Exercise: Find Output

```
class Program
{
static void Main()
{
string sampleText = "hello world, welcome to C# programming world";

Console.WriteLine("IndexOf examples:");
Console.WriteLine("IndexOf('w') -> " + sampleText.IndexOf('w'));
Console.WriteLine("IndexOf('o', 5) -> " + sampleText.IndexOf('o', 5));
Console.WriteLine("IndexOf(\"world\") -> " + sampleText.IndexOf("world"));
Console.WriteLine("IndexOf(\"world\", 10) -> " + sampleText.IndexOf("world", 10));

Console.WriteLine("\nLastIndexOf examples:");
Console.WriteLine("LastIndexOf('w') -> " + sampleText.LastIndexOf('w'));
Console.WriteLine("LastIndexOf('o', 20) -> " + sampleText.LastIndexOf('o', 20));
Console.WriteLine("LastIndexOf(\"world\") -> " + sampleText.LastIndexOf("world"));
Console.WriteLine("LastIndexOf(\"world\", 30) -> " + sampleText.LastIndexOf("world", 30));
}
}
```

Answer:

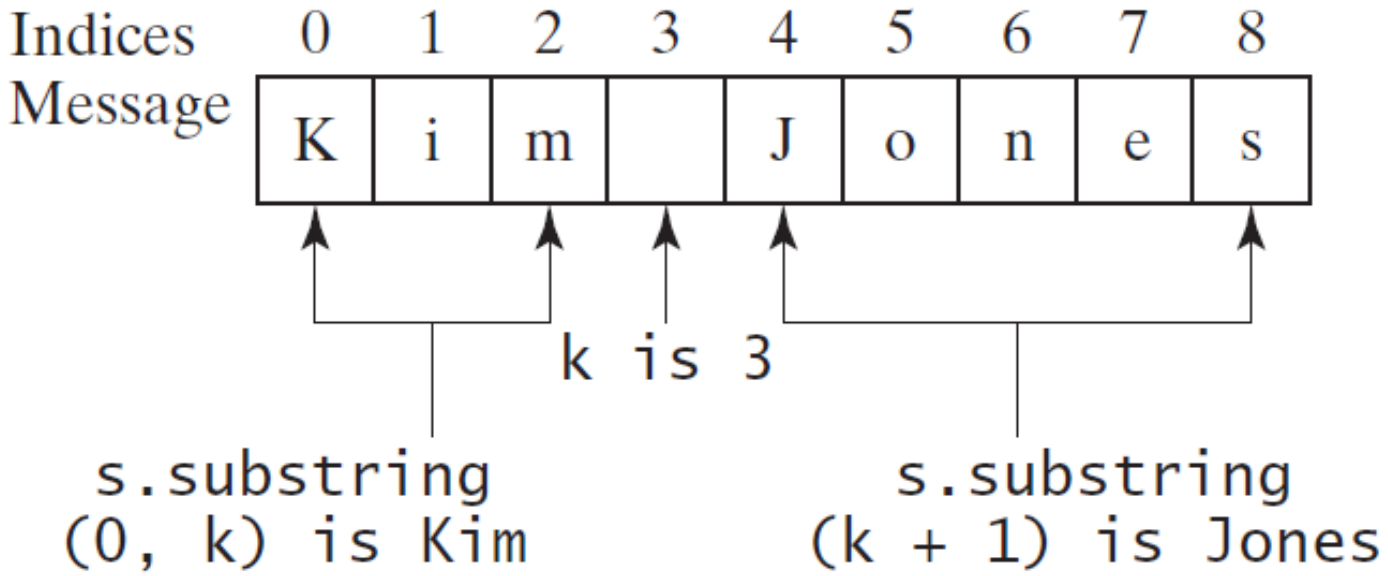
```
IndexOf examples:
IndexOf('w') -> 6
IndexOf('o', 5) -> 7
IndexOf("world") -> 6
IndexOf("world", 10) -> 39

LastIndexOf examples:
LastIndexOf('w') -> 39
LastIndexOf('o', 20) -> 17
LastIndexOf("world") -> 39
LastIndexOf("world", 30) -> 6
```


SDU

Finding a Character or a Substring in a String

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

What is special about substring method???



Answer: This substring method is an overloaded method and it either takes as input the beginIndex or the beginIndex and number of characters to extract.

SDU

Conversion between Strings and Numbers

Where do you need these conversions????? Any idea????

- We can convert a numeric string into a number using

```
int intValue = int.Parse(intString);
double doubleValue = double.Parse(doubleString);
```

- We can also convert numbers back to string using the string concatenation operator

```
String s = number + "";
```

- **Note** that if the string is not a numeric string, the conversion would cause a runtime error.

Answer: We need these methods while reading/writing from/to a text file.

Mini Exercise

- Show two ways to concatenate the following two strings together to get the string "Hi, mom.":

```
String hi = "Hi, ";
String mom = "mom.";
```

- How long is the string returned by the following expression? What is the string?

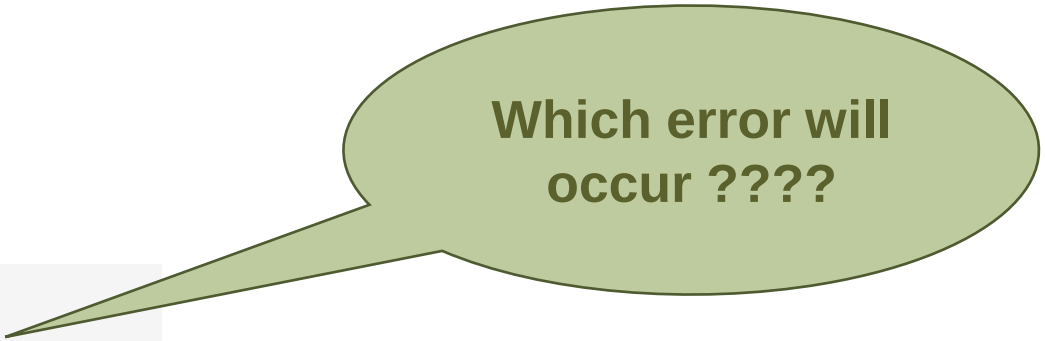
```
"Was it a car or a cat I saw?".Substring(9, 3)
```

Answer:

- string.Concat(hi,mom) or hi + mom.
- It's 3 characters in length: car. It does not include the space after car.
- 1020
- 30

```
string x = "10";
int y = 20;
string z = x + y;
int p = y + int.Parse(x);
System.Console.WriteLine(z);
System.Console.WriteLine(p);
```

Error Example




```
public static void Main(string[] args)
{
    int result = CalculateFactorial(5);
    Console.WriteLine($"Factorial: {result}");
}

static int CalculateFactorial(int n)
{
    return n * CalculateFactorial(n - 1);
}
```

Answer:

(Unhandled) **StackOverflowException:** This exception occurs when a program exceeds the allowed memory space on the call stack. This usually happens due to excessive recursion, as in this example. Each recursive call to CalculateFactorial adds a new frame to the call stack, and without a base case to stop the recursion, the stack eventually overflows.

SDU

SDU

Mini Exercise 1

Question: Is there anything wrong with this exception handler as written? Will this code compile?

```
try {

} catch (Exception e) {

} catch (ArithmeticException a) {

}
```

Answer:

This first handler catches exceptions of type Exception; therefore, it catches any exception, including ArithmeticException. The second handler could never be reached. This code will not compile.

When handling exceptions in C#, always place the more specific catch blocks before the more general ones to avoid unreachable code errors.

SDU

Mini Exercise 2

Question: What exception types can be caught by the following handler?

```
catch (Exception e) {

}
```

What is wrong with using this type of exception handler?

Answer: While this handler can catch any exception, it's generally not recommended to use it as the only exception handler in your code, because

Lack of specificity: It doesn't differentiate between different types of exceptions, so you can't handle specific exceptions in a targeted way.

Hides potential issues: It might catch exceptions that you didn't anticipate, potentially masking underlying problems in your code.

Mini Exercise 3: Write Output


```
public class Test: Exception { }

public class Example
{
    public static void Main(string args)
    {
        try
        {
            throw new Test();
        }
        catch (Test t)
        {
            Console.WriteLine("Got the Test Exception");
        }
        finally
        {
            Console.WriteLine("Inside finally block");
        }
    }
}
```

What is special about Test class?

Answer:

This a a custom exception
Got the Test Exception
Inside finally block

SDU

Mini Exercise 4a: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try {
            int[] a = new int[10];
            a[11] = 9;
        }
        catch (                                    ) {
            Console.WriteLine("                    ");
        }
    }
}
```

Answer:
IndexOutOfRangeException

https://beginnersbook.com/2013/04/exception-handling-examples/

SDU

Mini Exercise 4b: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            int num = int.Parse("XYZ");
            Console.WriteLine(num);
        }
        catch (                    e) {
            Console.WriteLine(                                  );
        }
    }
}
```

Answer:
FormatException

https://beginnersbook.com/2013/04/exception-handling-examples/

SDU

Mini Exercise 4c: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            string str = "beginnersbook";
            Console.WriteLine(str.Length);
            char c = str[0];
            c = str[40];
            Console.WriteLine(c);
        }
        catch (
            Console.WriteLine("
        }
    }
}
```

Answer:
IndexOutOfRangeException

https://beginnersbook.com/2013/04/exception-handling-examples/

Mini Exercise 4d: Identify Exception

```
public class ExceptionDemo {
    public static void Main(string[] args) {
        try
        {
            string str = null;
            Console.WriteLine(str.Length);
        }
        catch (                                    ) {
            Console.WriteLine(                                    );
        }
    }
}
```

Answer:
NullReferenceException

https://beginnersbook.com/2013/04/exception-handling-examples/

Mini Exercise 4e: Find Output


```
try
{
    string input = null;
    int result = int.Parse(input); // Null value parsing
}
catch (ArgumentNullException ex)
{
    Console.WriteLine("Null argument exception caught.");
}
catch (FormatException ex)
{
    Console.WriteLine("Format exception caught.");
}
catch (Exception ex)
{
    Console.WriteLine($"General exception caught: {ex.Message}");
}
```

Answer:

- When input=null, ArgumentNullException is thrown when a method receives a null argument where it's not allowed.
- When input= “”, int.Parse() expects a valid numeric string, and since “” is not a valid format for a number, it throws a FormatException.
- So, the **FormatException** is caught by the second catch block, and the message "Format exception caught." is displayed.