

# БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА C#

## Состав языка

*Алфавит* – совокупность допустимых в языке символов. Алфавит языка C# включает:

1. прописные и строчные латинские буквы и буквы национальных алфавитов (включая кириллицу);
2. арабские цифры от 0 до 9, шестнадцатеричные цифры от A до F;
3. специальные знаки: " { } , | ; [ ] ( ) + - / % \* . \ ' : ? < = > ! & ~ ^ @ \_
4. пробельные символы: пробел, символ табуляции, символ перехода на новую строку.

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы).

Границы лексем определяются другими лексемами, такими, как разделители или знаки операций. В свою очередь лексемы входят в состав выражений (выражение задает правило вычисления некоторого значения) и операторов (оператор задает законченное описание некоторого действия).

*Идентификатор* – это имя программного элемента: константы, переменной, метки, типа, класса, объекта, метода и т.д. Идентификатор может включать латинские буквы и буквы национальных алфавитов, цифры и символ подчеркивания. Прописные и строчные буквы различаются, например, `myname`, `myName` и `MyName` — три различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра.

Пробелы и другие разделители внутри имен не допускаются. Язык C# не налагает никаких ограничений на длину имен, однако для удобства чтения и записи кода не стоит делать их слишком длинными.

Для улучшения читабельности кода программным элементам следует давать осмысленные имена, составленные в соответствии с определенными правилами. Существует несколько видов нотаций – соглашений о правилах создания имен.

В нотации Pascal каждое слово, входящее в идентификатор, начинается с заглавной буквы. Например: `Age`, `LastName`, `TimeOfDeath`.

Венгерская нотация отличается от предыдущей наличием префикса, соответствующего типу величины. Например: `fAge`, `sName`, `iTime`.

В нотации Camel с заглавной буквы начинается каждое слово идентификатора, кроме первого. Например: `age`, `lastName`, `timeOfDeath`.

Наиболее часто используются нотации Pascal или Camel. Однако в простых программах будут использоваться однобуквенные переменные.

*Ключевые слова* – это зарезервированные идентификаторы, которые имеют специальное значение для компилятора, например, `static`, `int` и т.д. Ключевые слова можно использовать только по прямому назначению. Однако если перед ключевым словом поставить символ `@`, например, `@int`, `@static`, то полученное имя можно использовать в качестве идентификатора. С полным перечнем ключевых слов и их назначением можно ознакомиться в справочной системе C#.

**Замечание.** Другие лексемы (знаки операций и константы), а также правила формирования выражений и различные виды операторов будут рассмотрены чуть позже.

## Типы данных

C# является языком со строгой типизацией. В нем необходимо объявлять тип всех создаваемых программных элементов (например, переменных, объектов, окон, кнопок и т.д.), что позволяет среде CLR предотвращать возникновение ошибок, следя за тем, чтобы объектам присваивались значения только разрешенного типа. Тип программного элемента

сообщает компилятору о его размере (например, тип `int` показывает, что объект занимает 4 байта) и возможностях (например, кнопка может быть нарисована, нажата и т. д.).

В C# типы делятся на три группы:

1. *базовые* типы – предлагаемые языком;
2. типы, *определяемые пользователем*;
3. анонимные типы - типы, которые автоматически создаются на основе инициализаторов объектов (начиная с версии C# 3.0).

Кроме того, типы C# разбиваются на две другие категории: *размерные типы* (value type) и *ссылочные типы* (reference type). Почти все базовые типы являются размерными типами. Исключение составляют типы `Object` и `String`, которые являются базовыми, но ссылочными типами данных. Все пользовательские типы, кроме структур, являются ссылочными. Дополнительно к упомянутым типам, язык C# поддерживает типы *указателей*, однако они используются только с неуправляемым кодом.

Принципиальное различие между размерными и ссылочными типами состоит в способе хранения их значений в памяти. В первом случае фактическое значение хранится в стеке (или как часть большого объекта ссылочного типа). Адрес переменной ссылочного типа тоже хранится в стеке, но сам объект хранится в куче.

*Стек* – это структура, используемая для хранения элементов по принципу LIFO (Last input – first output или *первым ушел - последним пришел*). В данном случае под стекком понимается область памяти, обслуживаемая процессором, в которой хранятся значения локальных переменных. *Куча* – область памяти, используемая для хранения данных, работа с которыми реализуется через указатели и ссылки. Память для размещения таких данных выделяется программистом динамически, а освобождается сборщиком мусора.

Сборщик мусора уничтожает программные элементы в стеке через некоторое время после того, как закончит существование раздел стека, в котором они объявлены. То есть, если в пределах блока (фрагмента кода, помещенного в фигурные скобки `{}`) объявлена локальная переменная, соответствующий программный элемент будет удален по окончании работы данного блока. Объект в куче подвергается сборке мусора через некоторое время после того, как уничтожена последняя ссылка на него.

Язык C# предлагает обычный набор базовых типов, каждому из них соответствует тип, поддерживаемый общезыковой спецификацией .NET (CLS).

Тип в языке C#	Размер в байтах	Тип .NET	Описание
<b>Базовый тип</b>			
<code>object</code>		<code>Object</code>	Может хранить все что угодно, т.к. является всеобщим предком
<b>Логический тип</b>			
<code>bool</code>	1	<code>Boolean</code>	<code>true</code> или <code>false</code>
<b>Целые типы</b>			
<code>sbyte</code>	1	<code>SByte</code>	Целое со знаком (от -128 до 127)
<code>byte</code>	1	<code>Byte</code>	Целое без знака (от 0 до 255)
<code>short</code>	2	<code>Int16</code>	Целое со знаком (от -32768 до 32767)
<code>ushort</code>	2	<code>UInt16</code>	Целое без знака (от 0 до 65535)
<code>int</code>	4	<code>Int32</code>	Целое со знаком (от -2147483648 до 2147483647)
<code>uint</code>	4	<code>UInt</code>	Целое число без знака (от 0 до 4 294 967 295)
<code>long</code>	8	<code>Int64</code>	Целое со знаком (от -9223372036854775808 до 9223372036854775807)
<code>ulong</code>	8	<code>UInt64</code>	Целое без знака (от 0 до 0fffffffffffffff)
<b>Вещественные типы</b>			
<code>float</code>	4	<code>Single</code>	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 1.5 \cdot 10^{-45}$ до $\pm 3.4 \cdot 10^{38}$ с 7 значащими цифрами

double	8	Double	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ с 15-16 значащими цифрами
<b>Символьный тип</b>			
char	2	Char	Символ Unicode
<b>Строковый тип</b>			
string		String	Строка из Unicode-символов
<b>Финансовый тип</b>			
decimal	12	Decimal	Число до 28 знаков с фиксированным положением десятичной точки. Обычно используется в финансовых расчетах и требует суффикса <<m>> или <<M>>

## Переменные и константы

**Переменная** представляет собой типизированную область памяти. Программист создает переменную, объявляя ее тип и указывая имя. При объявлении переменной ее можно инициализировать (присвоить ей начальное значение), а затем в любой момент ей можно присвоить новое значение, которое заменит собой предыдущее.

```
static void Main()
{
    int i=10; //объявление и инициализация целочисленной переменной i
    Console.WriteLine(i); //просмотр значения переменной
    i=100; //изменение значение переменной
    Console.WriteLine(i);
}
```

В языках предыдущего поколения переменные можно было использовать без инициализации. Это могло привести к множеству проблем и долгому поиску ошибок. В языке C# требуется, чтобы переменные были явно проинициализированы до их использования. Проверим этот факт на примере.

```
static void Main()
{
    int i; //объявление переменной без инициализации
    Console.WriteLine(i); //просмотр значения переменной
}
```

При попытке скомпилировать этот пример в списке ошибок будет выведено следующее сообщение: «Использование локальной переменной *i*, которой не присвоено значение».

Инициализировать каждую переменную сразу при объявлении необязательно, но необходимо присвоить ей значение до того, как она будет использована.

**Константа**, в отличие от переменной, не может менять свое значение. Константы бывают трех видов: *литералы*, *типизированные константы* и *перечисления*.

В операторе присваивания:

```
x=32;
```

число 32 является *литеральной константой*. Его значение всегда равно 32 и его нельзя изменить.

*Типизированные константы* именуют постоянные значения. Объявление типизированной константы происходит следующим образом:

```
const <тип> <идентификатор> = <значение>;
```

Рассмотрим пример:

```
static void Main()
{
```

```

const int i=10;           //объявление целочисленной константы i
Console.WriteLine(i);   //просмотр значения константы
i=100;                  //ошибка – недопустимо изменять значение константы
Console.WriteLine(i);
}

```

**Задание.** Измените программу так, чтобы при объявлении константы не происходила инициализация. Как на это отреагирует компилятор и почему?

*Перечисления (enumerations)* являются альтернативой константам. Перечисление - это особый размерный тип, состоящий из набора именованных констант (называемых *списком перечисления*). Синтаксис объявления перечисления следующий:

```

[атрибуты] [модификаторы] enum <имя> [ : базовый тип]
{список-перечисления констант(через запятую)};

```

*Замечание.* Атрибуты и модификаторы являются необязательными элементами этой конструкции. Более подробные сведения о их можно найти в дополнительных источниках информации.

Базовый тип - это тип самого перечисления. Если не указать базовый тип, то по умолчанию будет использован тип `int`. В качестве базового типа можно выбрать любой целый тип, кроме `char`. Пример использования перечисления:

```

class Program
{
    enum gradus:int
    {
        min=0,
        krit=72,
        max=100, //1
    }
    static void Main()
    {
        Console.WriteLine("минимальная температура=" + (int) gradus.min);
        Console.WriteLine("критическая температура=" + (int)gradus.krit);
        Console.WriteLine("максимальная температура=" + (int)gradus.max);
    }
}

```

#### **Замечания**

1. В общем случае последнюю запятую в объявлении перечисления можно не ставить (см. строку 1). Но лучше ее поставить: если вам придется добавить еще несколько строк в перечисление, такая предусмотрительность избавит вас от возможных синтаксических ошибок.
2. Запись `(int) gradus.min` используется для явного преобразования перечисления к целому типу. Если убрать `(int)`, то на экран будет выводиться название констант.
3. Символ `+` в записи `"минимальная температура=" + (int) gradus.min` при обращении к методу `WriteLine` означает, что строка `"минимальная температура="` будет «склеена» со строковым представлением значения `(int) gradus.min`. В результате получится новая строка, которая и будет выведена на экран.

## **Организация ввода-вывода данных. Форматирование.**

Программа при вводе данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода (клавиатура) и вывода (экран) называется консолью. В языке C# нет операторов ввода и вывода. Вместо них для обмена данными с внешними устройствами используются специальные классы. В частности, для

работы с консолью используется стандартный класс *Console*, определенный в пространстве имен *System*.

### Вывод данных

В приведенных выше примерах мы уже рассматривали метод *WriteLine*, реализованный в классе *Console*, который позволяет организовывать вывод данных на экран. Однако существует несколько способов применения данного метода:

1. `Console.WriteLine(x);` //на экран выводится значение идентификатора *x*
2. `Console.WriteLine("x=" + x + "y=" + y);` /\* на экран выводится строка, образованная последовательным слиянием строки "x=", значения *x*, строки "y=" и значения *y* \*/
3. `Console.WriteLine("x={0} y={1}", x, y);` /\* на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение *x*, а вместо {1} – значение *y*\*/

Далее мы будем использовать только третий вариант, поэтому рассмотрим его более подробно. Пусть нам дан следующий фрагмент программы:

```
int i=3, j=4;
Console.WriteLine("{0} {1}", i, j);
```

При обращении к методу *WriteLine* через запятую перечисляются три аргумента: "{0} {1}", *i*, *j*. Первый аргумент "{0} {1}" определяет формат выходной строки. Следующие аргументы нумеруются с нуля, так переменная *i* имеет номер 0, *j* – номер 1. Значение переменной *i* будет помещено в выходную строку на место параметра {0}, а значение переменной *j* - на место параметра {1}. В результате на экран будет выведена строка: 3 4.

Если мы обратимся к методу *WriteLine* следующим образом:

```
Console.WriteLine("{0} {1} {0}", j, i);
```

то на экран будет выведена строка: 4 3 4.

Данный вариант использования метода *WriteLine* является наиболее универсальным, потому что он позволяет не только выводить данные на экран, но и управлять форматом их вывода. Рассмотрим несколько примеров:

#### 1) Использование управляющих последовательностей:

Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой. Данная совокупность символов интерпретируется как одиночный символ и используется для представления кодов символов, не имеющих графического обозначения (например, символа перевода курсора на новую строку) или символов, имеющих специальное обозначение в символьных и строковых константах (например, апостроф). Рассмотрим управляющие символы:

Вид	Наименование
<code>\a</code>	Звуковой сигнал
<code>\b</code>	Возврат на шаг назад
<code>\f</code>	Перевод страницы
<code>\n</code>	Перевод строки
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция
<code>\v</code>	Вертикальная табуляция
<code>\\</code>	Обратная косая черта
<code>\'</code>	Апостроф
<code>\"</code>	Кавычки

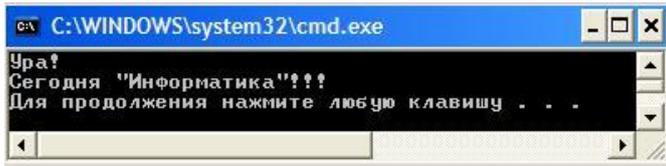
Пример:

```
static void Main()
{
```

```

Console.WriteLine("Ура!\nСегодня \"Информатика\"!!!");
}

```



*Замечание.* Вместо управляющей последовательности `\n` можно использовать константу `Environment.NewLine`. Она более универсальна, т.к. ее значение зависит от контекста и операционной системы, в которой запускается программа.

**Задание.** Измените программу так, чтобы все сообщение выводилось в одну строку, а после вывода сообщения раздавался звуковой сигнал.

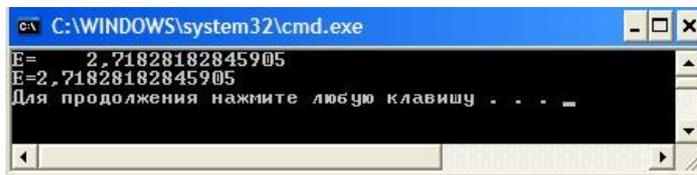
2) *Управление размером поля вывода:*

Первым аргументом `WriteLine` указывается строка вида `{n, m}` – где `n` определяет номер идентификатора из списка аргументов метода `WriteLine`, а `m` – количество позиций (размер поля вывода), отводимых под значение данного идентификатора. При этом значение идентификатора выравнивается по правому краю. Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций. Пример:

```

static void Main()
{
    double x= Math.E;
    Console.WriteLine("E={0,20}", x);
    Console.WriteLine("E={0,10}", x);
}

```



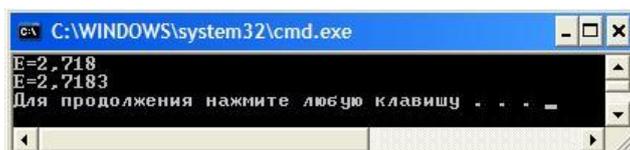
3) *Управление размещением вещественных данных:*

Первым аргументом `WriteLine` указывается строка вида `{n: ##.###}` – где `n` определяет номер идентификатора из списка аргументов метода `WriteLine`, а `##.###` определяет формат вывода вещественного числа. В данном случае, под целую часть числа отводится две позиции, под дробную – три. Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций. Пример:

```

static void Main()
{
    double x= Math.E;
    Console.WriteLine("E={0:##.###}", x);
    Console.WriteLine("E={0:.###}", x);
}

```



**Задание.** Измените программу так, чтобы число `e` выводилось на экран с точностью до 6 знаков после запятой.

#### 4) Управление форматом числовых данных:

Первым аргументом WriteLine указывается строка вида {n:<спецификатор>m} – где n определяет номер идентификатора из списка аргументов метода WriteLine, <спецификатор> - определяет формат данных, а m – количество позиций для дробной части значения идентификатора. В качестве спецификаторов могут использоваться следующие значения:

Параметр	Формат	Значение
C или c	Денежный. По умолчанию ставит денежный знак, определенный текущими региональными настройками. В русской Windows это р.	Задается количество десятичных разрядов.
D или d	Целочисленный (используется только с целыми числами)	Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями
E или e	Экспоненциальное представление чисел	Задается количество символов после запятой. По умолчанию используется значение 6.
F или f	Представление чисел с фиксированной точкой	Задается количество символов после запятой
G или g	Общий формат (или экспоненциальный, или с фиксированной точкой)	Задается количество символов после запятой. По умолчанию выводится целая часть
N или n	Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами	Задается количество символов после запятой. По умолчанию – 2, если число целое, то ставятся нули
X или x	Шестнадцатеричный формат	
P или p	Процентный	

Пример:

```
static void Main()
{
    Console.WriteLine("C Format:{0,14:C} \t{0:C2}", 12345.678);
    Console.WriteLine("D Format:{0,14:D} \t{0:D6}", 123);
    Console.WriteLine("E Format:{0,14:E} \t{0:E8}", 12345.6789);
    Console.WriteLine("G Format:{0,14:G} \t{0:G10}", 12345.6789);
    Console.WriteLine("N Format:{0,14:N} \t{0:N4}", 12345.6789);
    Console.WriteLine("X Format:{0,14:X} ", 1234);
    Console.WriteLine("P Format:{0,14:P} ", 0.9);
}
```

```
C:\WINDOWS\system32\cmd.exe
C Format: 12 345,68p.      12 345,68p.
D Format: 123             000123
E Format: 1,234568E+004   1,23456789E+004
G Format: 12345,6789      12345,6789
N Format: 12 345,68       12 345,6789
X Format: 4D2
P Format: 90,00%
Для продолжения нажмите любую клавишу . . .
```

#### Ввод данных

Для ввода данных обычно используется метод ReadLine, реализованный в классе Console. Данный метод в качестве результата возвращает строку, тип которой string. Пример:

```
static void Main()
{
    string s = Console.ReadLine();
    Console.WriteLine(s);
}
```

Для того чтобы получить числовое значение, необходимо воспользоваться преобразованием данных. Пример:

```
static void Main()
{
    string s = Console.ReadLine();
    int x = int.Parse(s); //преобразование строки в число
    Console.WriteLine(x);
}
```

Или сокращенный вариант:

```
static void Main()
{
    //преобразование введенной строки в число
    int x = int.Parse(Console.ReadLine());
    Console.WriteLine(x);
}
```

Для преобразования строкового представления целого числа в тип `int` мы используем метод `Parse()`, который реализован для всех числовых типов данных. Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом `float.Parse()` или `double.Parse()`. В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается и генерируется исключение. Например, если входная строка имела неверный формат, то будет сгенерировано исключение `System.FormatException`.

### **Задания.**

1. Подумайте, какие еще исключения могут возникнуть при использовании метода `Parse`. Проверьте свои предположения на практике.
2. Измените предыдущий фрагмент программы так, чтобы с клавиатуры вводилось вещественное число, а на экран это число выводилось с точностью до 3 знаков после запятой.

## **Операции**

Полный список операций языка `C#` в соответствии с их приоритетами (по убыванию приоритетов, операции с разными приоритетами разделены чертой) приведен в Приложении 1. В данном разделе мы подробно рассмотрим только часть операций, остальные операции будут вводиться по мере необходимости.

***Замечание.** Операции можно классифицировать по количеству операндов: унарные – воздействуют на один операнд, бинарные – воздействуют на два операнда, тернарные – воздействуют на три операнда. Некоторые символы используются для обозначения как унарных, так и бинарных операций. Например, символ «минус» используется как для обозначения унарной операции – арифметического отрицания, так и для обозначения бинарной операции вычитания. Будет ли данный символ обозначать унарную или бинарную операцию, определяется контекстом, в котором он используется.*

### 1. Инкремент (++) и декремент(--).

Эти операции имеют две формы записи — *префиксную*, когда операция записывается перед операндом, и *постфиксную* – операция записывается после операнда. Префиксная операция инкремента (декремента) увеличивает (уменьшает) свой операнд и возвращает измененное значение как результат. Постфиксные версии инкремента и декремента возвращают первоначальное значение операнда, а затем изменяют его.

Рассмотрим эти операции на примере.

```
static void Main()
{
    int i = 3, j = 4;
    Console.WriteLine("{0} {1}", i, j);
    Console.WriteLine("{0} {1}", ++i, --j);
    Console.WriteLine("{0} {1}", i++, j--);
    Console.WriteLine("{0} {1}", i, j);
}
```

~~~~~  
Результат работы программы:

```
3 4
4 3
4 3
5 2
~~~~~
```

**Задание.** Выясните, допустимы ли следующие способы записи ++(++i), (i--)--, ++(i--) и т.д. И почему.

**2. Операция new.** Используется для создания нового объекта. С помощью ее можно создавать как объекты ссылочного типа, так и размерные, например:

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

### 3. Отрицание:

1. Арифметическое отрицание (-) – меняет знак операнда на противоположный.
2. Логическое отрицание (!) – определяет операцию инверсии для логического типа.

Рассмотрим эти операции на примере.

```
static void Main()
{
    int i = 3, j=-4;
    bool a = true, b=false;
    Console.WriteLine("{0} {1}", -i, -j);
    Console.WriteLine("{0} {1}", !a, !b);
}
```

~~~~~  
Результат работы программы:

```
-3 4
False True
~~~~~
```

**Задание.** Выясните, допустимы ли следующие способы записи !(-i), -(!a). И почему.

**4. Явное преобразование типа.** Используется для явного преобразования из одного типа в другой. Формат операции:

(<тип> <выражение>;

Рассмотрим эту операцию на примере.

```
static void Main()
```

```

{
    int i = -4;
    byte j = 4;
    int a = (int)j; //преобразование без потери точности
    byte b = (byte)i; //преобразование с потерей точности
    Console.WriteLine("{0} {1}", a, b);
}

```

~~~~~  
*Результат работы программы:*

4 252  
 ~~~~~

**Задание.** Объясните, почему операция (byte)i вместо ожидаемого значения -4 дала нам в качестве результата значение 252.

*Замечание.* В Pascal, C++ и других языках допускается неявное преобразование типов, которое в рамках предыдущего примера позволило бы записать:  $b=i$ . В этом случае происходит потеря точности вычислений, о чем компилятор либо "умалчивает", либо сообщает в виде предупреждения. Возможность неявного преобразования чревата вычислительными ошибками, которые очень трудно найти. Чтобы избежать подобных ошибок, в C# запрещены некоторые виды неявных преобразований. Более подробно преобразование типов мы рассмотрим в следующем разделе.

**5. Умножение (\*), деление (/) и деление с остатком (%).** Операции умножения и деления применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам. При этом тип результата равен «наибольшему» из типов операндов, но не менее int. Если оба операнда при делении целочисленные, то и результат тоже целочисленный.

Рассмотрим эти операции на примере.

```

static void Main()
{
    int i = 100, j = 15;
    double a = 14.2, b = 3.5;
    Console.WriteLine("{0} {1} {2}", i*j, i/j, i%j);
    Console.WriteLine("{0} {1} {2}", a * b, a / b, a % b);
}

```

~~~~~  
*Результат работы программы:*

1500 6 10  
 49.7 4.05714285714286 0.1999999999999999  
 ~~~~~

**Задания.**

1. Выполните фрагмент программы и объясните полученный результат:

```

double a=100, b=33;
Console.WriteLine(a/b);
double d=100/33;
Console.WriteLine(d);

```

2. Выясните, чему будет равен результат операции, и объясните, как получился данный результат:

а) 1.0/0; б) 1/0

**6. Сложение (+) и вычитание (-).** Операции сложения и вычитания применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам.

**7. Операции отношения** (<, <=, >, >=, ==, !=). Операции отношения сравнивают значения левого и правого операндов. Результат операции логического типа: true – если значения совпадают, false – в противном случае. Рассмотрим операции на примере:

```
static void Main()
{
    int i = 15, j = 15;
    Console.WriteLine(i<j); //меньше
    Console.WriteLine(i<=j); //меньше или равно
    Console.WriteLine(i>j); //больше
    Console.WriteLine(i>=j); //больше или равно
    Console.WriteLine(i==j); //равно
    Console.WriteLine(i!=j); //не равно
}
```

~~~~~  
*Результат работы программы:*

False  
 True  
 False  
 True  
 True  
 False  
 ~~~~~

**Задание.** Выясните, чему равен результат данного выражения:

- 1)  $10 < 25 < 30$     2)  $true < false$

И объясните, как получился данный результат.

**8. Логические операции:** И (&&), ИЛИ (||).

Логические операции применяются к операндам логического типа.

Результат логической операции И имеет значение истина тогда и только тогда, когда оба операнда принимают значение истина.

Результат логической операции ИЛИ имеет значение истина тогда и только тогда, когда хотя бы один из операндов принимает значение истина.

Рассмотрим операции на примере:

```
static void Main()
{
    Console.WriteLine("x    y    x и y    x или y");
    Console.WriteLine("{0} {1} {2} {3}", false, false, false&&false, false||false);
    Console.WriteLine("{0} {1} {2} {3}", false, true, false&&true, false||true);
    Console.WriteLine("{0} {1} {2} {3}", true, false, true&&false, true||false);
    Console.WriteLine("{0} {1} {2} {3}", true, true, true&&true, true||true);
}
```

~~~~~  
*Результат работы программы:*

x    y    x и y    x или y  
 False False False False  
 False True False True  
 True False False True  
 True True True True  
 ~~~~~

*Замечание.* Фактически была построена таблица истинности для логических операций И и ИЛИ.

**Задание.** Объясните, какое значение примет переменная t в данном фрагменте программы:

```
int a=10, b=3;
```

```
bool t=(a>=b && a!=2*b || a<0);
```

### 9. Условная операция.

Формат: (<операнд1>)? <операнд2> : <операнд3>;

Операнд1 – это логическое выражение, которое оценивается с точки зрения его эквивалентности константам *true* и *false*. Если результат вычисления операнда1 равен *true*, то результатом условной операции будет значение операнда2, иначе — операнда3. Фактически условная операция является сокращенной формой условного оператора *if*, который будет рассмотрен позже.

Пример использования условной операции:

```
static void Main()  
{  
    int x=5; int y=10;  
    int max = (x > y) ? x : y;  
    Console.WriteLine(max);  
}
```

**Задание.** Измените программу так, чтобы:

- 1) вычислялось наименьшее значение из двух вещественных чисел *x* и *y*;
- 2) если число двузначное, то на экран выводилось «Да», и «Нет» в противном случае.

### 10. Операции присваивания: =, +=, -= и т.д.

Формат операции простого присваивания (=):

операнд\_2 = операнд\_1;

В результате выполнения этой операции вычисляется значение операнда\_1, и результат записывается в операнд\_2. Можно связать воедино сразу несколько операторов присваивания, записывая такие цепочки: *a=b=c=100*. Выражение такого вида выполняется справа налево: результатом выполнения *c=100* является число 100, которое затем присваивается переменной *b*, результатом чего опять является 100, которое присваивается переменной *a*.

Кроме простой операции присваивания существуют *сложные операции присваивания*, например, умножение с присваиванием (*\*=*), деление с присваиванием (*/=*), остаток от деления с присваиванием (*%=*), сложение с присваиванием (*+=*), вычитание с присваиванием (*-=*) и т.д.

В сложных операциях присваивания, например, при *сложении с присваиванием*, к операнду\_2 прибавляется операнд\_1, и результат записывается в операнд\_2. То есть, выражение *c += a* является более компактной записью выражения *c = c + a*.

**Задание.** Объясните, какие значения примут переменные *t* и *b* после выполнения данного фрагмента программы:

```
int a=10, b=3;  
int t=(a++)-b;  
b+=t*a;
```

## Выражения и преобразование типов

Выражение – это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения состоят из операндов, операций и скобок. Каждый операнд является в свою очередь выражением или одним из его частных случаев – константой, переменной или функций.

**Замечание.** Список математических функций, реализованных в C# приведен в приложении 2.

Примеры выражений:

$(a + 0.12)/6$        $x \ \&\& \ y \ || \ !z$        $(t * \text{Math.Sin}(x) - 1.05e4) / ((2 * k + 2) * (2 * k + 3))$

Вычисление значения выражения происходит с учетом приоритета операций (см. Приложение 1), которые в нем участвуют. Если в выражении соседствуют операции одного приоритета, то унарные операции, условная операция и операции присваивания выполняются *справа налево*, остальные — *слева направо*. Например,

$a = b = c$       означает       $a=(b=c)$ ,  
 $a+b+c$       означает       $(a + b) + c$ .

Если необходимо изменить порядок выполнения операций, то в выражении необходимо поставить круглые скобки.

### Задания

1. Укажите последовательность выполнения операций в данном выражении:  
 $(x*x+\text{Math.Sin}(x+1))/x-2$ .
2. Запишите заданное математическое выражение по правилам языка C#:

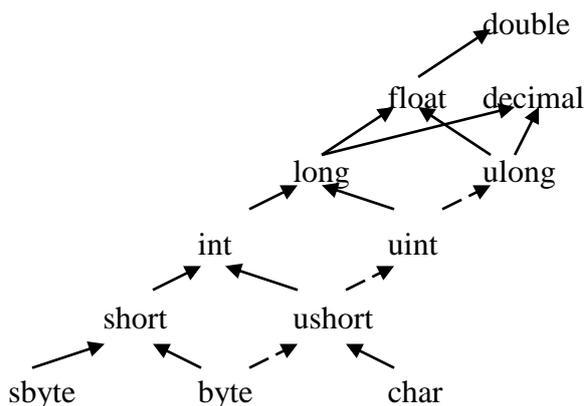
a)  $\sqrt{x^4 + \sqrt{|x+1|}}$ ;    b)  $\frac{a^2 + b^2}{1 - \frac{a^3 - b}{3}}$ ;    c)  $\ln\left(y - \sqrt{|x|}\right)\left(x - \frac{y}{x + \frac{x^2}{4}}\right)$

Результат вычисления выражения характеризуется значением и типом. Например, если  $a$  и  $b$  — переменные целого типа и описаны так:

`int a = 2, b = 5;`

то выражение  $a + b$  имеет значение 7 и тип `int`.

В выражение могут входить операнды различных типов. Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип. Если операнды разного типа, то перед вычислениями выполняются преобразования более коротких типов в более длинные для сохранения значимости и точности. Иерархия типов данных приведена в следующей схеме:



Преобразование типов в выражениях происходит *неявно* (без участия программистов) следующим образом. Если один из операндов имеет тип, изображенный на более низком уровне, чем другой, то он приводится к типу второго операнда при наличии пути между ними. Если пути нет, то возникает ошибка компиляции (чтобы ее избежать, необходимо воспользоваться операцией явного преобразования). Если путей преобразования несколько, то выбирается наиболее короткий, не содержащий пунктирных линий.

Вернемся к рассмотрению операции явного преобразования из предыдущего раздела:

```
static void Main()
{
    int i = -4;
    byte j = 4;
    int a = (int)j; //1
    byte b = (byte)i; //2
    Console.WriteLine("{0} {1}", a, b);
}
```

В строке 1 можно обойтись без явного преобразования типа, т.е. можно записать  $a=j$ , т.к. тип `int` в иерархии типов находится выше типа `byte` и существует путь для неявного преобразования типа `byte` в тип `int`. Но пути для неявного преобразования от типа `int` к типу `byte` нет, поэтому в строке 2 нельзя записать  $b=i$  – компилятор выдаст сообщение об ошибке.

**Замечание.** Рассмотрим один важный пример. В C++ допустим следующий фрагмент кода:

```
int x=10;
int y= (x)? 1: 2; // 3
```

В данном примере в строке 3 происходит неявное преобразование типа переменной  $x$ , т.е. типа `int`, к логическому типу. При этом, если  $x$  принимает значение 0 (или `null` для ссылочных типов), то ему ставится в соответствие логическая величина `false`, всем другим значениям в соответствие ставится логическое значение `true`. В нашем случае переменная  $x$  принимает ненулевое значение, поэтому оно будет преобразовано к значению `true`, и в переменную  $y$  запишется значение 1.

В языке C# подобное неявное преобразование невозможно. Необходимо обязательно выполнять операцию сравнения значения  $x$  с 0. Например, следующим образом:

```
int x=10;
int y= (x!=0)? 1: 2; // 3
```

## Примеры решения практических задач

1. Написать программу, подсчитывающую площадь квадрата, периметр которого равен  $p$ .

**Указания по решению задачи.** Прежде чем составить программу, проведем математические рассуждения. Пусть дан квадрат со стороной  $a$ , тогда:

$$\begin{aligned} \text{периметр вычисляется по формуле } p=4a & \Rightarrow a = \frac{p}{4} \\ \text{площадь вычисляется по формуле } s=a^2 & \Rightarrow a = \sqrt{s} \end{aligned} \Rightarrow \frac{p}{4} = \sqrt{s} \Rightarrow s = \left(\frac{p}{4}\right)^2$$

```
using System;
namespace Example
{
    class Program
    {
        static void Main()
        {
            Console.Write("p= ");
            double p = double.Parse(Console.ReadLine());
            double s = Math.Pow(p/4, 2);
            Console.WriteLine("s={0}",s);
        }
    }
}
```

```

    }
}
.....
Результат работы программы:
p=20
s=25
.....

```

**Задание.** Изменить программу так, чтобы она подсчитывала периметр квадрата, площадь которого равна s

2. Определить, является ли сумма цифр натурального двухзначного числа четной.

*Указание по решению задачи.* Напомним, что число является четным, если остаток от деления данного числа на 2 равен нулю. А для того, чтобы разложить двухзначное число на цифры, нужно разделить его нацело на 10 – так мы найдем старшую цифру числа, а затем взять остаток от деления на 10 исходного числа – так мы найдем его младшую цифру.

```

using System;
namespace Example
{
    class Program
    {
        static void Main()
        {
            Console.Write("a= ");
            byte a = byte.Parse(Console.ReadLine());
            string result=((a/10+a%10)%2==0)? "четное": "нечетное";
            Console.WriteLine(result);
        }
    }
}

```

*Результат работы программы:*

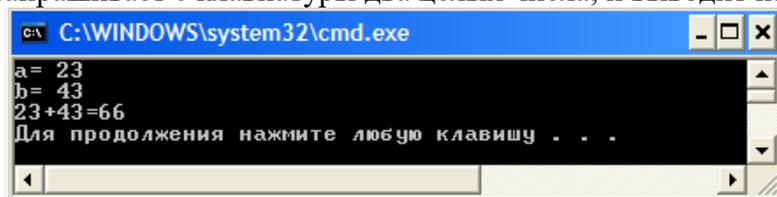
| x  | Сообщение на экране |
|----|---------------------|
| 45 | нечетное            |
| 88 | четное              |
| 55 | четное              |

**Задание.** Изменить программу так, чтобы она определяла, является ли сумма цифр трехзначного числа нечетной.

### Задания для самостоятельной работы

I. Написать программу, которая, реализует диалог с пользователем:

1) запрашивает с клавиатуры два целых числа, и выводит на экран сумму данных чисел:



2) запрашивает с клавиатуры два целых числа, и выводит на экран сумму данных чисел в прямом и обратном порядке:

```
C:\WINDOWS\system32\cmd.exe
a= 12
b= 34
12+34=34+12
Для продолжения нажмите любую клавишу . . .
```

3) запрашивает с клавиатуры три целых числа, и выводит на экран сумму данных чисел:

```
C:\WINDOWS\system32\cmd.exe
a= 24
b= 5
c= 36
24+5+36=65
Для продолжения нажмите любую клавишу . . .
```

4) запрашивает с клавиатуры два вещественных числа, и выводит на экран произведение данных чисел (вещественные числа выводятся с точностью до 1 знака после запятой):

```
C:\WINDOWS\system32\cmd.exe
a= 3,45
b= 12,1
3,5*12,1=41,7
Для продолжения нажмите любую клавишу . . .
```

5) запрашивает с клавиатуры два вещественных числа, и выводит на экран результат деления первого числа на второе (вещественные числа выводятся с точностью до 3 знаков после запятой):

```
C:\WINDOWS\system32\cmd.exe
a= 345,6
b= 12,345
345,600/12,345=27,995
Для продолжения нажмите любую клавишу . . .
```

6) запрашивает с клавиатуры три вещественных числа, и выводит на экран следующее сообщение (вещественные числа выводятся с точностью до 2 знаков после запятой):

```
C:\WINDOWS\system32\cmd.exe
a= 12,4
b= 2,567
c= 100
(12,40+2,57)+100,00=12,40+(2,57+100,00)
Для продолжения нажмите любую клавишу . . .
```

7) запрашивает с клавиатуры номинал купюры и количество купюр, и выводит экран следующее сообщение:

```
D:\c\PRIM\ConsoleApplication2\bin\Debug\Cons...
Номинал купюры = 100
Количество купюр = 23
Сумма денег = 2 300,00р.
Press any key to continue
```

8) запрашивает с клавиатуры сумму вклада и процент по вкладу, и выводит на экран следующее сообщение (вклад без капитализации – все начисления в конце года):

```
с:\ "D:\c\PRIM\ConsoleApplication2\bin\Debug\Console...
Введите сумму вклада= 12345,67
Введите процент по вкладу = 14,5
Через год начислят 1 790,12р.
Press any key to continue.
```

9) запрашивает с клавиатуры сумму вклада и процент по вкладу, и выводит на экран следующее сообщение (вклад без капитализации – все начисления в конце года):

```
с:\ "D:\c\PRIM\ConsoleApplication2\bin\Debug\ConsoleApplica...
Введите сумму вклада= 12345,67
Введите процент по вкладу = 14,5
Через год сумма на вкладе =14 135,79р.
Press any key to continue.
```

10) запрашивает с клавиатуры имя человека и его возраст, и выводит на экран следующее сообщение (текущий год – 2009):

```
с:\ "D:\c\PRIM\ConsoleApplication2\bin\Debug\...
Как тебя зовут? Вася
Сколько тебе лет? 15
Вася, ты родился в 1994 году.
Press any key to continue
```

II. Написать программу, которая подсчитывает:

- 1) площадь равностороннего треугольника, периметр которого равен  $p$ ;
- 2) расстояние между точками с координатами  $x_1, y_1$  и  $x_2, y_2$ ;
- 3) гипотенузу прямоугольного треугольника по двум данным катетам  $a, b$ .
- 4) площадь прямоугольного треугольника по двум катетам  $a, b$ .
- 5) периметр прямоугольного треугольника по двум катетам  $a, b$ .
- 6) ребро куба, площадь полной поверхности которого равна  $s$ ;
- 7) ребро куба, объем которого равен  $v$ ;
- 8) периметр треугольника, заданного координатами вершин  $x_1, y_1, x_2, y_2, x_3, y_3$ ;
- 9) площадь треугольника, заданного координатами вершин  $x_1, y_1, x_2, y_2, x_3, y_3$ ;
- 10) сумму членов арифметической прогрессии, если известен ее первый член, разность и число членов прогрессии

III. Написать программу, которая определяет:

- 1) наибольшую цифру в натуральном двухзначном числе;
- 2) наименьшую цифру в натуральном двухзначном числе;
- 3) одинаковы ли цифры данного двухзначного числа;
- 4) заканчивается ли сумма цифр двухзначного числа на 0;
- 5) кратна ли трем сумма цифр двухзначного числа;
- 6) кратна ли числу  $A$  сумма цифр двухзначного числа;
- 7) какая из цифр трехзначного числа больше: первая или последняя;
- 8) какая из цифр трехзначного числа больше: первая или вторая;
- 9) какая из цифр трехзначного числа больше: вторая или последняя;
- 10) все ли цифры трехзначного числа одинаковые.