

LESSON 06 - STRINGS, CHARS, & DATA TYPES



In this lesson we will explore the string data type and how to create string variables. We will also briefly look at all the basic data types that are available in C#.

Sections:

- I. [STRINGS & STRING VARIABLES PAGE 1](#)
- II. [STRING INPUT PAGE 3](#)
- III. [BASIC DATA TYPES IN C++ \(INT, FLOAT, DOUBLE, BOOLEAN, & CHAR\) PAGE 4](#)

I. STRINGS & STRING VARIABLES:

We have seen the use of strings/text when we want to output messages/text to the terminal:

```
Console.WriteLine("Hello World!");
```

Output will be:

Hello World!

Everything inside the quotations "" is considered a **string**. We can also store strings inside **string variables** using the 'string' keyword:

```
string s = "Hello World!";  
Console.WriteLine(s);
```

Output will be:

Hello World!

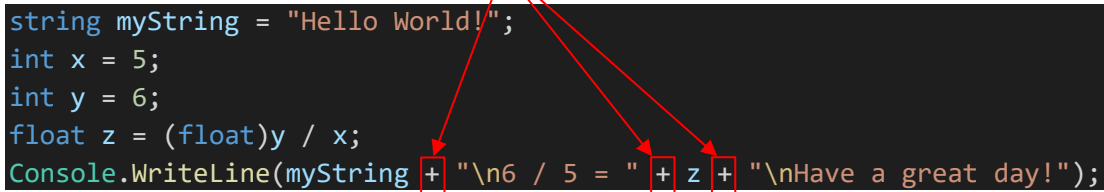
The keyword 'string' is used to declare a **variable** of a **string data type**.

Note: You may see some sample code that uses a capital 'S' on the keyword 'String', which is also acceptable:

```
String myString = "Hello World!";
```

As you can see, **strings** are used much the same as **ints** and **doubles**. Therefore, we can combine literal/variable strings, ints, and doubles in our **Console.WriteLine()** statements just like we did in previous lessons by using the plus '+' sign:

```
string myString = "Hello World!";  
int x = 5;  
int y = 6;  
float z = (float)y / x;  
Console.WriteLine(myString + "\n6 / 5 = " + z + "\nHave a great day!");
```

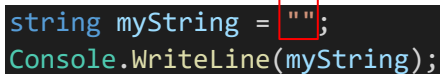


Output will be:

```
Hello World!  
6 / 5 = 1.2  
Have a great day!
```

Also know that if you do not give strings an **initial value** then you should default it to an **empty string**, which is a set of quotations "" with **nothing** inside. Without an **initial value** you will get a **build error**:

```
string myString = "";  
Console.WriteLine(myString);
```



Output will be:

Obviously, the output for the above code will be blank in this case.

You may also see sample code use the **empty string property** which looks like this:

```
string myString = string.Empty;
```

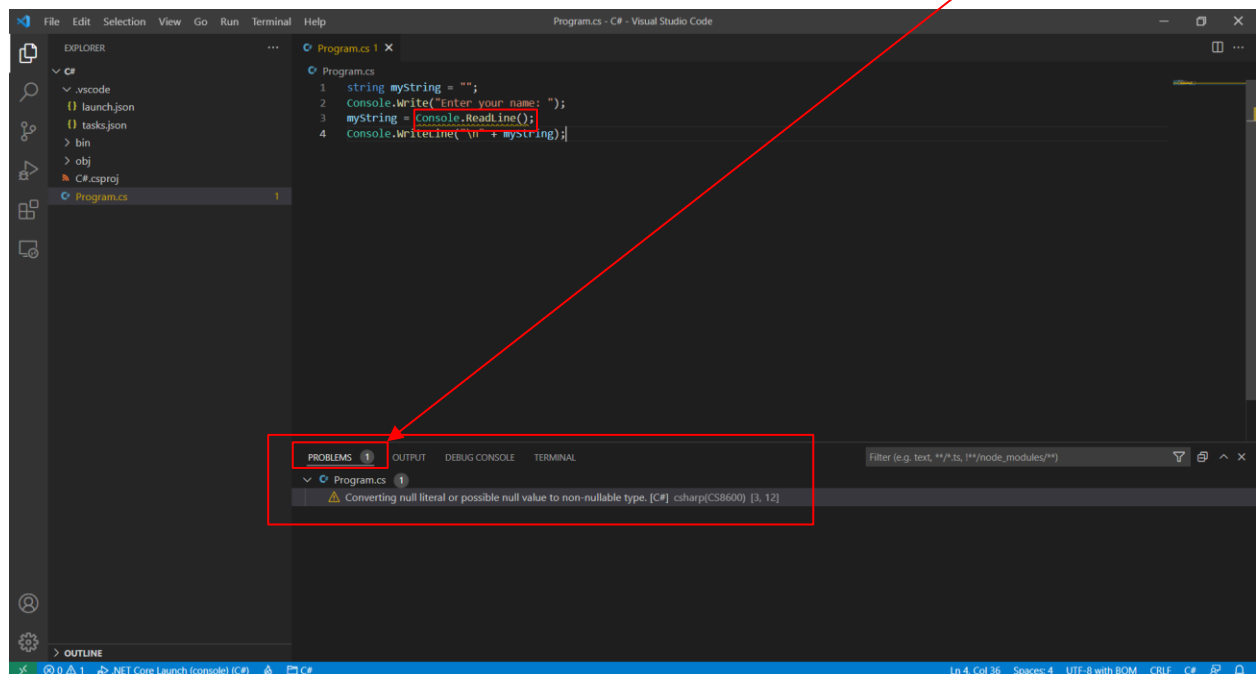


II. STRING INPUT:

When we want to receive **string input** from the user, we can use the standard **Console.ReadLine()** function:

```
string myString = "";
Console.Write("Enter your name: ");
myString = Console.ReadLine();
Console.WriteLine("\n" + myString);
```

We do not need to use the **Convert.ToInt32()** like we did in the last lesson since there is no need to convert anything to an integer (i.e., **Console.ReadLine()** returns a string which is what we want!). However, you will notice that we get a **warning** from the code above in the **problems** tab:



To remove this warning, simply put a question '?' mark in front of the keyword '**string**' of the '**myString**' variable and it will go away:

```
string? myString = "";
Console.Write("Enter your name: ");
myString = Console.ReadLine();
Console.WriteLine("\n" + myString);
```

We do not need to get into why using the question mark removes this warning. We will just do it! 😊

III. BASIC DATA TYPES IN C++ (INT, FLOAT, DOUBLE, BOOLEAN, & CHAR):

So far, we have used a few **data types**: **strings** for text, **ints** for integers, and **doubles** for decimal values. The following chart shows a more complete list of the **basic data types** used in **C#**:

Data Type	Size	Description
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
bool	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter, surrounded by single quotes
string	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

Notice both **'int'** and **'long'** hold **integer values**, however **'long'** uses more memory than **'int'**. The **'long'** data type allows for a wider range of numbers to store, but most of the time **'int'** is sufficient. We will mostly stick with **'int'** in this course.

Notice both **'float'** and **'double'** hold **decimal values**, however **'double'** uses **more memory** than **'float'**. The **'double'** data type allows for **up to 15 decimal digits**, whereas **'float'** only allows 6 to 7 decimal digits. We will mostly stick with **'double'** in this course.

```
int a = 1;
float b = 2.123456;
double c = 2.123456789012345;
```

The last two basic **data types** are **'bool'** and **'char'**. We will discuss **'bool'** variables in the next unit. The **'char'** data type holds a **single character**. In fact, **strings** are just chars grouped (stringed) together in a sequence. **Chars** are only **2 bytes** in size, so they are **very small**. Here is an example of using **'char'**:

```
char c1 = 'H';
char c2 = 'i';
char c3 = '!';
Console.Write(c1);
Console.Write(c1);
Console.Write(c1);
```

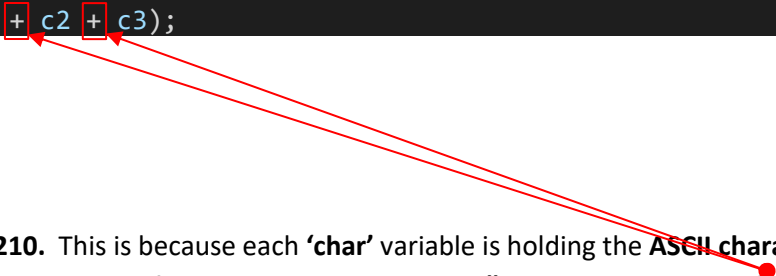
Output will be:

Hi!

One important thing to note about **chars** is that they use **single quotes** **' '** not double quotes **" "**.

Also notice that we used 3 separate **Console.Write()** statements instead of 1 **Console.WriteLine()** for our output above. Here is why:

```
char c1 = 'H';
char c2 = 'i';
char c3 = '!';
Console.WriteLine(c1 + c2 + c3);
```




Output will be:

210

In this case, output is **210**. This is because each **'char'** variable is holding the **ASCII character code** for each letter, **not** the letter itself. When **Console.WriteLine()** encounters the **plus '+'** sign it assumes you are adding these ASCII **'char'** values together, where **'H'** = 72, **'i'** = 105, and **'!'** = 33, therefore totalling to **'210'**. This is important to understand if you ever come across **chars** in sample code.

Also note that we have been using **chars** already, for example the **newline character "\n"**:

```
Console.WriteLine("Hello" + "\n" + "World!");
```




Output will be:

Hello World!

In this case, we can (should) use **single quotes ' '** for the **newline character** because the newline character is a **char**:

```
Console.WriteLine("Hello" + '\n' + "World!");
```



Output will be:

Hello World!

Both lines of code above will give the **same output**. You may be asking yourself why **'\n'** is considered a **char** if it contains 2 characters? The reason is because at runtime the compiler will see the forward slash **'\'** before the **'n'** which initiates an **'escape sequence'** to access special characters (in this case the **newline character**). In fact, there are many other special characters you can access using an **escape sequence** in **C#**, for example:

Simple Escape Sequences	
\'	single quote
\"	double quote
\\	backslash
\0	null
\a	audible bell
\b	backspace
\f	form feed - new page
\n	line feed - new line
\r	carriage return
\t	horizontal tab
\v	vertical tab

There are more if you search on google! Some of these come in useful when you need to output a specific character that is already used as part of the language syntax. For example, if we wanted to output the double quote " ", we would get an error if we tried the following:

```
Console.WriteLine("My name is \"Mr.Bellavia\"");
```

Trying to output **double quotes** " " inside our **Console.WriteLine()** will give you an error! The solution is to use **escape characters** (in this case ****):

```
Console.WriteLine("My name is \\\"Mr.Bellavia\\\"");
```

Output will be:

My name is "Mr. Bellavia"

One final note about **chars** is that the **default value** for a char value is **'\0'** which represents an **empty char**, also called the **null character**.

```
char myChar = '\0';
myChar = 'A';
Console.WriteLine(myChar);
```

Output will be:

A

There are also ways to get a **'char'** input from the user, but we are not going to spend too much time on dealing with chars.