Лекция 8

Стандартные типы данных

Цель: Познакомиться с основными типами данных языка C#

Тип данных определяет:

- размер памяти, выделяемой под хранение данных;

- внутреннее представление данных в памяти компьютера;

- множество значений, которые могут принимать величины этого типа;

- операции и функции, которые можно применять к величинам этого типа.

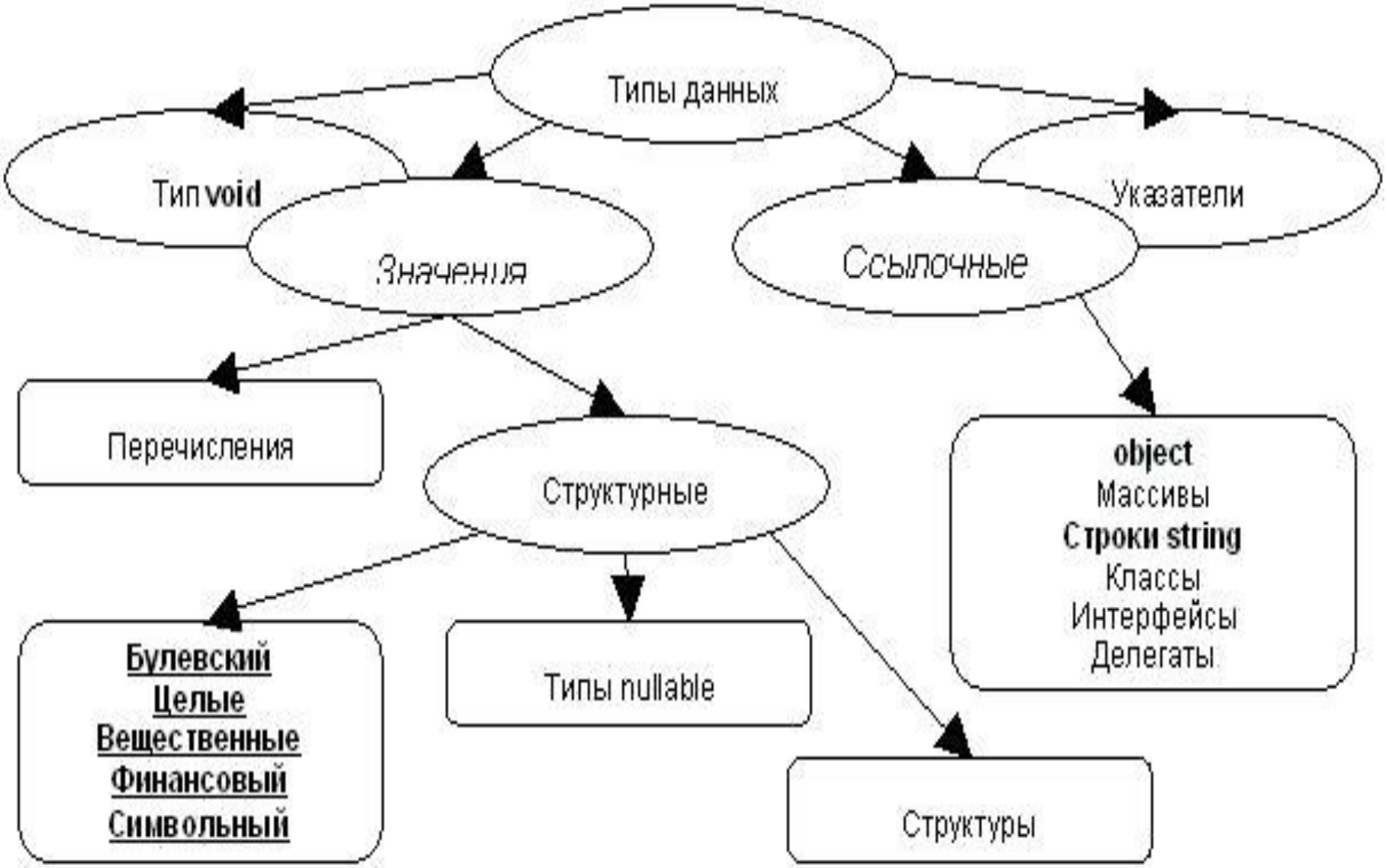
Классификация типов данных по разным признакам

По строению элементов, все типы можно разделить на ***простые*** (не имеют внутренней структуры) и ***структурированные*** (состоят из элементов других типов).

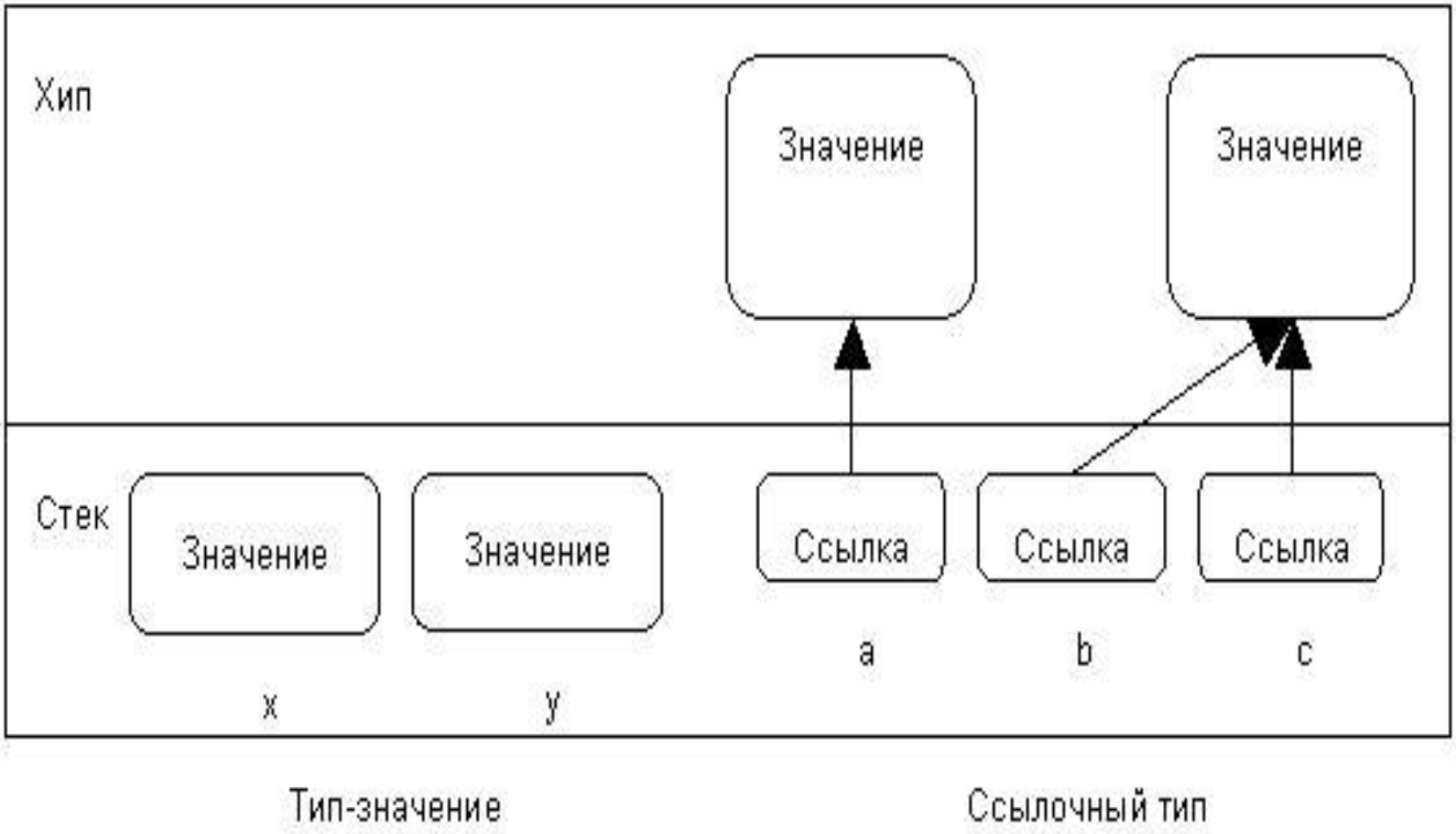
По способу создания типы можно разделить на ***встроенные*** (стандартные) и ***определяемые программистом***.

По способу хранения значений типы делятся на ***значимые***, или типы-значения, и ***ссылочные*** .

Классификация типов данных
по способу хранения значений



Разница между величинами значимого и ссылочного типов



Встроенные типы C#					
Название	**Ключевое слово**	**Тип .NET**	**Диапазон значений**	**Описание**	**Размер, битов**
Логический тип	bool	Boolean	true, false		
Целые типы	sbyte	SByte	От –128 до 127	Со знаком	8
	byte	Byte	От 0 до 255	Без знака	8
	short	Int16	От –32768 до 32767	Со знаком	16
	ushort	UInt16	От 0 до 65535	Без знака	16
	int	Int32	От -2×10^9 до 2×10^9	Со знаком	32
	uint	UInt32	От 0 до 4×10^9	Без знака	32
	long	Int64	От -9×10^{18} до 9×10^{18}	Со знаком	64
	ulong	UInt64	От 0 до 18×10^{18}	Без знака	64
Символьный тип	char	Char	От U+0000 до U+ffff	Unicode-символ	16

Встроенные типы C#					
Название	**Ключевое слово**	**Тип .NET**	**Диапазон значений**	**Описание**	**Размер, битов**
Вещественные	float	Single	От 1.5×10^{-45} до 3.4×10^{38}	7 цифр	32
	double	Double	От 5.0×10^{-324} до 1.7×10^{308}	15–16 цифр	64
Финансовый тип	decimal	Decimal	От 1.0×10^{-28} до 7.9×10^{28}	28–29 цифр	128
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Тип object	object	Object	Можно хранить все, что угодно	Всеобщий предок	

Логический, или булев, тип

true (истина) и **false** (ложь)

В **C#** не определено взаимное преобразование логических и целых значений.

Например, **1** не преобразуется в значение **true**, а **0** — в значение **false**.

Пример 1

```
using System;
class BoolDemo {
    static void Main() {
        bool b;
        b = false;
        Console.WriteLine ("b равно " + b);
        b = true;
        Console.WriteLine("b равно " + b);
        // Логическое значение может
        // управлять оператором if.
        if (b) Console.WriteLine("Выполняется.");
```

Пример 1

```
        b = false;
        if (b) Console.WriteLine("Не выполняется.");
         // Результатом выполнения оператора отношения
         // является логическое значение.
       Console.WriteLine("0 > 9 равно " + (10 > 9) );
    }
}
```

Эта программа дает следующий результат.

```
b равно False
b равно True
Выполняется.
10 > 9 равно True
```

Целые типы

Внутреннее представление величины ***целого типа*** — целое число в двоичном коде. В знаковых типах старший бит числа интерпретируется как знаковый (0 — положительное число, 1 — отрицательное).

Самым распространенным в программировании целочисленным типом является тип **int**. Переменные типа **int** нередко используются для управления циклами, индексирования массивов и математических расчетов общего назначения.

Пример 2

```
using System;
class Inches {
    static void Main() {
        long inches;
        long miles;
        miles = 93000000; // 93 000 000 миль до Солнца
                          // 5 280 футов в миле, 12 дюймов в футе,
        inches = miles * 5280 * 12;
        Console.WriteLine("Расстояние до Солнца:
  " + inches + " дюймов.");
    }
}
```

Пример 3

```
using System;
class Use_byte {
    static void Main() {
        byte x;
        int sum;
        sum = 0;
        for(x = 1; x <= 100; x++)
            sum = sum + x;
        Console.WriteLine("Сумма чисел от 1 до
   100 равна " + sum);
    }
}
```

Вещественные типы, или ***типы данных с плавающей точкой***

Внутреннее представление вещественного числа состоит из двух частей — **мантиссы** и **порядка**, причем каждая часть имеет знак. Длина мантиссы определяет *точность* числа, а длина порядка — его *диапазон*.

Чаще всего в программах используется тип **double**, поскольку его диапазон и точность покрывают большинство потребностей. Этот тип имеют вещественные литералы и многие стандартные математические функции. Например, метод **Sqrt**(), определенный в библиотеке классов **System. Math**, возвращает значение типа **double**, которое представляет собой квадратный корень из аргумента типа **double**, передаваемого данному методу.

Вещественные типы, или ***типы данных с плавающей точкой***

-25,994 → -0,25994e+2 - преобразование к каноническому виду (-0,25994*10^2)

В памяти компьютера хранится:

	Знак числа	Мантисса	Знак порядка	Порядок
Число	-	25994	+	2
Величина	1	2599400000000000	0	2
Двоичное представление величин	01	10010011110000100011111110011010000100010000000000	00	10

Вещественные типы, или ***типы данных с плавающей точкой***

0,000045 → +0,45e-4 - преобразование к каноническому виду $(+0,45*10^{-4})$

В памяти компьютера хранится:

	Знак числа	Мантисса	Знак порядка	Порядок
Число	+	45	-	4
Величина	0	4500000000000000	1	4
Двоичное представление величин	00	11111111110010111001111001010111101010000000000000	01	100

Пример 4

```
using System;
class Trigonometry {
    static void Main() {
        Double theta; // угол в радианах
        for(theta = 0.1; theta <= 1.0; theta = theta +0.1) {
            Console.WriteLine("Синус угла " + theta + " !
  равен " + Math.Sin(theta));
            Console.WriteLine("Косинус угла " + theta + "
  равен " + Math.Cos(theta));
            Console.WriteLine("Тангенс угла " + theta + "
  равен " + Math.Tan (theta));
            Console.WriteLine();
        }
    }
```

Пример 4

// Часть результата выполнения программы:

Синус угла 0.1 равен 0.0998334166468282

Косинус угла 0.1 равен 0.995004165278026

Тангенс угла 0.1 равен 0.100334672085451

Синус угла 0.2 равен 0.198669330795061

Косинус угла 0.2 равен 0.980066577841242

Тангенс угла 0.2 равен 0.202710035508673

// и т. д.

Синус угла 1.0 равен 0.000004848136811 1

Косинус угла 1.0 равен 0.999999999882477

Тангенс угла 1.557407724654902 3

Денежный тип

Тип **decimal** предназначен для ***денежных вычислений***, в которых критичны ошибки округления. Величины типа **decimal** позволяют хранить 28–29 десятичных разрядов.

Тип **decimal** не относится к вещественным типам, у них различное внутреннее представление. Величины *денежного типа* даже нельзя использовать в одном выражении с вещественными без явного преобразования типа. Использование величин финансового типа в одном выражении с целыми допускается.

Пример 5

```
using System;
class UseDecimal {
    static void Main() {
        decimal price;
        decimal discount;
        decimal discounted_price;
        // Рассчитать цену со скидкой
        price = 19.95m;
        discount = 0.15m; // норма скидки составляет 15%
        discounted_price = price - ( price * discount);
        Console.WriteLine("Цена со скидкой: $" +
   discounted_price);
    }
}
```

Символьный тип

В **C#** ***символы*** представлены не 8-разрядным кодом, как во многих других языках программирования, например C++, а 16-разрядным кодом, который называется уникодом (**Unicode**).

Тип **char** представляет 16-разрядные значения без знака в пределах от **0** до **65 535**. При этом стандартный набор символов в 8-разрядном коде ASCII является подмножеством уникода в пределах от 0 до 127.

```
char  ch;                    ch = 'X';
Console.WriteLine ("Значение ch равно: " + ch);
```

В C# отсутствует автоматическое преобразование символьных значений в целочисленные и обратно:

```
char ch;                          ch = 88; // ошибка
```

код	символ	код	символ	код	символ	код	Символ
0	NUL	32	BL	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16		48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	

Тип void

Тип **void** (пустой, неопределенный) используется для определения функции, не возвращающей никаких значений, для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

Преобразование и приведение типов

Если в одной операции присваивания смешиваются совместимые типы данных, то **значение в правой части оператора присваивания автоматически (неявно) преобразуется в тип, указанный в левой его части**.

```
int i;
float f;
i = 10;
f = i; // присвоить целое значение переменной типа float
```

Преобразование и приведение типов

В операции присваивания **неявное преобразование типов** (расширяющее преобразование) происходит автоматически при следующих условиях:

- оба типа совместимы;
- диапазон представления чисел целевого типа шире, чем у исходного типа.

```
long L;
double D;
L = 100123285L;
D = L; //неявное преобразование типа long в тип double
```

Преобразование и приведение типов

Если тип **long** может быть преобразован в тип **double** неявно, то обратное преобразование типа **double** в тип **long** неявным образом невозможно, поскольку оно не является расширяющим.

Не допускается неявное взаимное преобразование типов **decimal** и **float** или **double**, а также числовых типов и **char** или **bool**.

Типы **char** и **bool** несовместимы друг с другом.

Преобразование и приведение типов

Приведение типов (явное преобразование типов) — это команда компилятору преобразовать результат вычисления выражения в указанный тип.

(целевой_тип) выражение

Здесь **целевой_тип** обозначает тот тип, в который желательно преобразовать указанное выражение. Например:

```
double x, y;     int k;
k = (int) (x / y); // результат вычисления
    // выражения x/y должен быть типа int
```

Преобразование и приведение типов

Если приведение типов приводит к сужающему преобразованию, то часть информации может быть потеряна.

Например, в результате приведения типа **long** к типу **int** часть информации потеряется, если значение типа **long** окажется больше диапазона представления чисел для типа **int**, поскольку старшие разряды этого числового значения отбрасываются.

Когда значение с плавающей точкой приводится к целочисленному, то в результате усечения теряется дробная часть этого числа.

Пример 6

```
using System;
class CastDemo {
    static void Main() {
        double x, y;
        byte b;
        int i;
        char ch;
        uint u;
        short s;
        long l;
        x = 10.0;
        y = 3.0;
```

Пример 6

```
// Приведение типа double к типу int,
// дробная часть числа теряется
 i = (int) (x / y);
Console.WriteLine("Целочисленный результат деления x / y: " + i);
 // Приведение типа int к типу byte
 // без потери данных
i = 255;
b = (byte) i;
Console.WriteLine("b после присваивания 255: " + b + " — без потери данных.");
```

Пример 6

```
// Приведение типа int к типу byte
// с потерей данных
i = 257;
b = (byte) i;
Console.WriteLine("b после присваивания 257: " + b + " — с потерей данных.");
// Приведение типа uint к типу short
// без потери данных
 u = 32000;
 s = (short) u;
 Console.WriteLine("s после присваивания 32000: " + s + " — без потери данных.");
```

Пример 6

```
// Приведение типа uint к типу short
// с потерей данных
u = 64000;
s = (short) u;
Console.WriteLine("s после присваивания 64000: " + s + " — с потерей данных.");
// Приведение типа long к типу uint
// без потери данных
1 = 64000;
u = (uint) 1;
Console.WriteLine("u после присваивания 64000: " + и + " — без потери данных.");
```

Пример 6

```
    // Приведение типа long к типу uint
    // с потерей данных
      1 = -12;
      u = (uint) 1;
      Console.WriteLine("u после присваивания
-12: " + и + " — с потерей данных.");
    // Приведение типа int к типу char
      b = 88;   // код ASCII символа X
      ch = (char) b;
      Console.WriteLine("ch после присваивания
88: " + ch);
   }
}
```

Пример 6

Результат выполнения программы:

Целочисленный результат деления x / y: 3

b после присваивания 255: 255 — без потери данных.

b после присваивания 257: 1 — с потерей данных.i

s после присваивания 32000: 32000 — без потери данных,

s после присваивания 64000: -1536 — с потерей данных.

и после присваивания 64000: 64000 — без потери данных,

и после присваивания -12: 4294967284 — с потерей данных.

ch после присваивания 88: X

Преобразование типов в выражениях

Когда в выражении смешиваются разные типы данных, они преобразуются в один и тот же тип по порядку следования операций.

Преобразования типов выполняются по принятым в **C#** правилам преобразования типов.

Правила преобразования типов в выражении с двумя операндами

ЕСЛИ один операнд имеет тип **decimal**, ТО и второй операнд преобразуется к типу **decimal** (но если второй операнд имеет тип **float** или **double**, результат будет ошибочным).

ЕСЛИ один операнд имеет тип **double**, ТО и второй операнд преобразуется к типу **double**.

Правила преобразования типов в выражении с двумя операндами

ЕСЛИ один операнд имеет тип **float**, ТО и второй операнд преобразуется к типу **float**.

ЕСЛИ один операнд имеет тип **ulong**, ТО и второй операнд преобразуется к типу **ulong** (но если второй операнд имеет тип **sbyte**, **short**, **int** или **long**, результат будет ошибочным).

Правила преобразования типов в выражении с двумя операндами

ЕСЛИ один операнд имеет тип **long**, ТО и второй операнд преобразуется к типу **long**.

ЕСЛИ один операнд имеет тип **uint**, а второй — тип **sbyte**, **short** или **int**, ТО оба операнда преобразуются к типу **long**.

ЕСЛИ один операнд имеет тип **uint**, ТО и второй операнд преобразуется к типу **uint**.

ИНАЧЕ оба операнда преобразуются к типу **int**.

Важные замечания

Во-первых, не все типы могут смешиваться в выражении. В частности, неявное преобразование типа **float** или **double** в тип **decimal** невозможно, как, впрочем, и смешение типа **ulong** с любым целочисленным типом со знаком. Для смешения этих типов требуется явное их приведение.

Важные замечания

Во-вторых, особого внимания требует последнее из приведенных выше правил. Оно гласит: если ни одно из предыдущих правил не применяется, то все операнды продвигаются к типу **int**.

Следовательно, все значения типа **char**, **sbyte**, **byte**, **ushort** и **short** продвигаются к типу **int** в целях вычисления выражения (целочисленное продвижение типов). Это означает, что результат выполнения всех арифметических операций будет иметь тип не ниже **int**.

Важные замечания

Следует иметь в виду, что правила преобразования типов применяются только к значениям, которыми оперируют при вычислении выражения. Так, если значение переменной типа **byte** преобразуется к типу **int** внутри выражения, то вне выражения эта переменная по-прежнему относится к типу **byte**. Преобразование типов затрагивает только вычисление выражения.

Важные замечания

Преобразование типов может иногда привести к неожиданным результатам.

```
byte b;
b = 10;
b = b * b;   // byte = int  ???
```

Сначала операнды типа **byte** преобразуются к типу **int**. А затем выполняется операция, дающая результат типа **int**. Следовательно, результат выполнения операции, в которой участвуют два значения типа **byte**, будет иметь тип **int**.

В этом случае, чтобы сохранить тип **byte**, необходимо приведение типов.

```
b = (byte) (b * b); // !!!
```

Важные замечания

Аналогичная ситуация возникает при выполнении операций с символьными операндами. Например, в следующем фрагменте кода требуется обратное приведение к типу **char**, поскольку операнды **ch1** и **ch2** в выражении преобразуются к типу **int**.

```
char ch1 = 'a', ch2 = 'b';
ch1 = (char) (ch1 + ch2);
```

Без приведения типов результат сложения операндов **ch1** и **ch2** будет иметь тип **int**, и поэтому его нельзя присвоить переменной типа **char**.

Важные замечания

Преобразование типов происходит и при выполнении унарных операций, например с унарным минусом. Операнды унарных операций более мелкого типа, чем **int** (**byte**, **sbyte**, **short** и **ushort**), т.е. с более узким диапазоном представления чисел, преобразуются к типу **int**. То же самое происходит и с операндом типа **char**. Кроме того, если выполняется унарная операция отрицания значения типа **uint**, то результат преобразуется к типу **long**.

Методы Parse и TryParse

Метод **Parse()** в качестве параметра принимает строку и возвращает объект текущего типа.

```
int a = int.Parse("10");
double b = double.Parse("23,56");
decimal c = decimal.Parse("12,45");
byte d = byte.Parse("4");
Console.WriteLine($"a={a}  b={b}  c={c}  d={d}");
```

Пример 7

```
using System;
using System.Globalization;
namespace FirstApp {
    class Program {
        public static void Main(string[] args)  {
            IFormatProvider formatter = new
NumberFormatInfo{NumberDecimalSeparator = "."};
            double b = double.Parse("23.56", formatter);
            Console.WriteLine("b12 = {0}", b12);
        }
    }
}
```

Методы Parse и TryParse

В данном случае в качестве разделителя устанавливается точка. Однако тем не менее потенциально при использовании метода **Parse** можно столкнуться с ошибкой, например, при передачи алфавитных символов вместо числовых. И в этом случае более удачным выбором будет применение метода **TryParse()**. Он пытается преобразовать строку к типу и, если преобразование прошло успешно, то возвращает **true**. Иначе возвращается **false.**

Методы Parse и TryParse

```
int number;
Console.WriteLine("Введите строку:");
string input = Console.ReadLine();
bool result = int.TryParse(input, out number);
if (result == true)
    Console.WriteLine("Преобразование прошло успешно");
else
    Console.WriteLine("Преобразование завершилось неудачно");
```

Метод Convert()

Реализует преобразование значений с помощью статических методов:

ToBoolean(value) **ToByte(value)**
ToChar(value) **ToDateTime(value)**
ToDecimal(value) **ToDouble(value)**
ToInt16(value) **ToInt32(value)**
ToInt64(value) **ToSByte(value)**
ToSingle(value) **ToUInt16(value)**
ToUInt32(value) **ToUInt64(value)**

Метод Convert()

```
int n = Convert.ToInt32("23");
bool b = true;
double d = Convert.ToDouble(b);
Console.WriteLine($"n={n}  d={d}");
```

Если методу не удастся преобразовать значение к нужному типу, то он выбрасывает исключение **FormatException**.

Значение NaN

Метод или оператор возвращает **NaN,** если результат операции является неопределенным. Например, результат деления нуля на нуль является **NaN**.

```
double zero = 0.0;
Console.WriteLine("{0} / {1} = {2}", zero, zero, zero/zero);       //   0 / 0 = NaN
```

Кроме того, вызов метода с NaN значение или операцию над NaN значение возвращает NaN.

Значение NaN

```
double nan1 = Double.NaN;
Console.WriteLine("{0} + {1} = {2}", 3, nan1, 3 + nan1);       //   3 + NaN = NaN
Console.WriteLine("Abs({0}) = {1}", nan1, Math.Abs(nan1));    //  Abs(NaN) = NaN
```

```
Double zero = 0;
if ((0 / zero) == Double.NaN)
    Console.WriteLine("0 / 0 can be tested with Double.NaN.");
else    Console.WriteLine("0 / 0 cannot be tested with Double.NaN; use Double.IsNan() instead.");
```

Контрольные вопросы

1 Какие основные типы данных языка C# вы знаете?

2 Назовите спецификаторы типов данных.

3 Каков механизм преобразования типов?