**CSC 180 Assignment #5**

Exercise 7.4. The purpose of this exercise is to review encapsulation and generalization.

1. Encapsulate the following code fragment, transforming it into a method that takes a

string as an argument and that returns the value of count.

using System;

namespace CSC180

{

public class Program

{

public static void Main()

{

string s = "((3 + 7) \* 2)";

int len = s.Length;

int i = 0;

int count = 0;

while (i < len)

{

char c = s[i];

if (c == '(')

{

count = count + 1;

}

else if (c == ')')

{

count = count - 1;

}

i = i + 1;

}

Console.WriteLine(count);

}

}

}

2. In a sentence or two, describe what the resulting method does (without getting into

the details of how).

Takes an index of a string and runs a count to the end and back to the beginning of the string index.

3. Now that you have generalized the code so that it works on any string, what could

you do to generalize it more?

using System;

namespace CSC180

{

public class Program

{

public static void Main()

{

string s = "(abcdefg)";

int len = s.Length;

int i = 0;

int count = 0;

while (i < len)

{

char c = s[i];

if (c == 0)

{

count = count + 1;

}

else if (c == s.Length - 1)

{

count = count - 1;

}

i = i + 1;

}

Console.WriteLine(count);

}

}

}

Exercise 7.5. The point of this exercise is to explore types and fill in some of the details that aren't covered in the chapter.

1. Create a new program named Test.cs and write a Main method that contains expressions that combine various types using the + operator. For example, what happens when you \add" a string and a char? Does it perform addition or concatenation? What is the type of the result? (How can you determine the type of the result?)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Test

{

class Program

{

static void Main(string[] args)

{

string a = "testing";

int i = 123;

Console.WriteLine(a+i);

}

}

}

I got a concatenate when I used a + for a string and an integer. By definition, concatenate links strings or integers together in a series whereas addition is a mathematical function resulting in a new integer. You can also use the &= operator to eliminate ambiguity and to provide self-documenting code for concatenation

2. Make a bigger copy of the following table and fill it in. At the intersection of each pair of types, you should indicate whether it is legal to use the + operator with these types, what operation is performed (addition or concatenation), and what the type of the result is.

A screenshot of a computer

Description automatically generated with low confidence

Concat bool Concat char Concat bool Concat string

Concat char Concat char Concat char Concat string

Concat bool Concat char Add int Concat string

Concat string Concat string Concat string Concat string

3. Think about some of the choices the designers of C# made when they filled in this table. How many of the entries seem unavoidable, as if there were no other choice? How many seem like arbitrary choices from several equally reasonable possibilities? How many seem problematic?

I think that the rules of the + operator leave little to choice. If either operand is a string, char or bool, the result will be concatenation. Some seem problematic but if you want to change the result, you can specify that in the console or convert afterwards.

Exercise 7.6. What is the output of this program? Describe in a sentence what Mystery does (not how it works).

Mystery takes the index of a string and numbers each character in reverse alphabetical order. At the end, it also prints on the console the reverse string.

Exercise 7.10. A word is said to be "abecedarian" if the letters in the word appear in alphabetical order. For example, the following are all 6-letter English abecedarian words.

abdest, agnosy, biopsy, deux, dimpsy

1. Describe a process for checking whether a given word (string) is abecedarian, if the word contains only lower-case letters. Your process can be iterative or recursive.

The program must check whether a string is in alphabetical order or not. To do this, turn characters into integer values. If the current index element is less than the index element before it, it is false.

2. Implement your process in a method called IsAbecedarian.

using System;

public class Program

{

public static bool isAbecedarian(string s)

{

int n = s.Length;

for (int i = 1; i < n; i++)

{

if (s[i] < s[i - 1])

return false;

}

return true;

}

public static void Main()

{

string s = "zaz";

if (isAbecedarian(s))

Console.WriteLine("Yes");

else

Console.WriteLine("No");

}

}