Лекция 9

Литералы, переменные, выражения

Литералы

В **C# литералами** называются постоянные значения (**константы**), представленные в удобной для восприятия форме. Например, число 100 является литералом. В **C#** литералы могут быть любого простого типа. Представление каждого литерала зависит от конкретного типа.

В языке **C#** разделяют четыре типа литералов (констант):

- целые литералы,
- вещественные литералы или литералы с плавающей точкой (запятой),
- символьные литералы,
- строковые литералы.

Примеры целых литералов

Десятичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	020	-0x10
-127	0117	0x2B
240	-0360	0XF0

count = 0xFF; // 255 в десятичной системе

incr = 0x1a; // 26 в десятичной системе

Литерал (константа)с плавающей точкой

[+|-] [цифры].[цифры] [E|e [+|-] цифры]

115.75, 1.5E-2, -0.025, .075, -0.85E2

Тип данных	*Примеры констант*		
float	123.23F	4.34e-3F	
double	123.23	1.0	-0.9876324

Литералы

Если значение целого литерала находится внутри диапазона допустимых значений типа **int**, литерал рассматривается как **int**, иначе он относится к наименьшему из типов **uint**, **long** или **ulong**, в диапазон значений которого он входит. Вещественные литералы по умолчанию относятся к типу **double**.

Например, константа **10** относится к типу **int**, хотя для ее хранения достаточно и байта, а константа **2147 483 648** будет определена как **uint**.

Литералы

Для явного задания типа литерала служит суффикс. Так, для указания типа **long** к литералу присоединяется суффикс **l** или **L**. Например, **12** — это литерал типа **int**, а **12L** — литерал типа **long**. Для указания целочисленного типа без знака к литералу присоединяется суффикс **u** или **U**. Следовательно, **100** — это литерал типа **int**, а **100U** — литерал типа **uint**. А для указания длинного целочисленного типа без знака к литералу присоединяется суффикс **ul** или **UL**. Например, **984375UL** — это литерал типа **ulong**. Кроме того, для указания типа **float** к литералу присоединяется суффикс **F** или **f**.

Например, **10.19F** — это литерал типа **float**. Можно даже указать тип **double**, присоединив к литералу суффикс **d** или **D**, хотя это излишне (по умолчанию **double**).

И наконец, для указания типа **decimal** к литералу присоединяется суффикс **m** или **M**. Например, **9.95M** — это десятичный литерал типа **decimal**.

Литералы

Несмотря на то что целочисленные литералы образуют по умолчанию значения типа **int**, **uint**, **long** или **ulong**, их можно присваивать переменным типа **byte**, **sbyte**, **short** или **ushort**, при условии, что присваиваемое значение может быть представлено целевым типом.

```
float ft;
ft = 10.19;  // происходит неявное преобразование типов
ft = 100.95F;
```

Явное задание типа применяется в основном для уменьшения количества *неявных преобразований типа*, выполняемых компилятором.

Символьный литерал

'a'

'%'

' ' - пробел,

'Q'- буква Q,

'\n' - символ новой строки,

'\\' - обратная дробная черта,

'\v' - вертикальная табуляция.

Управляющие символьные константы

Код	*Значение*
\b	Пробел
\f	Прогон бумаги
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\"	Двойная кавычка
\'	Одинарная кавычка
\0	Нуль
\\	Обратная косая черта
\v	Вертикальная табуляция
\a	Звуковой сигнал

Строковый литерал

"Школа N 35", "город \'Тамбов\'", "YZPT КОД"

Буквальный строковый литерал начинается с символа *@*, после которого следует строка в кавычках. Содержимое строки в кавычках воспринимается без изменений и может быть расширено до двух и более строк. Это означает, что в буквальный строковый литерал можно включить символы новой строки, табуляции и прочие, не прибегая к управляющим последовательностям. Единственное исключение составляют двойные кавычки ("), для указания которых необходимо использовать две двойные кавычки подряд ("").

Пример 1

```
using System;
class Verbatim {
    static void Main() {
        Console.WriteLine(@"Это буквальный
            строковый литерал,
            занимающий несколько строк.
        ");
        Console.WriteLine(@"А это вывод с табуляцией:
            1    2    3    4
            5    6    7    8
        ");
        Console.WriteLine(@"Отзыв программиста: ""Мне нравится C#.""");
    }
}
```

Пример 1

Результат выполнения программы:

Это буквальный
строковый литерал,
занимающий несколько строк.

А это вывод с табуляцией:

1	2	3	4
5	6	7	8

Отзыв программиста: "Мне нравится C#."

Переменные

Переменная — это *именованная область* памяти, предназначенная для хранения данных определенного типа. Во *время выполнения* программы *значение* переменной можно изменять. Все переменные, используемые в программе, должны быть описаны явным образом. При описании (объявлении) для каждой переменной задаются ее *имя* и *тип*.

Общий способ объявления переменных следующий:

тип имя_переменной;

где **тип** — это тип данных, хранящихся в переменной;

имя_переменной — это ее имя (идентификатор).

```
int a;
float x;
```

Переменные

Имя переменной служит для обращения к области памяти, в которой хранится *значение* переменной. Имя дает программист. *Тип переменной выбирается, исходя из диапазона и требуемой точности представления данных*. Таким образом, **возможности переменной определяются ее типом**. Например, переменную типа **bool** нельзя использовать для хранения числовых значений с плавающей точкой. Кроме того, **тип переменной нельзя изменять в течение срока ее существования**. В частности, переменную типа **int** нельзя преобразовать в переменную типа **char**.

Все переменные в **C#** должны быть объявлены до их применения.

Переменные

Общая форма инициализации переменной:

тип имя_переменной = значение;

где **значение** — это конкретное значение указанного типа, задаваемое при создании переменной.

```
int count = 10;  // задать начальное значение 10 переменной count.
char ch = 'X'; // инициализировать переменную ch буквенным значением X.
float f = 1.2F // использование суффиксов
float bn = 30.6f;
decimal dmv = 334.8m;
System.Int32 bik = 4; // использование системных типов   <==>    int bik = 4;
int a1, b1 = 1; float x = 0.1, y = 0.1f; int a, b = 8, c = 19, d;
bool isEnabled = true;
double y=3.0;
string hello="Hello World";
int b = 1, a = 100;
int x = b * a + 25;
```

Пример 2

```
// динамическая инициализация переменных
using System;
class Dynlnit {
    static void Main() {
        // Длина сторон прямоугольного треугольника
        double s1 = 4.0;
        double s2 = 5.0;
        // Инициализировать переменную hypot динамически
        double hypot = Math.Sqrt( (s1 * s1) + (s2 * s2) );
        Console.Write("Гипотенуза треугольника со сторонами
   " + s1 + " и " + s2 + " равна ");
        Console.WriteLine("{0:#.###}.", hypot);
    }
}
```

Переменные

Неявно типизированная переменная объявляется с помощью ключевого слова **var** и должна быть непременно инициализирована. Для определения типа этой переменной компилятору служит тип ее инициализатора, т.е. значения, которым она инициализируется.

```
var e = 2.7183;
```

Переменная **e** инициализируется литералом с плавающей точкой, который по умолчанию имеет тип **double**, и поэтому она относится к типу **double**. Если бы переменная **e** была объявлена следующим образом:

```
var e = 2.7183F;
```

то она была бы отнесена к типу **float**.

Пример 3

```
using System;
class ImplicitlyTypedVar {
    static void Main() {
        var stroka = "Hell to World";
        var c = 20;
        // GetType() – определяет тип переменной
        Console.WriteLine(c.GetType().ToString());
        Console.WriteLine(stroka.GetType().ToString());
       // Следующий оператор не может быть скомпилирован,
       // поскольку переменная c имеет тип int и
       // ей нельзя присвоить вещественное значение
       // c = 12.2; // Ошибка!
    }
}
```

Переменные

Некоторые ограничения на применение неявно типизированных переменных.

Во-первых, нельзя сначала объявить неявно типизируемую переменную, а затем инициализировать:

```
var c;
c = 20; // Ошибка!
```

Во-вторых, нельзя указать в качестве значения неявно типизируемой переменной null:

```
var c = null; // Ошибка!
```

В-третьих, одновременно можно объявить только одну неявно типизированную переменную:

```
var s1 = 4.0, s2 = 5.0; // Ошибка!
```

Компилятор считает, что предпринимается попытка объявить обе переменные, **s1** и **s2**, одновременно.

Именованные константы

Можно запретить изменять *значение* переменной, задав при ее описании *ключевое слово* **const**, например:

```
const int b = 1;
// const распространяется на обе переменные
const float x = 0.1, y = 0.1f;
```

Такие величины называют *именованными константами*, или просто *константами*. Они применяются для того, чтобы вместо значений констант можно было использовать в программе их имена. Именованные *константы* должны обязательно инициализироваться при описании, например:

```
const int b = 1, a = 100;
const int x = b * a + 25;
```

Область действия и время существования переменных

Блок — это код, заключенный в *фигурные скобки*. Этот блок и определяет область действия. Следовательно, всякий раз, когда начинается блок, образуется новая область действия. Прежде всего область действия определяет видимость имен отдельных элементов, в том числе и переменных, в других частях программы без дополнительного уточнения. Она определяет также время существования локальных переменных.

Область действия и время существования переменных

В **C#** к числу наиболее важных относятся области действия, определяемые классом и методом.

Переменные, описанные непосредственно внутри класса, называются ***полями класса***. Им автоматически присваивается так называемое "*значение* по умолчанию" — как правило, это 0 соответствующего типа. Переменные, описанные внутри метода класса, называются ***локальными переменными***. Их *инициализация* возлагается на программиста.

Область действия и время существования переменных

Так называемая *область действия* переменной, то есть область программы, где можно использовать переменную, начинается в точке ее описания и длится до конца блока, внутри которого она описана. Основное назначение блока — группировка операторов. В C# любая *переменная* описана внутри какого-либо блока: класса, метода или блока внутри метода. *Имя переменной* должно быть уникальным в области ее действия. *Область действия* распространяется на вложенные в метод блоки.

Область действия и время существования переменных

Область действия, определяемая **методом**, начинается открывающей фигурной скобкой и оканчивается закрывающей фигурной скобкой. Но если у этого метода имеются параметры, то и они входят в область действия, определяемую данным методом.

Области действия могут быть **вложенными**. В этом случае внешняя область действия охватывает внутреннюю область. Это означает, что **локальные переменные, объявленные во внешней области действия, будут видимы для кода во внутренней области действия**. Но обратное не справедливо: локальные переменные, объявленные во внутренней области действия, **не будут видимы** вне этой области. Переменные могут быть объявлены в любом месте кодового блока, но они становятся действительными только после своего объявления.

Пример 4

```
class X    // начало описания класса X
{
    int A = 5;      // поле A класса X
    int B = 13;      // поле B класса X
    void Y()    // ---------------------------- метод Y класса X
    {
        int C = 1;   // локальная переменная C, область действия – метод Y
        int A = 8;   // локальная переменная A (НЕ конфликтует с полем A)
        int     i = 3;
        double  y = 4.12;
        decimal d = 600m;
        string  s = "Вася";
        Console.Write( "i = " );
        Console.WriteLine( i );
```

Пример 4

```
{   // ====== вложенный блок 1 =======
        int D;      // локальная переменная D,
                    // область действия – этот блок
//          int A;        недопустимо! Ошибка
//   компиляции, конфликт с локальной переменной A
        C = B;      // присваивание переменной C
                    //  поля B класса X (**)
        C = this.A; // присваивание переменной C
                    //  поля A класса X (***)
        Console.WriteLine("A равно " + A + ("      B
равно " + B);
        Console.Write( "y = " );  Console.WriteLine( y );
    }   // ======= конец блока 1 ==========
```

Пример 4

```
        {  // ====== вложенный блок 2 ========
            int D;    // локальная переменная D,
                      // область действия – этот блок
            D = C;
            Console.Write( "s = " );  Console.WriteLine(s);
        }  // ====== конец блока 2 ============
  //          D = B;      недопустимо!
  // Переменная D здесь недоступна
             Console.WriteLine("A равно " + A + ("      B
равно " + B);
             char H;   // переменная бесполезна
     } // --------- конец описания метода Y класса X
 //        Console.Write( "d = " );  Console.WriteLine( d ); !
}       // конец описания класса X
```

Выражения

Выражение — это правило вычисления значения.

Выражения состоят из ***операндов*** и ***операторов***. Операторы в выражении указывают, какие операции производятся с операндами.

К операторам относятся, например, +, -, *, / и new. К операндам относятся, например, литералы, именованные константы, поля класса, локальные переменные и вызовы функций. **a + 2**

Пробелы внутри знака *операции*, состоящей из нескольких символов, не допускаются: **n <= 10**

Когда выражение содержит несколько операторов, порядок вычисления отдельных операторов задается ***приоритетом*** операторов. Например, выражение x + y * z вычисляется как x + (y * z), поскольку оператор «*» имеет более высокий приоритет по сравнению с оператором «+».

Контрольные вопросы

1. Какие типы литералов (констант) разделяют в языке **C#**?
2. Каков общий способ объявления переменных?
3. Каким образом определяется область действия переменных?
4. Что такое блок?
5. Из чего состоит выражение?