Formatting a string using value of different types

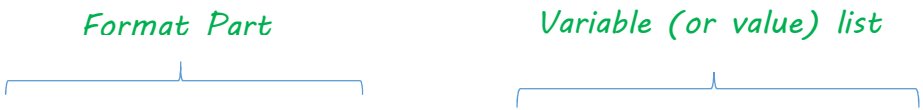
1. Objectives

- How to format a string to display values that constant or taken from variables.

2. String.Format (or string.Format),

The object string has several useful methods that you use to manipulate text. Use **String.Format** (or string.Format), if you need to insert the value of an object, variable, or expression into another string. For example, you can insert the value of a double type into a string to display it to the user as a single string. You can combine several values to display in the same string and you can format each value using formatting rules that are available in C# for different value types.

The string.Format ( ) method takes an argument which is a combination of a format part and a list variables. These are passed to the method inside the parentheses.( ). The formatting part is to be enclosed inside quotation marks, followed by a comma and then a list of variables, which you refer to in the formatting part. Here is an example:




```
string textOut = string.Format ( "{0,16:f2} C = {1,6:f2} F", index, convertedValue );
Console.WriteLine ( textOut );
```

where *index* is an integer and *convertedValue* is a double.

What happens in the above statement is that `"{0,16:f2}` will be replace by the current value of *index* and `{1,6:f2}` will be replaced by the value of the variable ***convertedValue***. The index-variable is the first variable in the list and therefore a 0 is used at the beginning of the expression `"{0,16:f2}` The variable **convertedValue** is second variable in the list.

The expression "`{0,16:f2}` has these parts:

- 0 the position of the variable in the list to be used.
- Comma is a separator
- **16** is the number of characters within which the value of index is to be aligned.
- :**f2** is the number of decimal positions to which the number is to be rounded off (rounding upwards). The letter “f” stands for floating-point and 2 sets the number of decimal positions.

If index has a value 2, it will be displayed as 2.00, right-adjusted within a 16-character width. Therefore, the method will be adding 12 spaces to the left of 2.00. The resulting substring will be displayed 12 spaces from the left.

In the same way, the expression `{1,6:f2}` instructs the computer to take the variable number 1 from the list, i.e. ***convertedValue***, round it up to two decimal places and display the result within a 6-character-width, right adjusted.

To left-adjust a substring, you need to add a minus sign before the width, {0, -16:f2}. There is no way for center adjusting.

All other characters and words that are not inside the curly braces are considered as constants and will be displayed as they are.

Important notes:

1. Console.WriteLine is designed to work in the same way as string.Format and you don't need to use string.Format. Instead of preparing a formatted string (textOUt in the code example above) and then using Console.WriteLine, you can write a formatted text to the Console directly:

```
Console.WriteLine ( "{0,16:f2} C = {1,6:f2} F", index, convertedValue );
```

   The good thing about string.Format is that you can use it at any time; you can use it with Console, with Windows Forms controls to display texts in components,other than a Console, and whenever you would need to have a formatted string.

2. To get straight columns when you have more lines using same formatting, you should use a font like "Courier New" which uses the same width for all characters. Such a font draws a space, a small 'i' and a big 'W', with the same number of pixels on the Console (or other components).

3. Using the $notation instead of String.Format

The $ special character identifies a string literal as an interpolated string. An interpolated string is a string literal that might contain interpolated expressions. When an interpolated string is resolved to a result string, items with interpolated expressions are replaced by the string representations of the expression results. This feature is available in C# 6 and later versions of the language.

String interpolation provides a more readable and convenient syntax to create formatted strings than a string.Format

```
string textOut = $"{index,16:f2} C = {convertedValue,6:f2} F";
Console.WriteLine ( textOut );
```

The formatting statement in above (also in the String.Format statement), we have values of string type ("C", and "F"), char (all blank spaces inside the quotations), int (index) and a double (convertedValue). Fantatic!

4. How to arrange values into columns

When you use **Console.Write(..)** , the text is written on the console but the cursor remains at the end of the text on the Console Window (when you run the program). Next time when you make a call to Console.Write, it continues from that position. **Console.WriteLine**, does the same thing but it moves the cursor to the beginning of next line (a carriage return), so next time the Console continues from the beginning of the line.

We make use of the above to arrange values in columns:

- Format a string within a certain width (12 chars). Make sure the width is greater than the number of chars in the longest value (to get spaces between the values). Use string.format: textOut = `string`.Format (….
- Write text using **Console.Write** for the first three values: **Console.Write(textOut);**
- Call **Console.WriteLine** after the third value is written: **Console.WriteLine();**
- Continue as above for the next three value, and the next three after that, etc.

To find out about "every third time", check if a counter mod 3 is zero (3, 6, 9, 12 …). The code for that is:

```
int p = 0;
int columns = 3;

for (int i=0; ...)
{
    //code for conversion, and prepare a string to write to Console
    Console.Write ( textOut );
    p++;
    if ((p % columns == 0) && (p >= columns))
        Console.WriteLine ( );
```

The % (modulus) operator gives the remainder after a whole number division. The expression p % 3 is read as p mod 3 and is true if the remainder after a whole number division of p by 3 is zero. Here we should perform the division so we separate the whole part and the fractional part. For example 11/5 = 2 1/5, the whole part is 2 (2*5 = 10) and then there 1 left which is not divisible by 5. The value 1 is the remainder and thus 11 % 5 = 1.

The table below shows what happens every time p is incremented. (A rather tough question: Why not use **i** as it also starts from 0?)

p	p % 3	Console.WriteLine?
0	`true`,0 /3 = 0, remainder 0	No because p is not > 3-1 = 2 (cols-1)
1, 2	`true`, 0/1 and 2 = 0	No because p is not > 2
3	3/3 leaves a remainder =0	Yes because p % 3 is 0 and p > 2
4	`false`, because 4/3 = 1 1/3. The whole part is 1 and the remainder is also 1.	No as p% 3 = 1 which is not 0
5	5/3 = 1 2/3, remainder is 2.	No as p% 3 = 2 which is not 0
6	6/3 = 2 remainder is 0	Yes as p%3 = 0
7	Continue as above	

5. Extra hints:

number += 1 is the same as: number = number + 1

text = “James “ + “Bond”; will give the result “James Bond”. The character '+' concatenates (puts together) strings. You can also use it as follows:

text = “James “;

text += ” Bond”; //a blank char at the beginning

and the result will be the same as above “James Bond”.

Remember:

A **for**-loop is used when the number of iterations is known on forehand.

A **while** loop is used when the number of iterations is not known. The loop will continue until one or more expressions are valid. The expression(s) is tested before the loop runs.

A **do while** loop is used during the same conditions as a **while** loop. The difference is that a do-while loop is executed at least once. The expression(s) are validated after the first iteration.

Good Luck.

Farid Naisan
Course Responsible and instructor