Formatting Text Messages and String Manipulation

Displaying text information

One thing you will learn about programming: if there is one way to do something, 9 times out of 10 there are 100 other ways to do the same thing.

Displaying text information is one of the most common things you will be doing. Most times you will want to create a message to display to the user that is comprised of many different pieces of information.

In C# it is easy to build text messages. Actually the term we use most often is **formatting** a text message. Also, remember that we call a text message a "string" in C#.

Types of strings

Thus far we have used strings in two different forms. The first will make obvious (I hope!) sense to you. When you asked the user to enter their name, we retrieved it from the keyboard (using the Console.ReadLine method) and stored it in a **string variable**.

You have been using strings in another form that isn't as obvious. Whenever you type in a text message in your code using the double quotes, you are making a string. It is called a **string literal** or a literal string. It is a string object in your program just like the string variable. Here is an example: "This is a string literal."

It is no less a string than your string variable userName. You can use it exactly like you use the variable. Next I will show you some of the things you can do with strings.

Formatting strings

Suppose in your program you want to say welcome to the user and remind them to feed their pet tonight. You have asked them (earlier) for their name and stored it in the string variable userName. You also asked them for the name of their pet and stored it in the string variable petName. You now want to tell them "Welcome <user name>! Don't forget to feed <pet name> tonight!"

Here are three ways you can do this:

Method 1:

```
Write("Welcome ");
Write(userName);
Write("! Don't forget to feed ");
Write(petName);
WriteLine(" tonight!");
```

Method 2:

```
WriteLine("Welcome " + userName + "! Don't forget to feed " + petName + " tonight!");
```

Method 3:

```
WriteLine("Welcome {0}! Don't forget to feed {1} tonight!", userName, petName);
```

Which do like the best? Which seems the most clear and understandable to you? Certainly the first is very messy … one thing to remember is the more lines of code you write, the greater the chance you will make a mistake! Keep it simple!!!!

You should always prefer the simplest solution. ***Always***. Learn to use the later two.

There is a big difference between methods 2 and 3 above.

Method 1

Forget it. It will be rare that you use this method. ☺ Too many lines of code, too much typing, too much opportunity for bugs.

Method 2

Method 2 is using "string arithmetic." It isn't really arithmetic but if you think about it, doesn't it make sense to be able to "add" two strings together? We call this concatenation and use the plus sign to represent it.

Next question: what types of strings are we concatenating? In order, we are concatenating first a string literal ("Welcome "), next a string variable (userName), next a different string literal ("! Don't forget to feed "), next a string variable (petName), and finally a third string literal (" tonight!").

One thing to note is the space at the end of the first string literal. Why is it there? If it wasn't, the word "Welcome" would be jammed right up against the user's name: "WelcomeJohn".

Method 3

This method allows you to write out the basic message the way you want it to appear and put placeholders in for the variable information. The string literal that contains the basic message with placeholders is called the **formatting string**. The way you identify the placeholders is with a number inside curly-brackets. Then after the formatting string, you list the string variables ***in order of the placeholder numbers*** to "plug into" the placeholder. The variables have to go in order. You can reference them in the formatting string in any order. So you could say:

```
WriteLine("Welcome {1}! Don't forget to feed {0} tonight!", petName, userName);
```

Be careful with this…to me the above is a bit confusing. Confusion = danger signal. It would be very easy to make a mistake doing this. Mistakes bad…clarity good.

Now remember, in computers we count from zero. So the first placeholder is {0}. The fifth is {4}.

Method 3 is a good route to go. It allows you to format the message in an easy to see way. With method 2 you have to make sure you have all the spacing right. Either works; it is more a matter of what is easier for you.

Formatting Text Messages and String Manipulation

Special characters in strings

Perhaps the question has occurred to you, what if I want to display a message to a user with a double quote in it. Let's say I want to display the following string literal:

> You said that your name is "Jane Doe". Is this correct?

How in the world do we format that? If I try to use the above in a string literal I have to put it inside double quotes. The first double quote starts the string, the next one ends it. The double quote right before the user's name will end the string. It would work out like this (bold red text is string literal):

> "**You said that your name is** "Jane Doe"**. Is this correct?**"

Obviously, that is not what we wanted. Moreover, the compiler is going to get very confused. What in the world is Jane Doe????

There is way of handling this. If we want to put in what is considered a special string formatting character (and there are a number of them), we simply precede it by a backslash:

> "**You said that your name is \"Jane Doe\". Is this correct?**"

The backslash says "just use the very next character exactly as it is; don't use it as a formatting instruction." Another example would be the curly-brackets around the placeholder numbers. If you wanted to use a { in your text you would write \{.

Control characters

You can use the backslash in a string in a slightly different way. You can use it to inject special string control characters. Suppose I wanted my message to appear on two lines. Rather then print two different messages I can inject a "newline" control command in the string with \n. Here are the most common control commands:

- **\n** – new line
- **\t** – tab character
- \\ – a backslash character
- **\"** – a double quote character