

Лекция 10

**Операции арифметические,
сравнения, логические и
поразрядные**

Операции

Операции часто разделяются на четыре основные группы: *арифметические* (arithmetic), *сравнения* (relational), *логические* (logical) и *побитовые* (bitwise).

Операнд - это константа, литерал, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное комбинацией операндов, знаков операций и круглых скобок. Каждый операнд имеет тип.

Таблица Операции C#

Категория	Знак операции	Название
Первичные	x()	Вызов метода или делегата
	x[]	Доступ к элементу
	x++	Постфиксный инкремент
	x--	Постфиксный декремент
	new	Выделение памяти
	&	Адресация (взятие адреса)
	*	Разадресация (разыменование)
	typeof	Получение типа
	checked	Проверяемый код
	unchecked	Непроверяемый код

Таблица Операции C#

Категория	Знак операции	Название
Унарные	+	Унарный плюс
	-	Унарный минус (арифметическое отрицание)
	!	Логическое отрицание
	~	Поразрядное отрицание
	++x	Префиксный инкремент
	--x	Префиксный декремент
	(тип)x	Преобразование типа

Таблица Операции C#

Категория	Знак операции	Название
Мультипликативные (типа умножения)	*	Умножение
	/	Деление
	%	Остаток от деления
Аддитивные (типа сложения)	+	Сложение
	-	Вычитание
Сдвига	<<	Сдвиг влево
	>>	Сдвиг вправо
Отношения и проверки типа	<	Меньше
	>	Больше
	<=	Меньше или равно
	>=	Больше или равно
	is	Проверка принадлежности типу
	as	Приведение типа
Проверки на	==	Равно
	!=	Не равно

Таблица Операции C#

Категория	Знак операции	Название
Поразрядные логические	&	Поразрядная конъюнкция (И)
	^	Поразрядное исключающее ИЛИ
	 	Поразрядная дизъюнкция (ИЛИ)
Условные логические	&&	Логическое И
	 	Логическое ИЛИ
Условная	?:	Условная операция
Присваивания	=	Присваивание
	*=	Умножение с присваиванием
	/=	Деление с присваиванием
	%=	Остаток от деления с присваиванием

Таблица Операции C#

Категория	Знак операции	Название
Присваивания	+=	Сложение с присваиванием
	-=	Вычитание с присваиванием
	<<=	Сдвиг влево с присваиванием
	>>=	Сдвиг вправо с присваиванием
	&=	Поразрядное И с присваиванием
	^=	Поразрядное исключающее ИЛИ с присваиванием
	=	Поразрядное ИЛИ с присваиванием

Арифметические операции

Унарные операции

Арифметическое отрицание (унарный минус $-$) меняет знак операнда на противоположный.

Стандартная операция отрицания определена для типов **int**, **long**, **float**, **double** и **decimal**. К величинам других типов ее можно применять, если для них возможно неявное преобразование к этим типам.

```
double u = 5;
```

```
u = -u;    /* переменной u присваивается ее
```

```
отрицание, т.е. u принимает значение -5 */
```

Унарный плюс - по умолчанию: $u = +u$;

Арифметические операции

Унарные операции

Арифметическое отрицание (унарный минус $-$) меняет знак операнда на противоположный.

Стандартная операция отрицания определена для типов **int**, **long**, **float**, **double** и **decimal**. К величинам других типов ее можно применять, если для них возможно неявное преобразование к этим типам.

```
double u = 5;
```

```
u = -u; /* переменной u присваивается ее
```

```
отрицание, т.е. u принимает значение -5 */
```

Унарный плюс - по умолчанию: $u = +u$;

Инкрементация и декрементация

операция	префиксная форма	постфиксная форма	
$x = x + 1;$	$++x;$	$x++;$	инкремент
$x = x - 1;$	$--x;$	$x--;$	декремент

а) `int t=1, s=2, z, f;`

Операторы: `z = t++ * 5;` `f = ++s/3;`

Результаты: `z = 5, t = 2` `s = 3, f = 1`

б) `int x = 10, z = 1;`

`y = ++x;` // `x = 11` , `y = 11`

`y = x++;` // `y = 11` , `x = 12`

`z++;` /* эквивалентно */ `++z;`

Инкарементация и декрементация

```
int x1 = 5;  
int z1 = ++x1;  
Console.WriteLine($" {x1} - {z1}");
```

```
int x2 = 5;  
int z2 = x2++;  
Console.WriteLine($" {x2} - {z2}");
```

Инкрементация и декрементация

```
int x1 = 5;
```

```
int z1 = --x1;
```

```
Console.WriteLine($" {x1} - {z1}");
```

```
int x2 = 5;
```

```
int z2 = x2--;
```

```
Console.WriteLine($" {x2} - {z2}");
```

Операция new

Операция **new** служит для создания нового объекта: **new** тип ([аргументы])

С помощью этой операции можно создавать объекты как ссылочных, так и значимых типов, например:

```
object z = new object();
```

```
int i = new int(); // то же самое, что int i = 0;
```

При выполнении операции **new** сначала выделяется необходимый объем памяти (для ссылочных типов в хипе, для значимых — в стеке), а затем вызывается так называемый *конструктор по умолчанию*, то есть метод, с помощью которого инициализируется объект.

Переменной *значимого типа* присваивается *значение по умолчанию*, которое равно нулю соответствующего типа.

Операции * и &

Объявить (определить) указатель можно с помощью операции *.

Получить адрес памяти, на который указывает указатель, можно с помощью оператора адресации &.

Ключевой при работе с указателями является операция *, которую еще называют **операцией разыменовывания (разадресации)**. Операция разыменовывания позволяет получить или установить значение по адресу, на который указывает указатель.

Операции * и &

Чтобы использовать небезопасный код в C#, надо первым делом указать проекту, что он будет работать с небезопасным кодом. Для этого надо установить в настройках проекта соответствующий флаг - в меню **Project (Проект)** найти **Свойства проекта**. Затем в меню **Build (Сборка)** установить флажок **Allow unsafe code (Разрешить небезопасный код)**.

Ключевое слово "**unsafe**" отключает систему безопасности так как работа с указателями небезопасна, поэтому для компиляции небезопасного кода необходимо указать параметр компилятора **unsafe**.

Пример 1

```
Console.WriteLine("Адрес переменной y:  
{0}", addr);
```

```
Console.WriteLine(*x); // 10
```

```
y = *x + 20;
```

```
Console.WriteLine(*x); // 30
```

```
*x = 50;
```

```
Console.WriteLine(y); // 50
```

```
} // ----- конец блока unsafe -----
```

```
Console.ReadLine();
```

```
}
```

Арифметические операции

Бинарные операции

Операции сложения, вычитания, умножения, деления определены для типов **int**, **uint**, **long**, **ulong**, **float**, **double** и **decimal**. К величинам других типов их можно применять, если для них существует неявное преобразование к этим типам. Тип результата операций равен "наибольшему" из типов операндов, но не менее **int**.

Арифметические операции

Бинарные операции

Операция сложения (+) возвращает сумму двух операндов.

Если оба операнда целочисленные или типа **decimal** и результат операции слишком велик для представления с помощью заданного типа, генерируется исключение **System.OverflowException**.

```
int x = 10;
```

```
int z = x + 12; // 22
```

Арифметические операции

Бинарные операции

Операция вычитания (-) возвращает разность двух операндов.

Если оба операнда целочисленные или типа **decimal** и результат операции слишком велик для представления с помощью заданного типа, генерируется исключение **System.OverflowException**.

```
int x = 10;
```

```
int z = x - 6; // 4
```

Арифметические операции

Бинарные операции

Операция умножения (*) возвращает результат перемножения двух операндов.

```
int x = 10;
```

```
int z = x * 5; // 50
```

Важно следить, чтобы результат операций не превышал диапазон значений типа переменной, в которую помещается результат.

Арифметические операции

Бинарные операции

Операция деления (/) вычисляет частное от деления первого операнда на второй.

Если оба операнда целочисленные, результат операции округляется вниз до ближайшего целого числа. Если делитель равен нулю, генерируется исключение **System.DivideByZeroException**.

Арифметические операции

Бинарные операции

При делении стоит учитывать, что если оба операнда представляют целые числа, то результат также будет округляться до целого числа:

```
double z = 10 / 4; // результат равен 2
```

Чтобы сохранить дробную часть, следует числитель и/или знаменатель преобразовать к вещественному типу явным или неявным способом:

```
double z = 10.0 / 4.0; // результат равен 2.5
```

```
double z = 10.0 / 4; // результат равен 2.5
```

```
int a = 4, b = 10;
```

```
double z = b / (a*1.0); // результат равен 2.5
```

```
// или
```

```
double z = (double) 10 / 4; // результат равен 2.5
```

Арифметические операции

Бинарные операции

Для финансовых величин (тип **decimal**)
при делении на 0 и переполнении
генерируются соответствующие
исключения, при *исчезновении*
порядка результат равен 0.

```
int x = 10;
```

```
int z = x / 5; // 2
```

```
double a = 10;
```

```
double b = 3;
```

```
double c = x / y; // 3.3333333333
```


Арифметические операции

Бинарные операции

Операция остатка от деления (%) :

- если оба операнда целочисленные, результат операции вычисляется по формуле $x - (x / y) * y$.

Если делитель равен нулю, генерируется исключение **System.DivideByZeroException.**

- если хотя бы один из операндов вещественный, результат операции вычисляется по формуле $x - n * y$, где n — наибольшее целое, меньшее или равное результату деления x на y .

Арифметические операции

Бинарные операции

Операция остатка от деления (%) :

Для финансовых величин (тип **decimal**) при получении остатка от деления на 0 и при переполнении генерируются соответствующие исключения, при *исчезновении порядка* результат равен 0. Знак результата равен знаку первого операнда.

Операция получение остатка от деления двух чисел (%):

```
double x = 10.0;
```

```
double z = x % 4.1; // результат равен 1.8
```

Приоритеты арифметических операций

Высший ++ --

- (унарный минус)

* / %

Низший + -

Операторы, имеющие одинаковый приоритет, выполняются слева направо.

Для изменения порядка следования операций применяются скобки.

Пример 2

```
int a = 3;
```

```
int b = 5;
```

```
int c = 40;
```

```
int d = c-- - b*a; // a=3 b=5 c=39 d=25
```

```
Console.WriteLine($"a={a} b={b} c={c} d={d}");
```

```
int d = c-- - b*a; ↔ int d = (c--)-(b*a);
```

```
int a = 3;
```

```
int b = 5;
```

```
int c = 40;
```

```
int d = (c-(--b))*a; // a=3 b=4 c=40 d=108
```

```
Console.WriteLine($"a={a} b={b} c={c} d={d}");
```

Арифметические операции

Арифметические *операции* не определены для более коротких, чем **int**, типов. Это означает, что если в выражении участвуют только величины типов **sbyte**, **byte**, **short** и **ushort**, перед выполнением *операции* они будут преобразованы в **int**. Таким образом, результат любой арифметической *операции* имеет тип не менее **int**.

```
byte a = 4;
```

```
byte b = a + 70; // ошибка, byte ≠ int
```

И чтобы выйти из этой ситуации, необходимо применить операцию преобразования типов:

```
byte a = 4;
```

```
byte b = (byte)(a + 70); // операция преобразования типов
```

Исключения

При вычислении выражений могут возникнуть ошибки, например, *переполнение*, *исчезновение порядка* или *деление на ноль*. Об ошибках система сигнализирует с помощью специального действия, называемого *выбрасыванием (генерированием) исключения*. Каждому типу ошибки соответствует свое *исключение*. В C# исключения являются классами, которые имеют общего предка — класс **Exception**, определенный в пространстве имен **System**.

Например, при делении на ноль будет сгенерировано *исключение* **DivideByZeroException**, при недостатке памяти — *исключение* **OutOfMemoryException**, при переполнении памяти — *исключение* **OverflowException**.

Исключения

В C# ключевое слово **checked** используется, если требуется указать, что выражение будет проверяться на переполнение, ключевое слово **unchecked** следует использовать, а требуется проигнорировать переполнение.

checked (выражение)

```
checked {  
    // проверяемые операторы  
}
```

unchecked (выражение)

```
unchecked {  
    // операторы, для которых переполнение игнорируется  
}
```

Исключения

В C# есть *механизм обработки исключительных ситуаций (исключений)* - конструкция **try...catch**.

Блок **try** содержит операторы, реализующие действия, в которых может потенциально возникнуть ошибка, а в блоке **catch** обрабатывается ошибка, если она возникла.

Внутри блока **catch** можно, например, вывести *предупреждающее сообщение* или скорректировать значения величин и продолжить выполнение программы. Если этот блок не задан, система выполнит *действия по умолчанию*, которые обычно заключаются в выводе диагностического сообщения и нормальном завершении программы.

Пример 3

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
namespace ConsoleApplication1 {
```

```
    class Program
```

```
    {
```

```
        static void Main()
```

```
        {
```

```
            byte a, b, result;
```

```
            Console.Write("Введите количество опросов: ");
```

```
            int i = int.Parse(Console.ReadLine());
```

Пример 3

```
for (int j = 1; j <= i; j++) { // ЦИКЛ
    try {
        Console.Write("Введите a: ");
        a = unchecked((byte)int.Parse(Console.ReadLine()));
        Console.Write("Введите b: ");
        b = unchecked((byte)int.Parse(Console.ReadLine()));
        checked {
            result = (byte)(a + b);
            Console.WriteLine("a + b = " + result);
            result = (byte)(a * b);
            Console.WriteLine("a*b = " + result + "\n");
        } // конец блока checked
    } // конец блока try
```

Пример 3

```
catch (OverflowException)
{
    Console.Write("Переполнение\n\n");
} // конец блока catch
} // конец цикла
    Console.ReadLine();
} // конец функции-метода Main
} // конец класса Program
} // конец namespace ConsoleApplication1
```

Операции доступа

В языке C# символы "." и "->" обозначают операции доступа к элементам класса. Операция доступа к члену класса (".") используется для непосредственного обращения к элементу класса (свойствам, методам). Операция ссылки на член класса ("->") используется для обращения к члену класса через указатель.

Пример 4

```
class Program {
    static void Main(string[] args) {
        unsafe {
            Person person;
            person.age = 29;
            person.height = 176;
            Person* p = &person;
            p->age = 30;
            Console.WriteLine(p->age);
            (*p).height = 180; // разыменовывание указателя
            Console.WriteLine((*p).height);
        } } }
public struct Person {
    public int age;
    public int height;
}
```

Операторы "[]" и "()"

Круглые скобки — это оператор, повышающий приоритет операций, заключенных внутри.

Квадратные скобки используются для индексации массива. Если в программе определен массив, то значение выражения, заключенного в квадратные скобки, равно индексу элемента массива.

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };  
for (int i = 0; i < numbers.Length; i++)  
{  
    numbers[i] = 1 + (numbers[i] * 2);  
    Console.WriteLine(numbers[i]);  
}
```

Операции сравнения (отношения)

Результаты операций отношения

Операция	Результат
$x == y$	true , если x <u>равно</u> y , иначе false
$x != y$	true , если x <u>не равно</u> y , иначе false
$x < y$	true , если x <u>меньше</u> y , иначе false
$x > y$	true , если x <u>больше</u> y , иначе false
$x <= y$	true , если x <u>меньше или равно</u> y , иначе false
$x >= y$	true , если x <u>больше или равно</u> y , иначе false

Операции сравнения (отношения)

```
int a = 10;
```

```
int b = 4;
```

```
bool cc = a == b; // false
```

```
bool c = a != b; // true
```

```
bool d = a != 10; // false
```

```
bool ccc = a < b; // false
```

```
bool c1 = a > b; // true
```

```
bool d1 = a > 25; // false
```

```
bool c2 = a <= b; // false
```

```
bool d2 = a <= 25; // true
```

```
bool c3 = a >= b; // true
```

```
bool d3 = a >= 25; // false
```


Логические операции

В C# определены логические операции, которые также возвращают значение типа **bool**. В качестве операндов они принимают значения типа **bool**.

Оператор	Действие
&&	И
	ИЛИ
!	НЕ

Таблица истинности для логических операторов.

p	q	p&&q	p q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Логические операции

Логическое отрицание (!) определено для типа **bool**. Результат операции — значение **false**, если операнд равен **true**, и значение **true**, если операнд равен **false**.

```
bool t, z = true;  
t = !z;    // false
```

Операция логического сложения (||). Возвращает **true**, если хотя бы один из операндов возвращает **true**.

```
bool x1 = (5 > 6) || (4 < 6);  
// 5 > 6 - false, 4 < 6 - true, поэтому возвращается true  
bool x2 = (5 > 6) | (4 > 6);  
// 5 > 6 - false, 4 > 6 - false, поэтому возвращается false
```

Логические операции

Операция логического умножения (&&). Возвращает **true**, если оба операнда одновременно равны **true**.

```
bool x1 = (5 > 6) && (4 < 6);
```

// 5 > 6 - false, 4 < 6 - true, поэтому возвращается false

```
bool x2 = (5 < 6) && (4 < 6);
```

// 5 > 6 - true, 4 > 6 - true, поэтому возвращается true

Ниже приведены **приоритеты** операций сравнения и логических операций.

ВЫСШИЙ	!			
	>	>=	<	<=
	==	!=		
	&&			
НИЗШИЙ				

Логические операции

Как и для арифметических операций, естественный порядок вычислений можно изменять с помощью скобок.

Операнды логических выражений вычисляются слева направо.

Если значения первого операнда достаточно, чтобы определить результат **логической** операции, то второй операнд не вычисляется.

Операции сравнения и логические операции имеют более низкий приоритет, чем арифметические операции. $10 > 1 + 12 \iff 10 > (1 + 12)$

В одном и том же выражении можно использовать несколько операций.

`bool x1 = 10 > 5 && ! (10 < 9) || 3 <= 4; // true`

Поразрядные (побитовые) операции

Поразрядные операции выполняются над отдельными разрядами числа. В этом плане числа рассматриваются в двоичном представлении, например, 2 в двоичном представлении 10 и имеет два разряда, число 7 - 111 и имеет три разряда.

Оператор	Действие
&	и
	или
^	Исключающее ИЛИ
~	Дополнение до единицы (НЕ)
>>	Сдвиг вправо
<<	Сдвиг влево

Поразрядные (побитовые) операции

Поразрядные логические операции

(**&**, **|**, **^**) применяются к целочисленным операндам и работают с их двоичными представлениями. При выполнении операций операнды сопоставляются побитно (первый бит первого операнда с первым битом второго, второй бит первого операнда со вторым битом второго, и т.д.). Стандартные операции определены для типов **int**, **uint**, **long** и **ulong**.

Поразрядные (побитовые) операции

Операция логического умножения или логическое И (&). Возвращает **true**, если оба операнда одновременно равны **true**.

```
bool x1 = (5 > 6) & (4 < 6); // false
```

```
bool x2 = (5 < 6) & (4 < 6); // true
```

```
int x1 = 2; // 010
```

```
int y1 = 5; //101
```

```
Console.WriteLine(x1 & y1); // (0*1, 1*0, 0*1) = 0
```

```
int x2 = 4; //100
```

```
int y2 = 5; //101
```

```
Console.WriteLine(x2 & y2); // (1*1, 0*0, 0*1) = 1002,
```

то есть число 4₁₀

Поразрядные (побитовые) операции

Операция логического сложения или логическое ИЛИ

(|). Возвращает **true**, если хотя бы один из операндов возвращает **true**.

```
bool x1 = (5 > 6) | (4 < 6); // true
```

```
bool x2 = (5 > 6) | (4 > 6); // false
```

```
int x1 = 2; // 010
```

```
int y1 = 5; // 101
```

```
Console.WriteLine(x1|y1); //  $7_{10} = 111_2$ 
```

```
int x2 = 4; // 100
```

```
int y2 = 5; // 101
```

```
Console.WriteLine(x2 | y2); //  $5_{10} = 101_2$ 
```

Поразрядные (побитовые) операции

Операция исключающего ИЛИ (^) (XOR).

Возвращает **true**, если либо первый, либо второй операнд (но не одновременно) равны **true**, иначе возвращает **false**

p	q	p ^ q
0	0	0
1	0	1
1	1	0
0	1	1

`bool x5 = (5 > 6) ^ (4 < 6);`

`// 5 > 6 - false, 4 < 6 - true, поэтому возвращается true`

`bool x6 = (50 > 6) ^ (4 / 2 < 3);`

`// 50 > 6 - true, 4/2 < 3 - true, поэтому возвращается false`

Поразрядные (побитовые) операции

Эту операцию нередко применяют для простого шифрования:

```
int x = 45;    // Значение, которое надо  
              // зашифровать - в двоичной форме 101101
```

```
int key = 102; // Пусть это будет ключ - в двоичной  
              // форме 1100110
```

```
int encrypt = x ^ key; // Результатом будет число  
                      // 1001011 или 75
```

```
Console.WriteLine("Зашифрованное число: " + encrypt);
```

```
int decrypt = encrypt ^ key; // Результатом будет  
                             // исходное число 45
```

```
Console.WriteLine("Расшифрованное число: " +  
decrypt);
```

Поразрядные (побитовые) операции

Поразрядное отрицание (\sim), часто называемое побитовым, инвертирует каждый разряд в двоичном представлении операнда типа **int**, **uint**, **long** или **ulong**.

```
char b = '9'; // '9' = шестнадцатеричному значению 39
```

```
unsigned char f;
```

```
f = ~b; // 'ц' = шестнадцатеричному значению С6
```

Оператор дополнения до единицы " \sim " инвертирует каждый бит операнда. Это значит, что каждая единица станет нулем, и наоборот.

Исходный байт	00101100
---------------	----------

После первого отрицания	11010011
-------------------------	----------

После второго отрицания	00101100
-------------------------	----------

Поразрядные (побитовые) операции

Две пары операций `|` и `||` (а также `&` и `&&`) выполняют похожие действия, однако же они не равнозначны.

В выражении `z = x | y`; будут вычисляться оба значения - `x` и `y`.

В выражении же `z = x || y`; сначала будет вычисляться значение `x`, и если оно равно `true`, то вычисление значения `y` уже смысла не имеет, так как в любом случае уже `z` будет равно `true`. Значение `y` будет вычисляться только в том случае, если `x` равно `false`.

То же самое касается пары операций `&`/`&&`.

`z = x & y`; // будут вычисляться оба значения - `x` и `y`

`z = x && y`; // сначала `x`, и если оно равно `true`, то `y`

Поразрядные (побитовые) операции

Операции сдвига (\ll и \gg) применяются к целочисленным операндам и определены для типов **int**, **uint**, **long** и **ulong**.

При *сдвиге влево* (\ll) освободившиеся разряды обнуляются.

$$4_{10} = 00000100_2$$
$$4 \ll 1 \rightarrow 8_{10} = 00001000_2$$

При *сдвиге вправо* (\gg) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае.

$$16_{10} = 00010000_2$$
$$16 \gg 1 \rightarrow 8_{10} = 00001000_2$$

Поразрядные (побитовые) операции

Операции сдвига можно использовать вместо непосредственного умножения или деления на два.

С помощью сдвига вправо можно эффективно поделить число на 2, а с помощью сдвига влево — умножить на 2.

byte x	Двоичное представление	Значение
x = 7;	0 0 0 0 0 1 1 1	7
x = x << 1;	0 0 0 0 1 1 1 0	14
x = x << 3;	0 1 1 1 0 0 0 0	112
x = x << 2;	1 1 0 0 0 0 0 0	192
x = x >> 1;	0 1 1 0 0 0 0 0	96
x = x >> 2;	0 0 0 1 1 0 0 0	24

Операция сложения

Правила сложения двух двоичных чисел

можно показать на следующем примере:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{и } 1 \text{ переносится в старший разряд}$$

Пример сложения многоразрядных чисел.

Требуется сложить два числа 18_{10} и 23_{10}

$$\begin{array}{r} 18 = 010010 \\ + 23 = 010111 \\ \hline 41 = 101001 \end{array}$$

1 1 1-перенос 1 в старшие разряды

Операция вычитания

Операция вычитания в цифровых системах реализуется с помощью операции сложения.

Вычитаемое при этом представляется в дополнительном коде (если расчет не требует высокой точности - в обратном коде).

Двоичный код со знаком называют также *прямым кодом*.

Операция вычитания

Положительное и отрицательное числа,
десятичный эквивалент которых равен

46_{10} :

$\boxed{0}0101110$ код положительного числа

$\boxed{1}0101110$ код отрицательного числа

Операция вычитания

Обратный код получается путем замены всех “0” на “1” и всех “1” на “0” прямого кода (двоичного числа со знаком). Причем, знаковый разряд при этом остается неизменным.

1 0101110 - прямой код отрицательного числа 46
1 1010001 - обратный код числа 46

Замена “0” на единицу (“1”) называется *инвертированием* (также и замена “1” на “0”).

Операция вычитания

Обратный код, дополненный единицей в младшем разряде, называется *дополнительным кодом*.

Последовательность действий при получении дополнительного кода:

$$\begin{array}{l} \boxed{1} 0101110 \text{ - прямой код} \\ \text{инвертирование всех разрядов кроме знакового} \\ \boxed{1} 1010001 \text{ - обратный код} \\ + \quad \quad \quad 1 \text{ - дополнение 1 к младшему разряду} \\ \hline \boxed{1} 1010010 \text{ - дополнительный код.} \end{array}$$

Операция вычитания

Задание: требуется из числа 23 отнять число 18. С начала вычитаемое следует представить в дополнительном коде:

010010 – 18 прямой код

101101 – 18 обратный код

+

 1

101110 – 18 дополнительный код

Операция вычитания

	<u>Разряд знака</u>	
23	0	010111 Прямой код
+		
-18	1	101110 Дополнительный код
<hr/>		
	0	000101 Результат в прямом коде

Так как вычитаемое (18) по модулю меньше уменьшаемого (23), то результат достигнут. Таким образом, получилось число плюс 5.

Операция вычитания (с/р)

Задание: требуется из числа 18 отнять 23.

С начала вычитаемое следует
представить в дополнительном коде:

010111 – 23 прямой код

101000 – 23 обратный код

+

 1

101001 – 23 дополнительный код

Операция вычитания

		Разряд знака	
18	0	010010	Прямой код
+			
-23	1	101001	Дополнительный код
<hr/>			
	1	111011	Результат в дополнительном коде
	1	000100	Инвертирование
		1	Прибавление "1"
<hr/>			
	1	000101	Результат в прямом коде

Так как вычитаемое (23) по модулю больше уменьшаемого (18), то результат из дополнительного кода следует перевести в прямой код.

Таким образом, получилось число плюс -5.

Контрольные вопросы

1. Где можно описывать переменные? Что входит в описание переменной?
2. Что происходит при использовании в выражении операндов различных типов? Приведите примеры.
3. Перечислите операции языка C#, сгруппировав их по приоритетам.
4. К операндам какого типа применимы операции сдвига?
5. Что такое исключительные ситуации?
6. Опишите принципы обработки исключительных ситуаций.