SCDT41 Programming and Software Fundamentals

Mikhail Nazarov

Table of Contents

**Task 12**

First solution2

Second solution3

Comparison4

**Task 25**

First solution5

Second solution7

Comparison8

**Task 39**

First solution9

Second solution9

Comparison10

**Task 411**

First solution11

Second solution12

Comparison13

**Task 514**

First solution14

Second solution14

Comparison15

**Reference16**

**Bibliography17**

**Task 1**

**First solution**

For the first task is required to find the maximum total number, without exceeding the set amount.

The algorithm represented for the first solution begins with declaring variables: an array with unique values, an integer T eques 14, and two integers for a total amount equal to 0. The next step is setting a for loop. For loop goes through each value of the array and contains an if statement inside. The if statement has a formula, where each value in the array equals total until is less or equal of value T. It then, assigns the number of values to integer answer. The last step is to output the result to the console.

**using** System;

**namespace** Task1

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set variables*

**int**[] A = { 3, 5, 2, 1, 7, 4 };

**int** T = 14;

**int** total = 0;

**int** answer = 0;

*// for loop to go through all numbers in array*

**for** (**int** i = 0; i < A.Length; i++)

{

*// adds numbers until is less than 14*

**if** ((total += A[i]) <= T)

{

answer += A[i];

}

**else**

{

Console.WriteLine("Total Items added: " + i);

Console.WriteLine("TOTAL VALUE: " + answer);

**break**;

}

}

}

}

}

**Second solution**

As well as the first step in the previous method, the algorithm begins with setting up the variables.

The next step is also the same: using a for loop to calculate the total amount of values in the array. However, instead of one if statement with a formula from the previous method, it uses a larger number of if statements. The sixth if statements perform a check every time if the total value is still less or equal to T, if not they add one to counter and move to the next value in the array.

**using** System;

**namespace** Task1\_1

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set variables*

**int**[] A = { 3, 5, 2, 1, 7, 4 };

**int** T = 14;

**int** total = 0;

**int** answer = 0;

*// for loop to check total + n*

**for** (**int** i = 0; i < A.Length; i++)

{

**if** (total <= T)

{

total += A[0];

i++;

}

**if** (total <= T)

{

total += A[1];

i++;

}

**if** (total <= T)

{

total += A[2];

i++;

}

**if** (total <= T)

{

total += A[3];

i++;

}

**if** (total <= T)

{

total += A[4];

i++;

}

**if** (total <= T)

{

total += A[5];

i++;

}

**else**

{

*// output the answer*

answer += A[i];

Console.WriteLine("Total Items added: " + answer);

**break**;

}

}

}

}

}

**Comparison**

The first solution compares to the second one using the mathematical formula, instead of comparing each value. The main advantage of the first solution, over the second, is compatibility with an array that includes more values in them. Besides, the constant use of if statements will affect badly on the performance of the program (Sharma, 2019). Besides the difference in performance, each solution contains readable and understandable code with comments. According to common programming principles (GeeksForGeeks, 2020), simplicity is one of the main factors. However, the disadvantage of both solutions, is their algorithm of going from the first value, while If numbers were different, a higher amount could have been achieved.

**Task 2**

**First Solution**

The second is required to create a program, which checks if the string password meets the requirements. The first solution will require the System.Linq namespace, which allows using Enumerable.Any Method.

As the first step, the variables have been set at the beginning of the program. It contains variables such as string P, to store the password, bool valid to check if all requirements have been met and string special that includes special characters. The purpose of integer criteria is to add 1, every time when one requirement is met. Then, the program checks if the first password is equal or has more than seven characters., by using an if statement. The next if statement, uses char methods, to check if password meets required rules: has uppercase character, lowercase character and contains a letter or a number. For the next check, the program uses foreach loop, which identifies if password contains at least one unique character. The final examination is to make sure the password does not have more than three repeated characters.

If the value of criteria is equal to 3, that means the password has matched all requirements; otherwise – the program ends with an error.

**using** System;

**using** System.Linq; *// Using System.Linq!!!*

**namespace** Task2\_1

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set variables*

**string** P = Console.ReadLine();

**bool** valid;

**string** special = @"|!#$%&/()=?@~£{}.-;'<>\_,";

**int** criteria = 0; *// criteria starts with 0 and adds 1 when it meets each criteria*

*// checks if password is at least 7 characters long*

**if** (P.Length >= 7)

{

criteria++;

}

**else** *//(P.Length <= 6)*

{

Console.WriteLine("Error: your password contains less than seven characters!");

}

*// checks if pasword contains letter of numbers and if there are at least one upper and lower*

**if** (P.Any(**char**.IsUpper) && P.Any(**char**.IsLower) && P.Any(**char**.IsLetterOrDigit))

{

criteria++;

}

**else**

{

Console.WriteLine("Error: your password should contain at least one uppercase, lowercase and a number!");

}

*// checks is password contains at least one special character*

**foreach** (**var** item **in** special)

{

**if** (P.Contains(item))

{

criteria++;

}

}

**if** (criteria != 3)

{

Console.WriteLine("Error: your password should contain at least one special character!");

}

*// checks if there are more than three repeating characters in a row*

**for** (**int** i = 0; i < P.Length - 2; i++)

{

**if** (P[i] == P[i + 1] && P[i] == P[i + 2] && P[i] == P[i + 3])

{

*//Console.WriteLine("Error: There were more than three repreating characters in a row!");*

criteria = 0;

}

}

**if** (criteria != 3)

{

Console.WriteLine("Error: There were more than three repreating characters in a row!");

}

*// checks if password matched all criterias*

**if** (criteria == 3)

{

valid = **true**;

Console.WriteLine("Successfull: your password match all requirements");

}

**else**

{

valid = **false**;

Console.WriteLine("Unsuccessful: your password did not meet one or more requirements");

}

}

}

}

**Second solution**

The second solution does not use System.Linq namespace to perform a check of the requirements; however, it uses System.Text.RegularExpressions namespace to register Regex classes.

As the first step, the program contains variables, which will be used to initialise the check all requirements. For example, hasUpperChar will check if the string includes uppercase characters from A to Z. The second solution uses the same for loop to identify if not the same three characters has been used next to each other. At the final, the program contains else if statements to perform checks of using Regex variables and returns true or false, depending on the result.

**using** System;

**using** System.Text.RegularExpressions;

**namespace** Task2\_2

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

**string** P = Console.ReadLine();

*// set variables*

**bool** valid;

**int** maxRepeat = 0;

**var** hasNumber = new Regex(@"[0-9]+");

**var** hasUpperChar = new Regex(@"[A-Z]+");

**var** hasLowerChar = new Regex(@"[a-z]+");

**var** hasMiniMaxChars = new Regex(@".{7,}");

**var** hasSymbols = new Regex(@"[!@#$%^&\*()\_+=\[{\]};:<>|./?,-]");

*// checks if there are more than three repeating characters in a row*

**for** (**int** i = 0; i < P.Length- 2; i++)

{

**if** (P[i] == P[i + 1] && P[i] == P[i + 2] && P[i] == P[i + 3])

{

maxRepeat = 1;

}

}

*// checks if password matched all criterias*

**if** (!hasLowerChar.IsMatch(P))

{

Console.WriteLine("Error: your password should contain at least one lowercase character!");

valid = **false**;

}

**else** **if** (maxRepeat == 1)

{

Console.WriteLine("Error: There were more than three repeating characters in a row!");

valid = **false**;

}

**else** **if** (!hasUpperChar.IsMatch(P))

{

Console.WriteLine("Error: your password should contain at least one uppercase character!");

valid = **false**;

}

**else** **if** (!hasMiniMaxChars.IsMatch(P))

{

Console.WriteLine("Error: your password contains less than seven characters!");

valid = **false**;

}

**else** **if** (!hasNumber.IsMatch(P))

{

Console.WriteLine("Error: your password should contain at least one number!");

valid = **false**;

}

**else** **if** (!hasSymbols.IsMatch(P))

{

Console.WriteLine("Error: your password should contain at least one special character!");

valid = **false**;

}

**else**

{

valid = **true**;

Console.WriteLine("Successful: your password match all requirements");

}

}

}

}

**Comparison**

Each solution uses an addition namespace, which restricts the using of these techniques in other programming languages. When comparing their performance, both explanations could be implemented in a shorter way to increase performance. According to the tests (donnetperls, n.d), the method with a switch case performance better than if-else statements, when applying to a similar program. Another disadvantage of both solutions, if the password did not match all criteria – program end. This could be improved by using a loop, which will only return true when all requirements have been met. In terms of simplicity of both solutions, second solutions have clearer and well-presented structure compared to the first solution.

**Task 3**

**First solution**

The first solution for task 3 is using two additional namespaces: System.Collections.Generic and System.Linq. As the first step, the program has a variable integer array N, that contains all values. The second step is to create a List, where the program will include an array N. The List will allow to use crucial Distinct method, which removes duplicates from the array. At the final step, the foreach loop has been used to output all the values, after they have been sorted and all repeated values have been removed.

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**namespace** Assignment1

{

**class** Task3\_1

{

**static** **void** Main(**string**[] args)

{

**int**[] N = { 9, 88, 1, 9, 88, 87, 35, 12, 50, 23, 12, 1, 4, 9 };

List<**int**> arraysList = new List<**int**>(); *// (System.Collections.Generic)*

arraysList.AddRange(N);

**int**[] C = arraysList.Distinct().ToArray(); *// System.Linq*

Array.Sort(C);

**foreach** (**int** i **in** C)

{

Console.WriteLine(i);

}

}

}

}

**Second solution**

The second solution is very similar to the first solution. Both solutions are using the same namespaces and adding N array to the list to perform Distinct method. However, the main difference is that the second solution does not use the Array.Sort method, by using a nested for loop instead. The nested for loop represents a sorting formula of bubble method.

**using** System;

**using** System.Linq;

**using** System.Collections.Generic;

**namespace** Task3\_2

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set variables*

**int**[] N = { 9, 88, 1, 9, 88, 87, 35, 12, 50, 23, 12, 1, 4, 9 };

**int** tempValue;

*// for loop which separates numbers from smallest to biggest. Replaces Array.Sort(C)*

**for** (**int** j = 0; j <= N.Length - 2; j++)

{

**for** (**int** i = 0; i <= N.Length - 2; i++)

{

**if** (N[i] > N[i + 1])

{

tempValue = N[i + 1];

N[i + 1] = N[i];

N[i] = tempValue;

}

}

}

List<**int**> arraysList = new List<**int**>();

arraysList.AddRange(N);

**int**[] C = arraysList.Distinct().ToArray();

**for** (**int** k = 0; k < C.Length; k++)

{

Console.WriteLine(C[k]);

}

}

}

}

**Comparison**

Both solutions perform at a very similar level; however, as they both use additional namespaces, it restricts the opportunity to reuse these solutions in other programming languages. As the advantage of the first solution is that it has a much more readable structure and compares nested for loop method in the second solution.

**Task 4**

**First solution**

For the first solution, as the first step, the program has set variables. One of the crucial variables is a char with MinValue method. This method is used to check if the previous characters in a string are not the same. The second step in the algorithm is for a loop, which contains two if statements. The loop goes through each character of the string, compares one character to the next one, and add counter if they are the same. The final step is to compare the temporary variable to the maximum repeated values and output the result to the console.

**using** System;

**namespace** Task4\_1

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

**string** S = "XXXYXYXXYYYXYYYYXX";

**char** letter = S[0];

**int** count = 1; *// characters will count at least once*

**int** currentCount = 0;

**char** previousChar = **char**.MinValue;

**char**[] array = S.ToCharArray();

**for** (**int** i = 0; i < array.Length; i++)

{

**if** (array[i] != previousChar) *//*

{

currentCount = 1;

}

**else**

{

currentCount++;

}

**if** (currentCount >= count)

{

letter = array[i];

count = currentCount;

}

previousChar = array[i];

}

**string** routput = new **string**(letter, count);

Console.WriteLine($"Number of consecutive characters: {count}. Substring: {routput}");

}

}

}

**Second solution**

The second solution follows the same steps as the first solution. However, this program contains a while loop, instead of a for loop and uses slightly different formula method.

**using** System;

**namespace** Task4\_2

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

**string** S = "XXXYXYXXYYYXYYYYXX";

**int** count = 1; *// characters will count at least once*

**char** letter = ' '; *// replacing an empty letter with most constant letter*

**int** tempCount = 0;

**char** previousChar = **char**.MinValue;

**char**[] array = S.ToCharArray();

*// While loop to find the highest consecutive number*

**int** i = 0;

**while** (i < S.Length)

{

**if** (array[i] == previousChar)

{

tempCount++;

}

**else**

{

tempCount = 1;

}

**if** (tempCount >= count)

{

letter = array[i];

count = tempCount;

}

previousChar = array[i];

i++;

}

*// outtput values*

**string** routput = new **string**(letter, count);

Console.WriteLine("Number of consecutive characters: {0}. Substring: {1}", count, routput);

}

}

}

**Comparison**

In comparison to the first solution, the main difference is using a different type of loop. A for loop commonly is used when you need to perform something in a particular range. On the other hand, while loop is used when you need to execute something an indefinite number of times before certain conditions are met. This could be applied if the string would have been longer or unknown size. However, in terms of performance, there is no visible difference when using a test to this program. Besides, some people mind finds a while loop much easier to read, than a for a loop.

**Task 5**

**First solution**

As the initial step for a first solution, variables have been set at the beginning of the program. One is a string S, containing the principal value, another is char temp, which will be holding a temporary value. Then, the program includes two nested for loops. In each of the loop, has been used String.Replace method, to shuffle values of the string between each other.

**using** System;

**namespace** Task5

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set variables*

**string** S = "ABC";

**char** temp = ' ';

*// for loop and replace method to replace each letter with temporary character and output the result*

**for** (**int** i = 0; i < S.Length; i++)

{

S = S.Replace(S[1], temp).Replace(S[2], S[1]).Replace(temp, S[2]);

**for** (**int** j = 1; j < S.Length; j++)

{

S = S.Replace(S[1], temp).Replace(S[2], S[1]).Replace(temp, S[2]);

Console.WriteLine(S);

}

**for** (**int** k = 2; k < S.Length; k++)

{

S = S.Replace(S[0], temp).Replace(S[1], S[0]).Replace(temp, S[1]);

}

}

}

}

}

**Second solution**

The idea of the second solution method, compared to the first one, is to avoid String.Replace method. The second solution contains same variables, expect new array for manipulation with a sting S. The program has same two nested for loops, however, uses an equal sign, to store a temporary letter from a string.

**using** System;

**namespace** Task5\_2

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

*// set vatiables*

**string** S = "ABC";

**char**[] array = S.ToCharArray();

**char** temp;

*// nested for loops with a temp variable to hold a character, while replacing places*

**for** (**int** i = 0; i < S.Length; i++)

{

**for** (**int** j = 0; j < S.Length ; j++)

{

temp = array[j];

array[j] = array[i];

array[i] = temp;

}

Console.WriteLine(array);

**for** (**int** k = 1; k < S.Length; k++)

{

temp = array[i];

array[i] = array[k];

array[k] = temp;

}

Console.WriteLine(array); *// output to console*

}

}

}

}

**Comparison**

The main difference between these solutions that the second solution does not use a predefined method. However, it has a disadvantage of not fully outputting all values. The second solution duplicates one answer twice. Besides, the first solution has a more straightforward readable code structure and performs well.

References:

Stack Overflow. 2015. *Find the longest repetition character in String*. [online] Available at: <https://stackoverflow.com/questions/32432281/find-the-longest-repetition-character-in-string>

Stack Overflow. 2014. *Check if string have uppercase, lowercase and number*. [online] Available at: https://stackoverflow.com/questions/27402126/check-if-string-have-uppercase-lowercase-and-number/27402159

Stack Overflow. 2013. *Detect if a string contains uppercase characters*. [online] Available at: <https://stackoverflow.com/questions/20032450/detect-if-a-string-contains-uppercase-characters>

Stack Overflow. 2013. *how to check if string has more than two repeating characters*. [online] Available at: <https://stackoverflow.com/questions/14634477/how-to-check-if-string-has-more-than-two-repeating-characters>

Sharma, A., 2019. *Performance Consideration For C# Conditional Statements*. [online] C-sharpcorner.com. Available at: <https://www.c-sharpcorner.com/article/performance-consideration-for-c-sharp-conditional-statements/>

GeeksforGeeks. 2020. *7 Common Programming Principles That Every Developer Must Follow - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/>

Dotnetperls.com. n.d. [online] Available at: <https://www.dotnetperls.com/if-switch-performance>

Bibliography:

Stack Overflow. 2010. *Which loop is faster, while or for?*. [online] Available at: <https://stackoverflow.com/questions/3629174/which-loop-is-faster-while-or-for>

Regex to include one (lowercase, g., 2017. *Regex to include one (lowercase, uppercase, number, given special characters) in c# Azure function*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/45957599/regex-to-include-one-lowercase-uppercase-number-given-special-characters-in>

Stack Overflow. 2010. *Check for special characters (/\*-+\_@&$#%) in a string?*. [online] Available at: <https://stackoverflow.com/questions/4503542/check-for-special-characters-in-a-string>

Stack Overflow. 2011. *Regular Expression for password validation*. [online] Available at: <https://stackoverflow.com/questions/5859632/regular-expression-for-password-validation>