

# ЛАБОРАТОРНАЯ РАБОТА № 1.

## Приложения C# для расчетов по формулам, консольный ввод-вывод.

### 1. Краткие теоретические сведения

#### Типы данных.

Язык C# имеет набор встроенных типов, которые рассматриваются как псевдонимы типов в пространстве имен `System`. Например, тип `string` – это псевдоним типа `System.String`, а тип `int` – псевдоним типа `System.Int32`. Все встроенные типы подразделены на группы: целочисленные типы; вещественные типы; логиче-

Таблица 1.1

Тип данных	Ключевое слово	Псевдоним класса библиотеки NET	Описание	Размер (бит)
логический	<code>bool</code>	<code>System.Boolean</code>	-	-
целый	<code>Int</code>	<code>System.Int32</code>	со знаком	32
	<code>short</code>	<code>System.Int16</code>	со знаком	16
	<code>byte</code>	<code>System.Byte</code>	без знака	8
	<code>sbyte</code>	<code>System.SByte</code>	со знаком	8
	<code>long</code>	<code>System.Int64</code>	со знаком	64
вещественный	<code>float</code>	<code>System.Single</code>	7 цифр	32
	<code>double</code>	<code>System.Double</code>	15 цифр	64
Строковый символьный	<code>string</code>	<code>System.String</code>	строка	-
	<code>char</code>	<code>System.Char</code>	символов Unicode	16
Любой тип	<code>object</code>	<code>System.Object</code>	объектный	-

ский тип; символьные типы; объектный тип (object). Описание типов приведено в таблице 1.1.

Иерархия классов NET Framework имеет один общий корень – класс `System.Object`. Все типы разделяются на две категории: размерные типы и ссылочные типы.

При создании переменной размерного типа под нее в стеке выделяется определенный объем памяти, соответствующий типу этой переменной. При передаче такой переменной в качестве параметра выполняется передача значения, а не ссылки на него. Значение размерного типа не может быть равным `null`. К размерным типам, например, относятся целочисленные и вещественные типы, структуры.

При создании переменной ссылочного типа память под созданный объект выделяется в другой области памяти, называемой кучей. Ссылка всегда указывает на объект заданного типа.

### **Структура приложения на языке C#.**

Проектом называется совокупность файлов, содержащих информацию об установках, конфигурации, ресурсах проекта, а также файлов исходного кода и заголовочных файлов.

Интегрированная среда проектирования Visual Studio позволяет для создания проектов на разных языках программирования использовать различные инструментальные средства проектирования (например, Microsoft Visual Basic, Microsoft Visual C#).

Любое приложение на языке C#, разрабатываемое в среде проектирования Visual Studio, реализуется как отдельный проект. Приложение на языке C# может состоять из нескольких модулей. Каждый модуль C# может содержать код нескольких классов (при соз-

дании приложения в среде Visual Studio.NET каждый класс C# автоматически помещается в отдельный модуль – файл с расширением cs).

Для консольного приложения один из классов, реализуемых модулем, должен содержать метод `Main`. В языке C# нет аппарата заголовочных файлов, используемого в языке C++, поэтому код модуля должен содержать как объявление, так и реализацию класса. По умолчанию весь код класса, представляющего консольное приложение, заключается в одно пространство имен, одноименное с именем приложения.

Точкой входа в программу на языке C# является метод `Main`. Этот метод может записываться как без параметров, так и с одним параметром типа `string` – указателем на массив строк, который содержит значения параметров, введенных при запуске программы. В отличие от списка параметров, задаваемых при запуске C-приложения, список параметров C#-приложения не содержит в качестве первого параметра имя самого приложения. Код метода указывается внутри фигурных скобок:

```
static void Main(string[] args)
{
}
```

Ключевое слово `static` определяет, что метод `Main` является статическим методом, вызываемым без создания экземпляра объекта типа класса, в котором этот метод определен. Метод, не возвращающий никакого значения, указывается с ключевым словом `void`. Однако метод `Main` может возвращать значение типа `int`.

**Пример 1.** Вывод сообщения на консоль.

```
static void Main()
```

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



**Замечание.** Для отладки можно использовать команду меню **Debug\Start Without Debugging**. На экране появится окно с результатом исполнения. Обратите внимание на надпись в конце программы: **Press any key to continue**, которая не была предусмотрена. При нажатии любой клавиши окно закрывается. Это результат срабатывания встроенной разработчиками компилятора функции «остановки экрана» для того, чтобы можно было бы сколь угодно долго его рассматривать.

Можно использовать команду **Debug\Start Debugging**, но тогда окно закроется и мы не сможем рассмотреть искомый результат. Для того чтобы обойти это неудобство, следует при разработке программы предусмотреть собственную остановку экрана. Для этого используется команда `Console.Read()`;

## Константы

Это неизменяемые в процессе выполнения программы величины.

**Целые константы** – наиболее распространенный тип `int`. Это целое число, которое может быть отрицательным, положительным или нулем `-12, 5, 0` (все целые со знаком 32 бита). Их можно за-

писывать с суффиксом `-12L` (длинное целое 64 бита), `5u` (целое без знака 8 бит)

**Вещественные константы с фиксированной точкой.** При записи константы типа `float` (32 бита) необходимо, чтобы за значением шел суффикс символ `f` или `F`: `1.2`, `-1.234`, при записи константы типа `double` (64 бита) можно записать суффикс «`d`» или «`D`», но это не является обязательным условием: `1234.5678`, `12.3d`. Дробная часть отделяется от целой части точкой.

**Вещественные константы с плавающей точкой.** При записи константы типа `float` (32 бита) необходимо, чтобы за значением шел суффикс символ `f` или `F`: `1.2E-3f` (число `0.0012`), при записи константы типа `double` (64 бита) `-1.34E5` (число `-134000`) наличие суффикса не требуется.

**Символьные константы.** Символьная константа `char` может представлять собой 16-битный символ `Unicode` (`'a'`) или управляющие символы (возврат каретки (`'\r'`), перевод строки (`'\n'`), горизонтальную табуляцию (`'\t'`), и другие), заключенный в апострофы.

**Строковые константы** — это последовательность символов, заключенная в кавычки, или константы `string`. Строка, состоящая из символов, например `"Ура!\nСегодня\"Информатика\"!!!"`

**Логическая константа.** Задается одним из двух значений `true` («истина») или `false` («ложь»). Используется в `C#` в логических выражениях, операторах условного перехода.

**Именованные константы.** Применяются для того, чтобы вместо значений констант, использовать в программе их имена, например константа `p` вещественная одинарной точности

```
const float p = 3.14159f
```

## Переменные

Переменная – именованная область памяти, для хранения данных определенного типа. При выполнении программы значение переменной величины можно изменять. Все переменные должны быть описаны явно, при описании переменной задается ее значение и тип. При объявлении переменной может быть задано начальное значение.

Имя переменной может содержать буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются. Например, переменные Long, LONG, long – три разных переменные.

Имя переменной может начинаться с буквы или знака подчеркивания, но не цифры. Имя переменной не должно совпадать с ключевыми словами. Не рекомендуется начинать имя с двух подчеркиваний (такие имена зарезервированы для служебного использования).

Правильные имена переменных: MaxLen, iMaxLen, Max\_Len

Неправильные имена переменных: 2Len, Le#

Примеры описания переменных:

```
int a = -14;           // числовая целая 32 бита
float c = -0.00151f;   // числовая вещественная 32
                        // бита
double i = 1234.56789; // числовая вещественная 64
                        // бита
bool l = false;        // логическая 16 бит
string name = "Petrov"; // строковая
```

Выражение – состоит из одного или более операндов (которые могут быть переменными, константами, функциями или символьными значениями), знаков операций и круглых скобок.

Примеры выражений:

$2 * 2 + 1$  полученное значение 5

$1 / 2 - 3$  полученное значение -3

$1. / 2 - 3$  полученное значение -2.5

Присвоение значения переменной представляет оператор присваивания (знаки основных операций приведены в таблице 1.2):

$y = 2*x*x + 3*x - 1.$

В этом примере сначала производятся вычисления правой части оператора присваивания « = », а затем полученное значение присваивается переменной *y*. Для текстовых данных выражение можно записать в следующем виде:

`string kaf = “Кафедра” + “ИСидТ”;`

В этом примере строки по правую сторону от оператора присваивания объединяются, чтобы получить строку “Кафедра + ИСидТ”, которая затем присваивается переменной *kaf*.

Таблица 1.2 Знаки операций

Знак операции	Название
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления

Если в арифметических выражениях используются целые числа, то результатом вычислений будет целое число, и любой остаток от деления будет отброшен. Для получения остатка можно использовать соответствующую операцию `%`, например `10 % 3` возвращает остаток от целочисленного деления, равный 1.

Когда в арифметических выражениях используются числа с плавающей точкой, то результатом деления `10f / 3f` будет число 3,333333.

### Математические функции

C# содержит большое количество встроенных математических функций, которые реализованы в классе `Math` пространства имен `System`.

Рассмотрим краткое описание некоторых математических функций, подробнее с ними можно познакомиться в справочной системе VS или технической документации. Особое внимание следует обратить на типы операндов и результатов, т. к. каждая функция может иметь несколько перегруженных версий.

**Замечание.** Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функции. Например, функция `Math.Abs()`, вычисляющая модуль числа, имеет 7 перегруженных версий: `double Math.Abs(double x)`, `float Math.Abs(float x)`, `int Math.Abs(int x)`, и т. д. (таблица 1.3)

Таблица 1.3 Математические функции

№	Название	Описание
1.	<code>Math.Abs(выражение)</code>	Модуль



2.	Math.Ceiling( <i>выражение</i> )	Округление до большего целого
3.	Math.Cos( <i>выражение</i> )	Косинус
4.	Math.E	Число e
5.	Math.Exp( <i>выражение</i> )	Экспонента
6.	Math.Floor( <i>выражение</i> )	Округление до меньшего целого
7.	Math.Log( <i>выражение</i> )	Натуральный логарифм
8.	Math.Log10( <i>выражение</i> )	Десятичный логарифм
9.	Math.Max( <i>выражение1</i> , <i>выражение2</i> )	Максимум из двух значений
10.	Math.Min( <i>выражение1</i> , <i>выражение2</i> )	Минимум из двух значений
11.	Math.PI	Число $\pi$
12.	Math.Pow( <i>выражение1</i> , <i>выражение2</i> )	Возведение в степень
13.	Math.Round( <i>выражение</i> ) Math.Round( <i>выражение</i> , <i>число</i> )	Простое округление Округление до заданного числа цифр
14.	Math.Sign( <i>выражение</i> )	Знак числа
15.	Math.Sin( <i>выражение</i> )	Синус
16.	Math.Sqrt( <i>выражение</i> )	Квадратный корень
17.	Math.Tan( <i>выражение</i> )	Тангенс

**Пример 2.** Вычислить значения функции  $Y = \frac{\cos \pi x}{1+x^2}$  при  $x = 2,5$

```
using System;
```

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

```

using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Example2        // начало описания класса
                           // Example2
    {
        static void Main()
        {
            double p = 3.14159;
            double x = 2.5;
            double y = Math.Cos(p * x)/(1 + x*x);
            Console.WriteLine();
            Console.WriteLine(" x = {0} \t y = {1} ",
                              x, y);
        }
    }
}

```

Эта программа выводит следующее окно с результатом:



**Замечание.** Функция `Console.WriteLine()` выводит на экран пустую строку. Это сделано для более комфортной работы

## Организация ввода-вывода данных.

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 объекты. В частности, для работы с консолью используется стандартный класс `Console`, определенный в пространстве имен `System`.

### Ввод данных

Для ввода данных обычно используется метод `ReadLine`, реализованный в классе `Console`. Особенностью данного метода является то, что в качестве результата он возвращает строку (`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 `int.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 `System.FormatException` (входная строка имела неверный формат).

### 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 (таблица 1.4):

Таблица 1.4. Способы вывода

<code>Console.WriteLine(x);</code>	на экран выводится значение идентификатора <code>x</code>
<code>Console.WriteLine("x=" + x + "y=" + y);</code>	на экран выводится строка, образованная последовательным слиянием строки <code>"x="</code> , зна-

	чения $x$ , строки " $y=$ " и значения $y$
<code>Console.WriteLine("x={0} y={1}", x, y);</code>	на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра <code>{0}</code> выводится значение $x$ , а вместо <code>{1}</code> – значение

Если использовать при выводе вместо метода `WriteLine` метод `Write`, вывод будет выполняться без перевода строки.

### **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 (таблица 1.5):

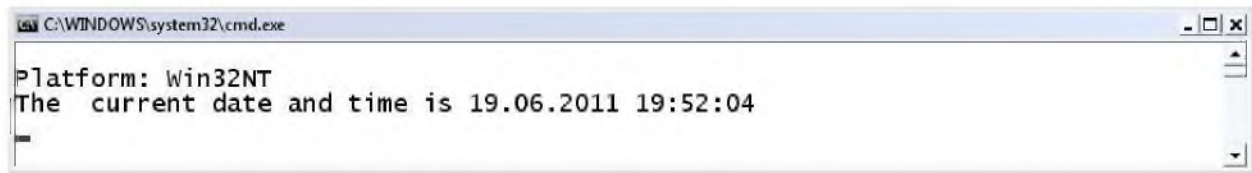
Таблица 1.5. Управляющие символы

Вид	Наименование	Вид	Наименование
<code>\a</code>	Звуковой сигнал	<code>\t</code>	Горизонтальная табуляция
<code>\b</code>	Возврат на шаг назад	<code>\v</code>	Вертикальная табуляция
<code>\f</code>	Перевод страницы	<code>\\</code>	Обратная косая черта

\n	Перевод строки	\'	Апостроф
\r	Возврат каретки	\"	Кавычки

**Пример 3.** Вывести сообщение о версии установленной операционной системы, текущую дату и время.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // вывести версию операционной системы
            OperatingSystem os =
                System.Environment.OSVersion;
            Console.WriteLine("Platform:
{0}",os.Platform);
            System.Console.WriteLine("The  current
date and time is " +
                System.DateTime.Now);    // дата и время
            System.Console.ReadLine();
        }
    }
}
```



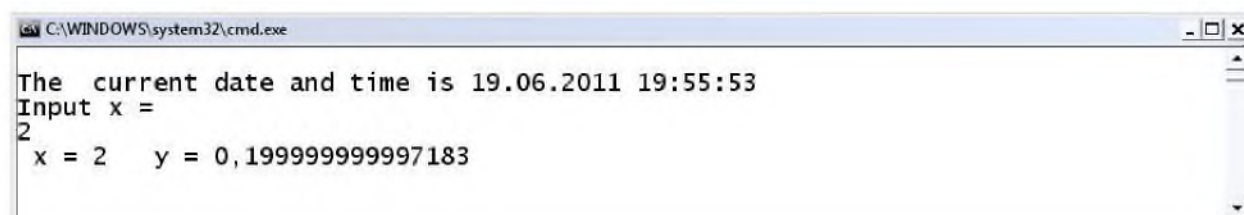
**Пример 4.** Использование консольного ввода для вычисления значений функции  $Y = \frac{\cos \pi x}{1+x^2}$

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace lab0
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("The      current
            date      and      time      is      "      +
            System.DateTime.Now);
            double pi = 3.14159;
            Console.WriteLine("Input x =\r");
            double x =
            Convert.ToDouble(Console.ReadLine());
            double y = Math.Cos(pi * x)/(1 + x*x);
            Console.WriteLine(" x = {0} \t y = {1}
            ",x,y);
        }
    }
}
```

```

        Console.ReadKey();
    }
}

```



```

C:\WINDOWS\system32\cmd.exe
The current date and time is 19.06.2011 19:55:53
Input x =
2
x = 2    y = 0,199999999997183

```

## 2. Практическая часть.

**Задание 1.** Напишите процедуру, выводящую сообщение о версии установленной операционной системы, текущей даты и времени (пример 3).

**Задание 2.** Составить процедуру для выполнения расчетов функции, значения задавать в диалоге с использованием метода `Console.ReadLine()` (пример 4) см. таблицу 1.6;.

Таблица 1.6

Вар.	Функция	x	y
1	$A = \sqrt{\ln(\frac{4}{3} + x) + \frac{9}{7}} - e^{-\sin(1,3x-0,7)}$	0,31 2,5	-0,0049
2	$B = (x + \frac{7}{6})^{\frac{4}{3}} + \sin^x + \arcsin(\cos px)$	-0,75 1,2	-0,018
3	$C = 3,7\sqrt{5-x}\cos(3,5-x) - \sqrt[5]{(5-x)^3}$	2,23 3,2	-0,018



4	$D = -e^{-\cos\sqrt{x+\frac{5}{3}}} - 1,7\arctg(\frac{x}{5} - \frac{3}{4})\sin 1,7x$	-0,35 1,5	-1,318
5	$E = 6,3\sin(1,3x - \frac{p}{3}) - x + \sqrt{x + \frac{9}{4}} + (x + \frac{7}{3})^{\frac{2}{3}}$	0,40 1,5	0,016
6	$F = \cos 1,5x - e^{\sin(x+\frac{4}{3})} + \sqrt{x + \frac{7}{6}}$	2,26 1,2	0,235
7	$G = \frac{5}{3} - \arctg\sqrt{2 - \cos 2x} - e^{-\frac{x}{5}}$	2,09 1,7	0,920
8	$H = \sin\ln(x + 2) - \cos(\pi\ln(x + \frac{5}{3})) + \frac{x}{5}$	-0,26 0,25	-0,0049
9	$I = 4\sin(15e^{\frac{x}{8}} + 10,2) - 9\cos e^{-x} + \sqrt{x + \frac{5}{3}}$	-0,61 0,5	-0,012
10	$J = e^{\frac{4x}{5}} + 2\sin(7\ln(x + \frac{5}{3})) - p$	0,97 -0,5	-0,0024
11	$K = 1,3e^{-\frac{x}{2}} + \left \cos(\frac{2\pi x}{3} - 1,4)\right  - \frac{6}{11}$	2,81 1,25	0,253
12	$L = p + \ln\left \frac{4}{7} - \frac{\sin \arctg x}{2}\right $	2,03 1,7	1,043
13	$M = e^{-\frac{x}{p}} + \frac{4}{3}\arcsin \cos x$	1,97 0,7	0,0017
14	$F = \cos 1,5x - e^{\sin(x + \frac{5}{3})} + \sqrt{x + \frac{7}{6}}$	0,96 1,23	-0,528

15	$O = \arccos \sin(3x + 1,3) - x e^{\arctg x} + 0,7$	1,32 -0,5	0,307
16	$P = 1,3x - 2,5 \sin(5\sqrt{\frac{4}{3} + \arctg x} - 0,7)$	-0,71 0,7	0,0252
17	$Q = e^{-\frac{x}{2}} \cos(2x - 0,3) + \frac{x^2}{2,7 + x}$	-0,73 1,53	-4,197
18	$R = \sqrt{e^{\frac{x}{2}} - 0,1} - x \cos(3x - 1,5)$	2,15 1,2	-1,485
19	$S = 5 \sin(x - 0,3) - \sqrt{2 + e^{-x} - 0,1x^2}$	0,62 1,1	-0,0082
20	$T = 20,7 + (\sin^2(1,2x) - \arccos \frac{x}{8}) \cdot e^{1,5x}$	2,07 1,35	-0,1699
21	$U = 1 - \sin \frac{2x}{3} \cdot \cos \frac{x}{3} - \ln(\frac{5}{3} + x)/1,5$	0,69 1,15	0,0038
22	$V = e^{-\sin \frac{2x}{5}} - e^{\cos \frac{x}{2}} + \sqrt{\frac{4}{3} + x}$	1,28 0,23	-0,0009
23	$W = 2 \sin \frac{4,5x + 0,2}{4} + \sqrt[5]{(x + \frac{7}{4})^2} \cdot e^{-\frac{x}{4}}$	-0,63 1,35	6,0827
24	$X = \cos(4 \ln(x + \frac{7}{3})) - \ln(4 - \frac{x}{7})/3 - \frac{4}{11}$	1,78 2,3	0,0064
25	$Y = \arcsin \sqrt{\frac{1}{3} + \frac{x}{7}} - 1,5 \sin \ln(x + \frac{4}{3})$	0,23 1,4	-0,0021

26	$Z = \cos \frac{x}{3} - e^{-(x+1)^2} - \frac{4}{9}$	-0,23 0,96	-1,0396
27	$A1 = 8x / \left( \frac{70}{3} + 7\sqrt{\frac{7}{6} + x} \right) - e^{-\sin^2\left(\frac{2x}{3}\right)}$	1,83 -0,5	0,2601
28	$B1 = \frac{15}{9} - \left( \sqrt{\frac{1}{3} + \frac{x}{5}} + \sqrt[3]{ x } \right) / e^{-\frac{x}{3}}$	0,47 1,2	-0,0073
29	$C1 = \ln\left(\frac{5}{3} + x\right) - \sin\sqrt{\frac{7}{3} + x} + \frac{1}{7}$	0,66 -0,5	-0,0001
30	$D1 = \frac{1}{3}e^{-\cos^2 x} + \sqrt{x + \frac{9}{7}} / (1 + \ln(x + 3)) - \frac{4}{5}$	0,71 1,4	-0,0012