Writing Exercise

Megha Bhatt

December 2014

1 Introduction

A technical document communicates information clearly, concisely and accurately to different audiences. The following document presents three ways of communicating the purpose of the original function to various audiences.

2 Description

This section briefly highlights the overall use of the function to an audience with limited or no understanding of programming languages.

3 Commented Code

The code has side comments in order for users to understand the function of each line by looking at the comments. This part of the document can be useful to developers, or others who work closely with the raw code. It can be useful for future users of the code, and will give them a better understanding in order to modify or change the code.

4 Technical Description

This section explains the function to users that are familiar with the technical vocabulary of programming languages. It also gives examples to users for a better understanding of the function.

Description

The function **findNeedles** has two parameters, a string (called **haystack**) and a list of strings (called **needles**). A string consists of one or more characters, and is written within quotations for distinction.

Firstly, the function checks to see if the number of items in the list, **needles** is greater than 5, which if true, displays the message “too many words!”

If the list, **needles**, contains 5 items or less, the function counts each item’s occurrence in the string, **haystack**. In order to do that, first the string, **haystack**, is split into a list of strings, called **words**, so that comparisons can be made between the two lists. The values contained inside of **haystack** are split by either a single space, a tab, a newline, or a linefeed.

Then the function checks whether the first item in the list, **needles**, appears in **words**. If it does not, then the same item in **needles** is compared to the next item in **words**, and so on. Once that item in **needles** has been compared to all items in **words**, the next item in **needles** is compared to each item in **words**. For every time an item in **needles** occurs in the list, **words**, the corresponding *countArray* value is increased by 1. Therefore, keeping a tally of the total number of occurrences of that item in **needles** contained in the list **words**. Once every item in the list, **needles**, is compared, each item in **needles** is displayed along with the number of times it occurred in the **haystack**.

```
haystack:   [string]            contains a string of unknown length
needles:    [list of strings]   contains a list of strings
countArray: [list of integers]  used to keep track/count the number of times each needle item occurs
haystack_ss:[list of strings]   contains the result of haystack being split up by multiple delimiters
words:      [list of strings]   contains the words that are in haystack
```

```
//Function: findNeedles(string)(list of string) -> string:integer
// Purpose: The purpose of this function is to evaluate the total number of occurrences of each item in the list, needles,
that occur in the string, haystack.

using System;
using System.Collections.Generic;   //Declaration of namespaces

class Program {     //Declaration of class Program
  public static void findNeedles(string haystack, List<string> needles) {
    if (needles.Count > 5) {
      Console.Error.WriteLine("too many words!");   //If needles contains more than 5 elements, display an error
    }

    else{   //Beginning of the else case
      int[] countArray = new int[needles.Count];    //Creates a new array of type integer with the same number of values
                                                    //as in needles
      for(int j = 0; j < needles.Count; j++){   //Evaluates every string value in needles in a loop
        countArray[j] = 0;      //Assign the value of 0 to all the elements in countArray
        }

      // Split haystack into words
      string[] haystack_ss =    //Create a new array of type string which contains the result of splitting haystack by
        haystack.Split(new Char[] {' ', '\t', '\n', '\r'});     //delimiters: single spaces, tabs, newlines, and line
      List<string> words = new List<string>();  //Create and initialize the list words, an empty list of type strings
      words.AddRange(haystack_ss);  //Add the elements of haystack to the words list

      for(int i = 0; i < needles.Count; i++) {  //Compare each string in needles to each string in words
        for(int j = 0; j < words.Count; j++) {
          if(words[j] == needles[i]){       //If needle matches word, increment the countArray's value that corresponds
            countArray[i]++;                // to the needle value by 1
          }
        }
      }
      for(int j = 0; j < needles.Count; j++){
        Console.WriteLine(needles[j], ": ", countArray[j]);  //Output out each needle and the number of times it
                                                             //occurred in haystack
```

Technical Description

The purpose of the function **findNeedles** is to count the number of times each string in the list, **needles**, occurs within the string, **haystack**. The list, **needles**, may contain up to a maximum of 5 items.

The snippet of code begins with the System, and System.Collections.Generic namespace declarations that will be used by functions within this program.

The function **findNeedles** accepts two parameters, a string **(haystack)** and a list of string values **(needles)**.

First, an if-else condition is declared within the function. The if statement checks the number of elements in the list of strings, **needles**, by using the *needles.Count* method. If the list of strings, **needles**, contains more than 5 items, the error *"too many words!"* is outputted to the console. Otherwise, the else condition is executed.

The else condition first initializes a single dimensional array of type integers, **countArray** with the same number of elements as in the list, **needles**. Then, inside a for-loop, every element of the array, **countArray**, is assigned a value of 0.

A list of type string, **haystackss**, is created and assigned values obtained from splitting the string, **haystack**, using delimiters: a single whitespace, tabs, newlines, and line feeds. A new list, **words**, is created, and the elements from **haystack_ss** are added to **words** using the method *AddRange.*

Then, **needles** is compared to **words** through the use of a nested for-loop. The first string in **needles** is compared with every string in the list, **words** using a nested for-loop. If they are equal, the value in **countArray** that corresponds to the current needle is incremented by 1. This cycle is repeated for each element in **needles** until every element in **needles** is compared with every element in **words**. Finally, the function then outputs, to the console, each string in **needles** and the value of the corresponding **countArray**, therefore displaying the total number of occurrences of each element of **needles** in **haystack**.

Examples

Case 1: Some needles occur in haystack

needles = "hi" , "is" , "Megha"
haystack = "Hello, my name is Megha. I am in the haystack."

Output:
```
hi : 0
is : 1
Megha: 1
```

Case 2: More than 5 needles in needles

needles = “hello” , “Megha” , “is” , “in” , “the” , “haystack”
haystack = “Hello, my name is Megha. I am in the haystack.”

Output:
```
too many words!
```

Case 3: Empty list

needles = “ ”
haystack = “It is a lovely day today.”

Output:

Case 4: Empty parameters

needles = “ ”
haystack = “ ”

Output:

Case 5: Spaces within elements of needles

needles = “A B C” , “1 2 3”
haystack = “A B C”

Output:
```
A B C: 0
1 2 3: 0
```