



Модуль 2

Переменные и типы данных



Обзор модуля

В этом модуле вы изучите:

- Переменные и типы данных в C#
- Комментарии и XML-документацию
- Константы и литералы
- Ключевые слова и Escape-последовательности
- Ввод и вывод



Занятие 1 - Переменные и типы данных в C#

На первом занятии, **Переменные и типы данных в C#**, вы научитесь:

- Описывать переменные и их цели.
- Описывать типы данных и их цели.
- Определять базовые типы данных в C#.
- Описывать ссылочные типы данных.
- Объяснять правила именования переменных.
- Объявлять и использовать переменные.



Определение

- **Переменная** - это сущность, значение которой может изменяться. Например, возраст студента, зарплата сотрудника.
- В C# **переменные** - это область компьютерной памяти, которая идентифицируется уникальным именем и используется для хранения значений.
- Они базируются на типах данных, которые необходимы для хранения (переменные могут быть различных типов).

Использование переменных

Пример

Объявление

переменная выделяется в момент их

```
<datatype> <variableName>;
```

```
string employee;
```

где,

`datatype`: допустимый в C# тип данных.

`variableName`: допустимое имя переменной.

Синтаксис - Инициализация

использовать переменную в момент

```
<variableName> = <value>;
```

где,

`=`: оператор присваивания, используемый для установки значения.

`value`: данные, хранящиеся в переменной.

переменной.



Типы данных

Типы значения

- Компилятор должен знать, какой тип данных хранит конкретная переменная.
- Хранят актуальное значение в стеке.
- Значения могут быть любого встроенного или определенного пользователем типа данных.
- Это позволяет переменным хранить значения соответствующих типов данных.
- Значения встроенных типов данных: int, float, double, char и bool

Ссылочные типы

- В языке программирования C# типы данных делятся на две категории. Это:
- Типы-значения
 - Хранят адреса размещения в памяти других переменных
 - Значения могут принадлежать любому встроенному или пользовательскому типу данных
- Ссылочные типы
- Большинство определенных пользователем типов, например, классы, являются ссылочными типами

Предопределённые типы данных

Тип данных	Размер	Диапазон
byte	Беззнаковое 8-битное целое	От 0 до 255
short	Знаковое 16-битное целое	От -32,768 до 32,767
int	знаковое 32-битное целое	От -2,147,483,648 до 2,147,483,647
long	знаковое 64-битное целое	От -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
float	32-битное вещественное число с точностью 7 знаков	От $\pm 1.5e-45$ до $\pm 3.4e38$
double	64-битное вещественное число с точностью 15-16 знаков	От $\pm 5.0e-324$ до $\pm 1.7e308$
decimal	128-битное вещественное целое	От $\pm 1.0 \times 10e-28$ до $\pm 7.9 \times 10e28$
char	16-битный символ Unicode	От U+0000 до U+ffff
bool	Хранит значения true или false	true или false



Классификация

Объект

- **Массив** ссылочные типы данных хранят в памяти ссылки
- Это встроенный ссылочный тип данных.
- Это определенный пользователем структура, поддерживающая значения одного типа данных, пользовательских типов данных.
- Например, оценки студентов.

Делегат

Классификация ссылочных типов данных:

- **Строка** объект
- Это определенный пользователем тип, содержащий ссылку на один или более методов.
- Представляет строку символов Unicode.

- **Интерфейс** Делегат присваивать и изменять строковые значения
- Это тип, определенный пользователем класса, который используется для сложного наследования.



Правила

Правила объявления переменных:

- Имя переменной может начинаться с буквы в верхнем или нижнем регистре. Имя может содержать буквы, цифры и символ подчеркивания (`_`).
- Первый символ в имени переменной должен быть буквой и не может быть цифрой. Подчеркивание также допустимый первый символ, но он не рекомендуется для начала имени.
- Язык C# чувствителен к регистру; таким образом переменные с именами `count` и `Count` - две разные переменные.
- Ключевые слова C# не могут быть использованы в качестве имен переменных. Если вам необходимо использовать ключевые слова C#, используйте символ '@' как префикс.



Проверка

Имя переменной	Допустимо/недопустимо
Employee	Допустимо
student	Допустимо
_Name	Допустимо
Emp_Name	Допустимо
@goto	Допустимо
static	Недопустимо, поскольку слово ключевое
4myclass	Недопустимо, поскольку имя переменной не может начинаться с цифры
Student&Faculty	Недопустимо, поскольку имя переменной не может содержать специальный символ &



Объявление

Пример

Объявление одной переменной

Инициализация

```
<data type> <variable name1> = <value1>, <variable name2>  
= <value2>;
```

data type: допустимый тип данных.

variable name: допустимое имя переменной или идентификатор.

value: значение, присваиваемое переменной.

присвоить ей значение.

Синтаксис – Объявление нескольких переменных

```
<data type> <variable name1>, <variable  
name2>, ..., <variable nameN>;
```

где,

data type: допустимый тип данных.

variable name1, variable name2, variable nameN:

имена переменных или идентификаторы.



Занятие 2 - Комментарии и XML-документация

На втором занятии, **Комментарии и XML-документация**, вы научитесь:

- Описывать комментарии в C# приложениях.
- Объяснять XML-документацию исходного кода.



Определение 1-2

Комментарии

- Предоставляют информацию о части кода.
- Делают программу более читаемой.
- Помогают программисту объяснить цель использования редких переменных или методов.
- Идентифицируются с помощью специальных символов.
- Игнорируются компилятором при выполнении программы

- C# поддерживает три типа комментариев:
 - Однострочные комментарии
 - Многострочные комментарии
 - XML-комментарии

Определение 2-2

XML-комментарии

- XML-комментарии начинаются с `<!--` и заканчиваются `-->`.
- В отличие от одно- и многострочных комментариев, XML-комментарии должны заключаться в XML-теги. Вам необходимо создать XML-тег для вставки XML-комментария.

```
/// <summary>  
/// Вы в тэге XML называемом summary.  
/// </summary>
```

перемножает два числа,
делит результат на 2 и
выводит частное */

```
int doMult = 5 * 20;  
int doDiv = doMult / 2;  
Console.WriteLine("Quotient is:" + doDiv)
```

Документация XML

```
class XMLComments
{
    /// <summary>
    /// This is the XML comment and can be extracted to a XML file.
    /// </summary>
    static void Main(string[] args)
    {
        Console.WriteLine("This program illustrates XML Comments");
    }
}
```

КО

Синтаксис

указанные в коде. В этом случае вы можете создать документ, который будет содержать всю необходимую

```
csc /doc: <XMLfilename.xml> <CSharpfilename.cs>
```

где,

XMLfilename.xml: имя создаваемого XML файла .

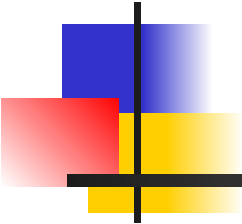
CSharpfilename.cs: имя C# файла, из которого извлекаются XML-комментарии.



Предопределенные XML тэги

- XML-комментарии вставляются в XML-тэги.
- Эти тэги могут быть предопределенными или определенными пользователем.

Предопределенные теги	Описание
<c>	Устанавливает для текста шрифт кода
<code>	Устанавливает одну или более строк исходного кода или вывода программы.
<example>	Обозначает пример.
<param>	Описывает параметр метода или конструктора.
<returns>	Описывает возвращаемое методом значение.
<summary>	Резюмирует общую информацию о коде.



Занятие 3 - Константы и литералы

На третьем занятии, **Константы и литералы**, вы научитесь:

- Описывать константы в C#.
- Перечислять различные типы литералов.



Необходимость констант

- Константы в C# - это фиксированные значения, присвоенные идентификаторам, которые не меняются при выполнении кода.
- Константы определяются тогда, когда значения должны быть фиксированными и повторно используемыми или для предотвращения их модификации.



Константы

Синтаксис константы могут быть определены для всех

ТИПОВ ДАННЫХ

```
const float _pi = 3.14F;  
float radius = 5;  
float area = _pi * radius * radius;  
Console.WriteLine("Area of the circle is " + area);
```

объявляется как константа.

`data type`: Тип данных константы.

`identifier name`: Имя идентификатора, который содержит константу.

`value`: Фиксированное значение, не изменяющееся во время выполнения кода.

- Для объявления идентификатора как константы, используется ключевое слово `const` в объявлении идентификатора. Компилятор идентифицирует константы при компиляции по ключевому слову `const`.



Использование литералов 1-4

- В C# литералы это типовые значения; присвоенное переменной литеральной константе.
- Логические литералы
- Целые литералы
- Литералы могут быть определены для любого вещественного литерала
- Символьные литералы
- Строковые литералы
- Числовые литералы могут содержать суффикс в виде символа алфавита, определяющего тип данных литерала. Литералы могут быть в верхнем или нижнем регистре. Например, `string bookName = "Csharp"`.

Использование литералов 2-4

Дольные литералы

- Дольные литералы могут принимать два значения: `true` или `false`, `int`, `long` или `ulong`. Суффиксы для

целых литералов включают `U`, `L`, `UL` или `LU`. `U`

Пример

обозначает `uint` или `ulong`, `L` обозначает `long`. `UL` и

```
bool val = true;
```

Пример

`true` — логический литерал, присвоенный переменной `val`.

```
long val = 53L;
```

где,

`53L`: целый литерал, присвоенный переменной `val`.

Использование литералов 3-4

Использование литералов

- Использование литералов привязываются к типам данных `char` (Символьный литерал) и `decimal`. Включая также ввод суффикса, добавляемый после присваиваемого значения. Вещественный

Пример

Литерал может заканчиваться на F, D или M. F

```
char val = 'A';
```

где,

A: символьный литерал, присвоенный переменной `val`.

```
float val = 1.66F;
```

где,

1.66F: вещественный литерал, присвоенный переменной `val`.

Использование литералов 4-4

Строковые литералы

Null Literal

- Есть два типа строковых литералов в C#, обычные и дословные.
 - Null-литерал имеет только одно значение - `null`.

Пример

строковый литерал - это стандартная строка.

```
string email = null;
```

где,

`null`: Определяет `email`, не ссылающийся ни на один объект (ссылку).

Пример

```
string mailDomain = "@gmail.com";
```

где,

`@gmail.com`: дословный строковый литерал.



Занятие 4 - Ключевые слова и escape-последовательности

На четвертом занятии, **Ключевые слова и escape-последовательности**, вы научитесь:

- перечислять ключевые слова в C#.
- Перечислять и объяснять символы escape-последовательностей.

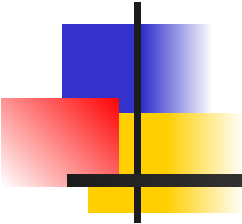


Ключевые слова

Ключевые слова - это зарезервированные слова, которые обрабатываются компилятором отдельно.

<code>abstarct</code>	<code>bool</code>	<code>break</code>	<code>byte</code>	<code>case</code>	<code>catch</code>
<code>char</code>	<code>class</code>	<code>const</code>	<code>continue</code>	<code>default</code>	<code>double</code>
<code>enum</code>	<code>else</code>	<code>false</code>	<code>finally</code>	<code>float</code>	<code>for</code>
<code>foreach</code>	<code>goto</code>	<code>if</code>	<code>int</code>	<code>interface</code>	<code>long</code>
<code>namespace</code>	<code>new</code>	<code>public</code>	<code>private</code>	<code>protected</code>	<code>return</code>
<code>sbyte</code>	<code>short</code>	<code>static</code>	<code>string</code>	<code>struct</code>	<code>switch</code>
<code>throw</code>	<code>true</code>	<code>try</code>	<code>ushort</code>	<code>void</code>	<code>while</code>

Эти ключевые слова не могут быть использованы в именах переменных, методов или классов (за исключением случаев использования префикса в виде символа "@").



Необходимость в символах `escape`-последовательностей

- Рассмотрим фонд заработной платы в какой-либо организации.
- Одна из его функций - показывать месячную зарплату, при этом каждое значение должно выводиться в новой строке.
- Программист хочет написать код, который всегда печатает зарплату в новой строке вне зависимости от длины строки, представляющей размер заработной платы.
- Для этого используются `escape`-последовательности.



Определение

- Символ `escape-последовательности` - это символ обратного слэша сообщает компилятору, специальный символ, следующий за символом обратного слэша (`\`), что следующий символ отмечен как непечатный.
- Например, `\n` используется для добавления новой строки подобно клавише `Enter` на клавиатуре. Символы `escape-последовательностей` используются для реализации специальных непечатных символов, например, новой строки, одиночного пробела или возврата каретки.
- В `C#` символы `escape-последовательности` должны всегда заключаться в двойные кавычки.
- Непечатные символы используются при форматированном выводе для повышения удобочитаемости.

Символы эскап-последовательностей в C#

В C# существует много символов эскап-

Пример

последовательностей, которые используются для

```
string str = "\u0048\u0065\u006C\u006C\u006F";  
Console.Write("\t" + str + "!\n");  
Console.WriteLine("David\u0020\"2007\" ");
```

Вывод

Одноточная кавычка, необходимая для символьных литералов.
Возврат каретки.
Горизонтальная табуляция.

```
Hello!  
David "2007"
```

	Шестнадцатеричное представление (две цифры).
\a	Предупреждение. Например, \b1 представляет символ 'a'.
\b	Backspace.
\f	Перевод страницы.
\n	Новая строка.



Занятие 5 - Ввод и вывод

На последнем занятии, **Ввод и вывод**, вы научитесь:

- Описывать методы консольного вывода в C#.
- Описывать операции консольного ввода в C#.
- Объяснять спецификаторы форматирования в C#.
- Объяснять спецификаторы форматирования даты и времени.



Консольные операции

- Консольные операции это задания, выполняемые консольными приложениями в виде командной строки с использованием исполняемых команд.
- Стандартный поток `in` берет ввод и передает его в консольное приложение для обработки.
- Все консольные приложения состоят из трех потоков, представляющих собой
- Стандартный поток `out` печатает последовательности байт, потому что они легко контролируются операциями вывода на мониторе.
- Эти потоки связаны с устройствами ввода и вывода компьютерной системы и обрабатывают операции ввода и вывода.
- Стандартный поток `err` показывает сообщения об ошибках на мониторе.



Методы вывода 1-2

- В C# все консольные операции содержатся в классе `Console` пространства имен `System`.
- Записывает любой тип данных.

Синтаксис

вывода данных в консоль вам необходимо

```
Console.Write("<data>" + variables);
```

где,

`data`: Заданная строка или символы escape-последовательности, заключенные в двойные кавычки.

`variables`: Заданные имена переменных, значения которых должны быть выведены на консоль.

- `Console.Write()`
- `Console.WriteLine()`

Методы вывода 2-2

Пример

`WriteLine()`

- ВЫВОДИТ ЛЮБОЙ ТИП ДАННЫХ И СИМВОЛ КОНЦА СТРОКИ

```
Console.WriteLine("C# is a powerful programming language");  
Console.WriteLine("C# is a powerful");  
Console.WriteLine("programming language");  
Console.Write("C# is a powerful");  
Console.WriteLine(" programming language");
```

В коде показана разница между двумя методами.

Вывод

```
.WriteLine("<data>" + variables);
```

```
C# is a powerful programming language  
C# is a powerful  
programming language  
C# is a powerful programming language
```


Указатели места заполнения

Пример

Методы `WriteLine()` и `Write()` принимают список

параметров для форматирования текста перед выводом

```
int number, result;
number = 5;
result = 100 * number;
Console.WriteLine("Result is {0} when 100 is multiplied
by {1}", result, number);
result = 150 / number;
Console.WriteLine("Result is {0} when 150 is divided by
{1}", +result, number);
```

Вывод

Например, для указания позиции первого параметра вы

```
Result is 500 when 100 is multiplied by 5
Result is 30 when 150 is divided by 5
```



Методы ввода

Пример

Для чтения данных вам необходим стандартный поток ввода

```
string name;
Console.Write("Enter your name: ");
name = Console.ReadLine();
Console.WriteLine("You are {0}", name);
```

Вывод

Методы ввода, которые позволяют программному

```
Enter your name: David Blake
You are David Blake
```

стандартного устройства ввода.

- `Console.Read()` - Читает один символ.
- `Console.ReadLine()` - Читает строку.

Методы преобразования

Вывод

ReadLine () может также использоваться для ввода целых значений

```
Enter your name: David Blake
Enter your age: 34
Enter the salary: 3450.50
Name: David Blake, Age: 34, Salary: 3450.50
```

```
userName = Console.ReadLine();
Console.Write("Enter your age: ");
age = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter the salary: ");
salary = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Name: {0}, Age: {1}, Salary: {2} ",
userName, age, salary);
```

Определение спецификаторов числового форматирования

Синтаксис

```
Console.WriteLine("{format specifier}", <variable name>);
```

где,

`format specifier`: числовой спецификатор форматирования.
`variable name`: имя целой переменной.

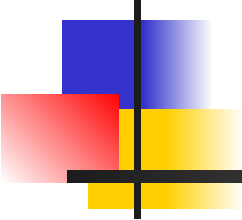
- В C# вы можете преобразовывать числовые значения в различные форматы.
- Это используется в методах вывода класса `Console`.
- Например, вы можете выводить большие числа в экспоненциальном формате.

Некоторые спецификаторы числового форматирования

Спецификатор форматирования	Пример	Название	Описание
C или D	<pre>int num = 456; Console.WriteLine("{0:C}", num); Console.WriteLine("{0:D}", num); Console.WriteLine("{0:E}", num);</pre>		<p>Цифра указывает количество нулей, вставляемых после десятичной точки (0-9) с префиксом в виде минуса, если число отрицательно. Спецификатор точности указывает минимальное количество</p>
	<p>Вывод</p> <pre>\$456.00 456 4.560000E+002</pre>		<p>например, если вы используете спецификатор вида C3, три нуля будут вставлены после десятичной точки для данного числа...E+ddd" или "-d.ddd...e+ddd", где каждая 'd' обозначает цифру (0-9).</p>

Еще числовые спецификаторы форматирования

Формат	Пример	Название	Описание
F	<pre>int num = 456; Console.WriteLine("{0:F}", num); Console.WriteLine("{0:N}", num); Console.WriteLine("{0:X}", num);</pre>		<p>Символа минус.</p>
N	<p>Вывод</p> <pre>456.00 456.00 1C8</pre>	Число	<p>Числа преобразовываются в форму "-d.ddd.ddd.ddd..." где каждая 'd'</p>
X или x		Шестнадцатеричное	<p>Число преобразуется в строку шестнадцатеричных цифр. Используйте "X" для "ABCDEF" и "x" для "abcdef".</p>



Стандартные спецификаторы форматирования даты и времени

Синтаксис

Спецификаторы форматирования даты и времени - это специальные символы, позволяющие вам

```
Console.WriteLine("{format specifier}", <datetime object>);
```

где,

`format specifier`: спецификатор форматирования даты и времени.
`datetime object`: объект класса `DateTime`.

Гринвичское время (GMT), то вы можете выводить его с использованием аббревиатуры GMT, используя спецификаторы форматирования даты и времени.

- Спецификаторы форматирования даты и времени позволяют вам выводить дату и время в 12-ти и 24-х часовых форматах.

Некоторые стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
dF	Полностью дата/время (длинное время)	Выводит дату в длинном формате и время в длинном формате (разделены пробелом). Формат по умолчанию "ddd dd/mm/yyyy HH:mm:ss".
D	Длинная дата	Выводит дату в длинном формате (разделены пробелом). Формат по умолчанию "dddd*, MMMM*, dd, yyyy HH*:mm*:ss*"
g	Обычное дата/время (короткое время)	Выводит дату в коротком формате и время в коротком формате (разделены пробелом). Формат по умолчанию "dd/mm/yyyy HH:mm"
f	Полностью дата/время (короткое время)	Выводит дату в длинном формате (разделены пробелом) и время в коротком формате (разделены пробелом). Формат по умолчанию "ddd dd/mm/yyyy HH*:mm*"

Некоторые стандартные спецификаторы форматирования даты и времени 2-2

Вывод

```
Short date format (d): 23/04/2007
Long date format (D): Monday, April 23, 2007
Full date with time without seconds (f):Monday, April
23, 2007 12:58 PM
Full date with time with seconds (F):Monday, April 23,
2007 12:58:43 PM
Short date and short time without seconds
(g):23/04/2007 12:58 PM
```

```
// Возвращает полностью дату и время с секундами
Console.WriteLine("Full date with time with seconds
(F):{0:F}", dt);
// Возвращает короткую дату и время без секунд
Console.WriteLine("Short date and short time without
seconds (g):{0:g}", dt);
```

Еще стандартные спецификаторы форматирования даты и времени 1-2

Спецификатор форматирования	Название	Описание
Г	Обычное время дата/время (длинное время)	Выводит время в коротком формате в виде индв. шаблоне (разделитель пробелом). Формат по умолчанию "MM/dd/yyyy".
у или Y	Шаблон месяцев	Выводит только месяц и
m или M	два месяца	Выводит только месяцу "дд. MM" по умолчанию "MMMM dd".
T	Короткое время	Выводит время в соответствии с коротким шаблоном. Формат по умолчанию "HH:mm".

Еще стандартные спецификаторы форматирования даты и времени 2-2

Вывод

```
Short date and short time with seconds (G):23/04/2007
12:58:43 PM
Month and day (m):April 23
Short time (t):12:58 PM
Short time with seconds (T):12:58:43 PM
Year and Month (y):April, 2007
```

```
Console.WriteLine("Month and day (m):{0:m}", dt);
// Возвращает короткое время
Console.WriteLine("Short time (t):{0:t}", dt);
// Возвращает короткое время с секундами
Console.WriteLine("Short time with seconds (T):{0:T}", dt);
// Возвращает год и месяц - Y также может быть использовано
Console.WriteLine("Year and Month (y):{0:y}", dt);
```



Заключение 1-3

Переменные и типы данных в C#

- Комментарии используются для предоставления дополнительных пояснений о различных аспектах кода.
- Вы можете вставить комментарии, добавив двойной слэш (//) перед пояснением к коду.
- Тип данных указывает вид данных, хранящихся в переменной.
- XML-комментарии начинаются с тройного обратного слэша (///) и заключаются в XML-тэги.
- Тип данных может быть любым типом-значением или ссылочным типом.
- Вы можете создать XML-документ, который будет содержать все XML-комментарии.



Заключение 2-3

Ключевые слова и escape последовательности

- Ключевые слова это специальные слова которые вы определяете в C# при выполнении программы.
- Вы можете использовать важные значения, которые передаются в программу.
- Escape последовательности это специальные символы для символов, которые предшествуют обратный слэш.
- Символы escape-последовательностей позволяют выводить непечатные символы.



Заключение 3-3

Ввод и вывод

- `Console.Read()` и `Console.ReadLine()` - методы ввода, которые всегда возвращают строковое значение.
- Вы можете преобразовать строковое значение в другой тип данных, используя методы преобразования класса `Convert`.
- `Console.Write()` и `Console.WriteLine()` - методы вывода.
- Спецификаторы форматирования позволяют вам настроить вывод в окно консоли.