Variables, Constants and Literals

A variable refers to the memory address. When you create variable, it creates holds space in the memory that is used for storing temporary data. As you know about c# data types, each data type has predefined size

Declaring Variables

- Data_type must be a valid C# data type including char, int, float, double, or any user-defined data type,
- and variable_list may consist of one or more identifier names separated by commas.

Syntax:

```
<data_type> <variable_list>;
```

Examples:

```
int i, j, k;

char c, ch;

float f, salary;
```

Initializing Variables

- Variables are initialized (assigned a value) with an equal sign followed by a constant expression

Syntax:

```
variable_name = value;
```

- Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression

Syntax:

```
<data_type> <variable_name> = value;
```

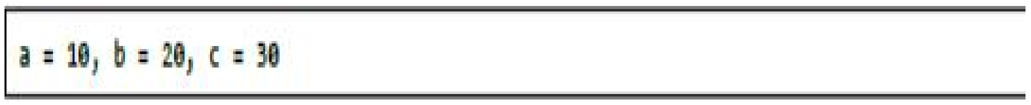
Examples:

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

Sample program using various types of variables:

```
using System;

namespace VariableDefinition

{

    class Program

    {

        static void Main(string[] args)

        {

            short a;

            int b ;

            double c;


            /* actual initialization */

            a = 10;

            b = 20;

            c = a + b;

            Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);

            Console.ReadLine();

        }

    }

}
```

When the above code is compiled and executed, it produces the following result:



```
a = 10, b = 20, c = 30
```

Sample Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Variable_Example
{
    class Program
    {
        static void Main(string[] args)
        {
            //cretaing integer type variable
            int num1, num2, result;
            //Displaying message
            Console.WriteLine("Please enter first value");

            //Accepting Value in num1
            num1 = Int32.Parse(Console.ReadLine());
            //Displaying message
            Console.WriteLine("Enter second Value");
            //Accepting Value
            num2 = Int32.Parse(Console.ReadLine());

            result = num1 + num2; //processing value

            Console.WriteLine("Add of {0} and {1} is {2}", num1, num2, result); //Output

            Console.ReadLine();
        }
    }
}
```

Output

```
Please enter first value
5
Enter second Value
7
Add of 5 and 7 is 12
_
```

In the preceding example we create three integer type variable num1,num2 and result. num1 and num2 is used for accepting user value and result is used for adding both number. The new thing in the preceding example is number of placeholders.

```
Console.WriteLine("Add of {0} and {1} is {2}", num1, num2,result);
```

If you want to display more than one variable values, then you will have to assign place holder for each variables. In the preceding line {0} denotes to num1, {1} denotes to num2 and {2} denotes to result.

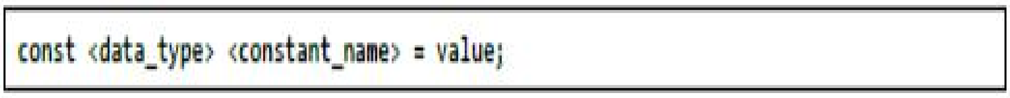
Constants and Literals

- The **constants** refer to **fixed values** that the program may not alter during its execution.
- Constants are **treated** just like regular variables except that their values **cannot be modified after their definition**
- Constant can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string constant.
- These fixed values are also called **literals**.

Defining Constants

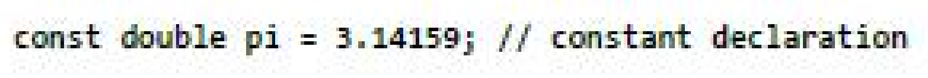
Constants are defined using the const keyword.

Syntax for defining a constant is:



```
const <data_type> <constant_name> = value;
```

Example:



```
const double pi = 3.14159; // constant declaration
```

Literals

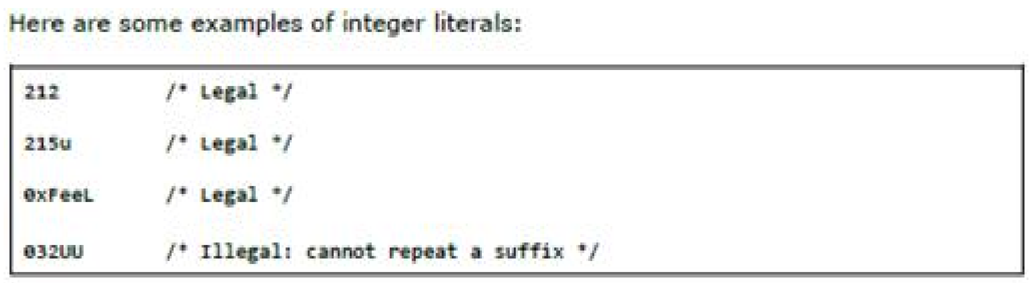
Integer Literals

An integer literal can be a **decimal** or **hexadecimal** constant.

- A prefix specifies the base or radix: **0x or 0X** for **hexadecimal**, and no prefix id for decimal.

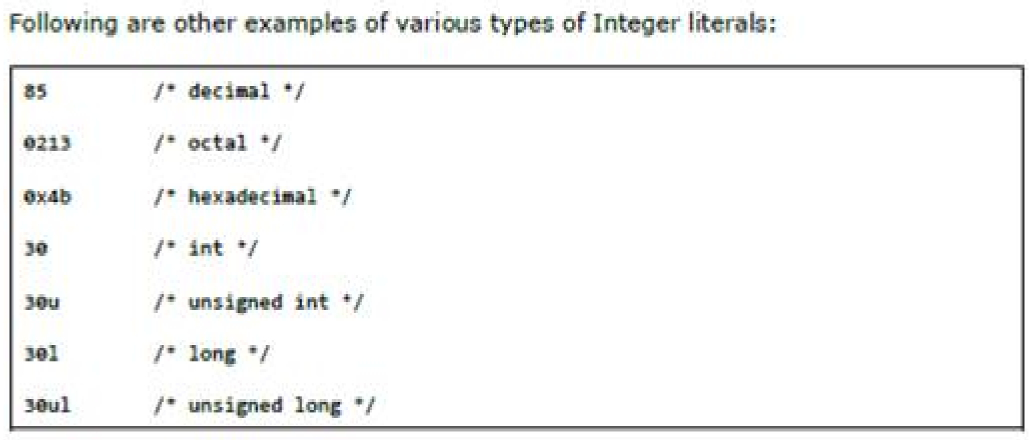
An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively.

- The suffix can be uppercase or lowercase and can be in any order.



Here are some examples of integer literals:

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
032UU       /* Illegal: cannot repeat a suffix */
```



Following are other examples of various types of Integer literals:

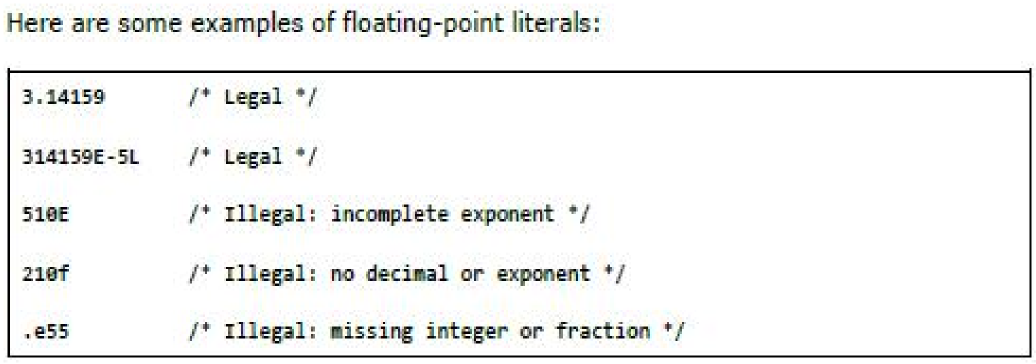
```
85         /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30         /* int */
30u        /* unsigned int */
30l        /* long */
30ul       /* unsigned long */
```

Floating Point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.

You can represent floating point literals either in **decimal form** or **exponential form**.

- While representing in decimal form, you must include the decimal point, the exponent, or both;
- and while representing using exponential form you must include the integer part, the fractional part, or both. The **signed exponent** is introduced by **e or E**.

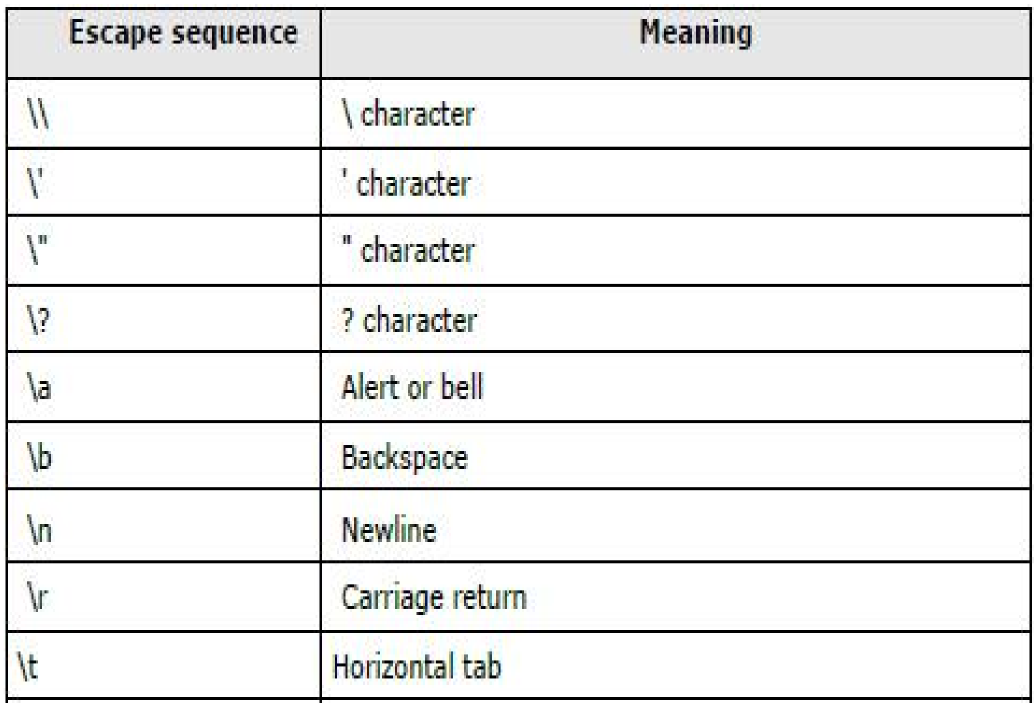

Here are some examples of floating-point literals:

```
3.14159       /* Legal */

314159E-5L    /* Legal */

510E          /* Illegal: incomplete exponent */

210f          /* Illegal: no decimal or exponent */

.e55          /* Illegal: missing integer or fraction */
```

Character Literals

A character literal can be a **plain character** (such as 'x'), an **escape sequence** (such as '\t'), or a **universal character** (such as '\u02C0').

Examples of escape sequence



Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\n	Newline
\r	Carriage return
\t	Horizontal tab

String Literals

String literals or constants are enclosed in double quotes **""** or with **@""**.

```
String str = "Hello \nWorld";
```

```
String str = @"Hello \nWorld";
```