Use the StringBuilder Class for Efficient String Manipulation Labs

Perform these labs on your own computer using Visual Studio 2022 or later, or VS Code 1.8x or later, to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Instantiating a StringBuilder Class

Using Visual Studio or VS code, create a console application named **StringBuilderSamples**.

Create an Empty StringBuilder Object

Open the **Program.cs** file and replace the contents of the file with the following code.

```
using System.Text;

StringBuilder sb = new();

DisplaySBInfo(sb);


//*****************************************
// Call to display StringBuilder information
//*****************************************
void DisplaySBInfo(StringBuilder sb)
{
  // Number of characters preallocated
  Console.WriteLine($"Capacity = {sb.Capacity}");
  // Total number of characters it can hold
  Console.WriteLine($"Max Capacity = {sb.MaxCapacity}");
  // Number of characters in the buffer
  Console.WriteLine($"Length = {sb.Length}");
}
```

Try It Out

Run the application and you should see something that looks like Figure 1. This figure is what is displayed on my Windows 11 machine.

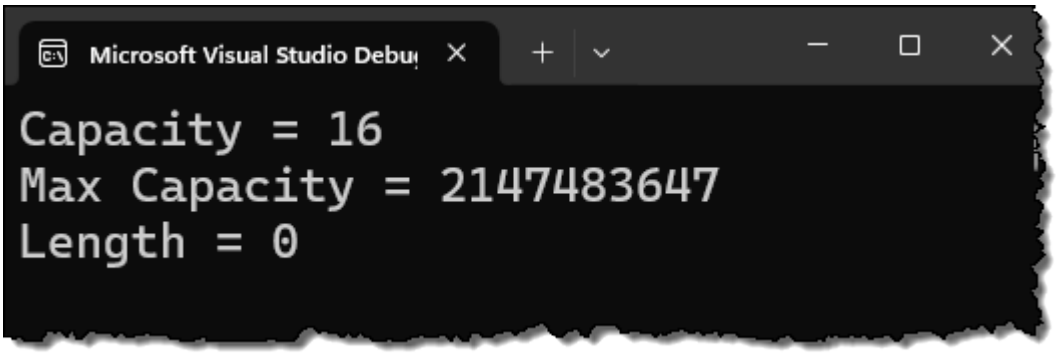

Figure 1: The results of initializing an empty StringBuilder object.

Lab 2: Initialize a StringBuilder Object with an Initial Capacity

Open the **Program.cs** file and modify the declaration of the StringBuilder object with the following.

```
StringBuilder sb = new(200);
```

Try It Out

Run the application and you should see that the capacity now reports a value of **200**. This means that you can put up to 200 characters of data into this instance of the StringBuilder object until it would have to reallocate more memory when you add more string data.

Lab 3: Initialize a StringBuilder Object with an Initial Value

Open the **Program.cs** file and modify the declaration of the StringBuilder object with the following.

```
StringBuilder sb = new("Acme Anvil");
```

Try It Out

Run the application and you should see the following data appear.

```
Capacity = 16
Max Capacity = 2147483647
Length = 12
```

Lab 4: Initialize a StringBuilder Object with an Initial Value and a Capacity

Open the **Program.cs** file and modify the declaration of the StringBuilder object with the following.

```
StringBuilder sb = new("Acme Anvil", 200);
```

Try It Out

Run the application and you should see the following data appear.

```
Capacity = 200
Max Capacity = 2147483647
Length = 12
```

Lab 5: Append Data to the StringBuilder Object Using the Append() Method

You can improve the efficiency of the StringBuilder object if you know approximately how much data you expect to add to the object. To illustrate, if you start with the default capacity of 16, and you append a string to the StringBuilder object that is more than 16 characters, this causes the **Capacity** property to double immediately. Once the length of the value within the StringBuilder object exceeds the value in the **Capacity** property, the buffer is doubled by the amount of the value in the **Capacity** property. To add new data into the StringBuilder object use the Append() method. Modify the declaration of the StringBuilder to the following code.

```
StringBuilder sb = new();
```

Add the following line of code immediately after the declaration to add a new sentence to the StringBuilder object.

```
sb.Append("The answer is always 42.");
```

Try It Out

Run the application and you should see the output looks like the following.

```
Capacity = 32
Max Capacity = 2147483647
Length = 24
```

As you can see, since you added a string that was 24 characters long, and that is greater than the initial capacity of 16, the StringBuilder added more buffer space that was equal to the initial capacity value times two.

Lab 6: Convert a StringBuilder Object to a String Object

So far all you have emitted on the console window are the number of characters within the StringBuilder object. What if you want to get the data back as a string so you can display it? For that you simply need to call the ToString() method on the StringBuilder object. Add the following code immediately following the call to the Append() method.

```
string value = sb.ToString();
Console.WriteLine(value);
Console.WriteLine();
```

Try It Out

Run the application and your console window should look something like Figure 2.

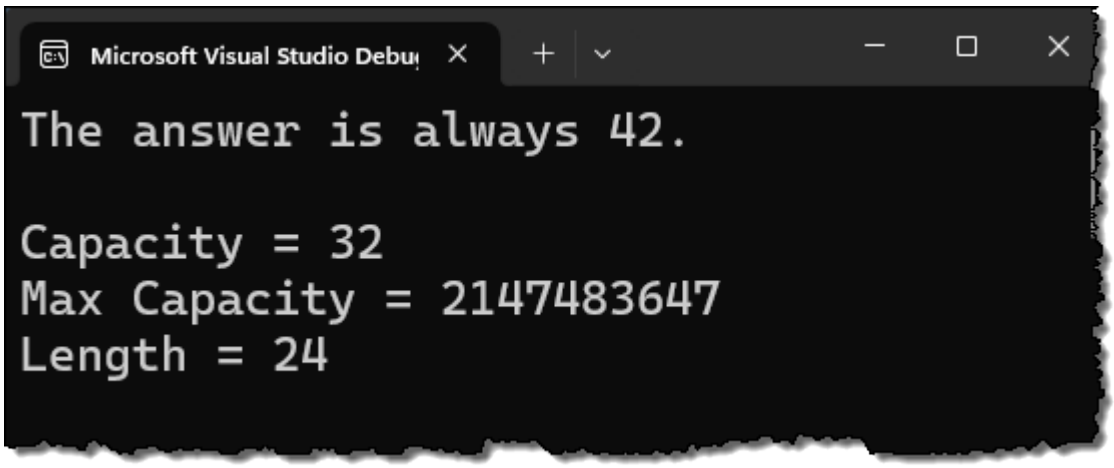

Figure 2: Use the ToString() method to convert the data to a string.

Of course, if all you want to do is to display the data on the console window, you do not need to place it into a string variable first, simply call the Console.WriteLine() as shown below.

```
Console.WriteLine(sb.ToString());
```

And, even simpler, you can just use the StringBuilder variable since if you reference any object by itself in C# it calls its ToString() method.

```
Console.WriteLine(sb);
```

Lab 7: String Class Versus a StringBuilder Class

You should not use a StringBuilder class every time you want to perform string manipulations, there are cases where using the String class is sufficient. For example, if the number of changes you are making to a string is small, use the String class. If you need to perform a lot of searching within the string you are building, use the String class because the StringBuilder class does not have searching methods. If you are concatenating hard-coded strings over a few lines in your method, use the String class because the compiler could combine those concatenations into a single operation.

The StringBuilder class is best when you are making an unknown number of changes to a string. For example, if you are looping through a set of data and concatenating that data into a single string value. Or, if you are making many changes to a string, then a StringBuilder class will be your best bet.

Lab 8: Append Lines to a StringBuilder Object Using the AppendLine() Method

If you want to add a carriage return/line feed to each value you append into your StringBuilder object, call AppendLine()instead of the Append() method. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

sb.AppendLine("Line 1");
sb.AppendLine("Line 2");
sb.AppendLine("Line 3");
Console.WriteLine(sb.ToString());
Console.WriteLine();

DisplaySBInfo(sb);
```

Try It Out

Run the application and your console window should look like Figure 3.

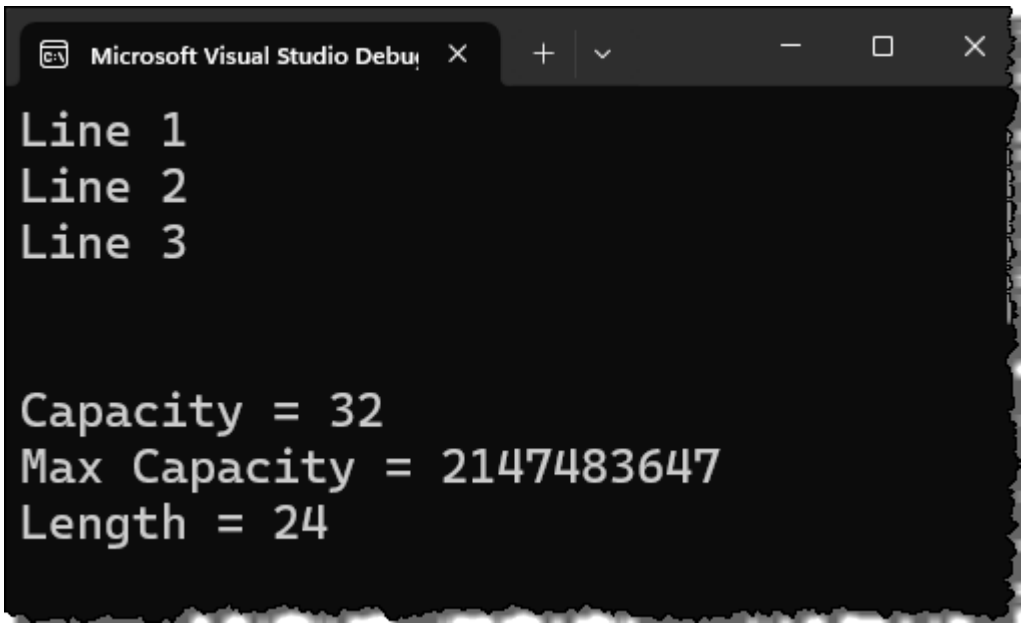

Figure 3: Use AppendLine() to add a CR/LF between each value added to the StringBuilder object.

Lab 9: Format Data Added to a StringBuilder Object Using the AppendFormat() Method

If you want to format the data going into the StringBuilder object, use the AppendFormat() method. This method allows you to build a composite format string with replaceable tokens. A composite format string contains curly braces with incrementing numbers into which you can substitute variable values. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

string name = "Acme Anvil";
decimal price = 99.99M;

sb.AppendFormat(
  "{0} is selling for {1:c}", name, price);
Console.WriteLine(sb.ToString());
```

Take a look at the string passed to the AppendFormat() method. You have two replaceable tokens {0} and {1:c}. Following the composite string you have two variables **name** and **price** that are passed to the AppendFormat() method. The **name** variable is replaced into the string at the location of the {0} token, and the **price** variable is replaced into the string at the location of the {1} token. You may use any of the standard .NET format string moniker such as "c" for current, "d" for date, etc.

Try It Out

Run the application and your console window should have a string that looks like the following.

```
Acme Anvil is selling for $99.99
```

Instead of composite string formatting you may use C# string interpolation in combination with the Append() method as shown in the following example.

```
sb.Append($"{name} is selling for {price:c}");
```

This line is equivalent to using the AppendFormat() method and the composite string. I prefer using string interpolation as it is much easier to read.

Lab 10: Append a String Array to a StringBuilder Object Using the AppendJoin() Method

If you have an array of strings that you wish to add to the StringBuilder object, you do not need to iterate over the array and add each element one at a time to it. Call the AppendJoin() method passing in the delimiter to separate each element in the array, and the array itself as shown in the following code. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

string[] names = ["Acme Anvil", "Acme Rocket Skates"];

sb.AppendJoin(", ", names);
Console.WriteLine(sb.ToString());
```

Try It Out

Run the application and your console window should have a string that looks like the following.

```
Acme Anvil, Acme Rocket Skates
```

Lab 11: Append a List<T> Collection to a StringBuilder Object Using the AppendJoin() Method

The AppendJoin() method not only supports a string array, but it also supports any IEnumerable<T> object. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

List<string> names = ["Acme Anvil", "Acme Rocket
Skates"];

sb.AppendJoin(", ", names);
Console.WriteLine(sb.ToString());
Console.WriteLine();
```

Try It Out

Run the application and your console window should have a string that looks like the following.

```
Acme Anvil, Acme Rocket Skates
```

Lab 12: Search and Replace in a StringBuilder Object Using the Replace() Method

Just like you can do a search and replace within a string object, the StringBuilder object has a Replace() method that allows you to replace one string, or one

character, with another. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

sb.AppendLine("Line 1");
sb.AppendLine("Line 2");
sb.AppendLine("Line 3");
sb.AppendLine("Line 4");

sb.Replace("Line ", "New Line ");
Console.WriteLine(sb.ToString());

DisplaySBInfo(sb);
```

Try It Out

Run the application and your console window should have a string that looks like the following.

```
New Line 1
New Line 2
New Line 3
New Line 4
```

Lab 13: Remove All Data From a StringBuilder Object Using the Clear() Method

If you are finished using the StringBuilder object you may remove all of the data from the object by calling the Clear() method. The Clear() method does not reset the **Capacity** property. The capacity of the StringBuilder object will still be as great, or just slightly less than the capacity of the object before the Clear() method was called. Modify the **Program.cs** file to look like the following (be sure to leave the DisplaySBInfo() method).

```
using System.Text;

StringBuilder sb = new();

List<string> names = ["Acme Anvil", "Acme Rocket
Skates"];

sb.AppendJoin(", ", names);
sb.AppendLine();
sb.AppendLine("Line 1");
sb.AppendLine("Line 2");
sb.AppendLine("Line 3");
sb.AppendLine("Line 4");
sb.AppendLine("Line 5");
sb.AppendLine("Line 6");
sb.AppendLine("Line 7");
sb.AppendLine("Line 8");
Console.WriteLine(sb.ToString());

DisplaySBInfo(sb);
Console.WriteLine();
sb.Clear();
DisplaySBInfo(sb);
```

Try It Out

Run the application and your console window should have a string that looks like the following.

```
Acme Anvil, Acme Rocket Skates
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8

Capacity = 128
Max Capacity = 2147483647
Length = 98

Capacity = 117
Max Capacity = 2147483647
Length = 0
```

Summary

When you have a lot of string manipulation to do in C# use the StringBuilder class. This class is more efficient than using the String class as it is mutable whereas the String class is immutable. You can add data easily using the different append methods, you can search and replace data, and you can clear out the data when you are through.