Лекция 4
Линейные программы

1. Простейший ввод-вывод
2. Ввод-вывод в файлы
3. Математические функции – класс `Math`

Линейной называется программа, все операторы которой выполняются последовательно в том порядке, в котором они записаны.

Пример. Простейшая линейная программа является программа расчета по заданной формуле. Она состоит из трех этапов:

- ввод исходных данных,
- вычисление по формуле,
- вывод результатов.

Чтобы написать подобную программу, нам пока не хватает знаний о том, как организовать ввод и вывод на языке C#. Подробно этот вопрос рассмотрим далее, приведем только минимально необходимые сведения.

1. Простейший ввод-вывод.

Любая программа при вводе исходных данных и выводе результатов взаимодействует с внешними устройствами.

Совокупность стандартных устройств ввода и вывода, то есть клавиатуры и экрана, называется ***консолью***. Обмен данными с консолью является частным случаем обмена с внешними устройствами.

В языке C#, как и во многих других, нет операторов ввода и вывода. Вместо них для обмена с внешними устройствами применяются стандартные объекты.

Для работы с консолью в C# применяется *класс* **`Console`**, определенный в пространстве имен **`System`**. *Методы* этого класса **`Write`** и **`WriteLine`** уже использовались в наших программах. Поговорим о них подробнее, для чего внесем некоторые изменения в листинг 1 (лекция 3).

Листинг 1 – Методы вывода

```
using System:
namespace ConsoleApplication1
{ class Class1
   { static void Main()
      {
       int i = 3;
       double y = 4.12;
       decimal d = 600m;
       string s = "Вася";
       Console.WriteLine("i = " + i );                  //1
       Console.WriteLine("y = {0} \nd = {1}", y, d); //2
       Console.WriteLine("s = " + s );                  //3
      }
   }
}
```

Результат работы программы:

```
i = 3
У = 4.12
d = 600
s = Вася
```

До сих пор мы использовали метод **`WriteLine`** для вывода значений переменных и литералов различных встроенных типов.

Это возможно благодаря тому, что

в классе **`Console`** существует несколько вариантов методов с именами **`Write`** и **`WriteLine`**, предназначенных для вывода значений различных типов.

Методы с *одинаковыми именами*, но *разными параметрами* называются ***перегруженными***.

Компилятор определяет, какой из методов вызван, *по типу передаваемых в него величин*.

Методы вывода в классе **`Console`**

перегружены для всех встроенных типов данных, кроме того,

предусмотрены варианты форматного вывода.

Листинг 1 содержит ***два*** наиболее употребительных *варианта вызова методов вывода*:

в строках 1 и 3;

в строке 2.

1. Способ вывода *пояснений к значениям переменных* в строках 1 и 3.

```
Console.WriteLine("i = " + i );                    //1
Console.WriteLine("s = " + s );                    //3
```

Пояснения представляют собой строковые литералы: **`"i = "`** и **`"s = "`**

Если метод **`WriteLine`** вызван ***с одним параметром***,

он может быть *любого встроенного типа*, например: числом, символом, строкой.

Нам же требуется вывести в каждой строке не одну, а ***две величины***:

1) ***текстовое пояснение***,

2) ***значение переменной***,

поэтому, прежде чем передавать их для вывода, требуется «склеить» их в одну строку с помощью операции **+**.

Перед объединением строки с числом надо преобразовать число из его внутренней формы представления в последовательность символов, то есть в ***строку***. Преобразование в строку определено во всех стандартных классах C# – для этого служит метод **`ToString()`**. В данном случае он выполняется неявно, но можно вызвать его и явным образом:

```
Console.WriteLine("1 = " + i.ToString());
```

2. Оператор 2 иллюстрирует ***форматный вывод***. В этом случае используется другой вариант метода **`WriteLine`**, который содержит *более одного параметра*.

```
Console.WriteLine("y = {0} \nd = {1}", y, d); //2
```

Первым параметром методу передается строковый литерал, содержащий помимо обычных символов, предназначенных для вывода на консоль, ***параметры в фигурных скобках***, а также ***управляющие последовательности***.

Параметры нумеруются с нуля, перед выводом они ***заменяются значениями соответствующих переменных в списке вывода***:

- нулевой параметр заменяется значением первой переменной (в данном примере – **`y`**),
- первый параметр – второй переменной (в данном примере – **`d`**) и т. д.

ПРИМЕЧАНИЕ – Для каждого параметра можно задать ширину поля вывода и формат вывода.

Из управляющих последовательностей чаще всего используются символы перевода строки (**`\n`**) и горизонтальной табуляции (**`\t`**).

Простейшие способы ввода с клавиатуры. В классе **Console** определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа.

Ввод числовых данных выполняется в два этапа:

1. Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.
2. Выполняется преобразование из строки в переменную соответствующего типа.

Преобразование можно выполнить с помощью:

1) специального класса **Convert**, определенного в пространстве имен **System**,

2) метода **Parse**, имеющегося в каждом стандартном арифметическом классе. В листинге 2 используются оба способа.

Листинг 2 – Методы ввода

```
using System;
namespace ConsoleApplication1
{ class Class1
    { static void Main()
       {Console.WriteLine("Введите строку");
        string s = Console.ReadLine(); // 1 – ввод строки
        Console.WriteLine( "s = " + s );
```

```
Console.WriteLine("Введите символ");
char c = (char)Console.Read();          // 2 – ввод символа
Console.ReadLine();                  // 3 – считывание остатка
строки
Console.WriteLine("c = " + c);
string buf;    // строка - буфер для ввода чисел
Console.WriteLine("Введите целое число");
buf = Console.ReadLine();
int i = Convert.ToInt32(buf);           // 4
Console.WriteLine(i);
 Console.WriteLine("Введите вещественное число");
buf = Console.ReadLine();
double x = Convert.ToDouble(buf);       // 5
Console.WriteLine(x);
 Console.WriteLine("Введите вещественное число");
buf = Console.ReadLine();
double y = double.Parse(buf);           // 6
Console.WriteLine(y);
 Console.WriteLine("Введите вещественное число");
buf = Console.ReadLine();
decimal z = decimal.Parse(buf);         // 7
Console.WriteLine(z);
}
```

Пояснения к примеру 2. Ввод строки выполняется в **операторе 1**. Длина строки не ограничена, ввод выполняется до символа перевода строки.

Ввод символа выполняется с помощью метода **`Read`**, который считывает один символ из входного потока – **оператор 2**. Метод возвращает значение типа **`int`**, представляющее собой код символа, или **`-1`**, если символов во входном потоке нет (например, пользователь нажал клавишу **`Enter`**). Поскольку нам требуется не **`int`**, а **`char`**, а неявного преобразования от **`int`** к **`char`** не существует, приходится применить операцию явного преобразования типа.

Оператор 3 считывает остаток строки и никуда его не передает. Это необходимо потому, что ввод данных выполняется через ***буфер*** – специальную область оперативной памяти. Фактически, данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши **`Enter`** вместе с ее кодом. Метод **`Read`**, в отличие от **`ReadLine`**, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий.

В **операторах 4 и 5** используются методы класса **`Convert`**, в **операторах 6 и 7** – методы **`Parse`** классов **`Double`** и **`Decimal`** библиотеки **`.NET`**, которые используются здесь через имена типов C# **`double`** и **`decimal`**.

ВНИМАНИЕ – При вводе вещественных чисел дробная часть отделяется от целой с помощью запятой, а не точки, т.е. при вводе используются правила операционной системы, а не языка программирования. Допускается задавать числа с порядком, например, 1.95e-8.

2. Ввод-вывод в файлы

При отладке даже небольших программ может потребоваться их выполнить не раз, не два и даже не десять. При этом *ввод исходных данных может стать утомительным* и испортить все удовольствие от процесса.

Удобно *заранее подготовить исходные данные в текстовом файле* и считывать их в программе. Кроме того, это дает возможность не торопясь продумать, какие исходные данные требуется ввести для полной проверки программы, и заранее рассчитать, что должно получиться в результате.

Вывод из программы тоже бывает полезно выполнить не на экран, а в текстовый файл для последующего неспешного анализа и распечатки.

Работу с файлами подробно рассмотрим далее, а здесь приведем лишь образцы для использования в программах.

В листинге 3 приведена версия программы из листинга 1, выполняющая вывод не на экран, а в текстовый файл с именем **`output.txt`**.

Файл создается в том же каталоге, что и исполняемый файл программы, по умолчанию – **`…\ConsoleApplication1\bin\Debug`**.

Листинг 3 – Вывод в текстовый файл

```
using System;
using System.I0;                                    // 1
namespace ConsoleApplication1
{ class Class1
   { static void Main()
      {
    StreamWriter f = new StreamWriter("output.txt"); // 2
        int     i = 3;
        double  y = 4.12;
        decimal d = 600m;
        string  s = "Вася";
f.WriteLine( "i = " + i );                         // 3
f.WriteLine( "y = {0} \nd = {1}", y, d );   // 4
f.WriteLine( "s = " + s );                          // 5
f.Close();                                          // 6
      }
    }
}
```

Для того чтобы использовать в программе файлы, необходимо:

1. Подключить пространство имен, в котором описываются стандартные классы для работы с файлами (оператор 1).
2. Объявить файловую переменную и связать ее с файлом на диске (оператор 2).
3. Выполнить операции ввода-вывода (операторы 3-5).
4. Закрыть файл (оператор 6).

СОВЕТ – При отладке программы бывает удобно выводить одну и ту же информацию и на экран, и в текстовый файл. Для этого соответствующие операторы дублируют.

Ввод данных из файла выполняется аналогично. В листинге 4 приведена программа, аналогичная листингу 2, но ввод выполняется из файла с именем **input.txt**, расположенного в каталоге **D:\C#**. Естественно, из программы убраны все приглашения к вводу.

Текстовый файл можно создать с помощью любого текстового редактора, но удобнее использовать **Visual Studio .NET**.

Для этого следует выбрать в меню команду **File ► New ► File...** и в появившемся диалоговом окне выбрать тип файла **Text File**.

Листинг 4 – Ввод из текстового файла

```
using System;
using System.I0;
namespace ConsoleApplication1
{ class Class1
  { static void Main()
   {StreamReader f = new StreamReader("d:\\C#\\input.txt");
string s = f.ReadLine();
 Console.WriteLine("s = " + s );
     char c = (char)f.Read();
     f.ReadLine();
 Console.WriteLine("c = " + c );
     string buf;
     buf = f.ReadLine();
     int i =
Convert.ToInt32(buf);
     Console.WriteLine(i);
     buf = f.ReadLine();
double x =
Convert.ToDouble(buf);
     Console.WriteLine(x);
     buf = f.ReadLine();
double y = double.Parse(buf);
      Console.WriteLine(y);
      buf = f.ReadLine();
decimal z = decimal.Parse(buf);
      Console.WriteLine(z);
      f.Close();
      }
   }
}
```

3. Математические функции – класс Math

В выражениях часто используются математические функции, например синус или возведение в степень. Они реализованы в классе **Math**, определенном в пространстве имен **System**. С помощью методов этого класса можно вычислить:

- тригонометрические функции: **Sin**, **Cos**, **Tan**;
- обратные тригонометрические функции: **ASin**, **ACos**, **ATan**, **ATan2**;
- гиперболические функции: **Tanh**, **Sinh**, **Cosh**;
- экспоненту и логарифмические функции: **Exp**, **Log**, **Log10**;
- модуль (абс. величину), квадратный корень, знак: **Abs**, **Sqrt**, **Sign**;
- округление: **Ceiling**, **Floor**, **Round**;
- минимум, максимум: **Min**, **Max**;
- степень, остаток: **Pow**, **IEEERemainder**;
- полное произведение двух целых величин: **BigMul**;
- деление и остаток от деления: **DivRem**.

Кроме того, у класса есть два полезных поля: число π и число e. Описание методов и полей приведено в табл. 1.

Таблица 1 – Основные поля и статические методы класса Math

Имя	Описание	Результат	Пояснения
Abs	Модуль	Перегружен[1]	|x| записывается как Abs(x)
Acos	Арккосинус[2]	**double**	Acos(double x)
Asin	Арксинус	**Double**	Asin(double x)
Atan	Арктангенс	**double**	Atan(double x)
Atan2	Арктангенс	**double**	Atan2(double x, double y) – угол, тангенс которого есть результат деления y на x
BigMul	Произведение	**Long**	BigMul (int x, int y)
Ceiling	Округление до большего целого	**double**	Ceiling(double x)
Cos	Косинус	**double**	Cos(double x)
Cosh	Гиперболический косинус	**double**	Cosh(double x)
DivRem	Деление и остаток	Перегружен	DivRem(x, y, rem)

[1]существует несколько версий метода для различных типов данных.
[2]Угол задается в радианах.

Имя	Описание	Результат	Пояснения
`E`	База натурального логарифма (число *e*)	`double`	`2,71828182845905`
`Exp`	Экспонента	`double`	e^x записывается как `Exp(x)`
`Floor`	Округление до меньшего целого	`double`	`Floor(double x)`
`IEEERemainder`	Остаток от деления	`double`	`IEEERemainder (double x, double y)`
`Log`	Натуральный логарифм	`double`	$\log_e$ `x` записывается как `Log(x)`
`Log10`	Десятичный логарифм	`double`	$\log_{10}$`x` записывается как `Log10(x)`
`Max`	Максимум из двух чисел	Перегружен	`Max(x, y)`
`Min`	Минимум из двух чисел	Перегружен	`Min(x, y)`
`PI`	Значение числа π	`double`	`3,14159265358979`
`Tan`	Тангенс	`double`	`Tan(double x)`
`Tanh`	Гиперболический тангенс	`double`	`Tanh(double x)`

В листинге 5 приведен пример применения двух методов класса Math. Остальные методы используются аналогично.

Листинг 5 – Применение методов класса **Math**

```
using System;
namespace ConsoleApplication1
{ class Class1
   { static void Main()
      {
      Console.Write( "Введите x: " );
      string buf = Console.ReadLine();  '
      double x = double.Parse(buf);
      Console.WriteLine("Значение sin: " + Math.Sin(x));
      Console.Write("Введите y: ");
      buf = Console.ReadLine();
      double y = double.Parse(buf);
      Console.WriteLine("Максимум : " + Math.Max(x, y));
      }
    }
}
```

В качестве примера рассмотрим программу расчета по заданной формуле.

$$y = \sqrt{\pi \cdot x} - e^{0{,}2\sqrt{a}} + 2\,tg\,2a + 1{,}6 \cdot 10^3 \cdot log_{10}x^2.$$

Из формулы видно, что исходными данными для программы являются две величины – **x** и **a**. Поскольку их тип и точность представления в условии не оговорены, выберем для них тип **double**. Программа приведена в листинге 6.

Листинг 6 – Программа расчета по заданной формуле

```
using System;
namespace ConsoleApplication1
{ class Class1
   { static void Main()
      {Console.WriteLine("Введите x");
       double x = Convert.ToDouble(Console.ReadLine());
       Console.WriteLine("Введите alfa");
       double a = Convert.ToDouble(Console.ReadLine());
       double y = Math.Sqrt(Math.PI * x) -
       Math.Exp(0.2 * Math.Sqrt(a)) + 2 * Math.Tan(2*a) +
       1.6e3 * Math.Log10(Math.Pow(x, 2));
       Console.WriteLine("Для x = {0} и alfa = {1}", x, a);
       Console.WriteLine("Результат = " + y);
        } } }
```

Итак, к настоящему моменту у вас накопилось достаточно сведений, чтобы писать на C# простейшие линейные программы, выполняющие вычисления по формулам. Далее займемся изучением операторов, позволяющих реализовывать более сложные алгоритмы.

Рекомендации по программированию. Приступая к написанию программы, четко определите, что является ее *исходными данными* и что требуется получить в *результате*.

Выбирайте **тип переменных** с учетом *диапазона* и требуемой *точности* представления данных.

Давайте переменным ***имена***, отражающие их назначение.

Правильно выбранные имена могут сделать программу в некоторой степени самодокументированной.

Неудачные имена, наоборот, служат источником проблем. В именах следует избегать сокращений. Они делают программу менее понятной, к тому же часто легко забыть, как именно было сокращено то или иное слово.

Общая тенденция такая: чем больше область действия переменной, тем более длинное у нее имя. Перед таким именем можно поставить префикс типа (одну или несколько букв, по которым можно определить тип переменной).

Напротив, для переменных, вся «жизнь» которых проходит на протяжении нескольких строк кода, лучше обойтись однобуквенными именами типа i или k.

Имена переменных логического типа, используемые в качестве флагов, должны быть такими, чтобы по ним можно было судить о том, что означают значения **`true`** и **`false`**. Например, признак «пусто» лучше описать не как **`bool flag`**, а как **`bool empty`**.

ПРИМЕЧАНИЕ – В C# принято называть:

- классы, методы и константы в соответствии с нотацией *Паскаля*,
- локальные переменные – в соответствии с нотацией *Camel*.

Переменные желательно *инициализировать при объявлении*, а объявлять как можно ближе к месту их непосредственного использования.

С другой стороны, удобно все объявления локальных переменных метода располагать в начале блока так, чтобы их было просто найти. При небольших размерах методов оба эти пожелания довольно легко совместить.

Избегайте использования в программе чисел в явном виде. *Константы должны иметь осмысленные имена*, заданные с помощью ключевого слова **`const`**. Символическое имя делает программу более понятной, а кроме того, при необходимости изменить значение константы потребуется изменить программу только в одном месте. Конечно, этот совет не относится к константам 0 и 1.

Ввод с клавиатуры предваряйте *приглашением*, а *выводимые* значения – *пояснениями*. Для контроля сразу же после ввода выводите исходные данные на дисплей (по крайней мере, в процессе отладки).

До запуска программы подготовьте *тестовые примеры*, содержащие *исходные данные* и *ожидаемые результаты*.

Отдельно проверьте *реакцию программы на неверные исходные данные*.

При записи выражений обращайте внимание на *приоритет* операций. Если в одном выражении соседствует несколько операций одинакового приоритета, операции присваивания и условная операция выполняются справа налево, остальные – слева направо. Для изменения порядка выполнения операций используйте круглые скобки.

Тщательно *форматируйте текст программы* так, чтобы его было удобно читать:

- ставьте пробелы после знаков препинания,
- отделяйте пробелами знаки операций,
- не пишите много операторов в одной строке,
- используйте *комментарии* и *пустые строки* для разделения логически законченных фрагментов программы.