В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

03

Выражения и Операции

Благодарность Д.Д. Сагалову за помощь в переработке презентаций

Основные Определения

Выражение – это правило для получения значения.
Выражение конструируется из операндов и операций.

Операнд – аргумент операции.

Операция – специальный способ записи некоторого действия.

Литерал – запись в исходном коде, представляющая фиксированное значение определённого типа.

Операнды Выражений

- Литералы;
- Именованные константы;
- Переменные;
- Вызовы методов;
- Аксессоры элементов массивов;
- Выражения, заключенные в круглые скобки;
- Другие выражения.

Синтаксис Простой Операции Присваивания

операнд = выражение;

В качестве операндов слева могут выступать:

- Переменные;
- Свойства;
- Индексаторы;
- События.

```
int g, s;
g = 5;
s = g * 5;
```

Примеры Выражений

«Сложное» выражение:

```
int a = 1, b = 2, c = 3;
int d = a + b + c;          // Равносильно int d = (a + b) + c;
```

Выражения с побочным эффектом:

```
int x = 1;
// Переменная z станет равна 5.
// Операция x++ возвращает копию x до изменений.
int z = x++ * 5;
```

Литералы – Константы Определённых Типов

```
using System;

Console.WriteLine("Тип int: \t" + 256);
Console.WriteLine("Тип double: \t" + 2.718282);
Console.WriteLine("Тип float: \t" + 2.718282F);
Console.WriteLine("Тип bool: \t" + false);
Console.WriteLine("Тип character: \t'" + 'G' + '\'');
Console.WriteLine("Тип string: \t\"" + "Строка!\"");
```

Результат выполнения:

```
Тип int:        256
Тип double:     2,718282
Тип float:      2,718282
Тип bool:       False
Тип character:  'G'
Тип string:     "Строка!"
```

Целочисленные Литералы

Примеры целочисленных <u>десятичных</u> литералов:

```
int num1 = 256;         // тип int – 32 бита.
long num2 = 256L;       // тип long – 64 бита.
uint num3 = 256U;       // тип uint – 32 бита (беззнаковый).
ulong num4 = 256UL;     // тип ulong – 64 бита (беззнаковый).
```

Форматы записи целочисленных литералов:

Decimal Digits
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Hex Digits
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
a, b, c, d, e, f, A, B, C, D, E, F }

Шестнадцатеричные и Двоичные Литералы

Примеры целочисленных шестнадцатеричных литералов:

```
int num1 = 0x2856;    // num1 равна 10326 в 10-чиной СС.
long num2 = 0xA48FL;  // num2 равна 42127 и имеет тип long.
ulong num3 = 0xECul;  // num3 равна 236 и имеет тип ulong.
```

Примеры целочисленных двоичных литералов:

```
int num1 = 0b000101010;    // num1 равна 42 в 10-чиной СС.
// Символ "_" используется для улучшения читаемости.
long num2 = 0b_1111_1111L; // num2 равна 255 и имеет тип long.
```

Целочисленные Литералы без Суффиксов

Язык IL требует, чтобы целочисленные литералы занимали в памяти или 32 или 64 бита:

- 236 – тип **int,** 32 разряда;
- 4000000000 – тип **uint,** 32 разряда;
- 5000000000 – тип **long**, 64 разряда;
- 9900000000000000000 – тип **ulong**, 64 разряда.

Суффиксы Целочисленных Литералов

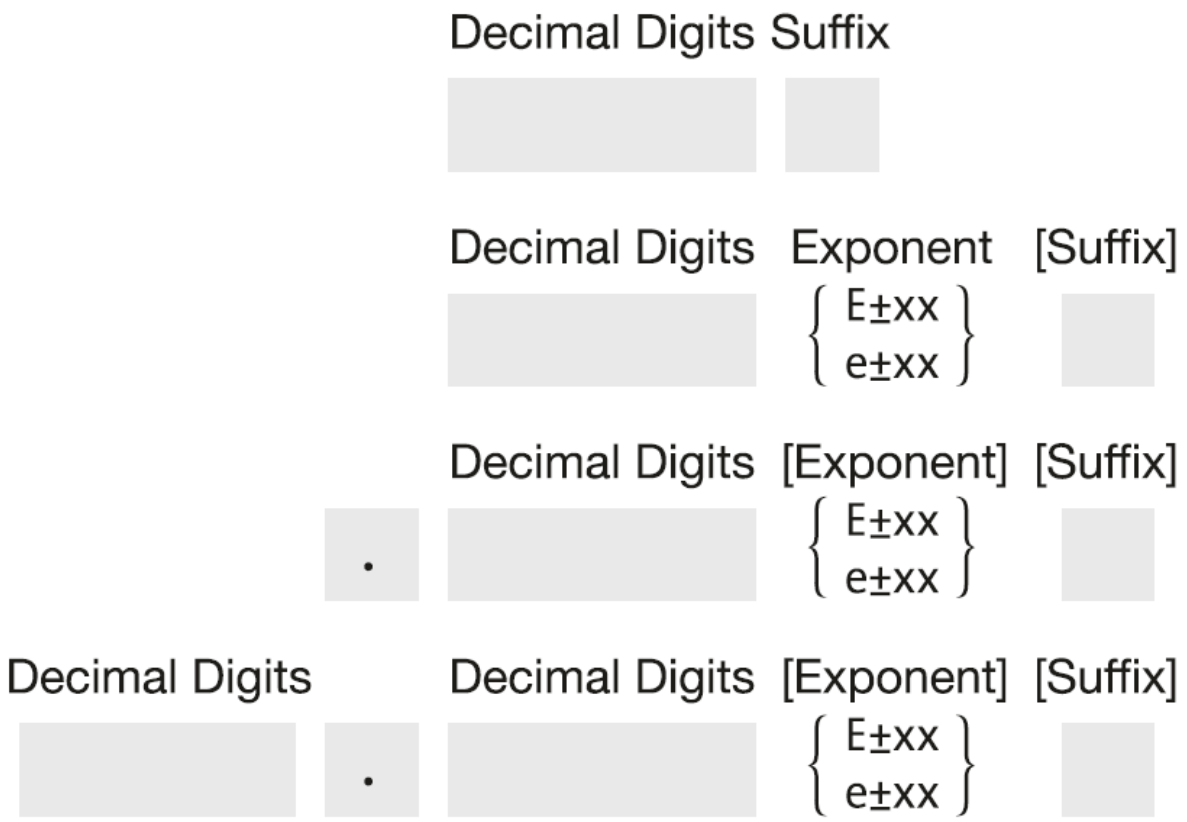
Суффикс	Тип, выбранный по длине записи			
Отсутствует	int	uint	long	ulong
L, l			long	ulong
U, u		uint		ulong
UL, Ul, uL, ul LU, lU, Lu, lu				ulong

Вещественные Литералы

Примеры целочисленных литералов:

```
float f1 = 765F;              // Нет дробной части и порядка.
double d1 = 456.789;          // Нет порядка.
double d2 = .12345;           // Нет целой части и порядка.
double d3 = 5.884e-20;        // Экспоненциальная запись.
decimal dc1 = 250.545m;       // Нет порядка.
decimal dc2 = .0012m;         // Нет целой части и порядка.
```

Форматы Вещественных Литералов



Decimal Digits
{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

The exponent part consists of
- An E, either upper- or lowercase
- An optional sign
- Decimal digits

Суффиксы Вещественных Литералов

Суффикс	Тип		
Отсутствует		double	
F, f	float		
D, d		double	
M, m			decimal

Примеры десятичных литералов:
12345678998765432 1m
0.44F
5.67e-12d

Вывод Значений Литералов Вещественных Типов

```
using System;

Console.WriteLine("0.12345678912345678F: \t" + 0.12345678912345678F);
Console.WriteLine("0.123456789123456789: \t" + 0.123456789123456789);
Console.WriteLine("0.123456789123456789m: \t" + 0.123456789123456789m);
Console.WriteLine("0.123456789123456789E+23: \t" + 0.123456789123456789E+23);
Console.WriteLine("0.123456789123456789E+23m: \t" + 0.123456789123456789E+23m);
Console.WriteLine("0.123456789123456789E+230: \t" + 0.123456789123456789E+230);
```

Результаты вывода:

0,1234568
0,123456789123457
0,123456789123456789
1,2345678912345678E+22
12345678912345678900000
1,2345678912345678E+229

Символьные Литералы

Примеры символьных литералов:

```
char c1 = 'd';          // Один символ.
char c2 = '\n';         // простая escape-последовательность.
char c3 = '\x61';       // 16-чная escape-последовательность.
char c4 = '\u005a';     // escape-последовательность юникода.
```

\x принимает 1-4 символов в 16-ричном представлении:

\x hex-digit hex-digit$_{opt}$ hex-digit$_{opt}$ hex-digit$_{opt}$

\u обязательно требует ровно 4 16-чных символа:

```
string good = "Tab\x9Good compiler";    // Вывод: Tab    Good compiler

string bad = "Tab\x9Bad compiler";      // Вывод: Tab? compiler
```

Escape-последовательности

Escape-последовательности – специальные сочетания символов, начинающиеся с \. Используются для представления специальных символов. В таблице ниже представлен список Escape-последовательностей:

Последова-тельность	Символ	Последова-тельность	Символ
\'	Одинарная кавычка	**\"**	Двойная кавычка
\n	Новая строка	**\r**	Возврат каретки
****	Обратный слеш	**\0**	Null
\t	Горизонтальная табуляция	**\v**	Вертикальная табуляция
\a	Звуковое предупреждение	**\b**	Backspace
\u	Escape-последовательность из ровно 4 шестнадцатеричных символов в кодировке UTF-16	**\U**	Escape-последовательность из ровно 8 шестнадцатеричных символов в кодировке UTF-32
\x	Аналог \u, однако может содержать 1-4 шестнадцатеричных символа	**\f**	Перевод страницы

Регулярные и Буквальные Строковые Литералы

```
using System;


Console.WriteLine("Pro Memoria - \n\t Для памяти");
Console.WriteLine(@"Pro Memoria - \n\t Для памяти");
Console.WriteLine("Есть сигнал!\u0007");
Console.WriteLine(@"Нет сигнала!\u0007");
```

Результаты вывода:

Pro Memoria -
        Для памяти
Pro Memoria - \n\t Для памяти
Есть сигнал!
Нет сигнала!\u0007

На заметку: буквальные строковые литералы удобно использовать для записи путей к файлам.

Вывод Строковых Литералов

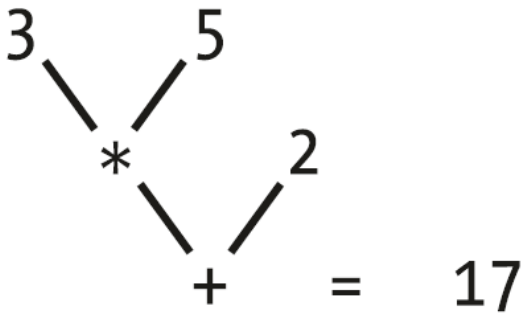
```
using System;

Console.WriteLine("It started, \"Four score and seven...\"");
Console.WriteLine(@"It started, ""Four score and seven...""");

Console.WriteLine("Value 1 \t 5, Val2 \t 10"); // Interprets tab esc sequence
Console.WriteLine(@"Value 1 \t 5, Val2 \t 10"); // Does not interpret tab

Console.WriteLine("C:\\Program Files\\Microsoft\\");
Console.WriteLine(@"C:\Program Files\Microsoft\");
```

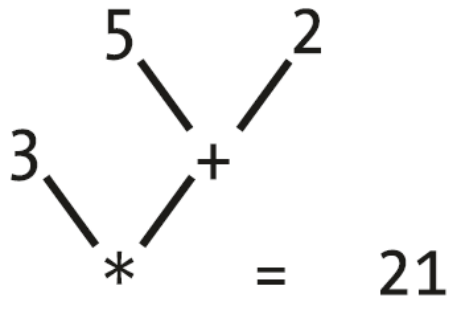
Результаты вывода:

```
It started, "Four score and seven..."
It started, "Four score and seven... "

Value 1         5, Val2         10
Value 1 \t 5, Val2 \t 10

C:\Program Files\Microsoft\
C:\Program Files\Microsoft\
```

Порядок Проведения Вычислений

Два варианта вычисления выражения 2+5*3:





Возникает необходимость в расстановке **приоритетов** и определении **ассоциативности** операций!

Приоритет Операций (Сверху Вниз, Слева Направо)

Категория или Имя	Операции	Категория или Имя	Операции
Первичные	x.y f(x) a[i] x?.y x?[y] x++ x-- x! new typeof checked unchecked default nameof delegate sizeof stackalloc x->y	Побитовое И	x & y
Унарные	+x -x !x ~x ++x --x ^x (T)x await &x *x *(указатели)* true false	Побитовое Исключающее И	x ^ y
Диапазон	x..y	Побитовое ИЛИ	x | y
Выражение switch	switch	Условное И	x && y
Мультипликативные	x * y x / y x % y	Условное ИЛИ	x || y
Аддитивные	x + y x - y	Операция Объединения с null	x ?? y
Операции Сдвига	x << y x >> y	Тернарная Условная Операция	cond ? t : f
Тестирование Типов	x < y x <= y x > y x >= y is as	Присваивание и Объявление Лямбда-Выражений	= += -= *= /= %= &= |= ^= <<= >>= ??= => *(лямбда-операция)*
Равенство	x == y x != y		

https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/

Associativity - Ассоциативность

Вопрос: Какова последовательность вычислений 2/6*4:

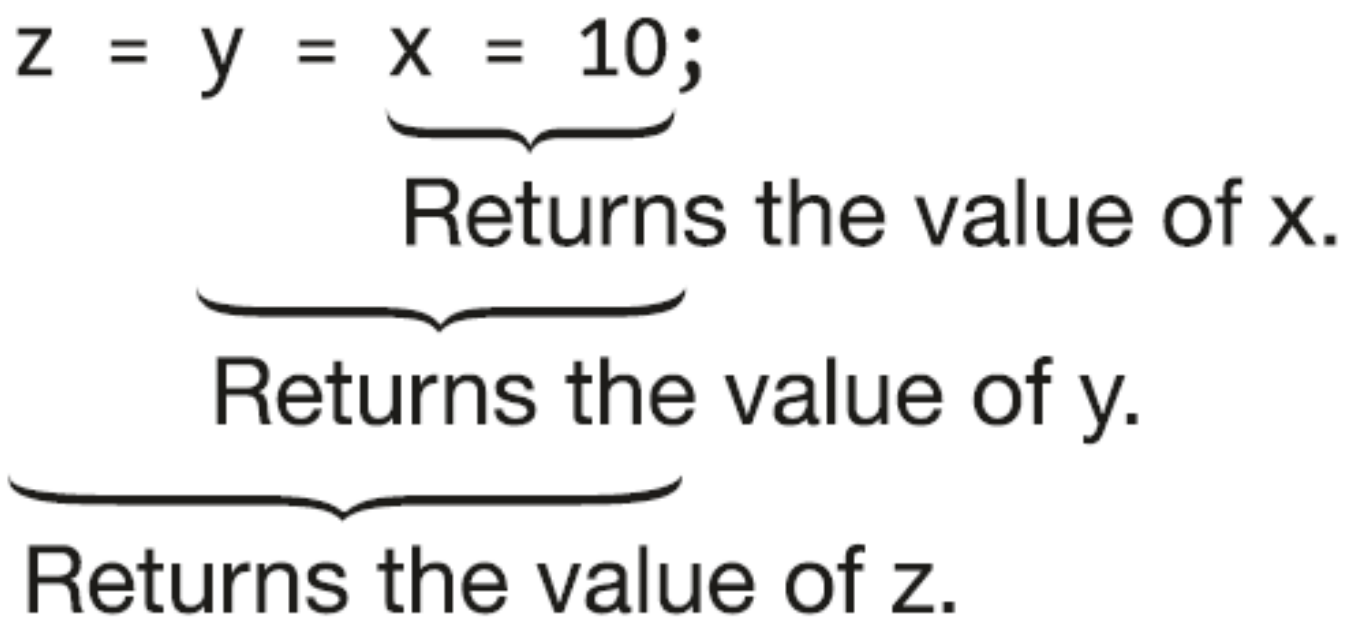
(2 / 6) * 4 ==> 4/3

или

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

Типы Операций	Ассоциативность
Операции присваивания	Правоассоциативная
Тернарная Условная операция	Правоассоциативная
Другие Бинарные Операции	Левоассоциативная

Правоассоциативность Операции Присваивания



Простые (Бинарные) Арифметические Операции

Операция	Наименование	Описание
+	Сложение	Складывает операнды.
-	Вычитание	Вычитает правый операнд из левого
/	Целая часть от деления	Целочисленное деление левого операнда на правый.
%	Остаток от деления	Остаток от деление левого операнда на правый.
*	Умножение	Умножает операнды.

Примеры Простых Арифметических Операций

```
int x1 = 5 + 6;
int x2 = 12 - 3;
int x3 = 3 * 4;
int x4 = 10 / 3;

byte b1 = 5 + 6;
sbyte sb1 = 6 * 5;
```

```
double d1 = 5.0 + 6.0;
double d2 = 12.0 - 3.0;
double d3 = 3.0 * 4.0;
double d4 = 10.0 / 3.0;

// Вычисляется на этапе компиляции.
```

Операция Остатка от Деления (Для Целочисленных Операндов)

0 % 3 = 0, т.к. 0 / 3 == 0 с остатком == 0.
1 % 3 = 1, т.к. 1 / 3 == 0 с остатком == 1.
2 % 3 = 2, т.к. 2 / 3 == 0 с остатком == 2.
3 % 3 = 0, т.к. 3 / 3 == 1 с остатком == 0.
4 % 3 = 1, т.к. 4 / 3 == 1 с остатком == 1.

(x / y *y + x % y) равно **x (для целых)**

- -5 % 2 равно -1
- 5 % -2 равно 1
- -5 % -2 равно -1

Операция Остатка от Деления (Для Вещественных Операндов)

```
0.0f % 1.5f   ==  0      // 0.0 / 1.5 = 0, остаток == 0
0.5f % 1.5f   ==  0.5    // 0.5 / 1.5 = 0, остаток == 0.5
1.0f % 1.5f   ==  1      // 1.0 / 1.5 = 0 ,остаток == 1
1.5f % 1.5f   ==  0      // 1.5 / 1.5 = 1, остаток == 0
2.0f % 1.5f   ==  0.5    // 2.0 / 1.5 = 1, остаток == 0.5
2.5f % 1.5f   ==  1      // 2.5 / 1.5 = 1, остаток == 1
```

- -0.5f % 1.5f равно -0,5
- 0.5f % -1.5f равно 0,5
- -0.5f % -1.5f равно -0,5

Унарные Арифметические Операции

Operator	Name	Description
+	Positive sign	Returns the numeric value of the operand
-	Negative sign	Returns the numeric value of the operand subtracted from 0

Примеры использования и результаты операций:

```
int x = +10;      // x = 10
int y = -x;       // y = -10
int z = -y;       // z = 10
```

Операции Инкремента и Декремента

Operator	Name	Description
++	Pre-increment ++var	Increment the value of the variable by 1 and save it back into the variable. Return the new value of the variable.
	Post-increment var++	Increment the value of the variable by 1 and save it back into the variable. Return the old value of the variable before it was incremented.
--	Pre-decrement --var	Decrement the value of the variable by 1 and save it back into the variable. Return the new value of the variable.
	Post-decrement var--	Decrement the value of the variable by 1 and save it back into the variable. Return the old value of the variable before it was decremented.

Поведение Префиксных и Постфиксных Форм Операций Инкремента и Декремента

	Starting Expression: x = 10	Value Returned to the Expression	Value of Variable After Evaluation
Pre-increment	++x	11	11
Post-increment	x++	10	11
Pre-decrement	--x	9	9
Post-decrement	x--	10	9

```
int a = 2;
a++;                  // эквивалентно x = x + 1;


int x = 5, z, y;
z = x++;              // z равно 5, x равно 6.
y = ++x;              // y равно 7, x равно 7.
```

Операции Сравнения и Отношений

Operator	Name	Description
<	Less than	`true` if the first operand is less than the second operand; `false` otherwise
>	Greater than	`true` if the first operand is greater than the second operand; `false` otherwise
<=	Less than or equal to	`true` if the first operand is less than or equal to the second operand; `false` otherwise
>=	Greater than or equal to	`true` if the first operand is greater than or equal to the second operand; `false` otherwise
==	Equal to	`true` if the first operand is equal to the second operand; `false` otherwise
!=	Not equal to	`true` if the first operand is not equal to the second operand; `false` otherwise

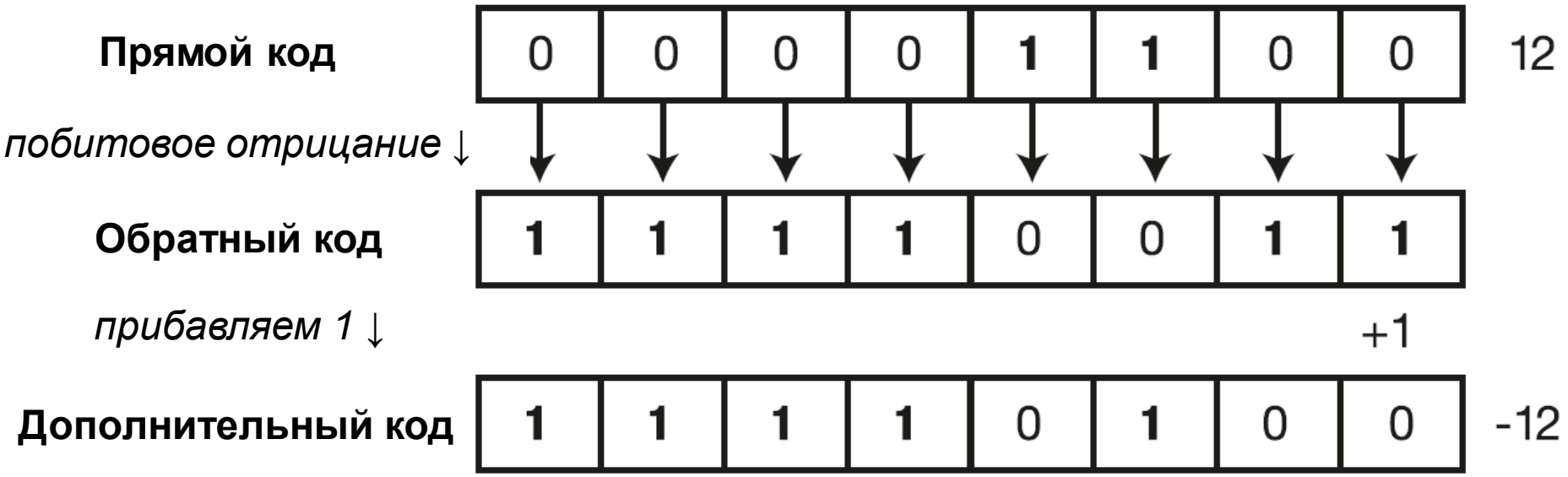
Операции Проверки на Равенство

Важно: В отличие от **C** и **C++**, числа в **C#** не могут быть неявно приведены к типу bool.

```
int x = 5;
if (x) {}           // Ошибка компиляции: x не приводится к типу bool неявно.
if (x == 5) {}      // OK: операция == возвращает значение типа bool.

int x = 5, y = 4;
Console.WriteLine($"x == x is {x == x}");
Console.WriteLine($"x == y is {x == y}");
```

Результаты вывода:
x == x is True
x == y is False

Дополнительный Код Числа -12



Поразрядные Логические Операции

Operator	Name	Description
&	Bitwise AND	Produces the bitwise AND of the two operands. The resulting bit is 1 only if both operand bits are 1.
|	Bitwise OR	Produces the bitwise OR of the two operands. The resulting bit is 1 if either corresponding operand bit is 1.
^	Bitwise XOR	Produces the bitwise XOR of the two operands. The resulting bit is 1 if one, but not both, of the corresponding operand bits is 1.
~	Bitwise negation	Each bit in the operand is switched to its opposite. This produces the one's complement of the operand. (The *one's complement* of a number is the inversion of every bit of its binary representation. That is, every 0 is switched to 1, and every 1 is switched to 0.)

Поразрядные Логические Операции

Операция	Наименование
~	Отрицание (Побитовое НЕ)
&	Конъюнкция (Побитовое И)
|	Дизъюнкция (Побитовое ИЛИ)
^	Исключающее ИЛИ

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

Результат: 0
```

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

Проверить принадлежность X отрезку [E;F]:

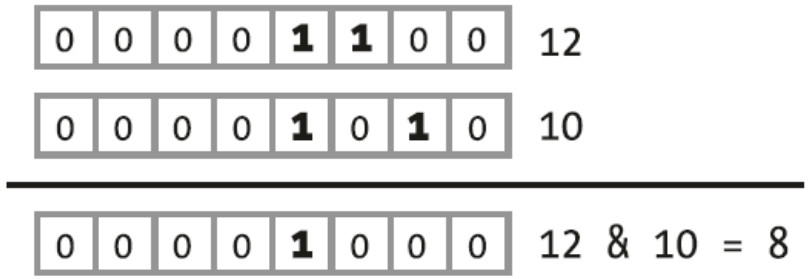
```
bool factor = X >= E & X <= F;
```

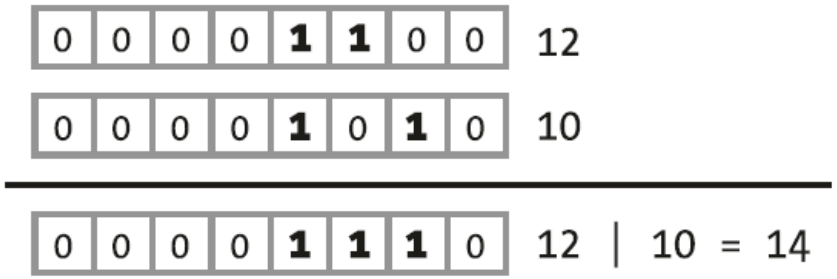
Проверить принадлежность точки (x, y) кругу с центром в (0,0) и радиусом r:

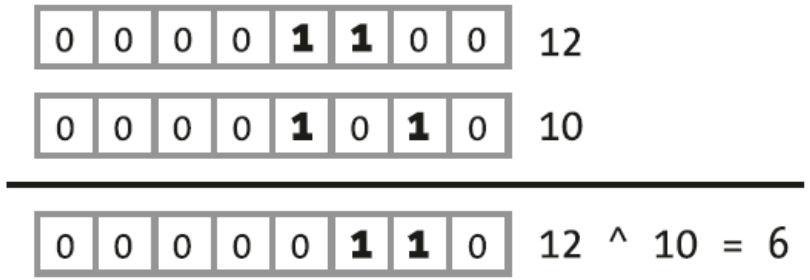
```
bool inside = x * x + y * y < r * r;
```

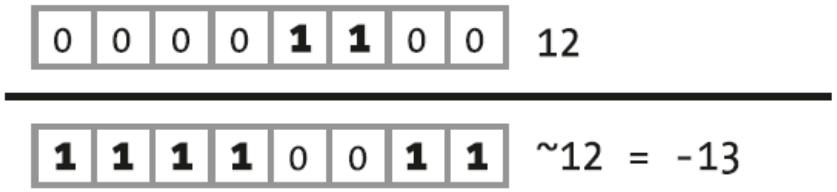
Поразрядные Логические Операции

```
const byte x = 12, y = 10; // Константы этапа компиляции.
sbyte a;
a = x & y;    // Побитовое И: a = 8
a = x | y;    // Побитовое ИЛИ: a = 14
a = x ^ y;    // Побитовое Исключающее ИЛИ: a = 6
a = ~x;       // Побитовое Дополнение: a = -13
```









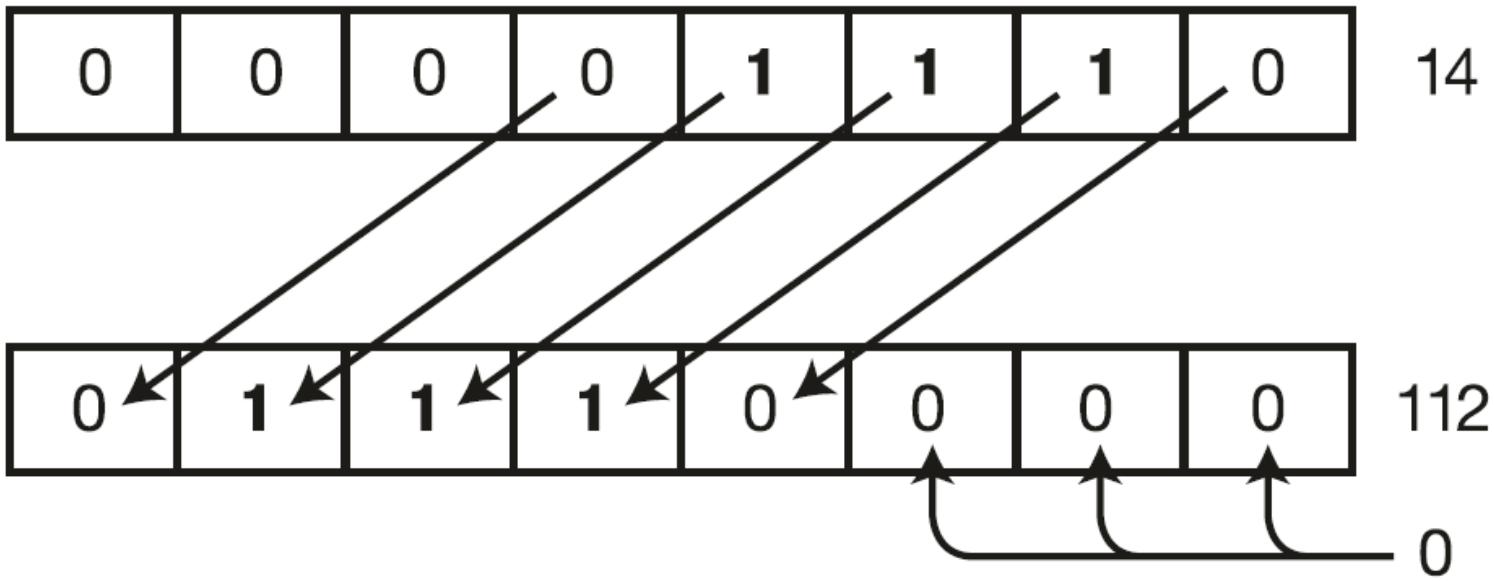
Операции Побитового Сдвига

Operator	Name	Description
<<	Left shift	Shifts the bit pattern left by the given number of positions. The bits shifted off the left end are lost. Bit positions opening up on the right are filled with 0s.
>>	Right shift	Shifts the bit pattern right by the given number of positions. Bits shifted off the right end are lost.

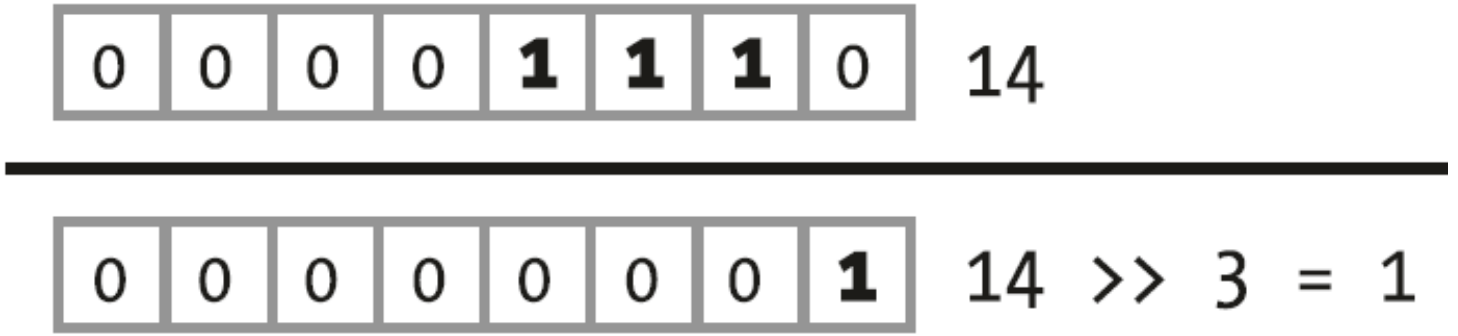
Синтаксис:

```
Operand << Count        // Сдвиг влево.
Operand >> Count        // Сдвиг вправо.
```

Операции Побитового Сдвига – Иллюстрация



14 << 3 = 112



Условные Логические Операции

Operator	Name	Description
&&	Logical AND	`true` if both operands are `true`; `false` otherwise
||	Logical OR	`true` if at least one operand is `true`; `false` otherwise
!	Logical NOT	`true` if the operand is false; `false` otherwise

Expr1 && Expr2
Expr1 || Expr2
! *Expr*

Объяснение действия “short circuit”:

x && y вычисляется (bool)x ? (bool)y : false

x || y эквивалентно (bool)x ? true : (bool)y

Условные Логические Операции

Операция	Наименование
!	Отрицание (Логическое НЕ)
&&	Конъюнкция (Логическое И)
||	Дизъюнкция (Логическое ИЛИ)
^	Исключающее ИЛИ

Важно: в отличие от побитовых И/ИЛИ, логигические И/ИЛИ работают по короткой схеме.

Условные Логические Операции

Пример 1:

```
bool bVal;
int iVal = 10;                          // Короткая схема – так как 1 == 2 == false
bVal = (1 == 2) && (9 == iVal++);       // правая часть после && не вычисляется.
System.Console.WriteLine(iVal);         // Вывод: 10.
```

Пример 2:

```
double x = 5, y = 8, z = 2;
// y < x + z == false, часть z < x + y не вычисляется!
bool sign = x < y + z && y < x + z && z < x + y;
```

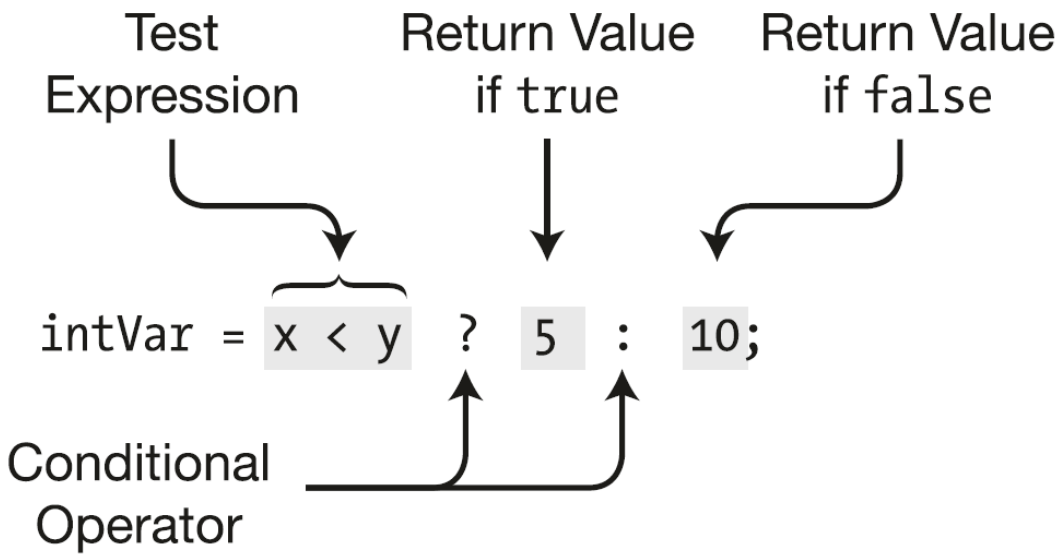
Тернарная Условная Операция

Operator	Name	Description
?:	Conditional operator	Evaluates an expression and returns one of two values, depending on whether the expression returns `true` or `false`

Синтаксис условной тернарной операции:
Условие ? *Выражение_1* : *Выражение_2*

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

Два одинаковых результата:



Логические и Условная (Тернарная) Операции

Пример:

```
using System;

double a = 3, b = 4, c = 5;
bool sign = a < b + c & b < a + c & c < a + b;
string report = sign ? "Это длины сторон треугольника!" :
  "Треугольник построить нельзя!";
Console.WriteLine(report);
```

Результаты вывода:
Это длины сторон треугольника!

Операции Присваивания

Operator	Description
=	Simple assignment; evaluate the expression on the right, and assign the returned value to the variable or expression on the left.
*=	Compound assignment; var *= expr is equal to var = var * (expr).
/=	Compound assignment; var /= expr is equal to var = var / (expr).
%=	Compound assignment; var %= expr is equal to var= var % (expr).
+=	Compound assignment; var += expr is equal to var = var + (expr).
-=	Compound assignment; var -= expr is equal to var = var- (expr).
<<=	Compound assignment; var <<= expr is equal to var = var << (expr).
>>=	Compound assignment; var >>= expr is equal to var = var >> (expr).
&=	Compound assignment; var &= expr is equal to var = var & (expr).
^=	Compound assignment; var ^= expr is equal to var = var ^ (expr).
|=	Compound assignment; var |= expr is equal to var = var | (expr).

Составные Операции Присваивания

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

```
x = x + expr;       // эквивалентно   x += expr;
x = x + (y – z);    // эквивалентно   x += y – z;

x *= y – z;         // эквивалентно   x = x * (y – z)
x /= y – z;         // эквивалентно   x = x / (y – z)
```

Составные Операции Присваивания

Эквивалентные выражения:

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

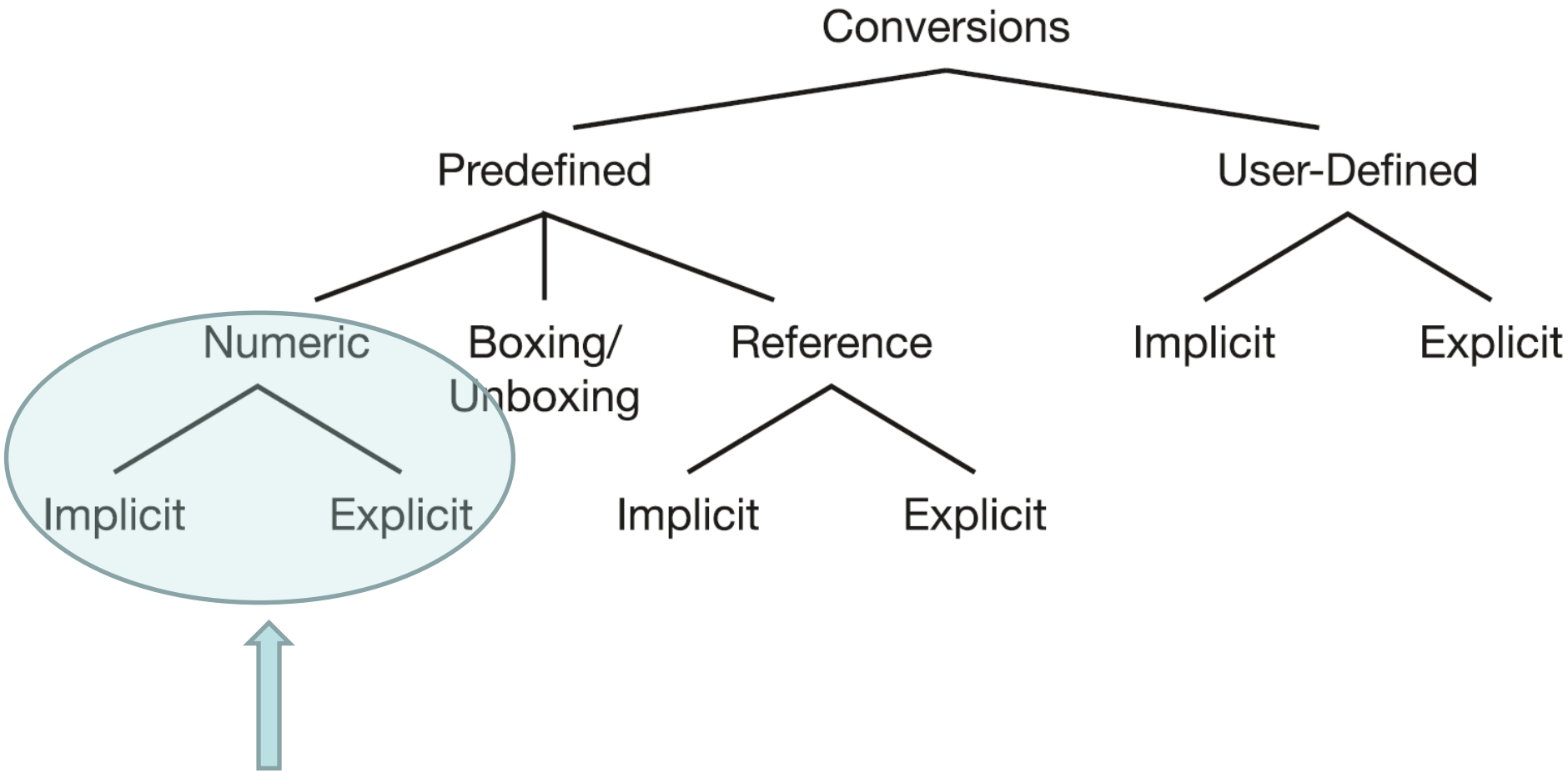
Не эквивалентные выражения:

x++ x += 1 *(x++ возвращает копию x)*

Иллюстрация:

```
using System;
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
```

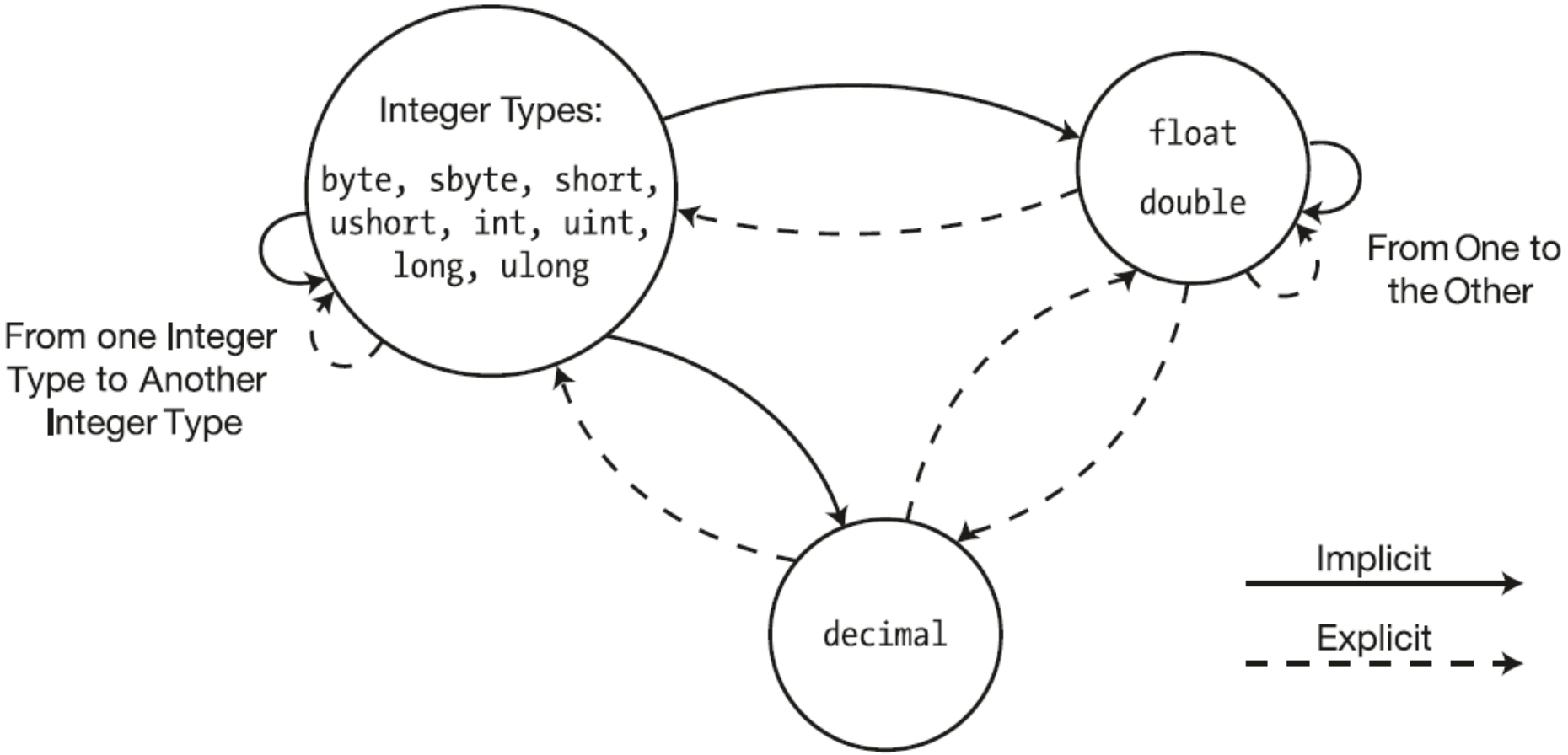
Виды преобразования типов



То, что мы рассмотрим сейчас

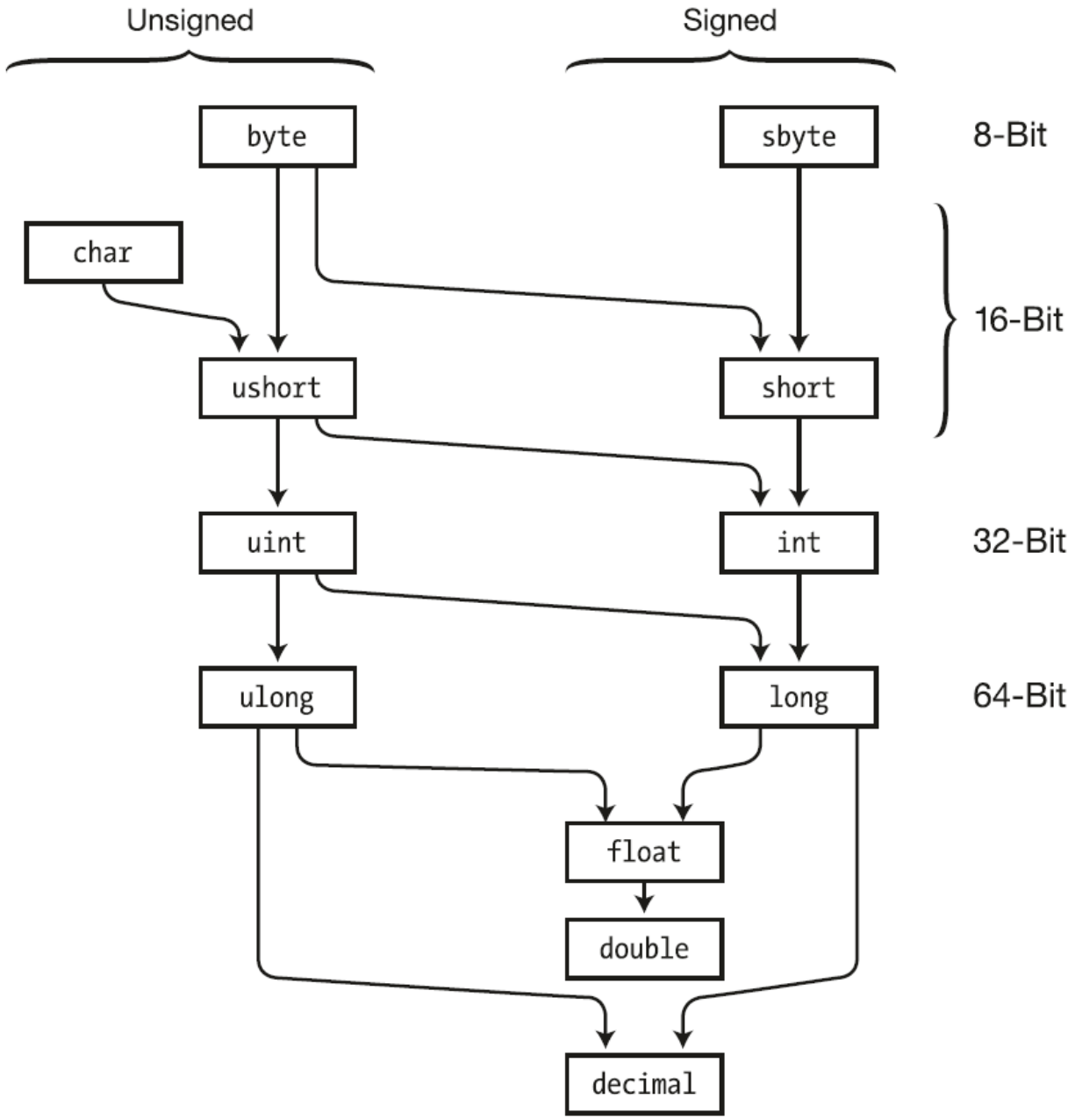
Не путайте: **implicit – неявное, explicit – явное!**

Схема Преобразования Числовых Типов



Не путайте: **implicit – неявное, explicit – явное!**

Неявные преобразования числовых типов



Преобразования Типов

Явное преобразование с помощью операции приведения типов:

Тип имяПеременной = (Тип)24;

Пример 1:

```
sbyte f = 14;                  // 00001110
sbyte r = (sbyte)(f << 3); // 01110000
sbyte t = (sbyte)(f << 4); // 11100000
```

Пример 2:

```
byte n = 2, m = 3;
byte h = m + n;                // Ошибка компиляции: m + n имеет тип int.
byte h = (byte)(m + n);    // OK – явное приведение к типу byte.
```

Операция typeof

Синтаксис операции:

Type t = **typeof**(SomeType);

System.Type – это класс в пространстве имен System.

Пример нескольких свойства класса Type:

- **Name** – имя типа;
- **BaseType** – базовый тип;
- **Assembly** – сборка;
- **IsPublic** – доступность.

Метод GetType()

```
using System;
class SomeClass {  }
class Program
{
    static void Main()
    {
        SomeClass s = new SomeClass();
        Console.WriteLine("Type s: {0}", s.GetType().Name);
    }
}
```

Вывод:
Type s: SomeClass