНАРНАЯ БИТОВАЯ ОПЕРАЦИЯ

**Префиксная**

~

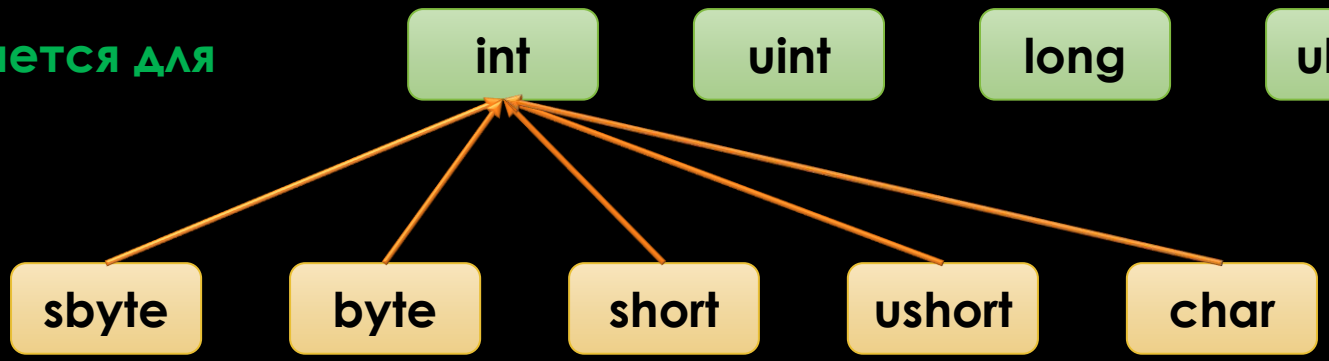
Побитовое дополнение

```
byte foo = 16;

Console.WriteLine($"{Convert.ToString(foo, 2)}");
Console.WriteLine($"{Convert.ToString(~foo, 2)}");
```

```
                10000
11111111111111111111111101111
```

**Поддерживается для**

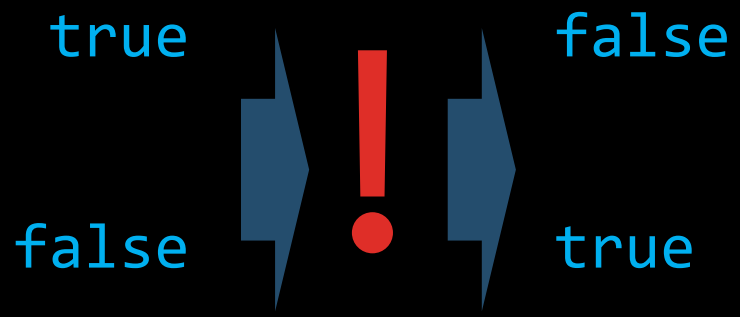


# УНАРНАЯ ЛОГИЧЕСКАЯ ОПЕРАЦИЯ

**Префиксная**

!

Логическое отрицание




```
bool logical = false;
Console.Write(!logical); // logical = false; output: True
Console.Write(logical);  // logical = false; output: False
Console.Write(!true);    // output: False
```

В постфиксной форме становится другим оператором!
Допускает значение **NULL**

# БИНАРНЫЕ ОПЕРАЦИИ

# БИНАРНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

\+ Сложение

```
int a = 5, b = 6;
int c = a + b;
```

**Переопределено**:
- конкатенация строк
- объединение списков вызовов делегатов

\- Вычитание

```
int a = 5, b = 6;
int c = a - b;
```

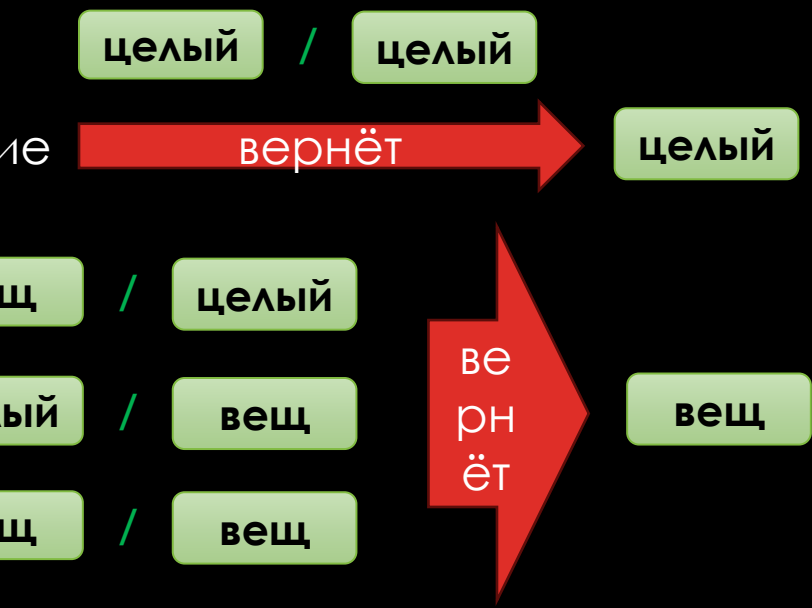
**Переопределено**:
- удаление из списков вызовов делегатов

* Умножение

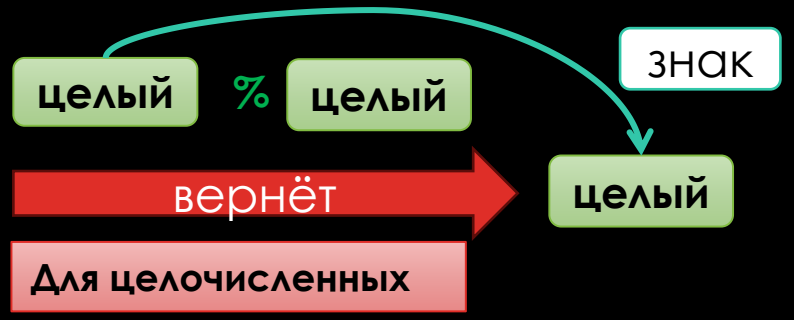
```
int a = 5, b = 6;
int c = a * b;
```

**Унарный *** - разыменовывание указателя

/ Деление




% Остаток от деления



# МЕХАНИКА РАБОТЫ ОПЕРАЦИИ ВЗЯТИЯ ОСТАТКА ОТ ДЕЛЕНИЯ

**Для целых:**

```
a % b
```

```
a - (a / b) * b
```

Если нужно и частное и остаток от деления – есть метод Math.DivRem() [http://learn.microsoft.com/ru-ru/dotnet/api/system.math.divrem?view=net-6.0]

**Для вещественных:**

`|a| - n * |b|`, `n` – наибольшее значение, которое меньше значения `|a| / |b|`

Знак остатка берётся по левому операнду

Для вычислений в соответствии с IEEE 754 существует метод Math.IEEERemainder() [http://learn.microsoft.com/ru-ru/dotnet/api/system.math.ieeeremainder?view=net-6.0]

# БИТОВЫЕ БИНАРНЫЕ ОПЕРАЦИИ

**Для целочисленных операндов - побитовое!**

& Конъюнкция (И) **Унарный &**- взятие адреса

| Дизъюнкция (ИЛИ)

^ Исключающее ИЛИ

```
int r = 3, d = 4;
int b = r & d;
Console.WriteLine(b);
```

r = $011_2$,
d = $100_2$,
b = $000_2$

**Что будет выведено?**

```
int r = 11, d = 7;
int b = r & d;
Console.Write(b);
```

$7_2 = 0111_2$
$11_2 = 1011_2$

$$\begin{array}{cc} & \begin{matrix}0 & 1 & 1 & 1\end{matrix} \\ \& & \begin{matrix}1 & 0 & 1 & 1\end{matrix} \\ \hline & \begin{matrix}0 & 0 & 1 & 1\end{matrix} \end{array}$$

# ВОПРОС

Что будет выведено?

```
uint a = 0b_1111_1001;
uint b = 0b_1001_1101;
uint c = a & b;
Console.Write(Convert.ToString(c, toBase: 2));
```

# БИТОВЫЕ БИНАРНЫЕ ОПЕРАЦИИ СДВИГОВ

## << Сдвиг влево

```
uint a = 0b_1111_1001;
Console.WriteLine($"{Convert.ToString(a, 2)}");
uint lShiftA = a << 4;
Console.WriteLine($"{Convert.ToString(lShiftA, 2)}");
```

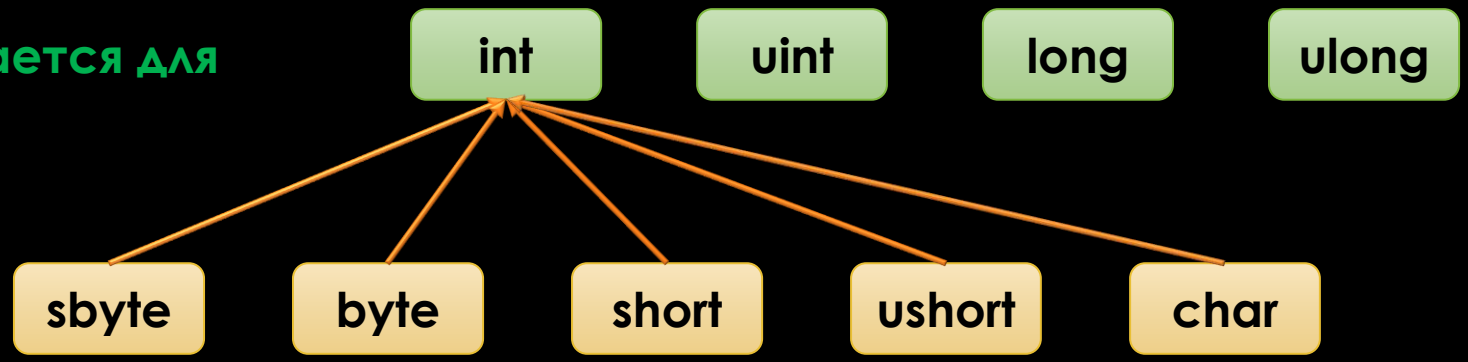
- Старшие биты за пределами диапазона типа будут отброшены
- Младшие биты заполняются нулями

## >> Сдвиг вправо

```
int a = -249;
Console.WriteLine($"{Convert.ToString(a, 2)}");
int rShiftA = a >> 2;
Console.WriteLine($"{Convert.ToString(rShiftA, 2)}");
```

- Старшие разряды заполняются знаковым битом
- Младшие биты за пределами диапазон типа будут отброшены

**Поддерживается для**



# ЛОГИЧЕСКИЕ БИНАРНЫЕ ОПЕРАЦИИ

**Для логических операндов - логическое!**

`&` Логическое И

`|` Логическое ИЛИ

`^` Исключающее ИЛИ

Правый операнд «ленивой операции» не вычисляется, если левый имеет значение

`&&` «ленивое» И — `false`

`||` «ленивое» ИЛИ — `true`

```
Console.WriteLine(true & true);
Console.WriteLine(true & false);
Console.WriteLine(false & true);
Console.WriteLine(false & false);
```

# БИНАРНЫЕ ОПЕРАЦИИ СРАВНЕНИЯ

Возвращают логический тип **bool**

| | |
|---|---|
| < | меньше чем |
| > | больше |
| =< | меньше или равно |
| >= | больше или равно |

```
int x = int.Parse(Console.ReadLine());
Console.WriteLine(x > 0);
Console.WriteLine(x >= 0);
Console.WriteLine(x < 0);
Console.WriteLine(x <= 0);
```

```
Console.WriteLine(double.NaN > 15);
Console.WriteLine(float.NaN < double.NaN);
```

# ИСПОЛЬЗУЕМ СЛОЖНЫЕ ВЫРАЖЕНИЯ

Проверить принадлежность X отрезку [E; F]:

```
bool factor = X >= E & X <= F;
```

Проверить принадлежность точки (x, y) кругу с центром в (0, 0) и радиусом r:

```
bool inside = x * x + y * y < r * r;
```

# БИНАРНЫЕ ОПЕРАЦИИ ПРОВЕРКИ НА РАВЕНСТВО

Возвращают логический тип **bool**

== равно

!= не равно

Эквивалентная запись: **!(a == b)**

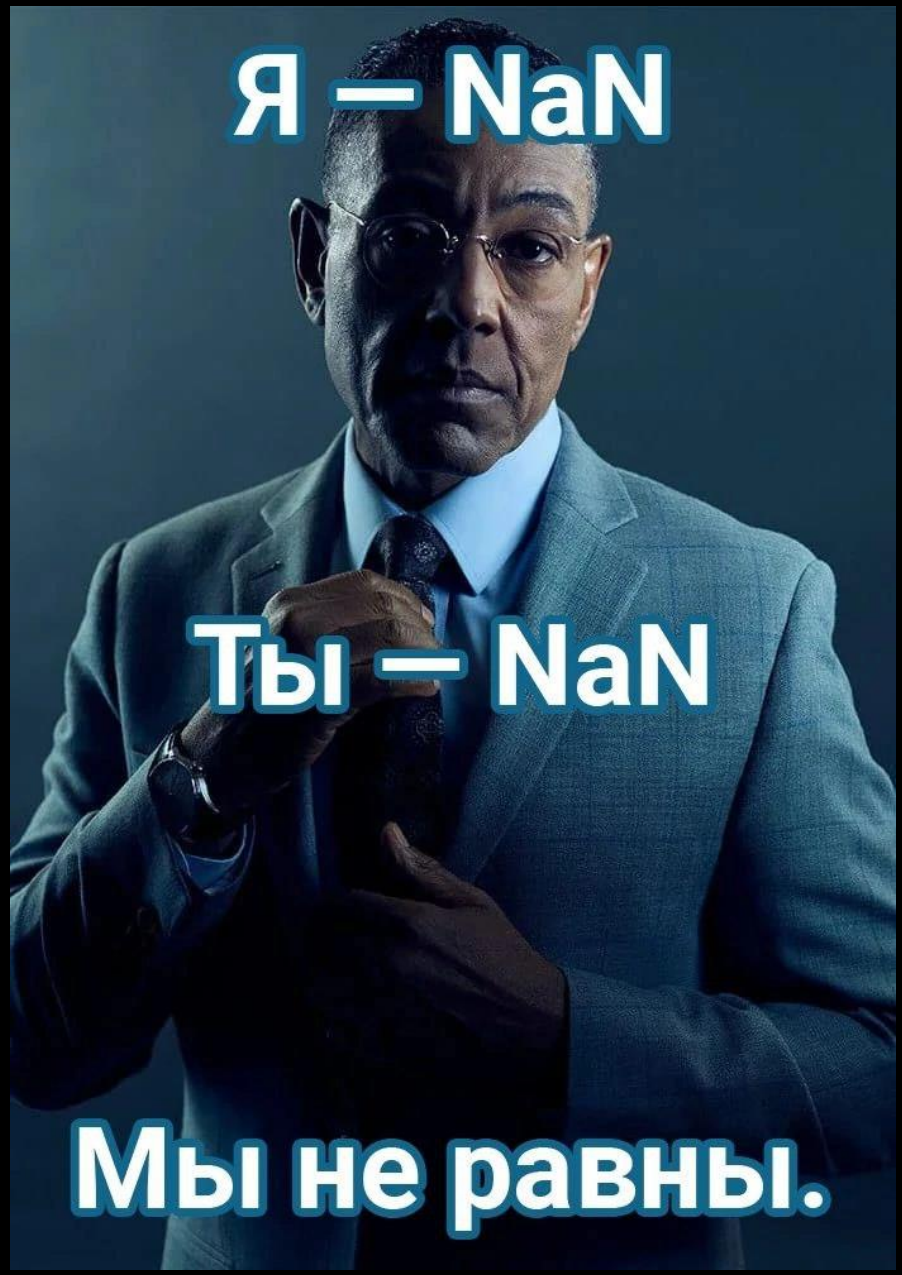
```
int a = 0b_0000_0110;
int b = 6;
Console.WriteLine($"a==b:: {a == b}");

char c1 = 'w';
char c2 = 'W';
Console.WriteLine($"c1 == c2:: {c1 == c2}");
Console.WriteLine($"c1 != c2:: {c1 != c2}");
```

# ОПЕРАЦИИ ОТНОШЕНИЙ И ВЕЩЕСТВЕННЫЕ ЗНАЧЕНИЯ

- **Эпсилон** вещественного формата задаёт границу неразличимости двух вещественных значений
  - То есть мы на бумаге можем видеть разницу, а компьютер этой разницы уже не видит

```
double eps = double.Epsilon;
double testval = 2.4;
Console.WriteLine(testval == testval + eps);
```


Я — NaN
Ты — NaN
Мы не равны.

# ПРИОРИТЕТЫ И АССОЦИАТИВНОСТЬ

# ПРИОРИТЕТ

Какова последовательность вычислений выражения **2 / 6 * 4** :

**(2 / 6) * 4 ==> 4/3**

или

**2 / (6 * 4) ==> 1/12**

**Приоритет** [precedence] – характеристика операции, определяющая порядок выполнения операций в сложных выражениях

убывание

| | |
|---|---|
| +, -, !, ~, ++, --, ^ | Унарные |
| *, /, % | Бинарные мультипликативные |
| +, - | Бинарные аддитивные |
| <<, >> | Битовые сдвиги |
| >, <, >=, <= | Отношений |
| ==, != | Равенства |
| & | И |
| ^ | Исключающее ИЛИ |
| \| | ИЛИ |
| && | Логическое И |
| \|\| | Логическое ИЛИ |
| ? : | Тернарная операция |
| = | Операции назначений |

# АССОЦИАТИВНОСТЬ

Какова последовательность вычислений выражения **2 / 6 * 4** :

```
(2 / 6) * 4 ==> 4/3
```

или

```
2 / (6 * 4) ==> 1/12
```

**Ассоциативность** [associativity] – это свойство операции, определяющее направление ее выполнения

Слева направо – операция с **левой ассоциативностью**

Справа налево – операция с **правой ассоциативностью**

**Левоассоциативные**:
все бинарные операции

**Правоассоциативные**:
Присваивание
Тернарная операция

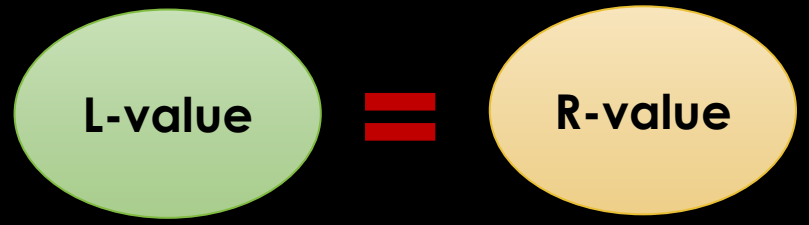
# ОПЕРАЦИЯ ПРИСВАИВАНИЯ

Присваивание

Составное присваивание

# ОПЕРАЦИЯ ПРИСВАИВАНИЯ

```
int a = 1, b = 2, c = 3;
int d = a + b + c;
```




- **Значение как «l-value»** – объект, существующий за пределами одного выражения
  - Значение l-value можно представить как объект с именем
  - В большинстве императивных ЯП все переменные (включая константы) являются значениями l-value
- **Значение как «r-value»** – временное значение, которое не сохраняется за пределами выражения, в котором оно получено

# ПРИМЕР НА L-VALUE И R-VALUE

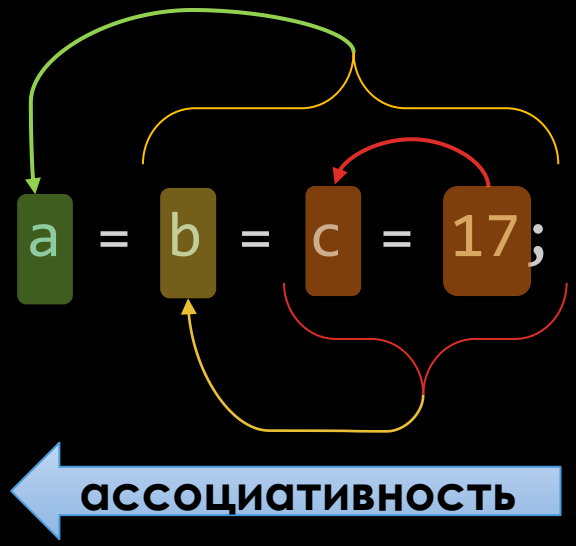
```
public class Program
{
    public static void Main(string[] args)
    {
        int x = 3 + 4;
        Console.WriteLine(x);
    }
}
```

Это выражение

Выражение **3 + 4 со значением равным 7 – это r-value**, т.к. оно возвращает временное значение, которое не сохраняется за пределами выражения, в котором оно определено

**x – это l-value**, т.к. оно продолжает существовать за пределами выражения, в котором определено

# АССОЦИАТИВНОСТЬ ОПЕРАЦИИ ПРИСВАИВАНИЯ




```
a = (b = (c = 17));
```

- **Левый операнд** получает **значение правого**
- Для **типов значений** копируется содержимое правого операнда – **назначение значения**
- Для **ссылочных типов** копируется ссылка на объект – **назначение ссылки**

# СОСТАВНАЯ ОПЕРАЦИЯ ПРИСВАИВАНИЯ

**X op= Y**, где op = { +, -, *, /, %, <<, >>, &, ^, | }

**X = X op Y** – эквивалентное выражение

```
int a = 5, b = 6;
a += b;
```

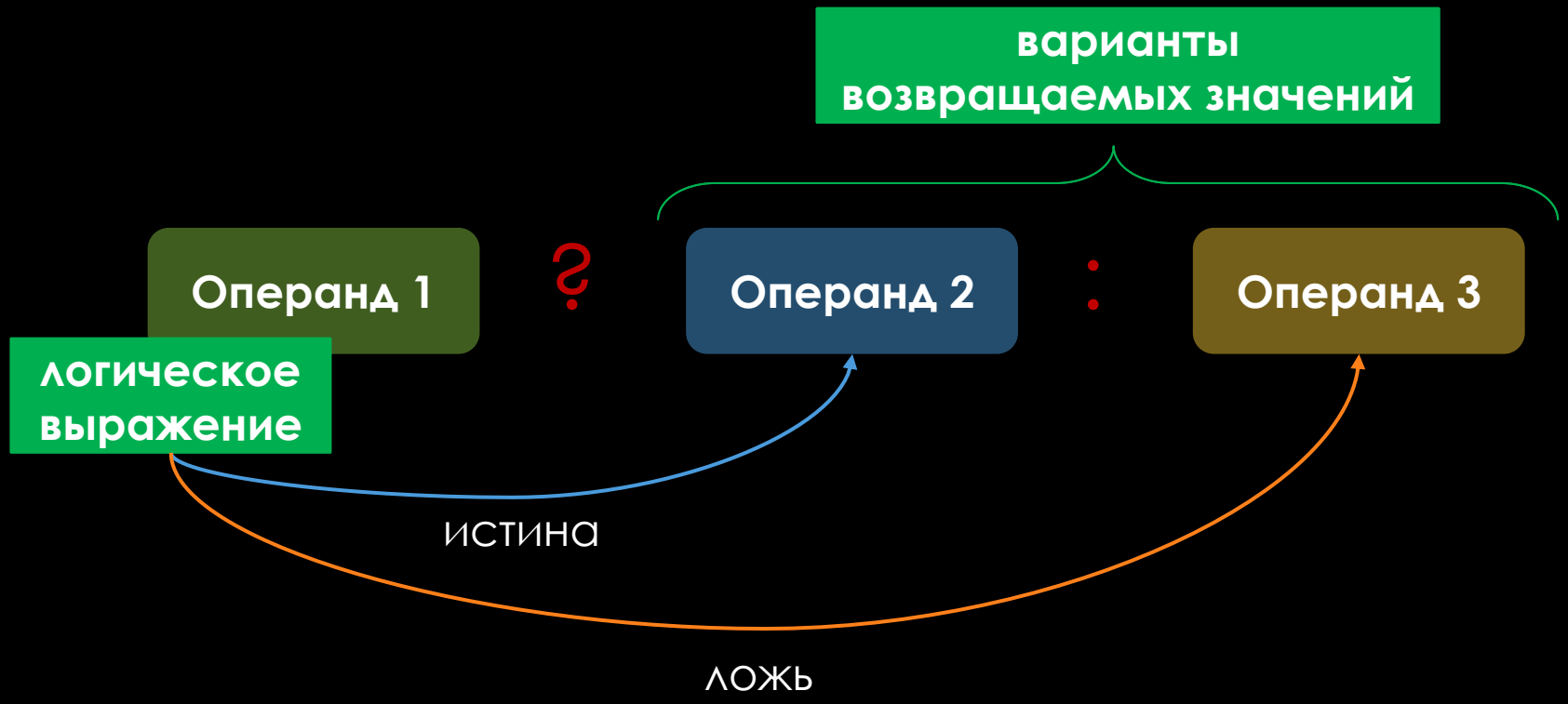
```
double mean = 0.0;

int a = int.Parse(Console.ReadLine());
mean += a;
int b = int.Parse(Console.ReadLine());
mean += b;
int c = int.Parse(Console.ReadLine());
mean += c;

mean /= 3;
Console.WriteLine($"Mean = {mean}");
```

# ТЕРНАРНАЯ ОПЕРАЦИЯ

# ТЕРНАРНАЯ УСЛОВНАЯ ОПЕРАЦИЯ



# ТЕРНАРНАЯ УСЛОВНАЯ ОПЕРАЦИЯ

Два одинаковых результата, но важно, что if – структурный оператор, а тернарная операция – нет

```
int x = 3, y = 4;
int intVar;
```

```
if (x < y)
{
    intVar = 5;
}
else
{
    intVar = 10;
}
```

```
intVar = x < y ? 5 : 10;
```

# ПРОВЕРЯЕМ СЕБЯ

# ОТВЕЧАЕМ НА ВОПРОСЫ

Что будет выведено на экран?

```
bool a = true, b = false;
byte x = 5, y = 7;
Console.Write(a & b);
Console.Write(x & y);
```

Вычислить значение выражения a + x и определить тип результата для следующих фрагментов кода:

```
// snipet 1
double a = 1.3;
int x = 3;
```

```
// snipet 2
byte a = 110;
sbyte x = 11;
```

```
// snipet 3
byte a = 110;
float x = 11.2f;
```

# ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА


- Арифметические операторы (справочник по C#) [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/arithmetic-operators]
- Логические операторы — AND, OR, NOT, XOR [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/boolean-logical-operators]
- Побитовые операторы и операторы сдвига (справочник по C#) [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators]
- Операторы равенства — проверьте, равны ли два объекта [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/equality-operators]
- Операторы сравнения (справочник по C#) [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/comparison-operators]
- Операторы присваивания [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/assignment-operator]
- Приоритет операторов C# [http://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/#operator-precedence]