## Переменные и константы

Для хранения данных в программе применяются **переменные**. Переменная представляет именнованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная.

Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

|  |  |
| --- | --- |
| 1 | тип имя\_переменной; |

Вначале идет тип переменной, потом ее имя. В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

* имя может содержать любые цифры, буквы и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания
* в имени не должно быть знаков пунктуации и пробелов
* имя не может быть ключевым словом языка C#. Таких слов не так много, и при работе в Visual Studio среда разработки подсвечивает ключевые слова синим цветом.

Хотя имя переменой может быть любым, но следует давать переменным описательные имена, которые будут говорить об их предназначении.

Например, определим простейшую переменную:

|  |  |
| --- | --- |
| 1 | string name; |

В данном случае определена переменная name, которая имеет тип **string**. то есть переменная представляет строку. Поскольку определение переменной представляет собой инструкцию, то после него ставится точка с запятой.

При этом следует учитывать, что C# является регистрозависимым языком, поэтому следующие два определения переменных будут представлять две разные переменные:

|  |  |
| --- | --- |
| 1  2 | string name;  string Name; |

После определения переменной можно присвоить некоторое значение:

|  |  |
| --- | --- |
| 1  2 | string name;  name = "Tom"; |

Так как переменная name представляет тип string, то есть строку, то мы можем присвоить ей строку в двойных кавычках. Причем переменной можно присвоить только то значение, которое соответствует ее типу.

В дальнейшем с помощью имени переменной мы сможем обращаться к той области памяти, в которой хранится ее значение.

Также мы можем сразу при определении присвоить переменной значение. Данный прием называется инициализацией:

|  |  |
| --- | --- |
| 1 | string name = "Tom"; |

Отличительной чертой переменных является то, что в программе можно многократно менять их значение. Например, создадим небольшую программу, в которой определим переменную, поменяем ее значение и выведем его на консоль:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | string name = "Tom"; // определяем переменную и инициализируем ее    Console.WriteLine(name); // Tom    name = "Bob"; // меняем значение переменной  Console.WriteLine(name); // Bob |

Консольный вывод программы:

Tom  
Bob

### Константы

Отличительной особенностью переменных является то, что мы можем изменить их значение в процессе работы программы. Но, кроме того, в C# есть константы. **Константа** должна быть обязательно инициализирована при определении, и после определения значение константы не может быть изменено

Константы предназначены для описания таких значений, которые не должны изменяться в программе. Для определения констант используется ключевое слово **const**, которое указывается перед типом константы:

|  |  |
| --- | --- |
| 1 | const string NAME = "Tom"; // определяем константу |

Так, в данном случае определена константа NAME, которая хранит строку "Tom". Нередко для название констант используется верхний регистр, но это не более чем условность.

При использовании констант надо помнить, что объявить мы их можем только один раз и что к моменту компиляции они должны быть определены. Так, в следующем случае мы получим ошибку, так как константе не присвоено начальное значение:

|  |  |
| --- | --- |
| 1 | const string NAME; // ! Ошибка - константа NAME не инициализирована |

Кроме того, мы ее не сможем изменить в процессе работы программы:

|  |  |
| --- | --- |
| 1  2 | const string NAME = "Tom"; // определяем константу  NAME = "Bob"; // !Ошибка - у констаты нельзя изменить значение |

Таким образом, если нам надо хранить в программе некоторые данные, но их не следует изменить, они определяются в виде констант. Если же их можно изменять, то они определяются в виде переменных.

## Литералы

Литералы представляют неизменяемые значения (иногда их еще называют константами). Литералы можно передавать переменным в качестве значения. Литералы бывают логическими, целочисленными, вещественными, символьными и строчными. И отдельный литерал представляет ключевое слово null.

### Логические литералы

Есть две логических константы - **true** (истина) и **false** (ложь):

|  |  |
| --- | --- |
| 1  2 | Console.WriteLine(true);  Console.WriteLine(false); |

### Целочисленные литералы

Целочисленные литералы представляют положительные и отрицательные целые числа, например, 1, 2, 3, 4, -7, -109. Целочисленные литералы могут быть выражены в десятичной, шестнадцатеричной и двоичной форме.

С целыми числами в десятичной форме все должно быть понятно, так как они используются в повседневной жизни:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine(-11);  Console.WriteLine(5);  Console.WriteLine(505); |

Числа в двоичной форме предваряются символами 0b, после которых идет набор из нулей и единиц:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine(0b11); // 3  Console.WriteLine(0b1011); // 11  Console.WriteLine(0b100001); // 33 |

Для записи числа в шестнадцатеричной форме применяются символы 0x, после которых идет набор символов от 0 до 9 и от A до F, которые собственно представляют число:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine(0x0A); // 10  Console.WriteLine(0xFF); // 255  Console.WriteLine(0xA1); // 161 |

### Вещественные литералы

Вещественные литералы представляют дробные числа. Этот тип литералов имеет две формы. Первая форма - вещественные числа с фиксированной запятой, при которой дробную часть отделяется от целой части точкой. Например:

|  |  |
| --- | --- |
| 1  2  3 | 3.14  100.001  -0.38 |

Также вещественные литералы могут определяться в экспоненциальной форме MEp, где M — мантисса, E - экспонента, которая фактически означает "\*10^" (умножить на десять в степени), а p — порядок. Например:

|  |  |
| --- | --- |
| 1  2 | Console.WriteLine(3.2e3); // по сути равно 3.2 \* 10<sup>3</sup> = 3200  Console.WriteLine(1.2E-1); // равно 1.2 \* 10<sup>-1</sup> = 0.12 |

### Символьные литералы

Символьные литералы представляют одиночные символы. Символы заключаются в одинарные кавычки.

Символьные литералы бывают нескольких видов. Прежде всего это обычные символы:

|  |  |
| --- | --- |
| 1  2  3 | '2'  'A'  'T' |

Также мы можем передать их вывести на консоль с помощью Console.WriteLine:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine('2');  Console.WriteLine('A');  Console.WriteLine('T'); |

Специальную группу представляют **управляющие последовательности** Управляющая последовательность представляет символ, перед которым ставится слеш. И данная последовательность интерпретируется определенным образом. Наиболее часто используемые последовательности:

* '\n' - перевод строки
* '\t' - табуляция
* '\\' - слеш

И если компилятор встретит в тексте последовательность \t, то он будет воспринимать эту последовательность не как слеш и букву t, а как табуляцию - то есть длинный отступ.

Также символы могут определяться в виде шестнадцатеричных кодов, также заключенный в одинарные кавычки.

Еще один способ определения символов представляет использования шестнадцатеричных кодов ASCII. Для этого в одинарных кавычках указываются символы '\x', после которых идет шестнадцатеричный код символа из таблицы ASCII. Коды символов из таблицы ASCII можно посмотреть [здесь](http://www.asciitable.com/).

Например, литерал '\x78' представляет символ "x":

|  |  |
| --- | --- |
| 1  2 | Console.WriteLine('\x78'); // x  Console.WriteLine('\x5A'); // Z |

И последний способ определения символьных литералов представляет применение кодов из таблицы символов [Unicode](https://unicode-table.com/ru/). Для этого в одинарных кавычках указываются символы '\u', после которых идет шестнадцатеричный код Unicode. Например, код '\u0411' представляет кириллический символ 'Б':

|  |  |
| --- | --- |
| 1  2 | Console.WriteLine('\u0420'); // Р  Console.WriteLine('\u0421'); // С |

### Строковые литералы

Строковые литералы представляют строки. Строки заключаются в двойные кавычки:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine("hello");  Console.WriteLine("фыва");  Console.WriteLine("hello word"); |

Если внутри строки необходимо вывести двойную кавычку, то такая внутренняя кавычка предваряется обратным слешем:

|  |  |
| --- | --- |
| 1 | Console.WriteLine("Компания \"Рога и копыта\""); |

Также в строках можно использовать управляющие последовательности. Например, последовательность '\n' осуществляет перевод на новую строку:

|  |  |
| --- | --- |
| 1 | Console.WriteLine("Привет \nмир"); |

При выводе на консоль слово "мир" будет перенесено на новую строку:

Привет  
мир

### null

**null** представляет ссылку, которая не указывает ни на какой объект. То есть по сути отсутствие значения.

## Типы данных

Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных. Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.

В языке C# есть следующие базовые типы данных:

* **bool**: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean

|  |  |
| --- | --- |
| 1  2 | bool alive = true;  bool isDead = false; |

* **byte**: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte

|  |  |
| --- | --- |
| 1  2 | byte bit1 = 1;  byte bit2 = 102; |

* **sbyte**: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte

|  |  |
| --- | --- |
| 1  2 | sbyte bit1 = -101;  sbyte bit2 = 102; |

* **short**: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16

|  |  |
| --- | --- |
| 1  2 | short n1 = 1;  short n2 = 102; |

* **ushort**: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16

|  |  |
| --- | --- |
| 1  2 | ushort n1 = 1;  ushort n2 = 102; |

* **int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

|  |  |
| --- | --- |
| 1  2  3 | int a = 10;  int b = 0b101; // бинарная форма b =5  int c = 0xFF; // шестнадцатеричная форма c = 255 |

* **uint**: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32

|  |  |
| --- | --- |
| 1  2  3 | uint a = 10;  uint b = 0b101;  uint c = 0xFF; |

* **long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64

|  |  |
| --- | --- |
| 1  2  3 | long a = -10;  long b = 0b101;  long c = 0xFF; |

* **ulong**: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64

|  |  |
| --- | --- |
| 1  2  3 | ulong a = 10;  ulong b = 0b101;  ulong c = 0xFF; |

* **float**: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта. Представлен системным типом System.Single
* **double**: хранит число с плавающей точкой от ±5.0\*10-324 до ±1.7\*10308 и занимает 8 байта. Представлен системным типом System.Double
* **decimal**: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0\*10-28 до ±7.9228\*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal
* **char**: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

|  |  |
| --- | --- |
| 1  2  3 | char a = 'A';  char b = '\x5A';  char c = '\u0420'; |

* **string**: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы.

|  |  |
| --- | --- |
| 1  2 | string hello = "Hello";  string word = "world"; |

* **object**: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

|  |  |
| --- | --- |
| 1  2  3 | object a = 22;  object b = 3.14;  object c = "hello code"; |

Например, определим несколько переменных разных типов и выведем их значения на консоль:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | string name = "Tom";  int age = 33;  bool isEmployed = false;  double weight = 78.65;    Console.WriteLine($"Имя: {name}");  Console.WriteLine($"Возраст: {age}");  Console.WriteLine($"Вес: {weight}");  Console.WriteLine($"Работает: {isEmployed}"); |

Для вывода данных на консоль здесь применяется интерполяция: перед строкой ставится знак $ и после этого мы можем вводить в строку в фигурных скобках значения переменных. Консольный вывод программы:

Имя: Tom  
Возраст: 33  
Вес: 78,65  
Работает: False

### Использование суффиксов

При присвоении значений надо иметь в виду следующую тонкость: все вещественные литералы (дробные числа) рассматриваются как значения типа **double**. И чтобы указать, что дробное число представляет тип **float** или тип **decimal**, необходимо к литералу добавлять суффикс: F/f - для float и M/m - для decimal.

|  |  |
| --- | --- |
| 1  2  3  4  5 | float a = 3.14F;  float b = 30.6f;    decimal c = 1005.8M;  decimal d = 334.8m; |

Подобным образом все целочисленные литералы рассматриваются как значения типа **int**. Чтобы явным образом указать, что целочисленный литерал представляет значение типа uint, надо использовать суффикс **U/u**, для типа **long** - суффикс **L/l**, а для типа **ulong** - суффикс **UL/ul**:

|  |  |
| --- | --- |
| 1  2  3 | uint a = 10U;  long b = 20L;  ulong c = 30UL; |

### Использование системных типов

Выше при перечислении всех базовых типов данных для каждого упоминался системный тип. Потому что название встроенного типа по сути представляет собой сокращенное обозначение системного типа. Например, следующие переменные будут эквивалентны по типу:

|  |  |
| --- | --- |
| 1  2 | int a = 4;  System.Int32 b = 4; |

### Неявная типизация

Ранее мы явным образом указывали тип переменных, например, int x;. И компилятор при запуске уже знал, что x хранит целочисленное значение.

Однако мы можем использовать и модель неявной типизации:

|  |  |
| --- | --- |
| 1  2 | var hello = "Hell to World";  var c = 20; |

Для неявной типизации вместо названия типа данных используется ключевое слово var. Затем уже при компиляции компилятор сам выводит тип данных исходя из присвоенного значения. Так как по умолчанию все целочисленные значения рассматриваются как значения типа int, то поэтому в итоге переменная c будет иметь тип int. Аналогично переменной hello присваивается строка, поэтому эта переменная будет иметь тип string

Эти переменные подобны обычным, однако они имеют некоторые ограничения.

Во-первых, мы не можем сначала объявить неявно типизируемую переменную, а затем инициализировать:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // этот код работает  int a;  a = 20;    // этот код не работает  var c;  c= 20; |

Во-вторых, мы не можем указать в качестве значения неявно типизируемой переменной null:

|  |  |
| --- | --- |
| 1  2 | // этот код не работает  var c=null; |

Так как значение null, то компилятор не сможет вывести тип данных.