

Объектно-ориентированное программирование

1. Объектно-ориентированное программирование
2. Инкапсуляция
3. Полиморфизм
4. Наследование
5. Простая программа
6. Инструкции управления

- класс,
- объект,
- интерфейс,
- инкапсуляция,
- наследование,
- полиморфизм,
- событие

Объект в программе

— это абстракция реального объекта.

Объект обладает атрибутами, поведением и индивидуальностью.

Атрибуты определяют основные черты объекта, поведение — действия над объектом, индивидуальность — отличие одного объекта от другого с такими же атрибутами по их конкретным значениям.

Класс

– это множество объектов с одинаковыми атрибутами и поведением, представляемое в языке программирования в виде абстрактного типа данных, который включает в себя члены класса.

Члены класса

- *поля* – непосредственно данные определенного типа для описания атрибутов;
- *методы* - функции, предназначенные для обработки внутренних данных объекта данного класса;
- *свойства* – это специальные поля данных, с помощью которых, можно управлять поведением объектов данного класса.

- Детали реализации объекта, то есть внутренние структуры данных и алгоритмы их обработки, скрыты от пользователя и недоступны для непреднамеренного изменения.
- Объект используется через его *интерфейс* - совокупность правил доступа.
- Скрытие деталей реализации называется *инкапсуляцией*.

- В ООП данные и методы одного класса могут передаваться другим классам с помощью механизма *наследования*. Порожденный класс (потомок), наследующий характеристики другого класса, обладает теми же возможностями, что и класс (предок), от которого он порожден. При этом класс-предок остается без изменения, а классу-потомку можно добавлять новые элементы (поля, методы, свойства) или изменять унаследованные методы.

- Классы-потомки некоторого класса являются разновидностями этого класса-предка. Т.е. к объектам классов-потомков можно обращаться с помощью одного и того же имени (но при этом могут выполняться различные действия) — что составляет суть *полиморфизма*.

Инкапсуляция — это

- механизм программирования, который связывает код (действия) и данные, которыми он манипулирует, и при этом предохраняет их от вмешательства извне и неправильного использования.
- В объектно-ориентированном языке код и данные можно связать таким образом, что будет создан автономный *черный ящик*. Внутри этого ящика находятся все необходимые данные и код.
- При таком связывании кода и данных создается *объект* (объект — это элемент, который поддерживает инкапсуляцию).

- Код, данные могут быть *закрытыми* или *открытыми*
- Основной единицей инкапсуляции в С# является *класс*. Класс определяет форму объекта (задает как данные, так и код, который будет оперировать этими данными).
- Объекты — это экземпляры класса.
- Код и данные, которые составляют класс, называются *членами* класса.
- Данные, определенные в классе, называются *переменными экземпляра* (instance variable), а код, который оперирует этими данными, — *методами-членами* (member method), или просто *методами* (в С# - подпрограмма, в С/С++ - функция).

Полиморфизм (от греческого слова *polymorphism*, означающего "много форм") — это

- качество, которое позволяет одному интерфейсу получать доступ к целому классу действий.
- Концепция полиморфизма : "один интерфейс — много методов" (для выполнения группы подобных действий можно разработать общий интерфейс).
- Полиморфизм позволяет понизить степень сложности программы, предоставляя программисту возможность использовать один и тот же интерфейс для задания *общего класса действий*. Конкретное (нужное в том или ином случае) действие (метод) выбирается компилятором.

Наследование — это

- процесс, благодаря которому один объект может приобретать свойства другого.
- Благодаря наследованию поддерживается концепция иерархической классификации.
- Благодаря наследованию объекту нужно доопределить только те качества, которые делают его уникальным внутри его класса, поскольку он (объект) наследует общие атрибуты своего родителя.
- Механизм наследования позволяет одному объекту представлять конкретный экземпляр более общего класса.

```
using System;

class Ex
{
//программа начинается с вызова метода Main()
    public static void Main()
    {
        int x=100; // объявляется переменная
        int y; // объявляется переменная

        //x = 100; // переменной x присваивается 100
        Console.WriteLine("x содержит " + x);

        y = x / 2;

        Console.Write("y содержит x / 2: ");
        Console.WriteLine(y);
    }
}
```

Переменная — это именованная область памяти, которой может быть присвоено определенное значение.

Типы значений в C#

- C# содержит две категории встроенных типов данных:
 - *типы значений*
 - *Ссылочные типы* (определяются в классах)

Термин "тип значения" применяется к переменным, которые непосредственно содержат значения.

переменные ссылочных типов содержат ссылки на реальные значения

Типы значений в C#

<i>Ключевое слово</i>	<i>ТИП</i>
<code>bool</code>	Логический, представляет значения ИСТИНА/ЛОЖЬ
<code>char</code>	Символьный (16-ти разрядный)
<code>decimal</code>	Числовой тип для финансовых вычислений
<code>double</code>	С плавающей точкой двойной точности
<code>float</code>	С плавающей точкой
<code>int</code>	Целочисленный
<code>byte</code>	8-разрядный целочисленный без знака
<code>long</code>	Тип для представления длинного целого числа
<code>sbyte</code>	8-разрядный целочисленный со знаком
<code>short</code>	Тип для представления короткого целого числа
<code>uint</code>	Целочисленный без знака
<code>ulong</code>	Тип для представления длинного целого числа без знака
<code>ushort</code>	Тип для представления короткого целого числа без знака

ВАРИАНТЫ ВЫВОДА ДАННЫХ

```
Console.WriteLine("Вы заказали "+2+"  
предмета по $" +3+" каждый.");
```

```
Console.WriteLine("В феврале {0} или  
{1} дней.", 28, 29);
```

В феврале 28 или 29 дней.

```
Console.WriteLine(  
"В феврале {0,10} или {1,5} дней.",  
28, 29);
```

В феврале 28 или 29 дней.

```
Console.WriteLine(  
"При делении 10/3 получаем: " + 10.0/3.0);
```

При делении 10/3 получаем: 3.3333333333333333

```
Console.WriteLine(  
"При делении 10/3 получаем: {0:#.##}",10.0/3.0);
```

При делении 10/3 получаем: 3.33

```
Console.WriteLine("{0:###,###.##}",123456.56);
```

123,456.56

```
decimal balance;  
balance = 12323.09m;  
Console.WriteLine("Текущий баланс равен {0:C},balance);
```

Текущий баланс равен \$12,323.09

Управляющие последовательности СИМВОЛОВ

<code>\a</code>	Звуковой сигнал (звонок)
<code>\b</code>	Возврат на одну позицию
<code>\f</code>	Подача страницы (для перехода к началу следующей страницы)
<code>\n</code>	Новая строка
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция
<code>\v</code>	Вертикальная табуляция
<code>\0</code>	Нуль-символ
<code>\'</code>	Одинарная кавычка (апостроф)
<code>\"</code>	Двойная кавычка
<code>\\</code>	Обратная косая черта

Литералы - фиксированные значения, представленные в понятной форме (100)

Литерал типа `long` - присоединить к его концу букву `l` или `L`. `12` автоматически приобретает тип `int`, но значение `12L` имеет тип `long`.

Целочисленное значение без знака - суффикс `u` или `U`.
`100` имеет тип `int`, но значение `100U` - тип `uint`.

Длинное целое без знака - суффикс `ul` или `UL` (например, значение `987 654UL` будет иметь тип `ulong`).

Литерал типа `float` - суффикс `f` или `F` (например, `10.19F`).

Литерал типа `decimal` - суффикс `m` или `M` (например, `9. 95M`).

Шестнадцатеричные литералы

`count = 0xFF; // 255 в десятичной системе`

`incr = 0x1a; // 26 в десятичной системе`

Инициализация переменной

```
int count = 10 ; // Присваиваем переменной count  
                // начальное значение 10
```

```
char ch = 'X' ; // Инициализируем ch буквой X
```

```
float f = 1.2F // Переменная f инициализируется  
              // числом 1.2
```

```
int a,b=8,c=19,d; // Переменные b и c  
                  // инициализируются числами.
```

Динамическая инициализация

// Динамическая инициализация

```
using System;
```

```
class DynInit
```

```
{
```

```
    public static void Main()
```

```
    {
```

```
        double s1 = 4.0, s2 = 5.0; // длины строк
```

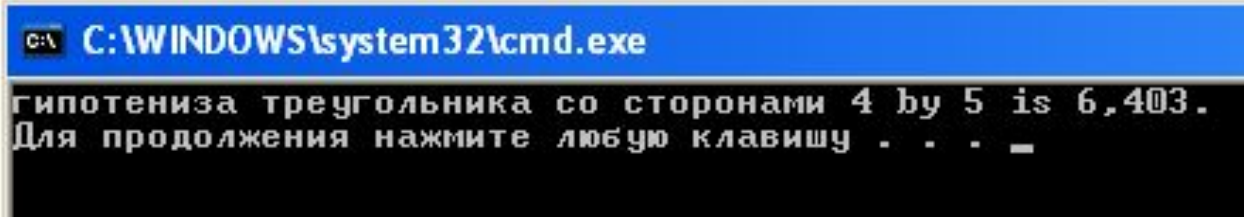
```
        // Динамически инициализируем hypot
```

```
        double hypot = Math.Sqrt((s1 * s1) + (s2 * s2));
```

```
        Console.Write("гипотениза треугольника со  
сторонами " + s1 + " by " + s2 + " is ");
```

```
        Console.WriteLine("{0:###.###}.", hypot);
```

```
    } }
```



C:\WINDOWS\system32\cmd.exe

гипотениза треугольника со сторонами 4 by 5 is 6,403.
Для продолжения нажмите любую клавишу . . .

Область видимости

- При создании блока создается и новая область видимости, которая определяет, какие объекты видимы для других частей программы. Область видимости также определяет время существования этих объектов.
- При объявлении переменной внутри области видимости мы локализуем ее и защищаем от неправомерного доступа и/или модификации(основа для инкапсуляции)


```
// демонстрация области видимости блока

using System;

class ScopeDemo {
    public static void Main() {
        int x; // известна всему коду в пределах метода Main()

        x = 10;
        if(x == 10) { // начало новой области видимости
            int y = 20; // известно только этому блоку

            // x и y известны
            Console.WriteLine("x и y: " + x + " " + y);
            x = y * 2;
        }
        // y = 100; // ошибка! y неизвестна

        // x известна
        Console.WriteLine("x равно " + x);
    }
}
```

Время существования переменных

- Переменные создаются после входа в их область видимости, а разрушаются при выходе из нее.
- Переменная, объявленная внутри некоторого метода, не будет хранить значение между вызовами этого метода.
- Время существования переменной ограничивается ее областью видимости.
- Если объявление переменной включает инициализатор, такая переменная будет повторно инициализироваться при каждом входе в блок, в котором она объявляется.

```
using System;
```

```
class VarInitDemo
```

```
{
```

```
public static void Main()
```

```
{
```

```
    int x;
```

```
    for (x = 0; x < 3; x++)
```

```
    {
```

```
        int y = -1; // y инициализируется при  
        //каждом входе в программный блок
```

```
        Console.WriteLine("y равно: " + y);
```

```
// всегда выводиться -1
```

```
        y = 100;
```

```
        Console.WriteLine("y теперь равно: "+y);
```

```
    }
```

```
}
```

```
}
```

C:\WINDOWS\system32\cmd.exe

y равно: -1

y теперь равно: 100

y равно: -1

y теперь равно: 100

y равно: -1

y теперь равно: 100

Для продолжения нажмите любую клавишу . . .

Преобразование и приведение типов

- автоматическое преобразование типов, выполняется если эти типы совместимы, тип приемника больше (т. е. имеет больший диапазон представления чисел), чем тип источника.

```
using System;
class LtoD {
    public static void Main() {
        long L;
        double D;
        //D=100123456.0
        //L=D; //неверно!!!
        L=100123285L;
        D=L;
        Console.WriteLine("L и D:" +L+ " "+D) ;
    }
}
```

Приведение несовместимых типов

- *Приведение к типу* — это явно заданная инструкция компилятору преобразовать один тип в другой

(тип_приемника) выражение

тип_приемника - тип для преобразования заданного выражения

```
double x, y;  
// . . .  
(int) (x / y) ;
```

```
// приведение типов
using System;
class CastDemo
{
    public static void Main()
    {
        double x, y;
        byte b;
        int i;
        char ch;
        uint u;
        short s;
        long l;
        x = 10.0;
        y = 3.0;
        //приведение типа double в int
        i = (int)(x / y); // дробная часть теряется
        Console.WriteLine("целочисленный результат
деления x / y: " + i);
        Console.WriteLine();
    }
}
```

```
//приведение типа int к byte, без потери дан.
```

```
    i = 255;
```

```
    b = (byte)i;
```

```
    Console.WriteLine("b после  
присваивания 255: "+b+" -- без потери  
данных");
```

```
//приведение типа int к byte, с потерей дан.
```

```
    i = 257;
```

```
    b = (byte)i;
```

```
    Console.WriteLine("b после  
присваивания 257: "+b+" -- с потерей  
данных");
```

```
    Console.WriteLine();
```



```
//приведение типа uint к short, без потери дан.
    u = 32000;
    s = (short)u;
    Console.WriteLine("s после присваивания
32000: "+s+ "--без потери данных");
//приведение типа uint к short, с потерей дан.
    u = 64000; s = (short)u;
    Console.WriteLine("s  после присваивания
64000: " + s + " -- с потерей данных");
    Console.WriteLine();
//приведение типа long к uint, без потери дан.
    l = 64000; u = (uint)l;
    Console.WriteLine("u  после присваивания
64000: " + u +" -- без потери данных");
```

```
//приведение типа long к uint, с потерей дан.
    l = -12;
    u = (uint)l;
    Console.WriteLine("u после присваивания
-12: " + u + " -- с потерей данных");
    Console.WriteLine();

    // приведение типа int к char
    b = 88; // ASCII-код для буквы X
    ch = (char)b;
    Console.WriteLine("ch после
присваивания 88: " + ch);
}
}
```

C:\Windows\system32\cmd.exe

целочисленный результат деления x / y : 3

b после присваивания 255: 255 -- без потери данных

b после присваивания 257: 1 -- с потерей данных

s после присваивания 32000: 32000 -- без потери данных

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

u после присваивания 64000: 64000 -- без потери данных

u после присваивания -12: 4294967284 -- с потерей данных

ch после присваивания 88: X

Преобразование типов в выражениях

- Преобразование типов выполняется на основе *правил продвижения по "типовой" лестнице*.
- Правило продвижения типов действует только при вычислении выражения.
- Для бинарных операций:

- ЕСЛИ один операнд имеет тип `decimal`, ТО и второй "возводится в ранг", т.е. "в тип" `decimal` (но если второй операнд имеет тип `float` или `double`, результат будет ошибочным).
- ЕСЛИ один операнд имеет тип `double`, ТО и второй преобразуется в значение типа `double`.
- ЕСЛИ один операнд имеет тип `float`, ТО и второй преобразуется в значение типа `float`.
- ЕСЛИ один операнд имеет тип `ulong`, ТО и второй преобразуется в значение типа `ulong` (но если второй операнд имеет тип `sbyte`, `short`, `int` или `long`, результат будет ошибочным).
- ЕСЛИ один операнд имеет тип `long`, ТО и второй преобразуется в значение типа `long`.
- ЕСЛИ один операнд имеет тип `uint`, а второй имеет тип `sbyte`, `short` или `int`, ТО оба операнда преобразуются в значения типа `long`.
- ЕСЛИ один операнд имеет тип `uint`, ТО и второй преобразуется в значение типа `uint`.
- ИНАЧЕ оба операнда преобразуются в значения типа `int`.

Приведение типов в выражениях

- Операцию приведения типов можно применить не ко всему выражению, а к конкретной его части.

```
//приведение типов в выражениях
using System;
class CastExpr
{
    public static void Main()
    {
        double n;
        for (n = 1.0; n <= 10; n++)
        {
            Console.WriteLine("квадратный корень из {0}
равен {1}", n, Math.Sqrt(n));
            Console.WriteLine("целая часть числа: {0}",
                (int)Math.Sqrt(n));
            Console.WriteLine("дробная часть числа:
{0}", Math.Sqrt(n) - (int)Math.Sqrt(n));
            Console.WriteLine();
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
квадратный корень из 1 равен 1  
целая часть числа: 1  
дробная часть числа: 0  
  
квадратный корень из 2 равен 1,4142135623731  
целая часть числа: 1  
дробная часть числа: 0,414213562373095  
  
квадратный корень из 3 равен 1,73205080756888  
целая часть числа: 1  
дробная часть числа: 0,732050807568877  
  
квадратный корень из 4 равен 2  
целая часть числа: 2  
дробная часть числа: 0  
  
квадратный корень из 5 равен 2,23606797749979  
целая часть числа: 2  
дробная часть числа: 0,23606797749979  
  
квадратный корень из 6 равен 2,44948974278318  
целая часть числа: 2  
дробная часть числа: 0,449489742783178  
  
квадратный корень из 7 равен 2,64575131106459  
целая часть числа: 2  
дробная часть числа: 0,645751311064591  
  
квадратный корень из 8 равен 2,82842712474619  
целая часть числа: 2  
дробная часть числа: 0,82842712474619  
  
квадратный корень из 9 равен 3  
целая часть числа: 3  
дробная часть числа: 0  
  
квадратный корень из 10 равен 3,16227766016838  
целая часть числа: 3  
дробная часть числа: 0,16227766016838
```

Для продолжения нажмите любую клавишу . . .

Арифметические операторы

Оператор Действие

+	Сложение
-	Вычитание, унарный минус
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

Операторы отношений

Оператор *Значение*

== Равно

!= Не равно

> Больше

< Меньше

>= Больше или равно

<= Меньше или равно

логические операторы

<i>Оператор</i>	<i>Значение</i>
&	И
	ИЛИ
^	Исключающее ИЛИ
&&	Сокращенное И
	Сокращенное ИЛИ
!	НЕ

различие между обычной и сокращенной версиями

при использовании обычной операции всегда вычисляются оба операнда, в случае же сокращенной версии второй операнд вычисляется только при необходимости.

Поразрядные операторы

<i>Оператор</i>	<i>Значение</i>
&	Поразрядное И
	Поразрядное ИЛИ
^	Поразрядное исключающее ИЛИ
>>	Сдвиг вправо
<<	Сдвиг влево
~	Дополнение до 1 (унарный оператор НЕ)

значение >> *число битов*;

значение << *число_битов*.

- *значение* — это объект операции сдвига, а элемент *число_битов* указывает, на сколько разрядов должно быть сдвинуто значение.

```
// использование операторов сдвига
//для умножения на 2
using System;
class MultDiv
{
    public static void Main()
    {
        int n;
        n = 10;
        Console.WriteLine("значение переменной n: " + n);
        // умножаем 2
        n = n << 1;
        Console.WriteLine("значение переменной после
n = n * 2: " + n);
        // умножаем 4
        n = n << 2;
        Console.WriteLine("значение переменной после
n = n * 4: " + n);
        // делим на 2
        n = n >> 1;
    }
}
```

```
Console.WriteLine("значение переменной после n = n
/ 2: " + n);
    // делим на 4
    n=n>>2;
    Console.WriteLine("значение переменной после
n = n / 4: " + n);
    Console.WriteLine();
    // reset n
    n=10;
    Console.WriteLine("значение переменной n: "
+ n);
    // умножаем 2, 30 раз
    n=n<<30; // данные потеряны
    Console.WriteLine("значение n после сдвига
на 30 разрядов: " + n);
}
}
```

C:\Windows\system32\cmd.exe

значение переменной n: 10

значение переменной после $n = n * 2$: 20

значение переменной после $n = n * 4$: 80

значение переменной после $n = n / 2$: 40

значение переменной после $n = n / 4$: 10

значение переменной n: 10

значение n после сдвига на 30 разрядов: -2147483648

Для продолжения нажмите любую клавишу . . .

Цикл for

*For (инициализация; условие; итерация)
инструкция;*


```
using System;
```

```
class For
```

```
{
```

```
    public static void Main()
```

```
    {
```

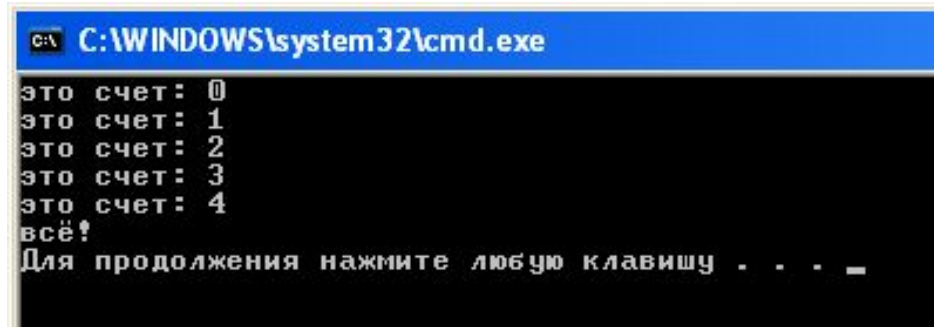
```
        int count;
```

```
        for (count = 0; count < 5; count ++)  
            Console.WriteLine("это счет:"+count);
```

```
        Console.WriteLine("всё!");
```

```
    }
```

```
}
```



```
C:\WINDOWS\system32\cmd.exe  
это счет: 0  
это счет: 1  
это счет: 2  
это счет: 3  
это счет: 4  
всё!  
Для продолжения нажмите любую клавишу . . . _
```

```
// сумма и произведение от 1 до 10.
```

```
using System;
```

```
class PrSum
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        int pr, sum, i;
```

```
        sum = 0;
```

```
        pr = 1;
```

```
        for (i = 1; i <= 10; i++)
```

```
        {
```

```
            sum = sum + i;
```

```
            pr = pr * i;
```

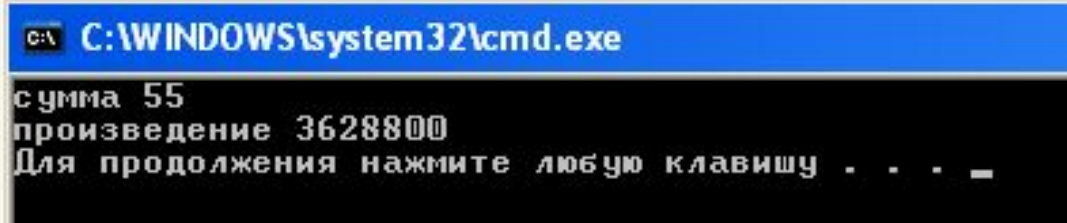
```
        }
```

```
        Console.WriteLine("сумма " + sum);
```

```
        Console.WriteLine("произведение " + pr);
```

```
    }
```

```
}
```



C:\WINDOWS\system32\cmd.exe

сумма 55

произведение 3628800

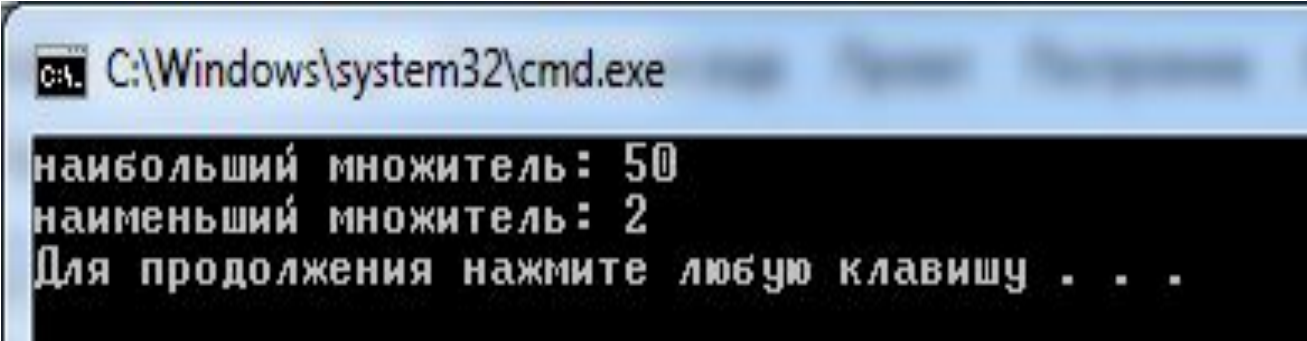
Для продолжения нажмите любую клавишу . . . _

- Программные блоки не снижают динамику выполнения программ, т.е. наличие фигурных скобок { и } не означает дополнительных затрат времени на выполнение программы.
- Благодаря способности блоков кода упрощать программирование алгоритмов, повышается скорость и эффективность выполнения программ в целом.

```
/* использование запятых в цикле for для
определения наибольшего и наименьшего
множителей числа*/
using System;
class Comma
{
    public static void Main()
    {
        int i, j;
        int smallest, largest;
        int num;
        num = 100;
        smallest = largest = 1;
        for (i=2, j=num/2; (i<=num/2) & (j>=2); i++, j)
        {
```

```
        if ((smallest==1) & ((num%i)==0))
            smallest = i;

        if ((largest==1) & ((num%j)==0))
            largest = j;
    }
    Console.WriteLine("наибольший
множитель: " + largest);
    Console.WriteLine("наименьший
множитель: " + smallest);
}
}
```



```
C:\Windows\system32\cmd.exe
наибольший множитель: 50
наименьший множитель: 2
Для продолжения нажмите любую клавишу . . .
```

if (условие) инструкция;

условие представляет собой булево выражение (которое приводится к значению ИСТИНА или ЛОЖЬ).

```
if (10 < 11) Console.WriteLine("10 меньше 11");
```

```
if(10 < 9) Console.WriteLine  
("Этот текст выведен не будет.");
```

Конструкция if - else - if

`if (условие)`

`инструкция;`

`else if (условие)`

`инструкция;`

`else if (условие)`

`инструкция;`

`else`

`инструкция;`

```
using System;
class IfDemo
{
```

```
    public static void Main()
```

```
    {    int a, b, c;
```

```
        a = 2;
```

```
        b = 3;
```

```
        if (a < b) Console.WriteLine("a меньше b");
```

```
        // следующая инструкция ничего не отобразит на экране
```

```
        if (a == b) Console.WriteLine("этого текста никто не увидит");
```

```
        Console.WriteLine();
```

```
        c = a - b; // c содержит -1
```

```
        Console.WriteLine("c содержит -1");
```

```
        if (c >= 0) Console.WriteLine("c неотрицательно");
```

```
        if (c < 0) Console.WriteLine("c отрицательно");
```

```
        Console.WriteLine();
```

```
        c = b - a; // c теперь содержит 1
```

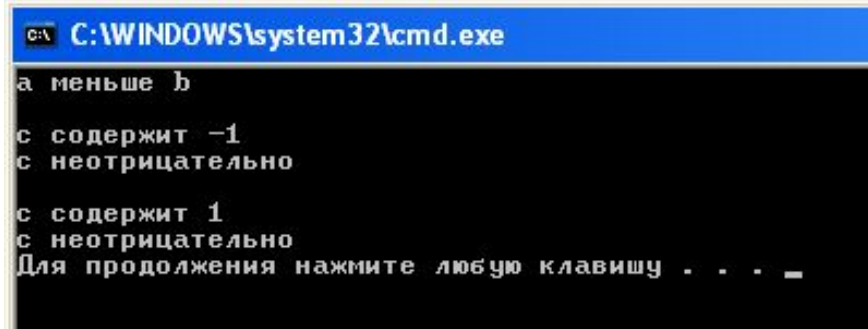
```
        Console.WriteLine("c содержит 1");
```

```
        if (c >= 0) Console.WriteLine("c неотрицательно");
```

```
        if (c < 0) Console.WriteLine("c отрицательно");
```

```
    }
```

```
}
```



```
C:\WINDOWS\system32\cmd.exe
a меньше b
c содержит -1
c неотрицательно

c содержит 1
c неотрицательно
Для продолжения нажмите любую клавишу . . . _
```



```
//определение наименьшего множителя
//состоящего из одной цифры
using System;
class Ladder
{
    public static void Main()
    {
        int num;
        for (num = 2; num < 12; num++)
        {
            if ((num % 2) == 0)
                Console.WriteLine("наименьший
множитель числа " + num + " равен 2.");
            else if ((num % 3) == 0)
```

```
        Console.WriteLine("наименьший  
множитель числа " + num + " равен 3.");  
        else if ((num % 5) == 0)  
            Console.WriteLine("наименьший  
множитель числа " + num + " равен 5.");  
        else if ((num % 7) == 0)  
            Console.WriteLine("наименьший  
множитель числа" + num + " равен 7");  
        else  
            Console.WriteLine(num + " не делится  
на 2, 3, 5, или 7");  
    }  
}  
}
```

C:\Windows\system32\cmd.exe

```
наименьший множитель числа 2 равен 2.  
наименьший множитель числа 3 равен 3.  
наименьший множитель числа 4 равен 2.  
наименьший множитель числа 5 равен 5.  
наименьший множитель числа 6 равен 2.  
наименьший множитель числа 7 равен 7.  
наименьший множитель числа 8 равен 2.  
наименьший множитель числа 9 равен 3.  
наименьший множитель числа 10 равен 2.  
11 не делится на 2, 3, 5, или 7  
Для продолжения нажмите любую клавишу . . . _
```

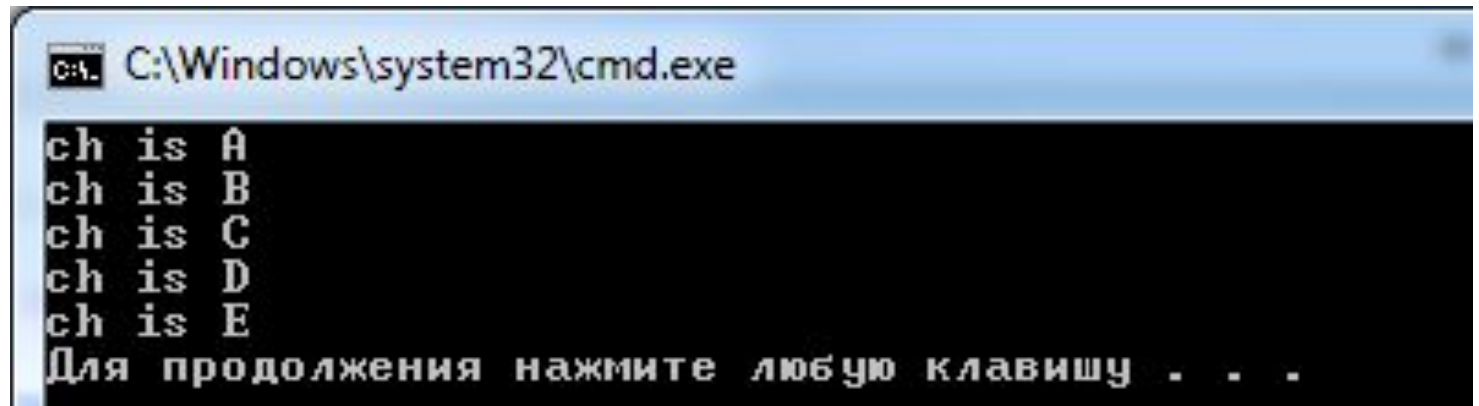
ИНСТРУКЦИЯ switch

```
switch (выражение) {  
    case константа1:  
        последовательность инструкций  
        break;  
    case константа2:  
        последовательность инструкций  
        break;  
    case константа3:  
        последовательность инструкций  
        break;  
    default:  
        последовательность инструкций  
        break; }  
}
```

- Элемент *выражение* инструкции *switch* должен иметь целочисленный тип (например, *char*, *byte*, *short* или *int*) или тип *string*.
- Выражения, имеющие тип с плавающей точкой, не разрешены.

```
// использование char для управления switch.
using System;
class Switch
{
    public static void Main()
    {
        char ch;
        for (ch = 'A'; ch <= 'E'; ch++)
            switch (ch)
            {
                case 'A':
                    Console.WriteLine("ch is A");
                    break;
                case 'B':
                    Console.WriteLine("ch is B");
                    break;
                case 'C':
                    Console.WriteLine("ch is C");
                    break;
            }
    }
}
```

```
case 'D':  
    Console.WriteLine("ch is D");  
    break;  
case 'E':  
    Console.WriteLine("ch is E");  
    break;  
}  
}  
}
```



```
C:\Windows\system32\cmd.exe  
ch is A  
ch is B  
ch is C  
ch is D  
ch is E  
Для продолжения нажмите любую клавишу . . .
```

Бесконечный цикл

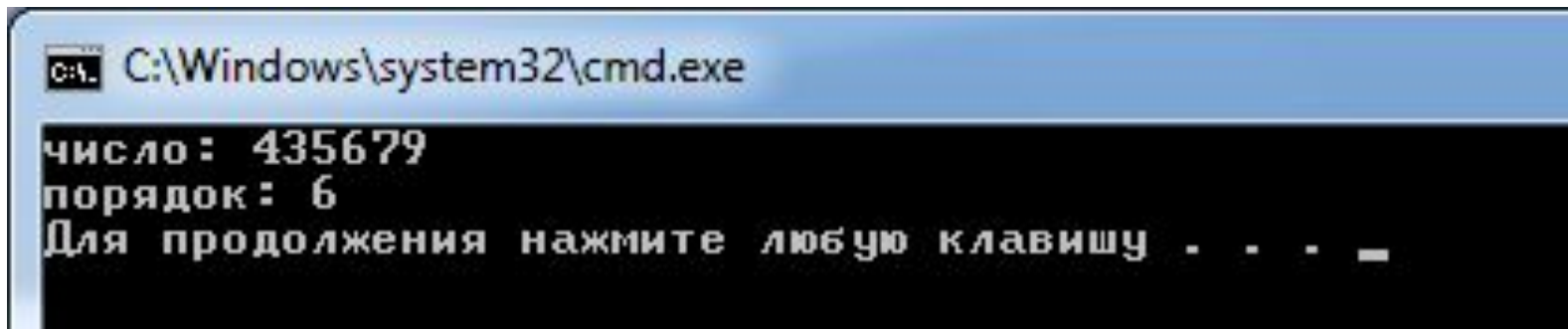
```
for (;;) // Специально созданный бесконечный цикл  
{  
//.....  
}
```


ЦИКЛ `while`

`while` (*условие*) *инструкция*

```
// вычисление порядка целого числа
using System;
class While
{
    public static void Main()
    {
        int num;
        int mag;
        num = 435679;
        mag = 0;
        Console.WriteLine("число: " + num);
        while (num > 0)
        {
```

```
mag++;  
    num = num / 10;  
};  
Console.WriteLine("порядок: " + mag);  
}  
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\system32\cmd.exe". The output of the program is displayed in Russian: "число: 435679", "порядок: 6", and "Для продолжения нажмите любую клавишу . . .".

```
C:\Windows\system32\cmd.exe  
число: 435679  
порядок: 6  
Для продолжения нажмите любую клавишу . . .
```

ЦИКЛ do - while

do {

инструкции;

} while (*условие*);

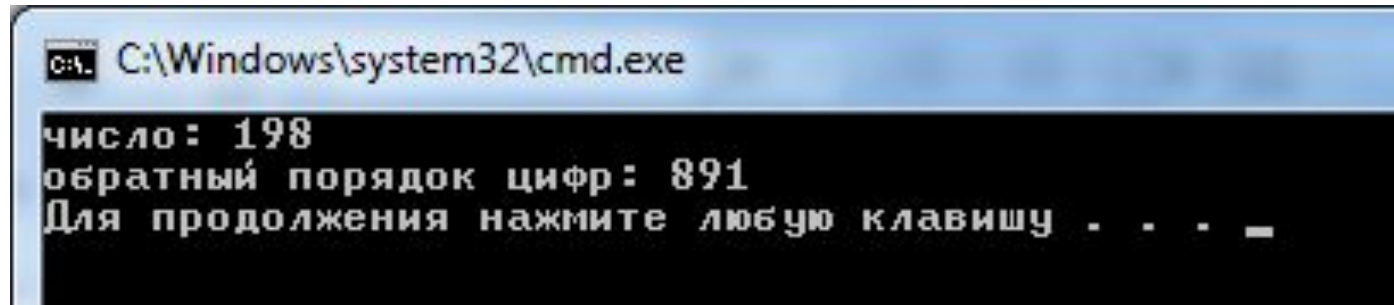
выполняется до тех пор, пока остается
ИСТИННЫМ ЭЛЕМЕНТ *условие*

```
/*отображение цифр целого числа в обратном
порядке*/
using System;
class DoWhileDemo
{
    public static void Main()
    {
        int num;
        int nextdigit;
        num = 198;
        Console.WriteLine("число: " + num);
        Console.Write("обратный порядок цифр: ");
    }
}
```

```
do
{
    nextdigit = num % 10;
    Console.Write(nextdigit);
    num = num / 10;
} while (num > 0);

Console.WriteLine();
}
```

```
}
```

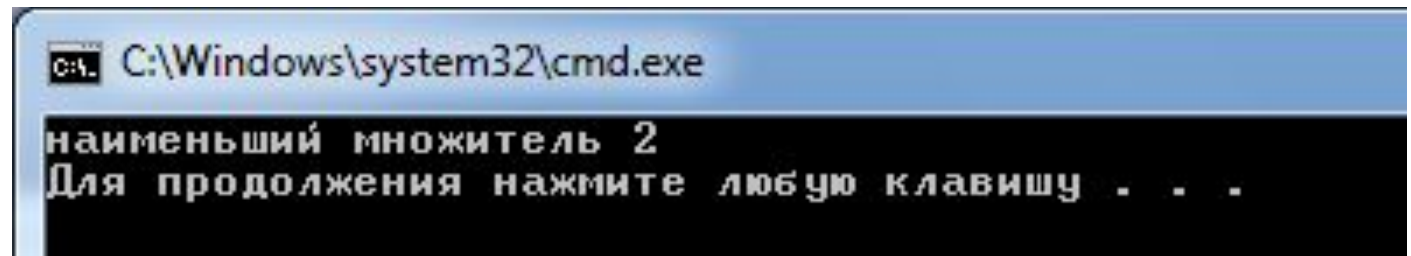


```
C:\Windows\system32\cmd.exe
число: 198
обратный порядок цифр: 891
Для продолжения нажмите любую клавишу . . . _
```

Использование инструкции break для выхода из цикла

```
//определение наименьшего множителя числа
using System;
class FSF
{
    public static void Main()
    {
        int factor = 1;
        int num = 1000;
        for (int i = 2; i < num / 2; i++)
```

```
{
    if ((num % i) == 0)
    {
        factor = i;
        break; // цикл прекращается,
        когда найден множитель
    }
}
Console.WriteLine("наименьший
множитель " + factor);
}
}
```



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the output of a program: "наименьший множитель 2" followed by "Для продолжения нажмите любую клавишу . . ." on the next line.

Инструкция goto

```
//использование goto и switch
using System;
class SwitchGoto
{
    public static void Main()
    {
        for (int i = 1; i < 5; i++)
        {
            switch (i)
            {
                case 1:
                    Console.WriteLine("в case 1");
                    goto case 3;
                case 2:
                    Console.WriteLine("в case 2");
                    goto case 1;
            }
        }
    }
}
```

case 3:

```
    Console.WriteLine("в case 3");
```

```
        goto default;
```

```
    default:
```

```
        Console.WriteLine("в default");
```

```
            break;
```

```
    }
```

```
    Console.WriteLine();
```

```
}
```

```
//goto case 1; //ошибка!нельзя впрыгнуть в switch.
```

```
}
```

```
}
```

C:\Windows\system32\cmd.exe

```
в case 1  
в case 3  
в default
```

```
в case 2  
в case 1  
в case 3  
в default
```

```
в case 3  
в default
```

```
в default
```

Для продолжения нажмите любую клавишу . . .

Оператор ?

тернарный оператор ? используется для замены определенных типов конструкций if-then-else. (работает с тремя операторами).

Выражение 1 ? Выражение2 : Выражение3;

Выражение1 должно иметь тип *bool*.

Типы элементов Выражение2 и Выражение 3 должны быть одинаковы.

- Вычисляется *Выражение1*.
- Если оно оказывается истинным, вычисляется *Выражение2*, и результат его вычисления становится значением всего ?-выражения.
- Если результат вычисления элемента *Выражение1* оказывается ложным, значением всего ?-выражения становится результат вычисления элемента *Выражение3*.

```
// способ обойти деление на 0 с помощью ?
using System;
class NoZeroDiv
{
    public static void Main()
    {
        int result;
        int i;
        for (i = -5; i < 6; i++)
        {
            result = i != 0 ? 100 / i : 0;
            if (i != 0)
                Console.WriteLine("100 / " +
i + " равно " + result);
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
100 / -5 равно -20
100 / -4 равно -25
100 / -3 равно -33
100 / -2 равно -50
100 / -1 равно -100
100 / 1 равно 100
100 / 2 равно 50
100 / 3 равно 33
100 / 4 равно 25
100 / 5 равно 20
```

Для продолжения нажмите любую клавишу . . .