200 XP

Exercise - Use the string's IndexOfAny() and LastIndexOf() helper methods

25 minutes

In this exercise, you use the `IndexOfAny()` method to find the first location of any of the `string` from selected array. You also use `LastIndexOf()` to find the final location of a string within another string.

Retrieve the last occurrence of a sub string

You increase the complexity of the `message` variable by adding many sets of parentheses, then write code to retrieve the content inside the **last** set of parentheses.

1. Select and delete all code lines in the Visual Studio Code Editor.
2. Update your code in the Visual Studio Code Editor as follows:

```
C#

string message = "(What if) I am (only interested) in the last (set of parentheses)?";
int openingPosition = message.LastIndexOf('(');

openingPosition += 1;
int closingPosition = message.LastIndexOf(')');
int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosition, length));
```

3. Save your code file, and then use Visual Studio Code to run your code. You should see the following output:

```
Output

set of parentheses
```

The key to this example is the use of `LastIndexOf()`, which you use to get the positions of the last opening and closing parentheses.

Retrieve all instances of substrings inside parentheses

This time, you update the `message` to have three sets of parentheses, and you write code to extract any text inside of the parentheses. You're able to reuse portions of the previous work, but need to add a `while` statement to iterate through the string until all sets of parentheses are discovered, extracted, and displayed.

1. Update your code in the Visual Studio Code Editor as follows:

C#

```
string message = "(What if) there are (more than) one (set of parentheses)?";
while (true)
{
    int openingPosition = message.IndexOf('(');
    if (openingPosition == -1) break;

    openingPosition += 1;
    int closingPosition = message.IndexOf(')');
    int length = closingPosition - openingPosition;
    Console.WriteLine(message.Substring(openingPosition, length));

    // Note the overload of the Substring to return only the remaining
    // unprocessed message:
    message = message.Substring(closingPosition + 1);
}
```

2. Save your code file, and then use Visual Studio Code to run your code. You should see the following output:

Output

```
What if
more than
set of parentheses
```

3. Take a minute to observe last line of code inside the `while` loop, pulled out in the following code:

C#

```
message = message.Substring(closingPosition + 1);
```

When you use `Substring()` without specifying a length input parameter, it will return every character after the starting position you specify. Since `message = "(What if) there are (more than) one (set of parentheses)?"` there's advantage to removing the first set of parentheses `(What if)` from the value of `message`. What remains is then processed in the next iteration of the `while` loop.

4. Take a minute to consider what happens during the final iteration of the `while` loop, when only the final `?` character remains.

   The followings code addresses handling the end of the string:

   ```
   C#

   int openingPosition = message.IndexOf('(');
   if (openingPosition == -1) break;
   ```

   The `IndexOf()` method returns `-1` if it can't find the input parameter in the string. You merely check for the value `-1` and `break` out of the loop.

Work with different types of symbol sets

This time, you search for several different symbols, not just a set of parentheses.

You update the `message` string, adding different types of symbols like square `[]` brackets and curly braces `{}`. To search for multiple symbols simultaneously, use on `.IndexOfAny()`. You search with `.IndexOfAny()` to return the index of the first symbol from the array `openSymbols` found in the `message` string.

1. Update your code in the Visual Studio Code editor as follows:

   ```
   C#

   string message = "Help (find) the {opening symbols}";
   Console.WriteLine($"Searching THIS Message: {message}");
   char[] openSymbols = { '[', '{', '(' };
   int startPosition = 5;
   int openingPosition = message.IndexOfAny(openSymbols);
   Console.WriteLine($"Found WITHOUT using startPosition:
   {message.Substring(openingPosition)}");

   openingPosition = message.IndexOfAny(openSymbols, startPosition);
   ```

```
Console.WriteLine($"Found WITH using startPosition {startPosition}:
{message.Substring(openingPosition)}");
```

2. Save your code file, and then use Visual Studio Code to run your code.

   You should see the following output:

   Output

   ```
   Searching THIS message: Help (find) the {opening symbols}
   Found WITHOUT using startPosition: (find) the {opening symbols}
   Found WITH using startPosition 6:  (find) the {opening symbols}
   ```

3. Take a minute to review the code previously entered.

   You used `.IndexOfAny()` *without*, and then *with*, the starting position overload.

   Now that you found an opening symbol, you need to find its matching closing symbol.

4. Update your code in the Visual Studio Code Editor as follows:

   C#

   ```
   string message = "(What if) I have [different symbols] but every {open symbol}
   needs a [matching closing symbol]?";

   // The IndexOfAny() helper method requires a char array of characters.
   // You want to look for:

   char[] openSymbols = { '[', '{', '(' };

   // You'll use a slightly different technique for iterating through
   // the characters in the string. This time, use the closing
   // position of the previous iteration as the starting index for the
   //next open symbol. So, you need to initialize the closingPosition
   // variable to zero:

   int closingPosition = 0;

   while (true)
   {
       int openingPosition = message.IndexOfAny(openSymbols, closingPosition);

       if (openingPosition == -1) break;

       string currentSymbol = message.Substring(openingPosition, 1);
   ```

```
    // Now  find the matching closing symbol
    char matchingSymbol = ' ';

    switch (currentSymbol)
    {
        case "[":
            matchingSymbol = ']';
            break;
        case "{":
            matchingSymbol = '}';
            break;
        case "(":
            matchingSymbol = ')';
            break;
    }

    // To find the closingPosition, use an overload of the IndexOf method to
specify
    // that the search for the matchingSymbol should start at the openingPosi-
tion in the string.

    openingPosition += 1;
    closingPosition = message.IndexOf(matchingSymbol, openingPosition);

    // Finally, use the techniques you've already learned to display the sub-
string:

    int length = closingPosition - openingPosition;
    Console.WriteLine(message.Substring(openingPosition, length));
}
```

5. Take a few minutes examine the previous code and reading the comments that help explain the code.

6. Continuing examining the code and locate the following line of code using `IndexOf()` to define `closingPosition`:

```
C#

closingPosition = message.IndexOf(matchingSymbol, openingPosition);
```

The variable `closingPosition` is used to find the length passed into the `Substring()` method, but it is also used to find the next `openingPosition` value:

```
C#
```

```
int openingPosition = message.IndexOfAny(openSymbols, closingPosition);
```

For this reason, the `closingPosition` variable is defined outside of the `while` loop scope and initialized to `0` for the first iteration.

7. Save your code file, and then use Visual Studio Code to run your code. You should see the following output:

Output

```
What if
different symbols
open symbol
matching closing symbol
```

Recap

Here's the most important things to remember:

- `LastIndexOf()` returns the last position of a character or string inside of another string.
- `IndexOfAny()` returns the first position of an array of `char` that occurs inside of another string.

Check your knowledge

1. What method should be used to search for the first occurrence of a search term in a long string? *

- ◯ `IndexOfAny()`
- ◯ `LastIndexOf()`
- ◯ `Substring()`

Check your answers